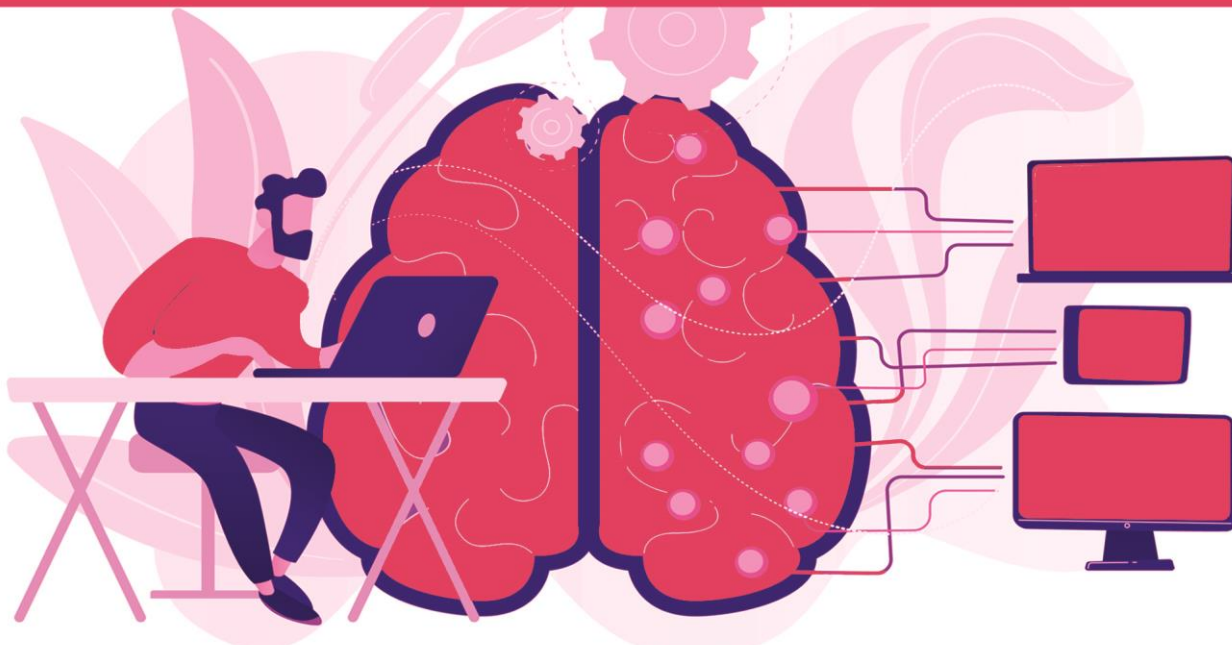
التعمق في التعلم العميق

الجزء الثاني: تقنيات التعلم العميق الحديثة

تأليف : أنـــتون زانخ وآخرون

ترجمة : د. علاء طعيمة



بـــمه تعالى

التعمق فى التعلم العميق

الجزء الثاني

تقنيات التعلم العميق الحديثة

تأليف:

آـــتون زانغ وآخرون

ترجمة:

د. علاء طعيمة

مقدمة المترجم

على مدى السنوات القليلة الماضية، طور فريق من علماء أمازون كتاباً "Dive into Deep Learning" يكتسب شعبية بين الطلاب والمطورين الذين ينجذبون إلى مجال التعلم العميق المزدهر، وهو مجموعة فرعية من التعلم الآلي تركز على الشبكات العصبية الاصطناعية واسعة النطاق.

عند انتهائي من قراءه هذا الكتاب، احببت ان اترجم هذا الكتاب وأشارككم هذه الترجمة لان هناك عدد من الأشياء الرائعة حول هذا الكتاب وأكثر ما يعجبني هو أنه يغطي كل مجالات التعلم العميق تقريبًا مثلاً للمبتدئين هناك فصول مثل الشبكات العصبية والبيرسيبترون متعدد الطبقات والانحدار والتصنيف بالإضافة الى المفاهيم الأساسية من الجبر الخطي، وحساب التفاضل والتكامل، والاحتمال الى فصول متقدمة مثل الشبكات العصبية الالتفافية CNN، تم تضمين الشبكات العصبية المتكررة RNN والرؤية الحاسوبية CV ومعالجة اللغات الطبيعية NLP أيضًا.

هذا كتاب تفاعلي مفتوح المصدر مقدمفي شكل فريد يدمج النص والرياضيات والكود، ويدعم الآن أطر برمجة TensorFlow وPyTorch وApache MXNet، والتي تمت صياغتها بالكامل من خلال Jupyter Notebook.

يمكن تقسيم الكتاب إلى ثلاثة أجزاء تقريبًا، لقد قمنافي الوقت الحالي بترجمة الجزء الأول والذي يشمل الأساسيات والمقدمات **والجزء الثاني والذي يشمل التقنيات الحديثة للتعلم العميق** وان شاء اللهفي المستقبل القريب سنقوم بترجمة الجزء الثالث والذي يشمل مواضيع مثل الرؤية الحاسوبية ومعالجة اللغات الطبيعية.

لقد اخترت كتاب "Dive into Deep Learning" لما رأيته من جودة هذا الكتاب، وللمنهجية التي اتبعها المؤلفونفي ترتيبه وبساطة شرحه. لقد حاولت قدر المستطاع ان اخرج بترجمة ذات جودة عالية، ومع هذا يبقى عملاً بشرياً يحتمل النقص، فاذاكان لديك أي ملاحظات حول هذا الكتاب، فلا تتردد بمراسلتنا عبر بريدنا الالكتروني alaa.taima@qu.edu.iq .

نأمل ان يساعد هذا الكتاب كل من يريد ان يدخلفي مجال التعلم العميق ومساعدة القارئ العربي على تعلم هذا المجال. اسأل الله التوفيقفي هذا العمل لأثراء المحتوى العربي الذي يفتقر أشد الافتقار إلى محتوى جيد ورصينفي مجال الذكاء الاصطناعي وتعلم الالة والتعلم العميق. ونرجو لك الاستمتاع مع التعلم العميق ولا تنسونا من صالح الدعاء.

د. علاء طعيمة/كلية علوم الحاسوب وتكنولوجيا المعلومات/جامعة القادسية/العراق

المحتويات

6

دليل البنائين

1. دليل البنائين Builders' Guide

إلى جانب مجموعات البيانات العملاقة والأجهزة القوية، لعبت أدوات البرامج الرائعة دورًا لا غنى عنهفي التقدم السريع للتعلم العميق. بدءًا من مكتبة Theano الرائدة التي تم إصدارهافي عام 2007، مكنت الأدوات مفتوحة المصدر المرنة الباحثين من وضع نماذج أولية للنماذج بسرعة، وتجنب العمل المتكرر عند إعادة تدوير المكونات القياسية مع الحفاظ على القدرة على إجراء تعديلات منخفضة المستوى. بمرور الوقت، تطورت مكتبات التعلم العميق لتقديم تجريدات خشنة بشكل متزايد. تمامًا كما انتقل مصممو أشباه الموصلات من تحديد الترانزستورات إلى الدوائر المنطقية إلى كتابة التعليمات البرمجية، انتقل باحثو الشبكات العصبية من التفكيرفي سلوك الخلايا العصبية الاصطناعية الفردية إلى تصور الشبكات من حيث الطبقات الكاملة، والآن غالبًا ما يصممون البنى مع وضع الكتل الخشنة coarser blocksفي الاعتبار.

حتى الآن، قدمنا بعض مفاهيم التعلم الآلي الأساسية، لتكثيف نماذج التعلم العميق كاملة الوظائف.في الفصل الأخير، قمنا بتنفيذ كل مكون من مكونات MLP من البداية وحتى أوضحنا كيفية الاستفادة من واجهات برمجة التطبيقات API عالية المستوى لطرح نفس النماذج دون عناء. للوصول إلى هذا الحد بهذه السرعة، استدعينا المكتبات، لكننا تخطينا المزيد من التفاصيل المتقدمة حول كيفية عملها.في هذا الفصل، سنقوم بكشف الستارة، والبحث بشكل أعمقفي المكونات الرئيسية لحساب التعلم العميق، وهي بناء النموذج model construction، والوصول إلى المعلمات وتهيئتها، وتصميم الطبقات والكتل المخصصة، وقراءة النماذج وكتابتها على القرص، والاستفادة من وحدات معالجة الرسومات GPU لتحقيق التعجيل الدراماتيكي. ستنقلك هذه الأفكار من مستخدم نهائي end user إلى مستخدم قوي power user، مما يمنحك الأدوات اللازمة لجني فوائد مكتبة التعلم العميق الناضجة مع الاحتفاظ بالمرونةفي تنفيذ نماذج أكثر تعقيدًا، بمافي ذلك النماذج التي تخترعها بنفسك!في حين أن هذا الفصل لا يقدم أي نماذج أو مجموعات بيانات جديدة، فإن فصول النمذجة المتقدمة التي تليها تعتمد بشكل كبير على هذه التقنيات.

6.1 الطبقات والوحدات النمطية Layers and Modules

عندما قدمنا الشبكات العصبية لأول مرة، ركزنا على النماذج الخطية بإخراج واحد. هنا، يتكون النموذج بأكمله من خلية عصبية واحدة فقط. لاحظ أن خلية عصبية واحدة (1) تأخذ مجموعة من المدخلات؛ (2) يولد مخرجات قياسية متطابقة. و (3) لديه مجموعة من المعلمات المرتبطة التي يمكن تحديثها لتحسين بعض الدوال الموضوعية ذات الأهمية. بعد ذلك، بمجرد أن بدأنا التفكيرفي الشبكات ذات المخرجات المتعددة، استفدنا من الحساب المتجهي vectorized arithmetic لتوصيف طبقة كاملة من الخلايا العصبية. تمامًا مثل الخلايا العصبية الفردية، فإن

الطبقات (1) تأخذ مجموعة من المدخلات، (2) تولد مخرجات مقابلة، و (3) موصوفة بمجموعة من المعلمات القابلة للضبط tunable parameters. عندما عملنا من خلال انحدار softmax، كانت طبقة واحدة هي نفسها النموذج. ومع ذلك، حتى عندما قدمنا لاحقًا MLPs، لا يزال بإمكاننا التفكيرفي النموذج على أنه يحتفظ بنفس البُنية الأساسية.

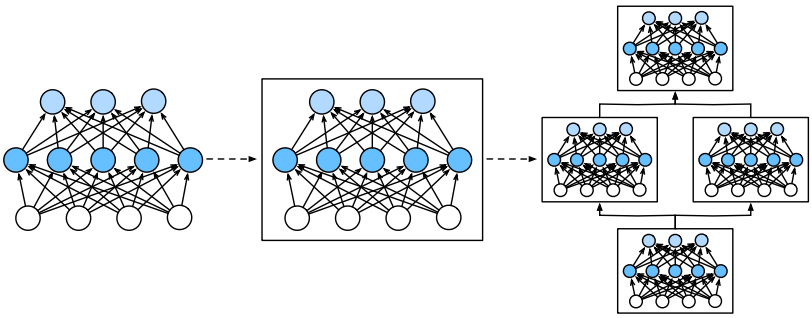
ومن المثير للاهتمام، بالنسبة لـ MLPs، أن كل من النموذج بأكمله والطبقات المكونة له تشتركفي هذا الهيكل. يأخذ النموذج بأكمله المدخلات الأولية (الميزات)، ويولد المخرجات (التنبؤات)، ويمتلك المعلمات (المعلمات المدمجة من جميع الطبقات المكونة). وبالمثل، فإن كل طبقة فردية تستوعب المدخلات (التي توفرها الطبقة السابقة) تولد النواتج (المدخلات إلى الطبقة اللاحقة)، وتمتلك مجموعة من المعلمات القابلة للضبط التي يتم تحديثها وفقًا للإشارة التي تتدفق للخلف من الطبقة اللاحقة.

بينما قد تعتقد أن الخلايا العصبية والطبقات والنماذج تعطينا أفكارًا تجريدية كافية لمواصلة أعمالنا، فقد اتضح أننا غالبًا ما نجد أنه من المناسب التحدث عن مكونات أكبر من طبقة فردية ولكنها أصغر من النموذج بأكمله. على سبيل المثال، تمتلك بنية ResNet-152، التي تحظى بشعبية كبيرةفي الرؤية الحاسوبية، مئات الطبقات. تتكون هذه الطبقات من أنماط متكررة لمجموعات من الطبقات. يمكن أن يصبح تنفيذ مثل هذه الشبكة طبقة واحدةفي كل مرة مملاً. هذا القلق ليس مجرد افتراض ــ أنماط التصميم هذه شائعةفي الممارسة. فازت بُنية ResNet المذكورة أعلاهفي مسابقات الرؤية الحاسوبية لعام ImageNet 2015 وCOCO لكل من التعرف والكشف (He et al.، 2016) ولا تزال بنية الانتقال للعديد من مهام الرؤية. إن البنى المماثلة التي يتم فيها ترتيب الطبقاتفي أنماط متكررة مختلفة موجودة الآنفي كل مكانفي مجالات أخرى، بمافي ذلك معالجة اللغة الطبيعية والكلام.

لتنفيذ هذه الشبكات المعقدة، نقدم مفهوم وحدة النمطية للشبكة العصبية module. يمكن أن تصف الوحدة النمطية طبقة واحدة، أو مكونًا يتكون من طبقات متعددة، أو النموذج بأكمله! تتمثل إحدى فوائد العمل مع تجريد الوحدة النمطية module abstractionفي أنه يمكن دمجها في قطع أثرية أكبر، غالبًا بشكل متكرر. وهذا موضحفي الشكل 6.1.1. من خلال تحديد الكود لإنشاء وحدات ذات تعقيد افتراضي عند الطلب، يمكننا كتابة كود مضغوط بشكل مدهش مع الاستمرارفي تنفيذ الشبكات العصبية المعقدة.

من وجهة نظر البرمجة، يتم تمثيل الوحدة النمطية بواسطة فئة class. يجب أن تحدد أي فئة فرعية منه طريقة انتشار أمامية تحول مدخلاتها إلى مخرجات ويجب أن تخزن أي معلمات ضرورية. لاحظ أن بعض الوحدات لا تتطلب أي معلمات على الإطلاق. أخيرًا، يجب أن تمتلك وحدة نمطية طريقة الانتشار الخلفي backpropagation، لأغراض حساب التدرجات

gradients. لحسن الحظ، نظرًا لبعض سحر ما وراء الكواليس الذي يوفره التفاضل التلقائي auto differentiation (المقدمفي القسم 2.5) عند تحديد الوحدة النمطية الخاصة بنا، نحتاج فقط إلى القلق بشأن المعلمات وطريقة الانتشار الأمامية forward propagation.



الشكل 6.1.1 يتم دمج الطبقات المتعددةفي وحدات مكونة أنماطًا متكررة لنماذج أكبر.

للبدء، نعيد النظرفي الكود الذي استخدمناه لتنفيذ MLPs (القسم 5.1). يُنشئ الكود التالي شبكة ذات طبقة مخفية واحدة متصلة بالكامل مع 256 وحدة وتنشيط ReLU، متبوعة بطبقة إخراج متصلة بالكامل مع 10 وحدات (بدون دالة تنشيط).

```
import tensorflow as tf

net = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dense(10),
])

X = tf.random.uniform((2, 20))
net(X).shape
```

```
TensorShape([2, 10])
```

في هذا المثال، قمنا ببناء نموذجنا عن طريق إنشاء مثيل لـ `keras.models.Sequential`، مع طبقات بالترتيب الذي يجب أن يتم تنفيذه بها كوسيطات arguments. باختصار، يحدد `Sequential` نوعًا خاصًا من `keras.Model`، وهو الكلاس (الفئة) الذي يقدم وحدةفي Keras. يحتفظ بقائمة مرتبة من النماذج المكونة `Models` constituent. لاحظ أن كل طبقة من الطبقتين المتصلتين بالكامل هي مثيل لكلاس `Dense` التي تعدفي حد ذاتها فئة فرعية من النموذج. كما أن طريقة الانتشار الامامي (الاستدعاء `call`) بسيطة بشكل ملحوظ: فهي تربط كل وحدةفي القائمة معًا، وتمرير ناتج كل منها كمدخل إلى التالي. لاحظ أنه حتى الآن، كنا نستدعي نماذجنا عبر البناء `net(X)`

للحصول على مخرجاتها. هذا في الواقع مجرد اختصار لـ `net.call(X)` ، خدعة بايثون الرائعة التي تم تحقيقها عبر طريقة `__call__` لكلاس الوحدة النمطية.

6.1.1. الوحدة النمطية المخصصة A Custom Module

ربما تكون أسهل طريقة لتطوير الحدس حول كيفية عمل الوحدة النمطية هي أن ننفذها بأنفسنا. قبل أن ننفذ الوحدة النمطية المخصصة custom module الخاصة بنا، نلخص بإيجاز الوظائف الأساسية التي يجب أن توفرها كل وحدة:

1. استوعب بيانات الإدخال كوسيطات arguments لطريقة الانتشار الأمامي forward propagation.
2. قم بتوليد مخرجات بجعل طريقة الانتشار الأمامي ترجع قيمة. لاحظ أن الإخراج قد يكون له شكل مختلف عن المدخلات. على سبيل المثال ، تستوعب أول طبقة متصلة بالكامل fully connected layer في نموذجنا أعلاه مدخلات ذات بُعد افتراضي ولكنها تُرجع ناتجًا من البعد 256.
3. احسب التدرج gradient لمخرجاته فيما يتعلق بمدخلاته، والتي يمكن الوصول إليها عبر أسلوب الانتشار الامامي backpropagation الخاص به. عادةً ما يحدث هذا تلقائيًا.
4. قم بتخزين وتوفير الوصول إلى تلك المعلمات اللازمة لتنفيذ حساب الانتشار الأمامي.
5. قم بتهيئة معلمات النموذج حسب الحاجة.

في المقتطف التالي، نقوم ببرمجة وحدة نمطية من البداية تتوافق مع MLP بطبقة مخفية واحدة تحتوي على 256 وحدة مخفية وطبقة إخراج 10 أبعاد. لاحظ أن فئة MLP أدناه ترث الفئة التي تمثل وحدة نمطية. سوف نعتمد بشكل كبير على طرق الفئة الأصلية parent class، ونزود المُنشئ الخاص بنا فقط (طريقة `__init__` في بايثون) وطريقة الانتشار الأمامي.

```
class MLP(tf.keras.Model):
    def __init__(self):
        # Call the constructor of the parent class
tf.keras.Model to perform
        # the necessary initialization
        super().__init__()
        self.hidden = tf.keras.layers.Dense(units=256,
activation=tf.nn.relu)
        self.out = tf.keras.layers.Dense(units=10)

    # Define the forward propagation of the model, that
is, how to return the
```

```
    # required model output based on the input X
    def call(self, X):
        return self.out(self.hidden((X)))
```

دعنا نركز أولاً على طريقة الانتشار الأمامي. لاحظ أنه يأخذ X كمدخل، ويحسب التمثيل المخفي مع تطبيق دالة التنشيط، ويخرج سجلاته logits.في تطبيق MLP هذا، تعد كلتا الطبقتين متغيرات حالة. لمعرفة سبب كون ذلك معقولاً، تخيل إنشاء مثيل لـ MLPs، net1 و net2، وتدريبهما على بيانات مختلفة. بطبيعة الحال، نتوقع منهم أن يمثلوا نموذجين متعلمين مختلفين.

نقوم بإنشاء مثيل لطبقات MLPفي المُنشئ ومن ثم استدعاء هذه الطبقاتفي كل استدعاء لطريقة الانتشار الأمامي. لاحظ بعض التفاصيل الأساسية. أولاً، تستدعي طريقة __init__ المخصصة لدينا طريقة __init__ الخاصة بالفئة الأصلية عبر super().__init__() يجنبنا ألم إعادة صياغة الكود المعياري المطبق على معظم الوحدات. ثم نقوم بإنشاء مثيل للطبقتين المتصلتين تمامًا، ونقوم بتعيينهما إلى self.hidden و self.out. لاحظ أنه ما لم ننفذ طبقة جديدة، فلا داعي للقلق بشأن طريقة backpropagation أو تهيئة المعلمة. سيقوم النظام بإنشاء هذه الطرق تلقائيًا. دعونا نجرب هذا.

```
net = MLP()
net(X).shape
```

```
TensorShape([2, 10])
```

من المزايا الرئيسية لتجريد الوحدة تعدد استخداماتها versatility. يمكننا تصنيف وحدة نمطية فرعية لإنشاء طبقات (مثل فئة الطبقة المتصلة بالكامل)، أو نماذج كاملة (مثل فئة MLP أعلاه)، أو مكونات مختلفة من التعقيد الوسيط. نستغل هذا التنوعفي الفصول التالية، على سبيل المثال عند معالجة الشبكات العصبية التلافيفية convolutional neural networks.

6.1.2. الوحدة النمطية المتسلسلة The Sequential Module

يمكننا الآن إلقاء نظرة فاحصة على كيفية عمل الكلاس التسلسلي Sequential class. تذكر أن Sequential تم تصميمه لربط الوحدات الأخرى معًا بطريقة سلسلة ديزي daisy-chain. لبناء MySequential المبسطة الخاصة بنا، نحتاج فقط إلى تحديد طريقتين رئيسيتين: 1. طريقة لإلحاق الوحدات واحدة تلو الأخرى بالقائمة. 2. طريقة انتشار أمامية لتمرير مُدخل من خلال سلسلة الوحدات النمطية، بنفس الترتيب الذي تم إلحاقه به.

يوفر كلاس MySequential التالي الوظيفة نفسها للكلاس Sequential الافتراضي.

```
class MySequential(tf.keras.Model):
    def __init__(self, *args):
        super().__init__()
        self.modules = args
```

```
    def call(self, X):
        for module in self.modules:
            X = module(X)
        return X
```

عندما يتم استدعاء طريقة الانتشار الأمامي الخاصة بـ `MySequential`، يتم تنفيذ كل وحدة نمطية مضافة بالترتيب الذي تمت إضافتها به. يمكننا الآن إعادة تطبيق MLP باستخدام كلاس `MySequential`.

```
net = MySequential(
    tf.keras.layers.Dense(units=256,
activation=tf.nn.relu),
    tf.keras.layers.Dense(10))
net(X).shape
```

```
TensorShape([2, 10])
```

لاحظ أن استخدام `MySequential` مطابق للرمز الذي كتبناه سابقًا لكلاس `Sequential` (كما هو موضح في القسم 5.1).

6.1.3. تنفيذ التعليمات البرمجية في طريقة النشر الأمامي Executing Code in the Forward Propagation Method

تجعل الفئة (الكلاس) Sequential إنشاء النموذج أمرًا سهلاً، مما يسمح لنا بتجميع بُنى جديدة دون الحاجة إلى تحديد فئتنا الخاصة. ومع ذلك، ليست كل الأبنية عبارة عن سلاسل أقحوان بسيطة. عندما يتطلب الأمر مزيدًا من المرونة، سنرغب في تحديد الكتل الخاصة بنا. على سبيل المثال، قد نرغب في تنفيذ تدفق التحكم control flow في بايثون داخل طريقة الانتشار الأمامي. علاوة على ذلك، قد نرغب في إجراء عمليات حسابية عشوائية، وليس مجرد الاعتماد على طبقات الشبكة العصبية المحددة مسبقًا.

ربما لاحظت أنه حتى الآن، عملت جميع العمليات في شبكاتنا على عمليات تنشيط شبكتنا ومعلماتها. ومع ذلك، في بعض الأحيان، قد نرغب في دمج المصطلحات التي ليست نتيجة الطبقات السابقة ولا المعلمات القابلة للتحديث. نسمي هذه المعلمات الثابتة constant parameters. لنفترض على سبيل المثال أننا نريد طبقة تحسب الدالة $f(\mathbf{x}, \mathbf{w}) = c \cdot \mathbf{w}^\top \mathbf{x}$، حيث $\mathbf{x}$ الإدخال، و$\mathbf{w}$ هي المعلمة الخاصة بنا، وc هي ثابتة محددة لم يتم تحديثها أثناء التحسين. لذلك قمنا بتطبيق فئة `FixedHiddenMLP` على النحو التالي.

```
class FixedHiddenMLP(tf.keras.Model):
    def __init__(self):
        super().__init__()
```

```
        self.flatten = tf.keras.layers.Flatten()
        # Random weight parameters created with
`tf.constant` are not updated
        # during training (i.e., constant parameters)
        self.rand_weight =
tf.constant(tf.random.uniform((20, 20)))
        self.dense = tf.keras.layers.Dense(20,
activation=tf.nn.relu)

    def call(self, inputs):
        X = self.flatten(inputs)
        # Use the created constant parameters, as well
as the `relu` and
        # `matmul` functions
        X = tf.nn.relu(tf.matmul(X, self.rand_weight) +
1)
        # Reuse the fully connected layer. This is
equivalent to sharing
        # parameters with two fully connected layers
        X = self.dense(X)
        # Control flow
        while tf.reduce_sum(tf.math.abs(X)) > 1:
            X /= 2
        return tf.reduce_sum(X)
```

في نموذج `FixedHiddenMLP` هذا، نقوم بتنفيذ طبقة مخفية يتم تهيئة أوزانها (`self.rand_weight`) بشكل عشوائي عند إنشاء مثيل لها ثم تصبح ثابتة بعد ذلك. هذا الوزن ليس معلمة نموذجية وبالتالي لا يتم تحديثه أبدًا عن طريق backpropagation. تقوم الشبكة بعد ذلك بتمرير ناتج هذه الطبقة "الثابتة" عبر طبقة متصلة بالكامل.

لاحظ أنه قبل إعادة الإخراج، فعل نموذجنا شيئًا غير عادي. قمنا بتشغيل حلقة while، واختبارنا على الشرط أن معيارها ℓ_1 أكبر من 1 ، وقسمنا متجه الإخراج على 2 حتى يستوفي الشرط. أخيرًا، قمنا بإرجاع مجموع الإدخالاتفي X. على حد علمنا، لا توجد شبكة عصبية قياسية تقوم بهذه العملية. لاحظ أن هذه العملية المحددة قد لا تكون مفيدةفي أي مهمةفي العالم الحقيقي. هدفنا هو فقط أن نوضح لك كيفية دمج رمز تعسفي arbitrary code في تدفق حسابات الشبكة العصبية الخاصة بك.

```
net = FixedHiddenMLP()
net(X)
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.81634015>
```

يمكننا المزج والتوفيق بين الطرق المختلفة لتجميع assembling الوحدات النمطية معًا.في المثال التالي، نقوم بدمج الوحدات النمطية ببعض الطرق الإبداعية.

```
class NestMLP(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.net = tf.keras.Sequential()
        self.net.add(tf.keras.layers.Dense(64,
activation=tf.nn.relu))
        self.net.add(tf.keras.layers.Dense(32,
activation=tf.nn.relu))
        self.dense = tf.keras.layers.Dense(16,
activation=tf.nn.relu)

    def call(self, inputs):
        return self.dense(self.net(inputs))

chimera = tf.keras.Sequential()
chimera.add(NestMLP())
chimera.add(tf.keras.layers.Dense(20))
chimera.add(FixedHiddenMLP())
chimera(X)
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.63493085>
```

6.1.4. الملخص

- الطبقات Layers عبارة عن وحدات نمطية modules.
- يمكن أن تشتمل العديد من الطبقات على وحدة نمطية.
- يمكن أن تشتمل العديد من الوحدات على وحدة نمطية.
- يمكن أن تحتوي الوحدة النمطية على كود.
- تقوم الوحدات النمطية بالكثير المهام، بمافي ذلك تهيئة المعلمات parameter initialization والانتشار الخلفي backpropagation.
- يتم التعامل مع التسلسل المتسلسل Sequential concatenations للطبقات والوحدات النمطية بواسطة الوحدة النمطية التسلسلية `Sequential`.

6.1.5. التمارين

1. ما أنواع المشاكل التي ستحدث إذا قمت بتغيير `MySequential` لتخزين الوحداتفي قائمة بايثون؟

2. قم بتنفيذ وحدة نمطية تأخذ وحدتين كوسيطة، على سبيل المثال `net1` و `net2` وإرجاع الإخراج المتسلسل لكلتا الشبكتينفي الانتشار الأمامي. وهذا ما يسمى أيضًا بالوحدة المتوازية parallel module.
3. افترض أنك تريد ربط مثيلات متعددة multiple instances من نفس الشبكة. قم بتنفيذ دالة المصنع factory function التي تنشئ مثيلات متعددة لنفس الوحدة وتبني منها شبكة أكبر.

6.2 إدارة المعلمات Parameter Management

بمجرد اختيار البنية architecture وتعيين المعلمات الفائقة hyperparameters الخاصة بنا، ننتقل إلى حلقة التدريب training loop، حيث يتمثل هدفنافي العثور على قيم المعلمات التي تقلل من دالة الخسارة (الخطأ) لدينا. بعد التدريب، سنحتاج إلى هذه المعلمات لعمل تنبؤات مستقبلية. بالإضافة إلى ذلك، نرغبفي بعض الأحيانفي استخراج المعلمات إما لإعادة استخدامها في سياق آخر، أو لحفظ نموذجنا على القرص بحيث يمكن تنفيذهفي برامج أخرى، أو للفحص على أمل اكتساب الفهم العلمي.

في معظم الأوقات، سنكون قادرين على تجاهل التفاصيل الدقيقة لكيفية الإعلان عن المعلمات والتلاعب بها، بالاعتماد على أطر التعلم العميق للقيام بالرفع الثقيل. ومع ذلك، عندما نبتعد عن البنى المكدسة بطبقات قياسية، سنحتاج أحيانًا إلى الدخولفي أعشاب إعلان المعلمات ومعالجتها.في هذا القسم، نغطي ما يلي:

- الوصول إلى معلمات التصحيح debugging والتشخيصات diagnostics والتصورات visualizations.
- مشاركة المعلمات Sharing parameters عبر مكونات النموذج المختلفة.

نبدأ بالتركيز على MLP بطبقة واحدة مخفية.

```
import tensorflow as tf

net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4, activation=tf.nn.relu),
    tf.keras.layers.Dense(1),
])

X = tf.random.uniform((2, 4))
net(X).shape
```

```
TensorShape([2, 1])
```

6.2.1. الوصول إلى المعلمة Parameter Access

لنبدأ بكيفية الوصول إلى المعلمات من النماذج التي تعرفها بالفعل. عندما يتم تعريف نموذج عبر الفئة `Sequential`، يمكننا أولاً الوصول إلى أي طبقة عن طريق فهرستهافي النموذج كما لو كانت قائمة list. يتم وضع معلمات كل طبقة بشكل ملائمفي جدولها. يمكننا فحص معلمات الطبقة الثانية المتصلة بالكامل على النحو التالي.

```
net.layers[2].weights
```

```
[<tf.Variable 'dense_1/kernel:0' shape=(4, 1)
dtype=float32, numpy=
 array([[ 0.2706747],
        [ 0.7514136],
        [-0.7881366],
        [-0.1292265]], dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32,
numpy=array([0.], dtype=float32)>]
```

يمكننا أن نرى أن هذه الطبقة المتصلة بالكامل تحتوي على معلمتين، تقابل أوزان weights تلك الطبقة وانحيازاتها biases، على التوالي.

6.2.1.1. المعلمات المستهدفةTargeted Parameters

لاحظ أنه يتم تمثيل كل معلمة كمثيل instance لفئة (كلاس) المعلمة parameter. للقيام بأي شيء مفيد مع المعلمات، نحتاج أولاً إلى الوصول إلى القيم العددية الأساسية. هناك عدة طرق للقيام بذلك. بعضها أبسط بينما البعض الآخر أكثر عمومية. يستخرج الكود التالي التحيز من طبقة الشبكة العصبية الثانية، والتي تعرض مثيل فئة المعلمة، وتصل إلى قيمة هذه المعلمة بشكل أكبر.

```
type(net.layers[2].weights[1]),
tf.convert_to_tensor(net.layers[2].weights[1])
```

```
(tensorflow.python.ops.resource_variable_ops.ResourceVar
iable,
 <tf.Tensor: shape=(1,), dtype=float32,
numpy=array([0.], dtype=float32)>)
```

6.2.1.2. كل المعلمات في وقت واحد All Parameters at Once

عندما نحتاج إلى إجراء عمليات على جميع المعلمات، فإن الوصول إليها واحدًا تلو الآخر يمكن أن يصبح مملاً. يمكن أن يصبح الموقف صعبًا بشكل خاص عندما نعمل مع وحدات أكثر تعقيدًا (على سبيل المثال، الوحدات المتداخلة nested modules)، نظرًا لأننا سنحتاج إلى التكرار من خلال الشجرة بأكملها لاستخراج معلمات كل وحدة فرعية. أدناه نوضح الوصول إلى معلمات جميع الطبقات.

```
net.get_weights()
```

```
[array([[-0.25718492, -0.02684402,  0.38722616, -
0.21718812],
        [ 0.8491538 ,  0.2304619 ,  0.37694377,
0.5665582 ],
        [-0.4560371 ,  0.7668019 , -0.52032065,
0.6611255 ],
        [ 0.04601806,  0.5825514 ,  0.7364692 ,
0.70636266]],
       dtype=float32),
 array([0., 0., 0., 0.], dtype=float32),
 array([[ 0.2706747],
        [ 0.7514136],
        [-0.7881366],
        [-0.1292265]], dtype=float32),
 array([0.], dtype=float32)]
```

6.2.2. المعلمات المرتبطة Tied Parameters

في كثير من الأحيان، نريد مشاركة المعلمات عبر طبقات متعددة. دعونا نرى كيف نفعل ذلك بأناقة. فيما يلي نخصص طبقة متصلة بالكامل ثم نستخدم معلماتها على وجه التحديد لتعيين تلك الخاصة بطبقة أخرى. نحتاج هنا إلى تشغيل الانتشار الأمامية net(X) قبل الوصول إلى المعلمات.

```
# tf.keras behaves a bit differently. It removes the
duplicate layer
# automatically
shared = tf.keras.layers.Dense(4, activation=tf.nn.relu)
net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    shared,
    shared,
    tf.keras.layers.Dense(1),
])
net(X)
# Check whether the parameters are different
print(len(net.layers) == 3)
```

```
True
```

يوضح هذا المثال أن معلمات الطبقة الثانية والثالثة مرتبطة. إنهما ليسا متساويين فقط، بل يتم تمثيلهما بنفس الموتر الدقيق exact tensor. وبالتالي، إذا قمنا بتغيير أحد المعلمات، يتغير الآخر أيضًا. قد تتساءل، عندما يتم ربط المعلمات، ماذا يحدث للتدرجات gradients؟ نظرًا

لأن معلمات النموذج تحتوي على تدرجات، يتم إضافة تدرجات الطبقة المخفية الثانية والطبقة المخفية الثالثة معًا أثناء النشر الخلفي backpropagation.

6.2.3. الملخص

لدينا عدة طرق للوصول إلى معلمات النموذج وربطها.

6.2.4. التمارين

1. استخدم نموذج `NestMLP` المحددفي القسم 6.1 وقم بالوصول إلى معلمات الطبقات المختلفة.
2. أنشئ MLP يحتوي على طبقة معلمة مشتركة وقم بتدريبها. أثناء عملية التدريب، لاحظ معلمات النموذج والتدرجات لكل طبقة.
3. لماذا تعتبر مشاركة المعلمات فكرة جيدة؟

6.3 تهيئة المعلمة Parameter Initialization

الآن بعد أن عرفنا كيفية الوصول إلى المعلمات، دعنا نلقي نظرة على كيفية تهيئتها initialize بشكل صحيح. ناقشنا الحاجة إلى التهيئة المناسبة proper initialization في القسم 5.4. يوفر إطار عمل التعلم العميق عمليات تهيئة عشوائية random initializations افتراضية لطبقاته. ومع ذلك، فإننا غالبًا ما نرغب في تهيئة أوزاننا وفقًا لبروتوكولات أخرى مختلفة. يوفر إطار العمل البروتوكولات الأكثر استخدامًا، ويسمح أيضًا بإنشاء مُهيئ مخصص custom initializer.

بشكل افتراضي، يقوم Keras بتهيئة مصفوفات الوزن weight matrices بشكل موحد من خلال الرسم من نطاق يتم حسابه وفقًا لأبعاد الإدخال والإخراج، ويتم تعيين جميع معلمات التحيز على الصفر. يوفر TensorFlow مجموعة متنوعة من طرق التهيئةفي كل من الوحدة النمطية الجذر `root` ووحدة `keras.initializers`.

```
import tensorflow as tf

net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4, activation=tf.nn.relu),
    tf.keras.layers.Dense(1),
])

X = tf.random.uniform((2, 4))
net(X).shape
```

```
TensorShape([2, 1])
```

6.3.1. التهيئة المدمجة Built-in Initialization

لنبدأ بالاتصال بالمهيئات المدمجين built-in initializers. يقوم الكود أدناه بتهيئة جميع معلمات الوزن كمتغيرات عشوائية غاوسية بانحراف معياري 0.01، بينما تم مسح معلمات التحيز إلى الصفر.

```
net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(
        4, activation=tf.nn.relu,

kernel_initializer=tf.random_normal_initializer(mean=0,
stddev=0.01),
        bias_initializer=tf.zeros_initializer()),
    tf.keras.layers.Dense(1)])

net(X)
net.weights[0], net.weights[1]
```

```
(<tf.Variable 'dense_2/kernel:0' shape=(4, 4)
dtype=float32, numpy=
 array([[-4.3637343e-03,  1.4685265e-02,  1.1814130e-02,
1.0973577e-02],
        [ 4.1117594e-03, -1.4918787e-02, -2.7245909e-03,
2.2734334e-03],
        [-2.3910873e-02, -2.1292072e-02, -1.7594380e-02,
2.9788772e-03],
        [-2.9245612e-05,  1.7144383e-03,  2.0546305e-03,
1.0586854e-03]],
       dtype=float32)>,
 <tf.Variable 'dense_2/bias:0' shape=(4,) dtype=float32,
numpy=array([0., 0., 0., 0.], dtype=float32)>)
```

يمكننا أيضًا تهيئة جميع المعلمات إلى قيمة ثابتة معينة (على سبيل المثال، 1).

```
net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(
        4, activation=tf.nn.relu,

kernel_initializer=tf.keras.initializers.Constant(1),
        bias_initializer=tf.zeros_initializer()),
    tf.keras.layers.Dense(1),
])
```

```
net(X)
net.weights[0], net.weights[1]
```

```
(<tf.Variable 'dense_4/kernel:0' shape=(4, 4)
dtype=float32, numpy=
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]], dtype=float32)>,
 <tf.Variable 'dense_4/bias:0' shape=(4,) dtype=float32,
numpy=array([0., 0., 0., 0.], dtype=float32)>)
```

يمكننا أيضًا تطبيق مُهيِّئات مختلفة لكتل معينة certain blocks. على سبيل المثال، نقوم أدناه بتهيئة الطبقة الأولى باستخدام مُهيئ Xavier وتهيئة الطبقة الثانية إلى قيمة ثابتة تبلغ 42.

```
net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(
        4,
        activation=tf.nn.relu,

kernel_initializer=tf.keras.initializers.GlorotUniform()
),
    tf.keras.layers.Dense(
        1,
kernel_initializer=tf.keras.initializers.Constant(42)),
])

net(X)
print(net.layers[1].weights[0])
print(net.layers[2].weights[0])
```

```
<tf.Variable 'dense_6/kernel:0' shape=(4, 4)
dtype=float32, numpy=
array([[ 0.62105197,  0.0110271 ,  0.77947575,
0.42430252],
       [ 0.7728824 , -0.06041008,  0.7212722 ,
0.3408653 ],
       [-0.23351735, -0.16163725,  0.8497241 , -
0.29418987],
       [-0.52923286, -0.5558266 , -0.1514526 , -
0.73771775]],
      dtype=float32)>
<tf.Variable 'dense_7/kernel:0' shape=(4, 1)
dtype=float32, numpy=
```

```
array([[42.],
       [42.],
       [42.],
       [42.]], dtype=float32)>
```

6.3.1.1. التهيئة المخصصة Custom Initialization

في بعض الأحيان، لا يتم توفير طرق التهيئة التي نحتاجها بواسطة إطار عمل التعلم العميق.في المثال أدناه، نحدد مُهيئًا لأي معلمة w وزن باستخدام التوزيع الغريب التالي:

$$w \sim \begin{cases} U(5,10) & \text{with probability } \frac{1}{4} \\ 0 & \text{with probability } \frac{1}{2} \\ U(-10,-5) & \text{with probability } \frac{1}{4} \end{cases}$$

هنا نحدد فئة فرعية subclass من `Initializer` وننفذ دالة `__call__` التي تُرجع الموتر المطلوب بالنظر إلى الشكل ونوع البيانات.

```
class MyInit(tf.keras.initializers.Initializer):
    def __call__(self, shape, dtype=None):
        data=tf.random.uniform(shape, -10, 10,
dtype=dtype)
        factor=(tf.abs(data) >= 5)
        factor=tf.cast(factor, tf.float32)
        return data * factor

net = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(
        4,
        activation=tf.nn.relu,
        kernel_initializer=MyInit()),
    tf.keras.layers.Dense(1),
])

net(X)
print(net.layers[1].weights[0])
```

```
<tf.Variable 'dense_8/kernel:0' shape=(4, 4)
dtype=float32, numpy=
array([[ 6.262211 ,  0.       ,  0.       ,  7.670639 ],
       [ 8.288603 ,  8.044296 ,  6.477192 , -0.       ],
       [-0.       ,  0.       , -0.       , -0.       ],
```

```
       [-0.       , -0.       , -7.4122763, -5.566807
]], dtype=float32)>
```

لاحظ أنه لدينا دائمًا خيار تعيين المعلمات مباشرةً.

```
net.layers[1].weights[0][:].assign(net.layers[1].weights
[0] + 1)
net.layers[1].weights[0][0, 0].assign(42)
net.layers[1].weights[0]
```

```
<tf.Variable 'dense_8/kernel:0' shape=(4, 4)
dtype=float32, numpy=
array([[42.       ,  1.       ,  1.       ,  8.670639 ],
       [ 9.288603 ,  9.044296 ,  7.477192 ,  1.       ],
       [ 1.       ,  1.       ,  1.       ,  1.       ],
       [ 1.       ,  1.       , -6.4122763, -4.566807
]], dtype=float32)>
```

6.3.2. الملخص

يمكننا تهيئة المعلمات باستخدام مُهيِّئات مضمنة (مدمجة) built-in ومخصصة custom.

6.3.3. التمارين

ابحث عن الوثائق عبر الإنترنت للحصول على المزيد من المهيئات المدمجة built-in initializers.

6.4. التهيئة الكسولة Lazy Initialization

حتى الآن، قد يبدو أننا أفلتنا من الوقوعفي الإهمالفي إنشاء شبكاتنا. على وجه التحديد، قمنا بالأشياء التالية غير البديهية، والتي قد لا يبدو أنها يجب أن تعمل:

- حددنا معماريات الشبكة دون تحديد أبعاد الإدخال.
- أضفنا طبقات دون تحديد أبعاد الإخراج للطبقة السابقة.
- حتى أننا "قمنا بتهيئة" هذه المعلمات قبل تقديم معلومات كافية لتحديد عدد المعلمات التي يجب أن تحتويها نماذجنا.

قد تتفاجأ من أن الكود الخاص بنا يعمل على الإطلاق. بعد كل شيء، لا توجد طريقة يمكن لإطار التعلم العميق أن يخبرنا بها عن أبعاد إدخال الشبكة. الحيلة هنا هي أن إطار العمل يؤجل defer التهيئة initialization،في انتظار أول مرة نقوم فيها بتمرير البيانات عبر النموذج، لاستنتاج أحجام كل طبقة أثناء الطيران.

في وقت لاحق، عند العمل مع الشبكات العصبية التلافيفية CNN، ستصبح هذه التقنية أكثر ملاءمة لأن أبعاد الإدخال (أي دقة الصورة) ستؤثر على أبعاد كل طبقة لاحقة. ومن ثم، فإن

القدرة على تعيين المعلمات دون الحاجة إلى معرفة،في وقت كتابة الكود، ما هي الأبعاد يمكن أن يبسط إلى حد كبير مهمة تحديد النماذج وتعديلها لاحقًا. بعد ذلك، نتعمقفي آليات التهيئة.

للبدء، دعنا ننشئ مثيلًا لـ MLP.

```
import tensorflow as tf

net = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dense(10),
])
```

في هذه المرحلة، لا يمكن للشبكة معرفة أبعاد أوزان طبقة الإدخال input layer's weights لأن بُعد الإدخال يظل غير معروف. وبالتالي، لم يقم إطار العمل بعد بتهيئة أي معلمات. نؤكد بمحاولة الوصول إلى المعلمات أدناه

```
[net.layers[i].get_weights() for i in
range(len(net.layers))]
```

```
[[], []]
```

لاحظ أن كل كائنات طبقة موجودة ولكن الأوزان فارغة. قد يؤدي استخدام `net.get_weights()` إلى حدوث خطأ لأن الأوزان لم تتم تهيئتها بعد.

بعد ذلك، دعنا نمرر البيانات عبر الشبكة لجعل إطار العمل يهيئ المعلمات أخيرًا.

```
X = tf.random.uniform((2, 20))
net(X)
[w.shape for w in net.get_weights()]
```

```
[(20, 256), (256,), (256, 10), (10,)]
```

بمجرد أن نعرف أبعاد الإدخال، 20، يمكن لإطار العمل تحديد شكل مصفوفة وزن الطبقة الأولى عن طريق توصيل القيمة 20. بعد التعرف على شكل الطبقة الأولى، ينتقل الإطار إلى الطبقة الثانية، وهكذا من خلال الرسم البياني الحسابي حتى تعرف كل الأشكال. لاحظ أنهفي هذه الحالة، تتطلب الطبقة الأولى فقط تهيئة كسولة lazy initialization، ولكن يتم تهيئة إطار العمل بالتسلسل sequentially. بمجرد معرفة جميع أشكال المعلمات، يمكن للإطار أخيرًا تهيئة المعلمات.

6.4.1. الملخص

- يمكن أن تكون التهيئة الكسولة مريحة، مما يسمح لإطار العمل باستنتاج أشكال المعلمات تلقائيًا، مما يجعل من السهل تعديل البُنى والقضاء على مصدر واحد شائع للأخطاء.
- يمكننا تمرير البيانات عبر النموذج لجعل الإطار يهيئ المعلمات أخيرًا.

6.4.2. التمارين

1. ماذا يحدث إذا قمت بتحديد أبعاد الإدخال للطبقة الأولى وليس للطبقات اللاحقة؟ هل تحصل على تهيئة فورية immediate initialization؟
2. ماذا يحدث إذا قمت بتحديد أبعاد غير متطابقة mismatching dimensions؟
3. ماذا يجب أن تفعل إذا كان لديك مدخلات ذات أبعاد متفاوتة varying dimensionality؟ تلميح: انظر إلى ربط المعلمة parameter tying.

6.5. الطبقات المخصصة Custom Layers

أحد العوامل الكامنة وراء نجاح التعلم العميق هو توافر مجموعة واسعة من الطبقات التي يمكن تكوينها بطرق إبداعية لتصميم بنى مناسبة لمجموعة متنوعة من المهام. على سبيل المثال، ابتكر الباحثون طبقات خصيصًا للتعامل مع الصور والنص والتكرار عبر البيانات المتسلسلة وأداء البرمجة الديناميكية. عاجلاً أم آجلاً، ستواجه أو تخترع طبقة غير موجودة بعدفي إطار التعلم العميق.في هذه الحالات، يجب عليك إنشاء طبقة مخصصة custom layer.في هذا القسم، نوضح لك كيف.

6.5.1. الطبقات بدون معلمات Layers without Parameters

للبدء، نقوم ببناء طبقة مخصصة لا تحتوي على أي معلمات خاصة بها. يجب أن يبدو هذا مألوفًا إذا كنت تتذكر مقدمتنا للوحدةفي القسم 6.1. فئة `CenteredLayer` التالية تطرح ببساطة المتوسط من مدخلاتها. لإنشائه، نحتاج ببساطة إلى الوراثة من فئة الطبقة الأساسية وتنفيذ دالة الانتشار الأمامي.

```
import tensorflow as tf
from d2l import tensorflow as d2l


class CenteredLayer(tf.keras.Model):
    def __init__(self):
        super().__init__()

    def call(self, inputs):
        return inputs - tf.reduce_mean(inputs)
```

دعنا نتحقق من أن طبقتنا تعمل على النحو المنشود عن طريق تغذية بعض البيانات من خلالها.

```
layer = CenteredLayer()
layer(tf.constant([1.0, 2, 3, 4, 5]))
```

```
<tf.Tensor: shape=(5,), dtype=float32, numpy=array([-2.,
-1.,  0.,  1.,  2.], dtype=float32)>
```

يمكننا الآن دمج incorporate طبقتنا كمكونفي بناء نماذج أكثر تعقيدًا.

```
net = tf.keras.Sequential([tf.keras.layers.Dense(128),
CenteredLayer()])
```

كتحقق إضافي من الصحة، يمكننا إرسال بيانات عشوائية عبر الشبكة والتحقق من أن المتوسط هوفي الواقع 0. نظرًا لأننا نتعامل مع أرقام الفاصلة العائمة floating point numbers، فقد لا نزال نرى عددًا صغيرًا جدًا غير صفري بسبب التكميم quantization.

```
Y = net(tf.random.uniform((4, 8)))
tf.reduce_mean(Y)
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=-4.656613e-
10>
```

6.5.2. الطبقات مع معلمات Layers with Parameters

الآن بعد أن عرفنا كيفية تحديد الطبقات البسيطة، دعنا ننتقل إلى تحديد الطبقات باستخدام المعلمات التي يمكن تعديلها من خلال التدريب. يمكننا استخدام دوال مدمجة لإنشاء معلمات توفر بعض الدوال الأساسية. على وجه الخصوص، تحكم الوصول والتهيئة والمشاركة والحفظ وتحميل معلمات النموذج. بهذه الطريقة، من بين الفوائد الأخرى، لن نحتاج إلى كتابة إجراءات تسلسل مخصصة لكل طبقة مخصصة.

الآن دعونا ننفذ نسختنا الخاصة من الطبقة المتصلة بالكامل fully connected layer. تذكر أن هذه الطبقة تتطلب معلمتين، أحدهما يمثل الوزن والآخر يمثل التحيز.في هذا التنفيذ، نختار دالة تنشيط ReLU كإعداد افتراضي. تتطلب هذه الطبقة وسيطتي إدخال: `in_units` وunits، والتي تشير إلى عدد المدخلات والمخرجات، على التوالي.

```
class MyDense(tf.keras.Model):
    def __init__(self, units):
        super().__init__()
        self.units = units

    def build(self, X_shape):
        self.weight = self.add_weight(name='weight',
            shape=[X_shape[-1], self.units],
            initializer=tf.random_normal_initializer())
        self.bias = self.add_weight(
            name='bias', shape=[self.units],
            initializer=tf.zeros_initializer())

    def call(self, X):
        linear = tf.matmul(X, self.weight) + self.bias
```

```
        return tf.nn.relu(linear)
```

بعد ذلك، نقوم بإنشاء مثيل لفئة `MyDense` والوصول إلى معلمات النموذج الخاصة بها.

```
dense = MyDense(3)
dense(tf.random.uniform((2, 5)))
dense.get_weights()
```

```
[array([[-0.02223266, -0.04409523,  0.01749811],
        [ 0.04932142, -0.07135182,  0.08249469],
        [ 0.05460888,  0.01363679,  0.0342483 ],
        [-0.02898769, -0.03092524, -0.02166725],
        [-0.05692595, -0.03148399,  0.05532912]],
dtype=float32),
 array([0., 0., 0.], dtype=float32)]
```

يمكننا إجراء حسابات الانتشار الأمامي forward propagation مباشرة باستخدام طبقات مخصصة custom layers.

```
dense(tf.random.uniform((2, 5)))
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[0.01219213, 0.        , 0.12613392],
       [0.        , 0.        , 0.02558431]],
dtype=float32)>
```

يمكننا أيضًا إنشاء نماذج باستخدام طبقات مخصصة. بمجرد أن نحصل على ذلك، يمكننا استخدامه تمامًا مثل الطبقة المضمنة المتصلة بالكامل.

```
net = tf.keras.models.Sequential([MyDense(8),
MyDense(1)])
net(tf.random.uniform((2, 64)))
```

```
<tf.Tensor: shape=(2, 1), dtype=float32, numpy=
array([[0.00192496],
       [0.        ]], dtype=float32)>
```

6.5.3. الملخص

- يمكننا تصميم طبقات مخصصة عبر فئة الطبقة الأساسية. يتيح لنا ذلك تحديد طبقات جديدة مرنة تتصرف بشكل مختلف عن أي طبقات موجودةفي المكتبة.
- بمجرد تحديدها، يمكن استدعاء الطبقات المخصصةفي سياقات وبنيات عشوائية.
- يمكن أن تحتوي الطبقات على معلمات محلية local parameters، والتي يمكن إنشاؤها من خلال الدوال المضمنة built-in functions.

6.5.4. التمارين

1. صمم طبقة تأخذ مدخلاً وتحسب تقليل الموتر tensor reduction، أي أنها ترجع $y_k = \sum_{i,j} W_{ijk} x_i x_j$.
2. صمم طبقة تُرجع النصف المتصدر leading half من معاملات فورييه Fourier coefficients للبيانات.

6.6. ملف الإدخال/الإخراج File I/O

ناقشنا حتى الآن كيفية معالجة البيانات وكيفية بناء نماذج التعلم العميق وتدريبها واختبارها. ومع ذلك،في مرحلة ما، نأمل أن نكون سعداء بدرجة كافية بالنماذج التي تم تعلمها والتي سنرغب في حفظ النتائج لاستخدامها لاحقًا في سياقات مختلفة (ربما حتى لعمل تنبؤات في النشر). بالإضافة إلى ذلك، عند إجراء عملية تدريب طويلة، فإن أفضل الممارسات هي حفظ النتائج الوسيطة بشكل دوري (checkpointing) للتأكد من أننا لا نفقد عدة أيام من العمليات الحسابية إذا كنا نتحرك عبر سلك الطاقة الخاص بخادمنا. وبالتالي فقد حان الوقت لمعرفة كيفية تحميل وتخزين متجهات الوزن الفردية والنماذج بأكملها. هذا القسم يعالج كلا المسألتين.

6.6.1. تحميل وحفظ الموترات Loading and Saving Tensors

بالنسبة إلى الموترات الفردية individual tensors، يمكننا مباشرة استدعاء دوال التحميل `load` والحفظ `save` لقراءتها وكتابتها على التوالي. تتطلب كلتا الدالتين توفير اسم، ويتطلب الحفظ حفظ المتغير كمدخل.

```
import numpy as np
import tensorflow as tf

x = tf.range(4)
np.save('x-file.npy', x)
```

يمكننا الآن قراءة البيانات من الملف المخزن مرة أخرى في الذاكرة.

```
x2 = np.load('x-file.npy', allow_pickle=True)
x2
```

```
array([0, 1, 2, 3], dtype=int32)
```

يمكننا تخزين قائمة الموترات وقراءتها مرة أخرى في الذاكرة.

```
y = tf.zeros(4)
np.save('xy-files.npy', [x, y])
x2, y2 = np.load('xy-files.npy', allow_pickle=True)
(x2, y2)
```

```
(array([0., 1., 2., 3.]), array([0., 0., 0., 0.]))
```

يمكننا حتى كتابة وقراءة قاموس يقوم بالتخطيط من السلاسل strings إلى الموترات tensors. هذا مناسب عندما نريد قراءة أو كتابة جميع الأوزانفي النموذج.

```
mydict = {'x': x, 'y': y}
np.save('mydict.npy', mydict)
mydict2 = np.load('mydict.npy', allow_pickle=True)
mydict2
```

```
array({'x': <tf.Tensor: shape=(4,), dtype=int32,
numpy=array([0, 1, 2, 3], dtype=int32)>, 'y':
<tf.Tensor: shape=(4,), dtype=float32, numpy=array([0.,
0., 0., 0.], dtype=float32)>},
```

6.6.2. تحميل وحفظ معلمات النموذج Loading and Saving Model Parameters

يعد حفظ متجهات الوزن الفردية (أو الموترات الأخرى) أمرًا مفيدًا، ولكنه يصبح مملاً للغاية إذا أردنا حفظ (وتحميل لاحقًا) نموذجًا كاملاً. بعد كل شيء، قد يكون لدينا مئات من مجموعات المعلمات متناثرةفي جميع الأنحاء. لهذا السبب، يوفر إطار عمل التعلم العميق دوال مدمجة لتحميل وحفظ الشبكات بأكملها. من التفاصيل المهمة التي يجب ملاحظتها أن هذا يحفظ معلمات النموذج وليس النموذج بأكمله. على سبيل المثال، إذاكان لدينا MLP ثلاثي الطبقات، فنحن بحاجة إلى تحديد البنية بشكل منفصل. والسببفي ذلك هو أن النماذج نفسها يمكن أن تحتوي على رمز تعسفي arbitrary code، وبالتالي لا يمكن إجراء تسلسل لها بشكل طبيعي. وبالتالي، من أجل إعادة النموذج إلى وضعه السابق، نحتاج إلى إنشاء البُنيةفي التعليمات البرمجية ثم تحميل المعلمات من القرص. لنبدأ بـ MLP المألوف لدينا.

```
class MLP(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.flatten = tf.keras.layers.Flatten()
        self.hidden = tf.keras.layers.Dense(units=256,
activation=tf.nn.relu)
        self.out = tf.keras.layers.Dense(units=10)

    def call(self, inputs):
        x = self.flatten(inputs)
        x = self.hidden(x)
        return self.out(x)

net = MLP()
X = tf.random.uniform((2, 20))
```

```
Y = net(X)
```

بعد ذلك، نقوم بتخزين معلمات النموذج كملف باسم "mlp.params"

```
net.save_weights('mlp.params')
```

لاستعادة النموذج، نقوم بإنشاء نسخة طبق الأصل من نموذج MLP الأصلي. بدلاً من التهيئة العشوائية لمعلمات النموذج، نقرأ المعلمات المخزنةفي الملف مباشرةً.

```
clone = MLP()
clone.load_weights('mlp.params')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoad
Status at 0x7fe7e6f9fdc0>
```

نظرًا لأن كلا المثالين لهما نفس معلمات النموذج، يجب أن تكون النتيجة الحسابية لنفس الإدخال X هي نفسها. دعونا نتحقق من هذا.

```
Y_clone = clone(X)
Y_clone == Y
```

```
<tf.Tensor: shape=(2, 10), dtype=bool, numpy=
array([[ True,  True,  True,  True,  True,  True,  True,
True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,
True,  True,
         True]])>
```

6.6.3. الملخص

- يمكن استخدام دوال الحفظ `save` والتحميل `load` لتنفيذ إدخال / إخراج الملف ile I/O لكائنات الموتر.
- يمكننا حفظ وتحميل مجموعات كاملة من المعلمات لشبكة عبر قاموس المعلمات parameter dictionary.
- يجب أن يتم حفظ البنيةفي الكود وليسفي المعلمات.

6.6.4. التمارين

1. حتى لو لم تكن هناك حاجة لنشر نماذج مدربة على جهاز مختلف، فما هي الفوائد العملية لتخزين معلمات النموذج؟
2. افترض أننا نريد إعادة استخدام أجزاء فقط من الشبكة ليتم دمجهافي شبكة ذات بنية مختلفة. كيف يمكنك استخدام، لنقل أول طبقتين من شبكة سابقةفي شبكة جديدة؟
3. كيف ستشرعفي حفظ بُنية الشبكة والمعلمات؟ ما هي القيود التي ستفرضها على المعمارية؟

6.7 وحدات معالجة الرسومات GPUs

في القسم 1.5، ناقشنا النمو السريع للحسابات على مدى العقدين الماضيين. باختصار، زاد أداء وحدة معالجة الرسومات GPU بمعدل 1000 عامل كل عقد منذ عام 2000. وهذا يوفر فرصًا رائعة ولكنه يشير أيضًا إلى الحاجة الملحة لتوفير مثل هذا الأداء.

في هذا القسم، نبدأفي مناقشة كيفية تسخير هذا الأداء الحسابي لبحثك. أولاً باستخدام وحدات معالجة رسومات واحدة وفي وقت لاحق، كيفية استخدام وحدات معالجة رسومات متعددة وخوادم متعددة (مع وحدات معالجة رسومات متعددة multiple GPUs).

على وجه التحديد، سنناقش كيفية استخدام وحدة معالجة رسومات NVIDIA واحدة لإجراء العمليات الحسابية. أولاً، تأكد من تثبيت NVIDIA GPU واحد على الأقل. بعد ذلك، قم بتنزيل برنامج تعريف NVIDIA وCUDA واتبع التعليمات لتعيين المسار المناسب. بمجرد اكتمال هذه الاستعدادات، يمكن استخدام الأمر `nvidia-smi` لعرض معلومات بطاقة الرسومات.

```
!nvidia-smi
```

```
Mon Aug 29 23:47:02 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.106.00   Driver Version: 460.106.00   CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:1B.0 Off |                    0 |
| N/A   48C    P0    52W / 300W |   1760MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

```
|   1  Tesla V100-SXM2...  Off  | 00000000:00:1C.0 Off |
0 |
| N/A   58C    P0    66W / 300W |      3MiB / 16160MiB |
0%      Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+
|   2  Tesla V100-SXM2...  Off  | 00000000:00:1D.0 Off |
0 |
| N/A   39C    P0    37W / 300W |      3MiB / 16160MiB |
0%      Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+
|   3  Tesla V100-SXM2...  Off  | 00000000:00:1E.0 Off |
0 |
| N/A   37C    P0    38W / 300W |      3MiB / 16160MiB |
0%      Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------
----------------------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|===========================================================
======================|
+-----------------------------------------------------------
----------------------+
```

لتشغيل البرامج الموجودةفي هذا القسم، تحتاج إلى وحدتي GPU على الأقل. لاحظ أن هذا قد يكون باهظًا بالنسبة لمعظم أجهزة الكمبيوتر المكتبية ولكنه متاح بسهولةفي السحابة cloud، على سبيل المثال، باستخدام مثيلات AWS EC2 متعددة وحدات معالجة الرسومات. لا تتطلب

جميع الأقسام الأخرى تقريبًا وحدات معالجة رسومات متعددة multiple GPUs. بدلاً من ذلك، هذا لتوضيح كيفية تدفق البيانات بين الأجهزة المختلفة.

6.7.1. أجهزة الحوسبة Computing Devices

يمكننا تحديد الأجهزة، مثل وحدات المعالجة المركزية CPU ووحدات معالجة الرسومات GPU، للتخزين storage والحساب calculation. بشكل افتراضي، يتم إنشاء الموترات في الذاكرة الرئيسية ثم استخدام وحدة المعالجة المركزية لحسابها.

```
import tensorflow as tf
from d2l import tensorflow as d2l

def cpu():  #@save
    return tf.device('/CPU:0')

def gpu(i=0):  #@save
    return tf.device(f'/GPU:{i}')

cpu(), gpu(), gpu(1)
```

```
(<tensorflow.python.eager.context._EagerDeviceContext at
0x7f5138482040>,
 <tensorflow.python.eager.context._EagerDeviceContext at
0x7f5138559b00>,
 <tensorflow.python.eager.context._EagerDeviceContext at
0x7f5139bdd540>)
```

يمكننا الاستعلام عن عدد وحدات معالجة الرسومات المتاحة.

```
def num_gpus():  #@save
    return
len(tf.config.experimental.list_physical_devices('GPU'))

num_gpus()
```

```
2
```

الآن نحدد دالتي ملائمة convenient functions تسمحان لنا بتشغيل التعليمات البرمجية حتى لو لم تكن وحدات معالجة الرسومات المطلوبة موجودة.

```
def try_gpu(i=0):  #@save
    """Return gpu(i) if exists, otherwise return
cpu()."""
    if num_gpus() >= i + 1:
        return gpu(i)
    return cpu()
```

```
def try_all_gpus():  #@save
    """Return all available GPUs, or [cpu(),] if no GPU
exists."""
    return [gpu(i) for i in range(num_gpus())]

try_gpu(), try_gpu(10), try_all_gpus()
```

```
(<tensorflow.python.eager.context._EagerDeviceContext at
0x7f5138500240>,
 <tensorflow.python.eager.context._EagerDeviceContext at
0x7f51381f92c0>,
 [<tensorflow.python.eager.context._EagerDeviceContext
at 0x7f51381f9180>,
  <tensorflow.python.eager.context._EagerDeviceContext
at 0x7f51381f93c0>])
```

6.7.2. الموترات ووحدات معالجة الرسومات Tensors and GPUs

بشكل افتراضي، يتم إنشاء الموترات على وحدة المعالجة المركزية CPU. يمكننا الاستعلام عن الجهاز الذي يوجد فيه الموتر.

```
x = tf.constant([1, 2, 3])
x.device
```

```
'/job:localhost/replica:0/task:0/device:GPU:0'
```

من المهم ملاحظة أنه كلما أردنا العمل بشروط متعددة، يجب أن يكونوا على نفس الجهاز. على سبيل المثال، إذا جمعنا اثنين من الموترات، فسنحتاج إلى التأكد من أن كلا الوسيطتين تعيشان على نفس الجهاز ـ وإلا فلن يعرف إطار العمل مكان تخزين النتيجة أو حتى كيفية تحديد مكان إجراء الحساب.

6.7.2.1. التخزين على وحدة معالجة الرسومات Storage on the GPU

هناك عدة طرق لتخزين موتر على وحدة معالجة الرسومات GPU. على سبيل المثال، يمكننا تحديد جهاز تخزين عند إنشاء موتر. بعد ذلك، نقوم بإنشاء متغير موتر Xفي وحدة معالجة الرسومات الأولى. يستهلك الموتر الذي تم إنشاؤه على وحدة معالجة الرسومات ذاكرة وحدة معالجة الرسومات هذه فقط. يمكننا استخدام الأمر `nvidia-smi` لعرض استخدام ذاكرة وحدة معالجة الرسومات. بشكل عام، نحتاج إلى التأكد من عدم إنشاء بيانات تتجاوز حد ذاكرة وحدة معالجة الرسومات.

```
with try_gpu():
    X = tf.ones((2, 3))
X
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[1., 1., 1.],
```

```
       [1., 1., 1.]], dtype=float32)>
```

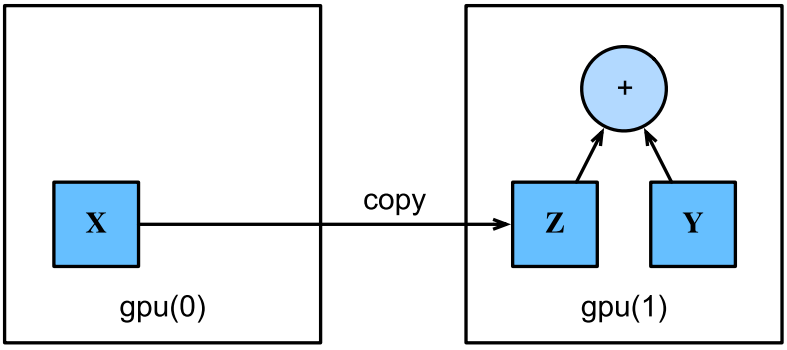
بافتراض أن لديك وحدتي GPU على الأقل، فإن الكود التالي سينشئ موترًا عشوائيًا على وحدة معالجة الرسومات الثانية.

```
with try_gpu(1):
    Y = tf.random.uniform((2, 3))
Y
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[0.87347054, 0.4167322 , 0.06983936],
       [0.37522686, 0.3176515 , 0.2823031 ]],
dtype=float32)>
```

6.7.2.2. النسخ Copying

إذا أردنا حساب X + Y، فنحن بحاجة إلى تحديد مكان إجراء هذه العملية. على سبيل المثال، كما هو موضحفي الشكل 6.7.1، يمكننا نقل X إلى وحدة معالجة الرسومات الثانية وإجراء العملية هناك. لا تضف X وY ببساطة، لأن هذا سيؤدي إلى استثناء. لن يعرف محرك وقت التشغيل ما يجب فعله: لا يمكنه العثور على البيانات على نفس الجهاز ويفشل. نظرًا لأن Y تعيش على وحدة معالجة الرسومات الثانية، فنحن بحاجة إلى نقل X إلى هناك قبل أن نتمكن من إضافة الاثنين.



شكل 6.7.1 انسخ البيانات لإجراء عملية على نفس الجهاز.

```
with try_gpu(1):
    Z = X
print(X)
print(Z)
```

```
tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
tf.Tensor(
[[1. 1. 1.]
```

```
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
```

الآن بعد أن أصبحت البيانات على نفس GPU (كلاهما Z و Y)، يمكننا جمعها.

```
Y + Z
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[1.8734705, 1.4167322, 1.0698394],
       [1.3752269, 1.3176515, 1.2823031]],
dtype=float32)>
```

تخيل أن المتغير Z الخاص بك موجود بالفعل في وحدة معالجة الرسومات الثانية. ماذا يحدث إذا استمررت في الاتصال بـ Z2 = Z ضمن نطاق الجهاز نفسه؟ سيعود Z بدلاً من عمل نسخة وتخصيص ذاكرة جديدة.

```
with try_gpu(1):
    Z2 = Z
Z2 is Z
```

```
True
```

6.7.2.3. ملاحظات جانبية Side Notes

يستخدم الناس وحدات معالجة الرسومات للقيام بالتعلم الآلي لأنهم يتوقعون أن تكون سريعة. لكن نقل المتغيرات بين الأجهزة بطيء. لذلك نريدك أن تكون متأكدًا بنسبة 100٪ أنك تريد أن تفعل شيئًا بطيئًا قبل أن نسمح لك بفعله. إذا قام إطار عمل التعلم العميق بالنسخ تلقائيًا دون تعطل، فقد لا تدرك أنك كتبت بعض التعليمات البرمجية البطيئة.

أيضًا، يعد نقل البيانات بين الأجهزة (وحدة المعالجة المركزية ووحدات معالجة الرسومات والأجهزة الأخرى) شيئًا أبطأ بكثير من الحساب. كما أنه يجعل الموازاة أكثر صعوبة، حيث يتعين علينا انتظار إرسال البيانات (أو بالأحرى استلامها) قبل أن نتمكن من متابعة المزيد من العمليات. هذا هو السبب في ضرورة توخي الحذر الشديد في عمليات النسخ. كقاعدة عامة، العديد من العمليات الصغيرة أسوأ بكثير من عملية واحدة كبيرة. علاوة على ذلك، فإن العديد من العمليات في وقت واحد أفضل بكثير من العديد من العمليات الفردية التي تتخللها التعليمات البرمجية إلا إذا كنت تعرف ما تفعله. هذا هو الحال لأن مثل هذه العمليات يمكن أن تمنع إذا كان على أحد الأجهزة انتظار الآخر قبل أن يتمكن من القيام بشيء آخر. إنه يشبه إلى حد ما طلب قهوتك في قائمة انتظار بدلاً من طلبها مسبقًا عبر الهاتف واكتشاف أنها جاهزة عندما تكون جاهزًا.

أخيرًا، عندما نطبع الموترات أو نحول الموترات إلى تنسيق NumPy، إذا لم تكن البيانات في الذاكرة الرئيسية، فسيقوم الإطار بنسخها إلى الذاكرة الرئيسية أولاً، مما يؤدي إلى زيادة نقل البيانات. والأسوأ من ذلك، أنه يخضع الآن لقفل المفسر العالمي global interpreter المخيف الذي يجعل كل شيء ينتظر بايثون حتى يكتمل.

6.7.3. الشبكات العصبية ووحدات معالجة الرسومات Neural Networks and GPUs

وبالمثل، يمكن لنموذج الشبكة العصبية تحديد الأجهزة. الكود التالي يضع معلمات النموذج على GPU.

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    net = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1)])
```

```
INFO:tensorflow:Using MirroredStrategy with devices
('/job:localhost/replica:0/task:0/device:GPU:0',
'/job:localhost/replica:0/task:0/device:GPU:1')
```

سنرى العديد من الأمثلة حول كيفية تشغيل النماذج على وحدات معالجة الرسوماتفي الفصول التالية، وذلك لأنها ستصبح إلى حد ما أكثر كثافة من الناحية الحسابية.

عندما يكون الإدخال موترًا على وحدة معالجة الرسومات، فإن النموذج سيحسب النتيجة على نفس وحدة معالجة الرسومات.

```
net(X)
```

```
<tf.Tensor: shape=(2, 1), dtype=float32, numpy=
array([[0.59118944],
       [0.59118944]], dtype=float32)>
```

دعنا نؤكد أن معلمات النموذج مخزنة على نفس وحدة معالجة الرسومات.

```
net.layers[0].weights[0].device,
net.layers[0].weights[1].device
```

```
('/job:localhost/replica:0/task:0/device:GPU:0',
 '/job:localhost/replica:0/task:0/device:GPU:0')
```

دع المدرب trainer يدعم GPU.

باختصار، طالما أن جميع البيانات والمعلمات موجودة على نفس الجهاز، يمكننا تعلم النماذج بكفاءة. سنرىفي الفصول التالية عدة أمثلة من هذا القبيل.

6.7.4. الملخص

- يمكننا تحديد أجهزة للتخزين والحساب، مثل وحدة المعالجة المركزية أو وحدة معالجة الرسومات. بشكل افتراضي، يتم إنشاء البياناتفي الذاكرة الرئيسية ثم تستخدم وحدة المعالجة المركزية لإجراء العمليات الحسابية.
- يتطلب إطار عمل التعلم العميق أن تكون جميع بيانات الإدخال للحساب على نفس الجهاز، سواء كانت وحدة المعالجة المركزية أو نفس وحدة معالجة الرسومات.

- يمكنك أن تفقد أداءً ملحوظًا عن طريق نقل البيانات دون عناية. الخطأ المعتاد هو كما يلي: حساب الخسارة (الخطأ) لكل دفعة صغيرة minibatch على وحدة معالجة الرسومات وإبلاغ المستخدم بها على سطر الأوامر (أو تسجيلهافي NumPy ndarray) سيؤدي إلى تشغيل قفل مترجم عام global interpreter lock يوقف جميع وحدات معالجة الرسومات. من الأفضل بكثير تخصيص ذاكرة للتسجيل داخل وحدة معالجة الرسومات ونقل السجلات الكبيرة فقط.

6.7.5. التمارين

1. جرب مهمة حسابية أكبر، مثل ضرب المصفوفات الكبيرة، ولاحظ الفرقفي السرعة بين وحدة المعالجة المركزية ووحدة معالجة الرسومات. ماذا عن مهمة بكمية صغيرة من الحسابات؟
2. كيف يجب أن نقرأ ونكتب معلمات النموذج على وحدة معالجة الرسومات؟
3. قم بقياس الوقت المستغرق لحساب 1000 مصفوفة من ضرب المصفوفات 100×100 وتسجيل معيار Frobenius لمصفوفة الإخراج نتيجة واحدةفي كل مرة مقابل الاحتفاظ بسجلفي وحدة معالجة الرسومات ونقل النتيجة النهائية فقط.
4. قم بقياس الوقت المستغرق لإجراء ضرب المصفوفةـ المصفوفة على وحدتي GPU في نفس الوقت مقابل التسلسل على وحدة معالجة رسومات واحدة. تلميح: يجب أن ترى تحجيمًا خطيًا linear scaling تقريبًا.

7

الشبكات العصبية الالتفافية

7. الشبكات العصبية التلافيفية Convolutional Neural Networks

يتم تمثيل بيانات الصورة كشبكة ثنائية الأبعاد من البكسل، سواء كانت أحادية اللون أو ملونة. وفقًا لذلك، يتوافق كل بكسل مع قيمة عددية واحدة أو عدة قيم رقمية على التوالي. حتى الآن تجاهلنا هذه البُنية الغنية وعاملناها كمتجهات للأرقام من خلال تسطيح flattening الصور، بغض النظر عن العلاقة المكانية بين وحدات البكسل. كان هذا النهج غير المرضي للغاية ضروريًا لتغذية المتجهات أحادية البعد الناتجة من خلال MLP متصلة بالكامل.

نظرًا لأن هذه الشبكات ثابتةفي ترتيب الميزات، يمكننا الحصول على نتائج مماثلة بغض النظر عما إذا كنا نحتفظ بترتيب يتوافق مع البُنية المكانية للبكسل أو إذا قمنا بتغيير أعمدة مصفوفة التصميم الخاصة بنا قبل ملاءمة fitting معلمات MLP. على نحو مفضل، سنستفيد من معرفتنا السابقة بأن وحدات البكسل القريبة ترتبط عادةً ببعضها البعض، لبناء نماذج فعالة للتعلم من بيانات الصورة.

يقدم هذا الفصل الشبكات العصبية التلافيفية (CNN)، (LeCun et al.، 1995)، وهي عائلة قوية من الشبكات العصبية المصممة لهذا الغرض تحديدًا. أصبحت البُنى القائمة على CNN موجودةفي كل مكان الآنفي مجال الرؤية الحاسوبية. على سبيل المثال،في مجموعة Imagnet، (Deng et al.، 2009)، كان استخدام الشبكات العصبية التلافيفية، باختصار Convnets هو الذي وفر تحسينات كبيرةفي الأداء (Krizhevsky et al.، 2012).

شبكات CNN الحديثة، كما يطلق عليها بالعامية، تدين بتصميمها إلى الإلهام من علم الأحياء biology، ونظرية المجموعة group theory، وجرعة صحية من الترقيع التجريبي experimental tinkering. بالإضافة إلى كفاءة العينة sample efficiencyفي تحقيق نماذج دقيقة achieving accurate models، تميل شبكات CNN إلى أن تكون فعالة من الناحية الحسابية، وذلك لأنها تتطلب معلمات أقل من البُنى المتصلة تمامًا ولأن الالتفافات convolutions سهلة الموازاة عبر نوى وحدة معالجة الرسومات (Chetlur et al.، 2014). وبالتالي، غالبًا ما يطبق الممارسون شبكات CNN كلما أمكن ذلك، وقد برزوا بشكل متزايد كمنافسين موثوق بهم حتىفي المهام ذات الهيكل التسلسلي أحادي البعد، مثل الصوت (Abdel-Hamid et al.، 2014)، النص (Kalchbrenner et al.، 2014)، وتحليل السلاسل الزمنية time series analysis (LeCun et al.، 1995)، حيث يتم استخدام الشبكات العصبية المتكررة recurrent neural networks (RNN) بشكل تقليدي. كما أن بعض التعديلات الذكية لشبكات CNN قد جعلتها تؤثر على البيانات المهيكلة بالرسوم البيانية

graph-structured data (Kipf and Welling، 2016) وفي أنظمة التوصية recommender systems.

أولاً، سوف نتعمق أكثرفي دوافع الشبكات العصبية التلافيفية. ويلي ذلك جولةفي العمليات الأساسية التي تشكل العمود الفقري لجميع الشبكات التلافيفية. وتشمل هذه الطبقات التلافيفية نفسها convolutional layers، والتفاصيل الدقيقة بمافي ذلك الحشو padding والخطوة stride، وطبقات التجميع pooling layers المستخدمة لتجميع المعلومات عبر المناطق المكانية المجاورة، واستخدام قنوات متعددةفي كل طبقة، ومناقشة دقيقة لهيكل البُنى الحديثة. سنختتم الفصل بمثال عملي كامل لـ LeNet، أول شبكة تلافيفية تم نشرها بنجاح، قبل وقت طويل من ظهور التعلم العميق الحديث.في الفصل التالي، سوف نتعمقفي التطبيقات الكاملة لبعض أبنية CNN الشائعة والحديثة نسبيًا والتي تمثل تصميماتها معظم التقنيات المستخدمة بشكل شائع من قبل الممارسين المعاصرين.

7.1. من الطبقات المتصلة بالكامل إلى التلافيف From Fully Connected Layers to Convolutions

حتى يومنا هذا، تظل النماذج التي ناقشناها حتى الآن خيارات مناسبة عندما نتعامل مع البيانات المجدولة tabular data. نعني بالجدول أن البيانات تتكون من صفوف تتوافق مع الأمثلة والأعمدة المقابلة للميزات. باستخدام البيانات المجدولة، قد نتوقع أن الأنماط التي نسعى إليها قد تتضمن تفاعلات بين الميزات، لكننا لا نفترض أي بُنية مسبقًا تتعلق بكيفية تفاعل الميزات.

في بعض الأحيان، نفتقر حقًا إلى المعرفة لتوجيه بناء البُنى الحِرَفية.في هذه الحالات، قد يكون MLP هو أفضل ما يمكننا القيام به. ومع ذلك، بالنسبة للبيانات الإدراكية عالية الأبعاد high-dimensional perceptual data، يمكن أن تنمو هذه الشبكات التي تفتقر إلى البُنية بشكل غير عملي.

على سبيل المثال، دعنا نعود إلى مثالنا الجاري للتمييز بين القطط والكلاب. لنفترض أننا نقوم بعمل شاملفي جمع البيانات، حيث نجمع مجموعة بيانات مشروحة من صور فوتوغرافية بدقة واحدة ميغا بكسل. هذا يعني أن كل مدخلفي الشبكة له مليون بعد. حتى التخفيض الشديد إلى ألف بُعد مخفي سيتطلب طبقة متصلة بالكامل تتميز بالمعلمات $10^3 \times 10^6 = 10^9$. ما لم يكن لدينا الكثير من وحدات معالجة الرسومات GPU، وموهبة التحسين الموزع، وقدر غير عادي من الصبر، فقد يصبح تعلم معلمات هذه الشبكة غير ممكن.

قد يعترض القارئ الحريص على هذه الحجة على أساس أن دقة واحدة ميغا بكسل قد لا تكون ضرورية. ومع ذلك،في حين أننا قد نكون قادرين على التخلص من مائة ألف بكسل، فإن الطبقة

المخفية التي يبلغ حجمها 1000 بكسل تقلل بشكل كبير من عدد الوحدات المخفية التي يتطلبها تعلم التمثيل الجيد للصور، لذلك سيظل النظام العملي يتطلب مليارات من المعلمات. علاوة على ذلك، قد يتطلب تعلم المصنف من خلال ملاءمة العديد من المعلمات تجميع مجموعة بيانات هائلة. ومع ذلك، أصبح بإمكان كل من البشر وأجهزة الكمبيوتر اليوم التمييز بين القطط والكلاب جيداً، الأمر الذي يتعارض على ما يبدو مع هذه البديهيات. وذلك لأن الصور تظهر بُنية غنية يمكن أن يستغلها البشر ونماذج التعلم الآلي على حد سواء. الشبكات العصبية التلافيفية (CNN) هي إحدى الطرق الإبداعية التي تبناها التعلم الآلي لاستغلال بعض الهياكل المعروفة في الصور الطبيعية.

قد يعترض القارئ الحريص على هذه الحجة على أساس أن دقة واحدة ميغا بكسل قد لا تكون ضرورية. ومع ذلك،في حين أننا قد نكون قادرين على التخلص من مائة ألف بكسل، فإن الطبقة المخفية التي يبلغ حجمها 1000 بكسل تقلل بشكل كبير من عدد الوحدات المخفية التي يتطلبها تعلم التمثيل الجيد للصور، لذلك سيظل النظام العملي يتطلب مليارات من المعلمات. علاوة على ذلك، قد يتطلب تعلم المصنف من خلال ملاءمة العديد من المعلمات تجميع مجموعة بيانات هائلة. ومع ذلك، أصبح بإمكان كل من البشر وأجهزة الكمبيوتر اليوم التمييز بين القطط والكلاب جيداً، الأمر الذي يتعارض على ما يبدو مع هذه البديهيات. وذلك لأن الصور تظهر بُنية غنية يمكن أن يستغلها البشر ونماذج التعلم الآلي على حد سواء. الشبكات العصبية التلافيفية (CNN) هي إحدى الطرق الإبداعية التي تبناها التعلم الآلي لاستغلال بعض الهياكل المعروفة في الصور الطبيعية.

7.1.1. الثبات Invariance

تخيل أننا نريد اكتشاف كائنفي صورة ما. يبدو من المعقول أنه أيا كانت الطريقة التي نستخدمها للتعرف على الأشياء لا ينبغي أن تهتم بشكل مفرط بالموقع الدقيق للكائنفي الصورة. من الناحية المثالية، يجب أن يستغل نظامنا هذه المعرفة. وعادة لا تطير الخنازير ولا تسبح الطائرات عادة. ومع ذلك، لا يزال يتعين علينا التعرف على الخنزير الذي ظهرفي أعلى الصورة. يمكننا استلهام بعض الإلهام هنا من لعبة الأطفال "أين والدو Where's Waldo" (الموضحةفي الشكل 7.1.1). تتكون اللعبة من عدد من المشاهد الفوضوية المليئة بالأنشطة. يظهر والدوفي مكان مافي كل منهما، وعادة ما يتربصفي مكان غير متوقع. هدف القارئ هو تحديد مكانه. على الرغم من مظهره المميز، قد يكون هذا صعبًا بشكل كبير، بسبب العدد الكبير من المشتتات. ومع ذلك، لا يعتمد شكل والدو على مكان وجود والدو. يمكننا مسح الصورة باستخدام كاشف والدو Waldo detector الذي يمكنه تعيين درجة لكل رقعة patch، مما يشير إلى احتمالية احتواء التصحيح على والدو.في الواقع، تعتمد العديد من خوارزميات اكتشاف الكائنات وتجزئتها على هذا النهج

(Long et al.، 2015). تنظم شبكات CNN فكرة الثبات المكاني spatial invariance هذه، وتستغلها لتعلم تمثيلات representations مفيدة بمعلمات أقل.

الشكل 7.1.1 صورة للعبة "Where's Waldo".

يمكننا الآن جعل هذه البديهيات أكثر واقعية من خلال تعداد بعض الرغبات لتوجيه تصميمنا لبُنية شبكة عصبية مناسبة للرؤية الحاسوبية computer vision:

1. في الطبقات الأولى، يجب أن تستجيب شبكتنا بشكل مشابه لنفس الرقعة ، بغض النظر عن مكان ظهوره في الصورة. يسمى هذا المبدأ بثبات الترجمة translation invariance (أو تساوي الترجمة translation equivariance).
2. يجب أن تركز الطبقات الأولى للشبكة على المناطق المحلية local regions، دون اعتبار لمحتويات الصورة في المناطق البعيدة. هذا هو مبدأ المكان. في النهاية ، يمكن تجميع هذه التمثيلات المحلية local representations لعمل تنبؤات على مستوى الصورة بأكملها.
3. أثناء تقدمنا، يجب أن تكون الطبقات الأعمق قادرة على التقاط ميزات أطول مدى longer-range featuresللصورة ، بطريقة مشابهة للرؤية ذات المستوى الأعلى في الطبيعة.

دعونا نرى كيف يترجم هذا إلى رياضيات.

7.1.2. تقييد MLP (Constraining the MLP)

للبدء، يمكننا اعتبار MLP مع صور ثنائية الأبعاد **X** كمدخلات وتمثيلاتها المخفية الفورية **H** ممثلة بشكل مشابه كمصفوفات (هما موترات ثنائية الأبعادفي الكود)، حيث **X** و **H** لهما نفس الشكل. نحن الآن نتصور ليس فقط المدخلات ولكن أيضًا التمثيلات الخفية على أنها تمتلك بنية مكانية spatial structure.

ليكن $[\mathbf{X}]_{i,j}$ و$[\mathbf{H}]_{i,j}$ يدلان على البكسلفي الموقع (i,j) في صورة الإدخال input image والتمثيل المخفي hidden representation، على التوالي. وبالتالي، لكي تتلقى كل وحدة من الوحدات المخفية مدخلات من كل بكسل من وحدات البكسل المدخلة، سننتقل من استخدام مصفوفات الوزن weight matrices (كما فعلنا سابقًافي MLPs) لتمثيل معلماتنا كموترات للوزن من الدرجة الرابعة W. لنفترض أن **U** يحتوي على تحيزات، فيمكننا التعبير رسميًا عن الطبقة المتصلة بالكامل كـ

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l}[\mathbf{X}]_{k,l}$$
$$= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b}[\mathbf{X}]_{i+a,j+b}.$$

يعتبر التبديل من W إلى V تجميليًا تمامًافي الوقت الحالي نظرًا لوجود تطابق واحد إلى واحد بين المعاملاتفي كلا الموترات من الدرجة الرابعة. نحن ببساطة نعيد فهرسة الرموز المنخفضة (k,l) مثل ذلك $k = i + a$ و $l = j + b$. بعبارة أخرى، وضعنا $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}$. المؤشرات a وb تتخطى كل من الإزاحات offsets الإيجابية والسلبية، وتغطي الصورة بأكملها. بالنسبة إلى أي موقع (i, j)في التمثيل المخفي $[\mathbf{H}]_{i,j}$ ، نحسب قيمته عن طريق جمع أكثر من بكسلفي x ، وتوسيطها (i,j) ، واخذ الاوزان لها بواسطة $[\mathsf{V}]_{i,j,a,b}$. قبل أن نواصل العمل، دعنا نفكرفي العدد الإجمالي للمعلمات المطلوبة لطبقة واحدة singlefي هذه المعلمات: يتم تعيين صورة 1000×1000 (1 ميغا بكسل) إلى تمثيل مخفي 1000×1000. يتطلب هذا معلمات 10^{12} ، تتجاوز بكثير ما يمكن لأجهزة الكمبيوتر التعامل معه حاليًا.

7.1.2.1. ثبات الترجمة Translation Invariance

الآن دعونا نستدعي المبدأ الأول الذي تم وضعه أعلاه: ثبات الترجمة (Zhang and others، 1988). هذا يعني أن التحولفي المدخلات **X** يجب أن يؤدي ببساطة إلى تحولفي التمثيل الخفي **H**. هذا ممكن فقط إذا V و **U** لا تعتمدان على (i,j) بالفعل. على هذا النحو، لدينا $[\mathsf{V}]_{i,j,a,b} = [\mathbf{V}]_{a,b}$ و**U** هو ثابت، على سبيل المثال u. نتيجة لذلك، يمكننا تبسيط تعريف **H**:

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}. \qquad (7.1.2)$$

هذا هو الالتواء convolution! نحن نأخذ الاوزان بشكل فعال وحدات البكسل في $(i+a, j+b)$ بالقرب من الموقع (i,j) باستخدام المعاملات $[\mathbf{V}]_{a,b}$ للحصول على القيمة $[\mathbf{H}]_{i,j}$. لاحظ أن $[\mathbf{V}]_{a,b}$ يحتاج إلى عدد أقل من المعاملات من $[V]_{i,j,a,b}$ لأنه لم يعد يعتمد على الموقع داخل الصورة. وبالتالي، لم يعد عدد المعلمات المطلوبة 10^{12} ولكن أكثر معقولية هو 10^{12}: لا يزال لدينا الاعتماد على $a, b \in (-1000,1000)$. باختصار، لقد أحرزنا تقدمًا كبيرًا. الشبكات العصبية ذات التأخير الزمني Time-delay neural networks (TDNNs) هي بعض الأمثلة الأولى لاستغلال هذه الفكرة (Waibel et al.، 1989).

7.1.2.2. المحلية Locality

الآن دعونا نستدعي المبدأ الثاني: المحلية Locality. كما تم تحفيزنا أعلاه، نعتقد أنه لا ينبغي علينا أن ننظر بعيدًا جدًا عن الموقع (i,j) من أجل جمع المعلومات ذات الصلة لتقييم ما يجري في $[\mathbf{H}]_{i,j}$. هذا يعني أنه خارج نطاق ما $|a| > \Delta$ ، أو $|b| > \Delta$ يجب علينا ضبط $[\mathbf{V}]_{a,b} = 0$. بالتساوي، يمكننا إعادة كتابة $[\mathbf{H}]_{i,j}$ كـ

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \qquad (7.1.3)$$

يؤدي هذا إلى تقليل عدد المعلمات من $4 \cdot 10^6$ إلى $4\Delta^2$ ، حيث يكون عادةً أصغر من 10. على هذا النحو، قمنا بتقليل عدد المعلمات بمقدار 4 أوامر أخرى من حيث الحجم. لاحظ أن (7.1.3)، باختصار، هي ما يسمى بالطبقة التلافيفية convolutional layer. الشبكات العصبية التلافيفية (CNN) هي عائلة خاصة من الشبكات العصبية التي تحتوي على طبقات تلافيفية.في مجتمع أبحاث التعلم العميق، $\mathbf{V}$ يُشار إلى نواة الالتفاف convolution kernel, أو عامل التصفية filter، أو ببساطة أوزان الطبقة layer's weights التي تُعد معلمات قابلة للتعلم.

بينمافي السابق، ربما كنا قد طلبنا بلايين من المعلمات لتمثيل طبقة واحدة فقطفي شبكة معالجة الصور، فنحن نحتاج الآن عادةً إلى بضع مئات فقط، دون تغيير أبعاد المدخلات أو التمثيلات المخفية. الثمن المدفوع لهذا التخفيض الكبيرفي المعلمات هو أن ميزاتنا أصبحت الآن ترجمة ثابتة translation invariant وأن طبقتنا يمكنها فقط دمج المعلومات المحلية، عند تحديد قيمة كل تنشيط مخفي. كل التعلم يعتمد على فرض التحيز الاستقرائي inductive bias. عندما يتفق هذا التحيز مع الواقع، نحصل على نماذج ذات كفاءةفي العينة تُعمم جيدًا على البيانات غير المرئية. لكن بالطبع، إذا كانت هذه التحيزات لا تتفق مع الواقع، على سبيل المثال، إذا تبين أن الصور ليست ثابتةفي الترجمة، فقد تكافح نماذجنا حتى لتلائم بيانات التدريب الخاصة بنا.

يقودنا هذا الانخفاض الدراماتيكي dramatic reduction في المعلمات إلى آخر رغباتنا، أي أن الطبقات العميقة يجب أن تمثل جوانب أكبر وأكثر تعقيدًا للصورة. يمكن تحقيق ذلك عن طريق ادخال interleaving الطبقات اللاخطية والطبقات التلافيفية بشكل متكرر.

7.1.3. التلافيف Convolutions

دعونا نراجع بإيجاز سبب تسمية (7.1.3) بالالتفاف convolution. في الرياضيات، يتم تعريف الالتفاف بين دالتين (Rudin، 1973)، على سبيل المثال $f, g: \mathbb{R}^d \to \mathbb{R}$ كالاتي:

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z}) g(\mathbf{x} - \mathbf{z}) d\mathbf{z}. \qquad (7.1.4)$$

وهذا يعني أننا نقيس التداخل overlap بين f وg متى يتم "قلب flipped" إحدى الدوال وإزاحتها "shifted " بواسطة $\mathbf{x}$. عندما يكون لدينا كائنات منفصلة discrete objects، يتحول التكامل إلى مجموع. على سبيل المثال، بالنسبة إلى المتجهات من مجموعة متجهات الأبعاد اللانهائية التي يمكن جمعها مربعة مع مؤشر يعمل فوق $\mathbb{Z}$، نحصل على التعريف التالي:

$$(f * g)(i) = \sum_a f(a) g(i - a). \qquad (7.1.5)$$

بالنسبة إلى الموترات ثنائية الأبعاد، لدينا مجموع مقابل مع مؤشرات (a, b) لـ f و$(i - a, j - b)$ لـ g، على التوالي:

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b). \qquad (7.1.6)$$

يبدو هذا مشابهًا لـ (7.1.3)، مع اختلاف رئيسي واحد. بدلاً من استخدام $(i + a, j + b)$، نستخدم الاختلاف difference بدلاً من ذلك. لاحظ، مع ذلك، أن هذا التمييز distinction هو في الغالب تجميلي حيث يمكننا دائمًا مطابقة التدوين بين (7.1.3) و (7.1.6). يصف تعريفنا الأصلي في (7.1.3) بشكل أكثر ملاءمة الارتباط المتبادل cross-correlation. سنعود إلى هذا في القسم التالي.

7.1.4. القنوات Channels

بالعودة إلى كاشف والدو Waldo detector، دعونا نرى كيف يبدو هذا. تلتقط الطبقة التلافيفية نوافذ ذات حجم معين وتزن شدة وفقًا للفلتر V، كما هو موضح في الشكل 7.1.2. قد نهدف إلى تعلم نموذج بحيث حيثما تكون "waldoness" هي الأعلى، يجب أن نجد ذروة في تمثيلات الطبقة المخفية.

الشكل 7.1.2 كشف والدو.

هناك مشكلة واحدة فقطفي هذا النهج. حتى الآن، تجاهلنا بسعادة أن الصور تتكون من 3 قنوات: الأحمر والأخضر والأزرق. باختصار، الصور ليست كائنات ثنائية الأبعاد بل هي موترات من الدرجة الثالثة، تتميز بالارتفاع والعرض والقناة، على سبيل المثال، مع شكل $1024 \times 1024 \times 3$ بكسل.في حين أن أول محورين يتعلقان بالعلاقات المكانية spatial relationships، يمكن اعتبار الثالث على أنه تعيين تمثيل متعدد الأبعاد لكل موقع بكسل. وبالتالي نقوم بفهرسة X كـ $[X]_{i,j,k}$. يجب أن يتكيف الفلتر التلافيفي convolutional filter وفقًا لذلك. بدلا من $[\mathbf{V}]_{a,b}$، لدينا الآن $[V]_{a,b,c}$.

علاوة على ذلك، كما أن مدخلاتنا تتكون من موتر من الدرجة الثالثة، فقد تبين أنها فكرة جيدة لصياغة تمثيلاتنا المخفية بالمثل كموترات من الدرجة الثالثة H. بعبارة أخرى، بدلاً من مجرد وجود تمثيل مخفي واحد يتوافق مع كل موقع مكاني، نريد متجهًا كاملاً للتمثيلات المخفية المقابلة لكل موقع مكاني. يمكننا التفكيرفي التمثيلات المخفية على أنها تتألف من عدد من الشبكات ثنائية الأبعاد المكدسة فوق بعضها البعض. كما هو الحالفي المدخلات، تسمى هذه أحيانًا القنوات channels. يطلق عليها أيضًا أحيانًا خرائط المعالم feature maps، حيث يوفر كل منها مجموعة مكانية من المعالم المكتسبة للطبقة التالية. بشكل بديهي، قد تتخيل أنهفي الطبقات السفلية الأقرب إلى المدخلات، يمكن أن تصبح بعض القنوات متخصصة للتعرف على الحواف edges بينما يمكن للآخرين التعرف على الأنسجة textures.

لدعم قنوات متعددةفي كل من المدخلات (X) والتمثيلات المخفية (H)، يمكننا إضافة إحداثي رابع إلى V: $[V]_{a,b,c,d}$. نجمع كل شيء معًا لدينا:

$$[\mathrm{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathrm{V}]_{a,b,c,d} [\mathrm{X}]_{i+a,j+b,c}, \qquad (7.1.6)$$

حيث d فهرسة قنوات الإخراجفي التمثيلات المخفية H. ستستمر الطبقة التلافيفية اللاحقةفي أخذ موتر من الدرجة الثالثة H، كمدخل. لكونها أكثر عمومية، (7.1.7) هو تعريف الطبقة التلافيفية لقنوات متعددة، حيث تكون النواة kernel أو فلتر الطبقة.

لا يزال هناك العديد من العمليات التي نحتاج إلى معالجتها. على سبيل المثال، نحتاج إلى معرفة كيفية دمج جميع التمثيلات المخفيةفي ناتج واحد، على سبيل المثال، ما إذا كان هناك والدو Waldo في أي مكانفي الصورة. نحتاج أيضًا إلى تحديد كيفية حساب الأشياء بكفاءة، وكيفية الجمع بين طبقات متعددة، ودوال التنشيط المناسبة، وكيفية اتخاذ خيارات تصميم معقولة لإنتاج شبكات فعالةفي الممارسة. ننتقل إلى هذه القضاياافي بقية الفصل.

7.1.5. الملخص والمناقشة

في هذا القسم اشتقنا بُنية الشبكات العصبية التلافيفية من المبادئ الأولى.في حين أنه من غير الواضح ما إذا كان هذا هو ما أدى إلى اختراع شبكات CNN، فمن المرضي معرفة أنها الخيار الصحيح عند تطبيق مبادئ معقولة لكيفية عمل معالجة الصور وخوارزميات الرؤية الحاسوبية، على الأقل عند المستويات الأدنى. على وجه الخصوص، يشير ثبات الترجمة translation invarianceفي الصور إلى أنه سيتم التعامل مع جميع بقع الصورة بنفس الطريقة. تعني المنطقة المحلية Locality أنه سيتم استخدام جوار صغير فقط من وحدات البكسل لحساب التمثيلات المخفية المقابلة. بعض الإشارات المبكرة لشبكات CNN هيفي شكل Neocognitron (Fukushima، 1982).

المبدأ الثاني الذي واجهناهفي تفكيرنا هو كيفية تقليل عدد المعلماتفي فئة دالة دون الحد من قوتها التعبيرية، على الأقل، عندما تصمد افتراضات معينةفي النموذج. لقد رأينا انخفاضًا كبيرًافي التعقيد نتيجة لهذا التقييد، وتحويل المشكلات غير القابلة للتنفيذ من الناحية الحسابية والإحصائية إلى نماذج يمكن تتبعها.

سمحت لنا إضافة القنوات channels بإعادة بعض التعقيد الذي فقد بسبب القيود المفروضة على النواة التلافيفية convolutional kernel من خلال المحلية وثبات الترجمة. لاحظ أن القنوات هي إضافة طبيعية إلى حد ما تتجاوز الأحمر والأخضر والأزرق. العديد من صور الأقمار الصناعية، خاصة للزراعة والأرصاد الجوية، لديها عشرات إلى مئات القنوات، وتولد صورًا فائقة الطيف بدلاً من ذلك. يقدمون بيانات حول العديد من الأطوال الموجية المختلفة. فيما يلي سوف نرى كيفية استخدام التلافيف بشكل فعال لمعالجة أبعاد الصور التي تعمل عليها، وكيفية الانتقال

من التمثيل القائم على الموقع location-based إلى التمثيل القائم على القناة channel-based وكيفية التعامل مع عدد كبير من الفئات بكفاءة.

7.1.6. التمارين

1. افترض أن حجم نواة الالتفاف convolution kernel هو $\Delta = 0$. أظهر أنه في هذه الحالة ، تنفذ نواة الالتفاف MLP بشكل مستقل لكل مجموعة من القنوات. هذا يؤدي إلى الشبكة في معماريات الشبكة (لين وآخرون ، 2013).
2. غالبًا ما يتم تمثيل البيانات الصوتية على أنها تسلسل أحادي البعد one-dimensional sequence.
   1. متى قد ترغب في فرض المحلية وثبات الترجمة translation /locality invariance على الصوت؟
   2. اشتق عمليات الالتفاف للصوت.
   3. هل يمكنك معالجة الصوت باستخدام نفس أدوات الرؤية الحاسوبية؟ تلميح: استخدم المخطط الطيفي spectrogram.
3. لماذا قد لا يكون ثبات الترجمة فكرة جيدة بعد كل شيء؟ اعط مثالا.
4. هل تعتقد أن الطبقات التلافيفية قد تكون قابلة للتطبيق أيضًا على البيانات النصية؟ ما هي المشاكل التي قد تواجهها مع اللغة؟
5. ماذا يحدث مع التلافيف عندما يكون الكائن في حدود الصورة.
6. إثبت أن الالتفاف متماثل symmetric، أي $f * g = g * f$.
7. إثبت نظرية الالتفاف convolution theorem، أي $f * g = \mathcal{F}^{-1}[\mathcal{F}[f] \cdot \mathcal{F}[g]]$. هل يمكنك استخدامه لتسريع التلافيف؟

7.2 التلافيف للصور Convolutions for Images

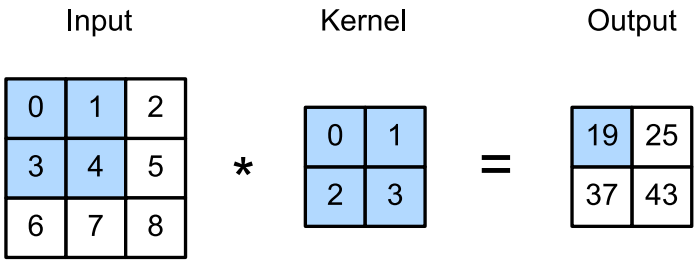
الآن بعد أن فهمنا كيفية عمل الطبقات التلافيفية convolutional layers من الناحية النظرية، نحن مستعدون لنرى كيف تعمل في الممارسة العملية. بناءً على دافعنا للشبكات العصبية التلافيفية CNN باعتبارها بنيات فعالة لاستكشاف البُنية في بيانات الصورة، فإننا نتمسك بالصور كمثال قيد التشغيل.

7.2.1. عملية الارتباط المتبادل The Cross-Correlation Operation

تذكر أن الطبقات التلافيفية convolutional layers هي تسمية خاطئة بالمعنى الدقيق للكلمة، لأن العمليات التي تعبر عنها توصف بدقة أكبر على أنها ارتباطات متبادلة cross-correlations. بناءً على أوصافنا للطبقات التلافيفية في القسم 7.1، في مثل هذه الطبقة، يتم دمج موتر الإدخال

input tensor وموتّر النواة kernel tensor لإنتاج موتر ناتج output tensor من خلال عملية الارتباط المتبادل cross-correlation.

دعنا نتجاهل القنوات channels في الوقت الحالي ونرى كيف يعمل ذلك مع البيانات ثنائية الأبعاد والتمثيلات المخفية.في الشكل 7.2.1، المدخلات عبارة عن موتر ثنائي الأبعاد بارتفاع 3 وعرض 3. ونضع علامة على شكل موتر كالتالي 3×3 أو (3،3). ارتفاع وعرض النواة كلاهما 2. شكل نافذة النواة kernel window (أو نافذة الالتفاف convolution window) يُعطى بارتفاع وعرض النواة (هنا 2×2).



الشكل 7.2.1 عملية الارتباط المتبادل ثنائية الأبعاد Two-dimensional cross-correlation operation. الأجزاء المظللة هي أول عنصر إخراج بالإضافة إلى عناصر موتر الإدخال والنواة المستخدمة لحساب الإخراج:$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$.

في عملية الارتباط المتبادل ثنائية الأبعاد، نبدأ مع نافذة الالتفاف الموضوعةفي الزاوية اليسرى العليا من موتر الإدخال ونزلقها عبر موتر الإدخال، من اليسار إلى اليمين ومن أعلى إلى أسفل. عندما تنزلق نافذة الالتفاف إلى موضع معين، يتم ضرب مستشعر الإدخال الفرعي الموجودفي تلك النافذة وموتّر النواة بطريقة عنصرية elementwise ويتم تلخيص الموتر الناتج لإنتاج قيمة عددية واحدة. تعطي هذه النتيجة قيمة موتر الإخراجفي الموقع المقابل. هنا، موتر الإخراج له ارتفاع 2 وعرض 2 ويتم اشتقاق العناصر الأربعة من عملية الارتباط المتبادل ثنائية الأبعاد:

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

لاحظ أنه على طول كل محور، يكون حجم الإخراج أصغر قليلاً من حجم الإدخال. نظرًا لأن النواة تحتوي على عرض وارتفاع أكبر من واحد، يمكننا فقط حساب الارتباط المتبادل للمواقع التي تتلاءم فيها النواة بالكامل مع الصورة، ويعطى حجم الإخراج من خلال حجم الإدخال $n_h \times n_w$ مطروحًا منه حجم نواة الالتفاف $k_h \times k_w$ عبر

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

هذا هو الحال لأننا نحتاج إلى مساحة كافية "لنقل shift" نواة الالتفاف convolution kernel عبر الصورة. سنرى لاحقًا كيفية الحفاظ على الحجم دون تغيير عن طريق حشو padding الصورة بأصفار حول حدودها بحيث يكون هناك مساحة كافية لإزاحة النواة. بعد ذلك، نقوم بتنفيذ هذه العمليةفي دالة corr2d، التي تقبل موتر الإدخال X وموتّر النواة K وتعيد موتر الإخراج Y.

```
import tensorflow as tf
from d2l import tensorflow as d2l


def corr2d(X, K):  #@save
    """Compute 2D cross-correlation."""
    h, w = K.shape
    Y = tf.Variable(tf.zeros((X.shape[0] - h + 1,
X.shape[1] - w + 1)))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i, j].assign(tf.reduce_sum(
                X[i: i + h, j: j + w] * K))
    return Y
```

يمكننا إنشاء موتر الإدخال X وموتر النواة K من الشكل 7.2.1 للتحقق من صحة إخراج التنفيذ أعلاه لعملية الارتباط المتبادل ثنائية الأبعاد.

```
X = tf.constant([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0,
7.0, 8.0]])
K = tf.constant([[0.0, 1.0], [2.0, 3.0]])
corr2d(X, K)
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32,
numpy=
array([[19., 25.],
       [37., 43.]], dtype=float32)>
```

7.2.2. الطبقات التلافيفية Convolutional Layers

تربط الطبقة التلافيفية Convolutional Layer بين المدخلات والنواة وتضيف تحيزًا قياسيًا لإنتاج مخرجات. المعلمتان للطبقة التلافيفية هما النواة kernel والتحيز القياسي scalar bias. عند تدريب النماذج على أساس الطبقات التلافيفية، نقوم عادةً بتهيئة النواة بشكل عشوائي، تمامًا كما نفعل مع طبقة متصلة بالكامل fully connected layer.

نحن الآن جاهزون لتنفيذ طبقة تلافيفية ثنائية الأبعاد بناءً على دالة corr2d المحددة أعلاه.في طريقة المُنشئ __init__، نعلن الوزن weight والتحيز bias كمعاملين للنموذج. تستدعي دالة الانتشار الأمامي الدالة corr2d وتضيف التحيز.

```
class Conv2D(tf.keras.layers.Layer):
    def __init__(self):
        super().__init__()

    def build(self, kernel_size):
        initializer = tf.random_normal_initializer()
        self.weight = self.add_weight(name='w',
shape=kernel_size,

initializer=initializer)
        self.bias = self.add_weight(name='b', shape=(1,
),

initializer=initializer)

    def call(self, inputs):
        return corr2d(inputs, self.weight) + self.bias
```

في الالتفاف $h \times w$ أو نواة الالتفاف $h \times w$، يكون ارتفاع وعرض نواة الالتفاف h و w، على التوالي. نشير أيضًا إلى الطبقة التلافيفية ذات النواة الالتفافية $h \times w$ باعتبارها طبقة تلافيفية $h \times w$.

7.2.3. كشف حواف الكائن في الصور Object Edge Detection in Images

دعنا نتوقف لحظة لتحليل تطبيق بسيط لطبقة تلافيفية: اكتشاف حافة كائنفي صورة من خلال إيجاد موقع تغيير البكسل. أولاً، نقوم ببناء "صورة" لـ 6×8 بكسل. الأعمدة الأربعة الوسطى سوداء (0) والباقي بيضاء (1).

```
X = tf.Variable(tf.ones((6, 8)))
X[:, 2:6].assign(tf.zeros(X[:, 2:6].shape))
X
```

```
<tf.Variable 'Variable:0' shape=(6, 8) dtype=float32,
numpy=
array([[1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
```

```
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.]],
dtype=float32)>
```

بعد ذلك، نقوم ببناء kernel K بارتفاع 1 وعرض 2. عندما نجري عملية الارتباط التبادلي مع الإدخال، إذا كانت العناصر المتجاورة adjacent elements أفقيًا هي نفسها، يكون الناتج 0. خلاف ذلك، يكون الناتج غير صفري. لاحظ أن هذه النواة هي حالة خاصة لمشغل الفروق المحدودة.في الموقع (i,j)، يحسب $x_{i,j} - x_{(i+1),j}$، أي أنه يحسب الفرق بين قيم وحدات البكسل المتجاورة أفقيًا. هذا تقريب متقطع للمشتق الأولفي الاتجاه الأفقي. بعد كل شيء، من أجل دالة $f(i,j)$ مشتقها $-\partial_i f(i,j) = \lim_{\epsilon \to 0} \frac{f(i,j) - f(i+\epsilon,j)}{\epsilon}$. دعونا نرى كيف يعمل هذافي الممارسة.

```
K = tf.constant([[1.0, -1.0]])
```

نحن على استعداد لإجراء عملية الارتباط المتبادل مع الوسيطات X (المدخلات الخاصة بنا) وK (النواة الخاصة بنا). كما ترى، نكتشف 1 للحافة من الأبيض إلى الأسود و -1 للحافة من الأسود إلى الأبيض. جميع النواتج الأخرى تأخذ القيمة 0.

```
Y = corr2d(X, K)
Y
```

```
<tf.Variable 'Variable:0' shape=(6, 7) dtype=float32,
numpy=
array([[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.]],
dtype=float32)>
```

يمكننا الآن تطبيق النواة على الصورة المنقولة transposed image. كما هو متوقع، فإنه يتلاشى. يكتشف النواة K الحواف الرأسية vertical edges فقط.

```
corr2d(tf.transpose(X), K)
```

```
<tf.Variable 'Variable:0' shape=(8, 5) dtype=float32,
numpy=
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
```

```
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]], dtype=float32)>
```

7.2.4. تعلم النواة Learning a Kernel

يُعد تصميم كاشف الحواف بالاختلافات المحدودة `[1, -1]` أمرًا رائعًا إذا علمنا أن هذا هو بالضبط ما نبحث عنه. ومع ذلك، عندما ننظر إلى نوى أكبر، ونأخذفي الاعتبار طبقات التلافيف المتتالية، فقد يكون من المستحيل التحديد الدقيق لما يجب أن يفعله كل فلتر يدويًا.

الآن دعونا نرى ما إذا كان بإمكاننا تعلم النواة التي ولدت Y من X من خلال النظر إلى أزواج الإدخال والإخراج فقط. نقوم أولاً ببناء طبقة تلافيفية وتهيئة نواتها كموتّر عشوائي. بعد ذلك،في كل تكرار، سنستخدم الخطأ التربيعي لمقارنة Y بإخراج الطبقة التلافيفية. يمكننا بعد ذلك حساب التدرج لتحديث النواة. من أجل البساطة، فيما يلي نستخدم الكلاس المدمج للطبقات التلافيفية ثنائية الأبعاد ونتجاهل التحيز.

```
# Construct a two-dimensional convolutional layer with 1
output channel and a
# kernel of shape (1, 2). For the sake of simplicity, we
ignore the bias here
conv2d = tf.keras.layers.Conv2D(1, (1, 2),
use_bias=False)

# The two-dimensional convolutional layer uses four-
dimensional input and
# output in the format of (example, height, width,
channel), where the batch
# size (number of examples in the batch) and the number
of channels are both 1
X = tf.reshape(X, (1, 6, 8, 1))
Y = tf.reshape(Y, (1, 6, 7, 1))
lr = 3e-2  # Learning rate

Y_hat = conv2d(X)
for i in range(10):
    with tf.GradientTape(watch_accessed_variables=False)
as g:
        g.watch(conv2d.weights[0])
        Y_hat = conv2d(X)
        l = (abs(Y_hat - Y)) ** 2
        # Update the kernel
```

```
        update = tf.multiply(lr, g.gradient(l,
conv2d.weights[0]))
        weights = conv2d.get_weights()
        weights[0] = conv2d.weights[0] - update
        conv2d.set_weights(weights)
        if (i + 1) % 2 == 0:
            print(f'epoch {i + 1}, loss
{tf.reduce_sum(l):.3f}')
```

```
epoch 2, loss 1.306
epoch 4, loss 0.400
epoch 6, loss 0.141
epoch 8, loss 0.054
epoch 10, loss 0.022
```

لاحظ أن الخطأ قد انخفض إلى قيمة صغيرة بعد 10 تكرارات. الآن سوف نلقي نظرة على موتر النواة الذي تعلمناه.

```
tf.reshape(conv2d.get_weights()[0], (1, 2))
```

```
<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[
1.0111188 , -0.98112327]], dtype=float32)>
```

في الواقع، موتر النواة المكتسبة قريب بشكل ملحوظ من موتر النواة K الذي حددناه سابقًا.

7.2.5. الارتباط المتبادل والالتفاف Cross-Correlation and Convolution

تذكر ملاحظتنا من القسم 7.1 من المراسلات بين الارتباط المتبادل cross-correlation وعمليات الالتفاف convolution operations. دعنا هنا نواصل النظرفي الطبقات التلافيفية ثنائية الأبعاد. ماذا لو قامت هذه الطبقات بإجراء عمليات التفاف صارمة strict convolution operation كما هو محددفي (7.1.6) بدلاً من الارتباطات المتقاطعة؟ من أجل الحصول على ناتج عملية الالتفاف الصارمة، نحتاج فقط إلى قلب موتر النواة ثنائي الأبعاد أفقيًا وعموديًا، ثم إجراء عملية الارتباط المتبادل مع موتر الإدخال.

من الجدير بالذكر أنه نظرًا لأن النوى يتم تعلمها من البياناتفي التعلم العميق، فإن مخرجات الطبقات التلافيفية تظل غير متأثرة بغض النظر عن أن هذه الطبقات تؤدي إما عمليات الالتفاف الصارمة أو عمليات الارتباط المتبادل.

لتوضيح ذلك، افترض أن الطبقة التلافيفية تقوم بعمل ارتباط متبادل وتتعرف على النواةفي الشكل 7.2.1، والتي يشار إليها هنا بالمصفوفة **K**. بافتراض أن الشروط الأخرى تظل دون تغيير، عندما تقوم هذه الطبقة بإجراء التفاف صارم strict convolution بدلاً من ذلك، فإن النواة المتعلمة learned kernel $\mathbf{K}'$ ستكون كما **K** بعد قلبها أفقيًا وعموديًا. وهذا يعني أنه عندما تقوم الطبقة

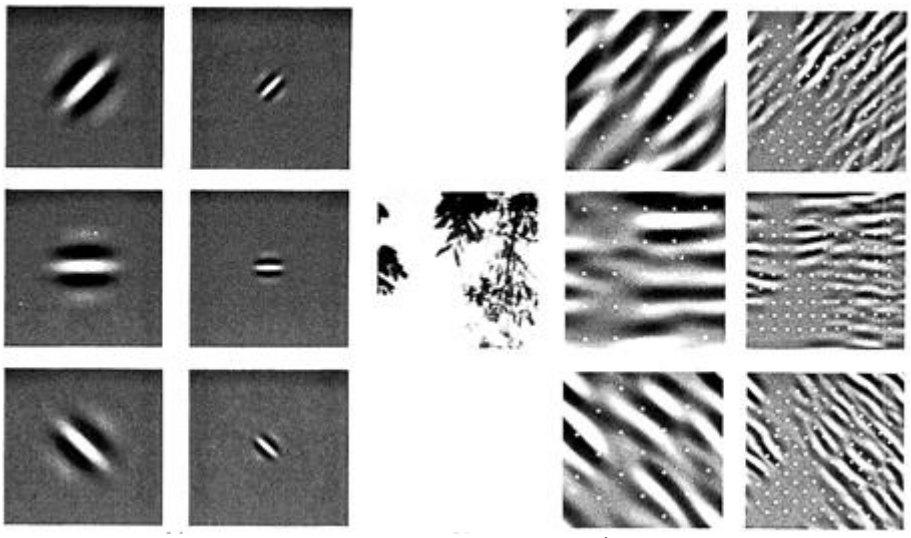
التلافيفية بإجراء التفاف صارم للمدخلاتفي الشكل 7.2.1، وسيتم الحصول على نفس الخرج **K**´في الشكل 7.2.1 (الارتباط المتبادل للمدخل و **K**).

7.2.6. خريطة المعالم وحقل التأثير Feature Map and Receptive Field

كما هو موضحفي القسم 7.1.4، يُطلق أحيانًا على ناتج الطبقة التلافيفيةفي الشكل 7.2.1 اسم خريطة المعالم feature map، حيث يمكن اعتبارها التمثيلات المتعلمة learned representations (السمات)في الأبعاد المكانية spatial dimensions (مثل العرض والارتفاع) إلى الطبقة اللاحقة.في شبكات CNN، بالنسبة لأي عنصرفي طبقة ما، يشير مجالها التأثيري receptive field إلى جميع العناصر (من جميع الطبقات السابقة) التي قد تؤثر على حساب x أثناء الانتشار الأمامي. لاحظ أن حقل التأثير قد يكون أكبر من الحجم الفعلي للإدخال.

دعونا نواصل استخدام الشكل 7.2.1 لشرح مجال التأثير. بالنظر إلى نواة الالتفاف 2 × 2، فإن مجال التأثير لعنصر الإخراج المظلل (للقيمة 19) هو العناصر الأربعةفي الجزء المظلل من المدخلات. دعنا الآن نشير إلى المخرجات 2 × 2 على **Y** وهي أعمق CNN مع طبقة تلافيفية إضافية 2 × 2 تأخذ **Y** كمدخلاتها، وتخرج عنصرًا واحدًا z.في هذه الحالة، يشتمل حقل التأثير لـ z على **Y** لجميع العناصر الأربعة لـ **Y**، بينما يشتمل حقل التأثير على الإدخال على جميع عناصر الإدخال التسعة. وبالتالي، عندما يحتاج أي عنصرفي خريطة المعالم إلى مجال تأثير أكبر لاكتشاف ميزات الإدخال عبر منطقة أوسع، يمكننا بناء شبكة أعمق.

تستمد حقول التأثير اسمها من الفسيولوجيا العصبية neurophysiology.في سلسلة من التجارب (Hubel and Wiesel، 1959، Hubel and Wiesel، 1962، Hubel and Wiesel، 1968) على مجموعة من الحيوانات والمحفزات stimuli المختلفة، استكشف Hubel and Wiesel استجابة ما يسمى القشرة البصرية visual cortex على المحفزات المذكورة. بشكل عام، وجدوا أن المستويات الأدنى تستجيب للحواف والأشكال ذات الصلة.في وقت لاحق، أوضح فيلد (1987) هذا التأثير على الصور الطبيعية مع ما لا يمكن تسميته إلا النوى التلافيفية convolutional kernels. نعيد طبع رقم رئيسيفي الشكل 7.2.2 لتوضيح أوجه التشابه المذهلة.



الشكل 7.2.2 الشكل والتعليق مأخوذ من Field (1987): مثال على الترميز بست قنوات مختلفة. (يسار) أمثلة على ستة أنواع من أجهزة الاستشعار المرتبطة بكل قناة. (يمين) التفاف للصورةفي (الأوسط) مع ستة أجهزة استشعار موضحةفي (يسار). يتم تحديد استجابة أجهزة الاستشعار الفردية عن طريق أخذ عينات من هذه الصور التي تمت تصفيتها على مسافة تتناسب مع حجم المستشعر (موضح بالنقاط). يوضح هذا الرسم البياني استجابة المستشعرات المتماثلة فقط.

كما اتضح، فإن هذه العلاقة تنطبق حتى على الميزات المحسوبة بواسطة طبقات أعمق من الشبكات المدربة على مهام تصنيف الصور، كما هو موضح على سبيل المثال،في Kuzovkin et al. (2018). يكفي القول، لقد أثبتت التلافيف أنها أداة قوية بشكل لا يصدق للرؤية الحاسوبية، سواءفي علم الأحياء أوفي الكود. على هذا النحو، ليس من المستغرب (بعد فوات الأوان) أنهم بشروا بالنجاح الأخيرفي التعلم العميق.

7.2.7. الملخص

الحساب الأساسي المطلوب للطبقة التلافيفية هو عملية الارتباط المتبادل cross-correlation. لقد رأينا أن حلقة for-loop البسيطة المتداخلة هي كل ما هو مطلوب لحساب قيمتها. إذا كان لدينا مدخلات متعددة وقنوات إخراج متعددة، فإننا نقوم بإجراء عملية مصفوفة-مصفوفة بين القنوات. كما يمكن أن نرى، فإن الحساب مباشر، والأهم من ذلك أنه محلي للغاية. يوفر هذا تحسينًا كبيرًا للأجهزة والعديد من النتائج الحديثةفي الرؤية الحاسوبية ممكنة فقط بسبب ذلك. بعد كل شيء، هذا يعني أن مصممي الشرائح يمكنهم الاستثمارفي الحساب السريع بدلاً

من الذاكرة، عندما يتعلق الأمر بتحسين التلافيف.في حين أن هذا قد لا يؤدي إلى تصميمات مثالية للتطبيقات الأخرى، إلا أنه يفتح الباب أمام الرؤية الحاسوبيةفي كل مكان وبأسعار معقولة.

من حيث التلافيف نفسها، يمكن استخدامها لأغراض عديدة مثل اكتشاف الحواف والخطوط، أو تعتيم الصور blur images، أو شحذها sharpen. والأهم من ذلك، ليس من الضروري أن يخترع الإحصائي (أو المهندس) الفلاتر المناسبة. بدلاً من ذلك، يمكننا ببساطة تعلمها من البيانات. يحل هذا محل الاستدلال الهندسي المميز بالإحصاءات القائمة على الأدلة. أخيرًا، ومن دواعي سرورنا أن هذه المرشحات ليست مفيدة فقط لبناء شبكات عميقة ولكنها تتوافق أيضًا مع الحقول المستقبلة وخرائط الميزاتفي الدماغ. وهذا يمنحنا الثقة بأننا نسير على الطريق الصحيح.

7.2.8. التمارين

1. أنشئ صورة X ذات حواف قطرية diagonal edges.
   1. ماذا يحدث إذا قمت بتطبيق النواة K في هذا القسم عليها؟
   2. ماذا يحدث إذا غيرت موضع X؟
   3. ماذا يحدث إذا قمت بتغيير موضع K؟
2. صمم بعض النوى kernels يدويًا.
   1. بالنظر إلى متجه اتجاهي $\mathbf{v} = (v_1, v_2)$، قم باشتقاق نواة لاكتشاف الحواف والتي تكتشف الحواف المتعامدة إلى $\mathbf{v}$، أي ، الحواف في الاتجاه $(v_2, -v_1)$.
   2. اشتق عامل الفروقات المحدودة finite difference operator للمشتق الثاني. ما هو الحد الأدنى لحجم النواة التلافيفية المرتبطة بها؟ ما هي الهياكلفي الصور التي تستجيب لها بشدة؟
   3. كيف تصمم نواة ضبابية(معتمة) blur kernel؟ لماذا قد ترغبفي استخدام مثل هذه النواة؟
   4. ما هو الحد الأدنى لحجم النواة للحصول على مشتق من الرتبة d؟
3. عندما تحاول العثور تلقائيًا على التدرج gradient لكلاس Conv2Dالتي أنشأناها، ما نوع رسالة الخطأ التي تراها؟
4. كيف تمثل عملية الارتباط المتبادل كضرب مصفوفة عن طريق تغيير موترات الإدخال والنواة؟

7.3. الحشو والخطوة Padding and Stride

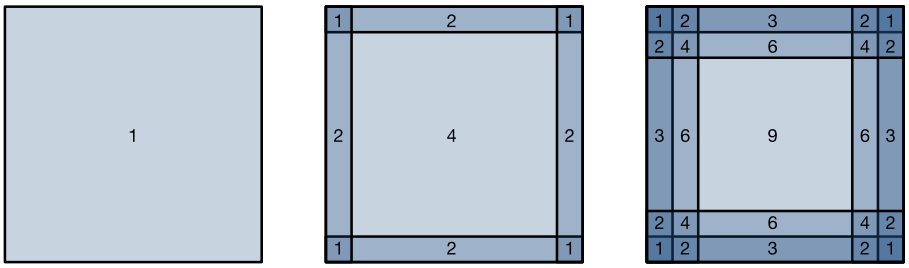
استدعي مثال الالتفافالشكل 7.2.1. كان لكل من المدخلات ارتفاع وعرض 3 وكان ارتفاع وعرض نواة الالتفاف 2، مما أدى إلى تمثيل إخراج بأبعاد 2×2. بافتراض أن شكل الإدخال هو $n_h \times n_w$وشكل نواة الالتفاف $k_h \times k_w$، سيكون شكل الإخراج $(n_h - k_h +$

$(n_w - k_w + 1) \times (1$: يمكننا فقط تحويل shift نواة الالتفاف حتى الآن حتى نفاد وحدات البكسل لتطبيق الالتفاف عليها.

فيما يلي سوف نستكشف عددًا من التقنيات، بمافي ذلك الحشو والتلافيف المتدرجة strided convolutions، التي توفر مزيدًا من التحكمفي حجم الإخراج. كدافع، لاحظ أنه نظرًا لأن النوى عمومًا لها عرض وارتفاع أكبر من 1، بعد تطبيق العديد من التلافيف المتتالية، فإننا نميل إلى الوصول إلى مخرجات أصغر بكثير من مدخلاتنا. إذا بدأنا بصورة بكسل 240 × 240، فإن 10طبقات 5 × 5تلافيفية تقلل الصورة إلى بكسل 200 × 200، وتقطع 30% من الصورة وتزيل أي معلومات مثيرة للاهتمام حول حدود الصورة الأصلية. الحشو Padding هو الأداة الأكثر شيوعًا للتعامل مع هذه المشكلة.في حالات أخرى، قد نرغبفي تقليل الأبعاد بشكل كبير، على سبيل المثال، إذا وجدنا أن دقة الإدخال الأصلية غير عملية. التلافيف المتوترة Strided convolutions هي تقنية شائعة يمكن أن تساعدفي هذه الحالات.

7.3.1. الحشو Padding

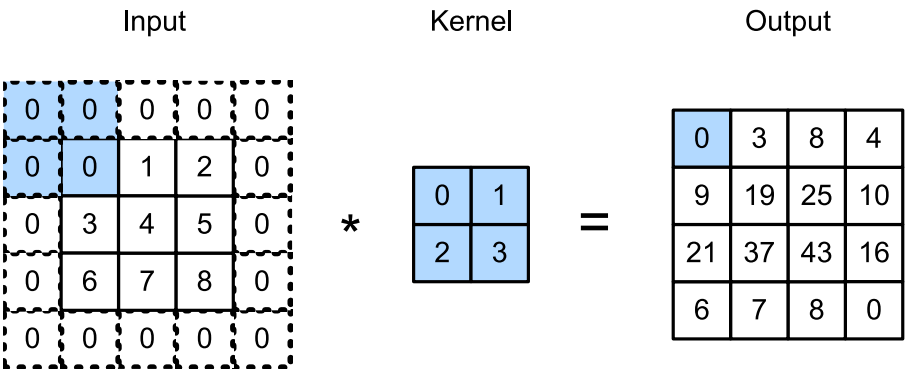
كما هو موضح أعلاه، فإن إحدى المشكلات الصعبة عند تطبيق الطبقات التلافيفية هي أننا نميل إلى فقد وحدات البكسلفي محيط صورتنا. ضعفي اعتبارك الشكل 7.3.1 الذي يصور استخدام البكسل كدالة لحجم نواة الالتفاف والموضع داخل الصورة. نادرًا ما يتم استخدام البكسلفي الزوايا على الإطلاق.



الشكل 7.3.1 استخدام البكسل لتلافيف ذات الحجم 1 × 1، و3 × 3 على التوالي.

نظرًا لأننا عادةً ما نستخدم نوى صغيرة small kernels، لأي التفاف معين، فقد نفقد عددًا قليلاً فقط من وحدات البكسل، لكن هذا يمكن أن يضيف لأننا نطبق العديد من الطبقات التلافيفية المتعاقبة. يتمثل أحد الحلول المباشرة لهذه المشكلةفي إضافة وحدات بكسل إضافية من الحشو حول حدود صورة الإدخال لدينا، وبالتالي زيادة الحجم الفعال للصورة. عادة، نقوم بتعيين قيم وحدات البكسل الإضافية على صفر.في الشكل 7.3.2، نقوم بتعبئة (حشو) أحد المدخلات 3 × 3، وزيادة حجمه إلى 5 × 5. ثم يزيد الناتج المقابل إلى مصفوفة 4 × 4. الأجزاء المظللة

هي أول عنصر إخراج بالإضافة إلى عناصر موتر الإدخال والنواة المستخدمة لحساب الإخراج:
$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$.



الشكل 7.3.2 الارتباط المتبادل ثنائي الأبعاد مع الحشو.

بشكل عام، إذا أضفنا إجمالي صفوف الحشو p_h (نصفها تقريبًافي الأعلى ونصفهفي الأسفل) ومجموعًا من أعمدة الحشو p_w (نصفها تقريبًا على اليسار ونصفها على اليمين)، فسيكون شكل الإخراج

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1).$$

هذا يعني أن ارتفاع الناتج وعرضه سيزدادان بمقدار p_h وp_w على التوالي.

في كثير من الحالات، نريد أن نضبط $p_h = k_h - 1$ و$p_w = k_w - 1$ نعطي المدخلات والمخرجات بنفس الارتفاع والعرض. سيسهل هذا التنبؤ بشكل الإخراج لكل طبقة عند إنشاء الشبكة. بافتراض أن k_h فردي هنا، سنعبئ الصفوف $p_h/2$ على جانبي الارتفاع. إذا كان k_h زوجي، فإن أحد الاحتمالات هو حشو الصفوف $\lceil p_h/2 \rceil$في الجزء العلوي من الإدخال والصفوف $\lfloor p_h/2 \rfloor$ في الأسفل. سنبطئ جانبي العرض بنفس الطريقة.

تستخدم شبكات CNN نواة التفاف ذات قيم عرض وعرض فردية، مثل 1 أو 3 أو 5 أو 7. اختيار أحجام نواة فردية له فائدة أنه يمكننا الحفاظ على الأبعاد أثناء الحشو بنفس عدد الصفوف في الأعلى والأسفل، ونفس عدد الأعمدة على اليسار واليمين.

علاوة على ذلك، فإن هذه الممارسة المتمثلةفي استخدام نوى فردية وحشو للحفاظ على الأبعاد بدقة تقدم فائدة كتابية. بالنسبة لأي موتر ثنائي الأبعاد `X` ، عندما يكون حجم النواة فرديًا ويكون عدد صفوف وأعمدة الحشو متماثلًافي جميع الجوانب، مما ينتج عنه مخرجات بنفس ارتفاع وعرض المدخلات ، فنحن نعلم أن الناتج `Y[i, j]` يتم حسابه من خلال الارتباط المتبادل بين المدخلات ونواة الالتفاف مع النافذة المتمركزة على `X[i, j]`.

في المثال التالي، قمنا بإنشاء طبقة تلافيفية ثنائية الأبعاد بارتفاع وعرض 3 ونطبق 1 بكسل من الحشو على جميع الجوانب. بإدخال ارتفاع وعرض 8، نجد أن ارتفاع وعرض المخرج يساوي 8 أيضًا.

```
import tensorflow as tf


# We define a helper function to calculate convolutions.
It initializes
# the convolutional layer weights and performs
corresponding dimensionality
# elevations and reductions on the input and output.
def comp_conv2d(conv2d, X):
    # (1, 1) indicates that batch size and the number of
channels are both 1
    X = tf.reshape(X, (1, ) + X.shape + (1, ))
    Y = conv2d(X)
    # Strip the first two dimensions: examples and
channels
    return tf.reshape(Y, Y.shape[1:3])
# 1 row and column is padded on either side, so a total
of 2 rows or columns are added
conv2d = tf.keras.layers.Conv2D(1, kernel_size=3,
padding='same')
X = tf.random.uniform(shape=(8, 8))
comp_conv2d(conv2d, X).shape
```

```
TensorShape([8, 8])
```

عندما يختلف ارتفاع وعرض نواة الالتفاف، يمكننا أن نجعل الناتج والمدخل لهما نفس الارتفاع والعرض عن طريق تعيين أرقام حشو مختلفة للارتفاع والعرض.

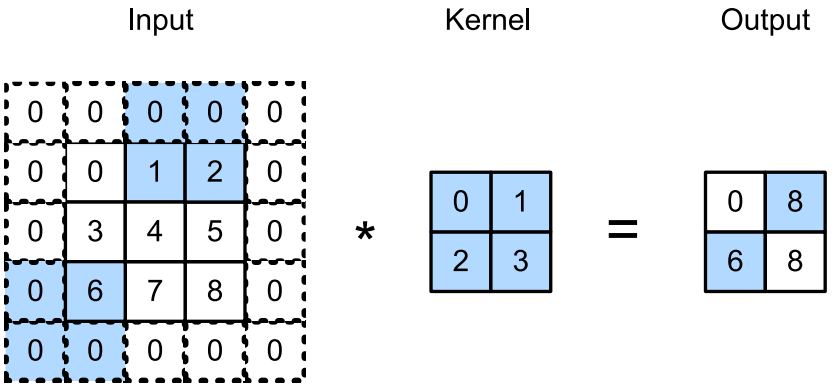
```
# We use a convolution kernel with height 5 and width 3.
The padding on
# either side of the height and width are 2 and 1,
respectively.
conv2d = tf.keras.layers.Conv2D(1, kernel_size=(5, 3),
padding='same')
comp_conv2d(conv2d, X).shape
```

```
TensorShape([8, 8])
```

7.3.2. الخطوة Stride

عند حساب الارتباط التبادلي cross-correlation، نبدأ بنافذة الالتفاففي الزاوية العلوية اليسرى من موتر الإدخال، ثم نمرره على جميع المواقع لأسفل وإلى اليمين.في الأمثلة السابقة، تخلفنا عن تحريك عنصر واحدفي كل مرة. ومع ذلك،في بعض الأحيان، إما من أجل الكفاءة الحسابية أو لأننا نرغبفي الاختزال downsample، فإننا ننقل نافذتنا أكثر من عنصر واحدفي كل مرة، متخطين المواقع الوسيطة. هذا مفيد بشكل خاص إذا كانت نواة الالتفاف كبيرة لأنها تلتقط مساحة كبيرة من الصورة الأساسية.

نشير إلى عدد الصفوف والأعمدة التي يتم اجتيازهافي كل شريحة كخطوة stride. حتى الآن، استخدمنا عدد الخطوات 1، لكل من الطول والعرض.في بعض الأحيان، قد نرغبفي اتخاذ خطوة أكبر. يوضح الشكل 7.3.3 عملية الارتباط المتبادل ثنائية الأبعاد بخطوة 3 عموديًا و2 أفقيًا. الأجزاء المظللة هي عناصر الإخراج بالإضافة إلى عناصر موتر المدخلات والنواة المستخدمة لحساب الإخراج: $0 \times 0 + 6 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$، $0 \times 0 + 0 \times 3 + 2 \times 2 = 6$. يمكننا أن نرى أنه عند إنشاء العنصر الثاني من العمود الأول، تنزلق نافذة الالتفاف إلى أسفل ثلاثة صفوف. تنقل نافذة الالتفاف عمودين إلى اليمين عند إنشاء العنصر الثاني من الصف الأول. عندما تستمر نافذة الالتفاففي تحريك عمودين إلى اليمين على الإدخال، لا يوجد إخراج لأن عنصر الإدخال لا يمكن أن يملأ النافذة (إلا إذا أضفنا عمودًا آخر من الحشو).



الشكل 7.3.3 الارتباط المتبادل مع خطوات 3 و2 للارتفاع والعرض، على التوالي.

بشكل عام، عندما تكون خطوة الارتفاع s_h وخطوة العرض s_w، يكون شكل الإخراج

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor.$$

إذا قمنا بتعيين $p_h = k_h - 1$ ثم $p_w = k_w - 1$، فيمكن تبسيط شكل الإخراج إلى $\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$. للمضي قدمًا، إذا كان ارتفاع الإدخال وعرضه قابلين للقسمة على خطوات الارتفاع والعرض، فسيكون شكل الإخراج $(n_h/s_h) \times (n_w/s_w)$.

أدناه، قمنا بتعيين الخطوات على كل من الارتفاع والعرض على 2، وبالتالي خفض ارتفاع المدخلات وعرضها إلى النصف.

```
conv2d = tf.keras.layers.Conv2D(1, kernel_size=3,
padding='same', strides=2)
comp_conv2d(conv2d, X).shape
```

```
TensorShape([4, 4])
```

لنلق نظرة على مثال أكثر تعقيدًا بعض الشيء.

```
conv2d = tf.keras.layers.Conv2D(1, kernel_size=(3,5),
padding='valid',
                                strides=(3, 4))
comp_conv2d(conv2d, X).shape
```

```
TensorShape([2, 1])
```

7.3.3. الملخص والمناقشة

يمكن أن يؤدي الحشو Padding إلى زيادة ارتفاع وعرض الإخراج. غالبًا ما يستخدم هذا لإعطاء الإخراج نفس الارتفاع والعرض مثل المدخلات لتجنب الانكماش shrinkage غير المرغوب فيه للإخراج. علاوة على ذلك، فإنه يضمن استخدام جميع وحدات البكسل بشكل متكرر. عادةً ما نختار حشوة متماثلة symmetric padding على جانبي ارتفاع المدخلات وعرضها.في هذه الحالة نشير إلى الحشو (p_h, p_w). الأكثر شيوعًا التي نضعها $p_h = p_w$ ، وفي هذه الحالة نذكر ببساطة أننا نختار الحشو p.

اتفاقية مماثلة تنطبق على الخطوات. عندما تتطابق الخطوة الأفقية s_h والخطوة العمودية s_w، نتحدث ببساطة عن الخطوة s. يمكن أن تقلل الخطوة من دقة الإخراج، على سبيل المثال تقليل ارتفاع وعرض الإخراج إلى $1/n$ من ارتفاع وعرض المدخلات لـ $n > 1$. بشكل افتراضي، تكون المساحة المتروكة 0 والخطوة 1.

حتى الآن كل الحشو الذي ناقشناه ببساطة الصور الممتدة مع الأصفار. هذا له فائدة حسابية كبيرة لأنه من السهل تحقيقه. علاوة على ذلك، يمكن تصميم المشغلين للاستفادة من هذه الحشوة ضمنيًا دون الحاجة إلى تخصيص ذاكرة إضافية.في الوقت نفسه، يسمح لشبكات CNN بتشفير معلومات الموقع الضمنية داخل الصورة، وذلك ببساطة عن طريق التعرف على مكان وجود

"المسافة البيضاء whitespace". هناك العديد من البدائل alternatives للحشو الصفري. (Alsallakh et al.، 2020) يقدم نظرة عامة شاملة للبدائل (وإن لم يكن هناك حالة واضحة لاستخدام حشوات غير صفرية ما لم تحدث القطع الأثرية unless artifacts occur).

7.3.4. التمارين

1. بالنظر إلى مثال الكود الأخير في هذا القسم مع حجم النواة والحشو والخطوة ، احسب شكل الإخراج للتحقق مما إذا كان متوافقًا مع النتيجة التجريبية.
2. بالنسبة للإشارات الصوتية audio signals، ما الذي يتوافق مع الخطوة 2؟
3. قم بتنفيذ حشوة متطابقة mirror padding، أي الحشو حيث يتم عكس قيم الحدود ببساطة لتوسيع الموترات.
4. ما هي الفوائد الحسابية computational benefits لخطوة أكبر من 1؟
5. ما الفوائد الإحصائية statistical benefits لخطوة أكبر من 1؟
6. كيف ستنفذ خطوة من $\frac{1}{2}$ ؟ مع ماذا يتوافق؟ متى يكون هذا مفيدا؟

7.4. مدخلات متعددة وقنوات إخراج متعددة Multiple Input and Multiple Output Channels

بينما وصفنا القنوات المتعددة multiple channels التي تتكون منها كل صورة (على سبيل المثال، تحتوي الصور الملونة على قنوات RGB القياسية للإشارة إلى مقدار الأحمر والأخضر والأزرق) والطبقات التلافيفية لقنوات متعددةفي القسم 7.1.4، حتى الآن، قمنا بتبسيط كل أمثلتنا العددية من خلال العمل بمدخل واحد فقط وقناة إخراج واحدة. سمح لنا هذا بالتفكيرفي مدخلاتنا، ونواة الالتفاف، والمخرجات على أنها موترات ثنائية الأبعاد.

عندما نضيف قنوات إلى المزيج، تصبح كل من مدخلاتنا والتمثيلات المخفية موترات ثلاثية الأبعاد. على سبيل المثال، كل صورة إدخال RGB لها شكل $3 \times h \times w$. نشير إلى هذا المحور، بحجم 3، كبعد القناة. إن فكرة القنوات قديمة قدم شبكات CNN نفسها. على سبيل المثال، يستخدمها LeNet5.(LeCun et al., 1995).في هذا القسم، سوف نلقي نظرة أعمق على نواة الالتفاف ذات المدخلات المتعددة وقنوات الإخراج المتعددة.

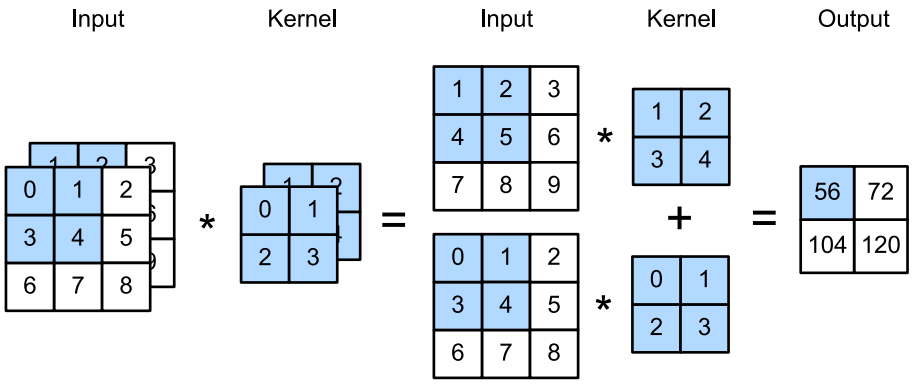
7.4.1. قنوات إدخال متعددة Multiple Input Channels

عندما تحتوي بيانات الإدخال على قنوات متعددة multiple channels، نحتاج إلى إنشاء نواة التفاف بنفس عدد قنوات الإدخال مثل بيانات الإدخال، بحيث يمكنها إجراء ارتباط متبادل مع بيانات الإدخال. بافتراض أن عدد القنوات لبيانات الإدخال هو c_i، يجب أيضًا أن يكون عدد

قنوات الإدخال لنواة الالتفاف c_i. إذا كان شكل نافذة نواة الالتفاف لدينا هو $k_h \times k_w$، عندما $c_i = 1$ ، يمكننا التفكير في نواة الالتواء على أنها مجرد موتر ثنائي الأبعاد للشكل $k_h \times k_w$.

ومع ذلك، عندما $c_i > 1$ نحتاج إلى نواة تحتوي على موتر الشكل $k_h \times k_w$ لكل قناة إدخال. يؤدي ربط هذه الموترات معًا إلى إنتاج نواة التفاف للشكل $c_i \times k_h \times k_w$. نظرًا لأن لكل من نواة الإدخال والالتفاف لديها c_i قنوات، يمكننا إجراء عملية الارتباط المتبادل على موتر ثنائي الأبعاد للإدخال والموتر ثنائي الأبعاد لنواة الالتفاف لكل قناة، مع إضافة نتائج c_i معًا (التجميع عبر القنوات) لإنتاج موتر ثنائي الأبعاد. هذا هو نتيجة الارتباط المتبادل ثنائي الأبعاد بين إدخال متعدد القنوات ونواة التفاف متعددة المدخلات.

يقدم الشكل 7.4.1 مثالاً على الترابط الثنائي الأبعاد مع قناتي الادخال. الأجزاء المظللة هي أول عنصر إخراج بالإضافة إلى عناصر موتر الإدخال والنواة المستخدمة لحساب الإخراج: $(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$.



الشكل 7.4.1 حساب الارتباط المتبادل مع قناتي إدخال.

للتأكد من أننا نفهم حقًا ما يجري هنا، يمكننا تنفيذ عمليات الارتباط المتبادل مع قنوات الإدخال المتعددة بأنفسنا. لاحظ أن كل ما نقوم به هو إجراء عملية الارتباط المتبادل لكل قناة ثم جمع النتائج.

```
import tensorflow as tf
from d2l import tensorflow as d2l


def corr2d_multi_in(X, K):
    # Iterate through the 0th dimension (channel) of K
first, then add them up
```

```
    return tf.reduce_sum([d2l.corr2d(x, k) for x, k in
zip(X, K)], axis=0)
```

يمكننا إنشاء موتر الإدخال X وموتر النواة K المطابق للقيم الواردةفي الشكل 7.4.1 للتحقق من صحة خرج عملية الارتباط المتبادل.

```
X = tf.constant([[[0.0, 1.0, 2.0], [3.0, 4.0, 5.0],
[6.0, 7.0, 8.0]],
                 [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0,
8.0, 9.0]]])
K = tf.constant([[[0.0, 1.0], [2.0, 3.0]], [[1.0, 2.0],
[3.0, 4.0]]])

corr2d_multi_in(X, K)
```

```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[ 56.,  72.],
       [104., 120.]], dtype=float32)>
```

7.4.2. قنوات إخراج متعددة Multiple Output Channels

بغض النظر عن عدد قنوات الإدخال، فقد انتهى بنا المطاف دائمًا بقناة إخراج واحدة. ومع ذلك، كما ناقشنافي القسم 7.1.4، فقد تبين أنه من الضروري وجود قنوات متعددةفي كل طبقة.في أكثر هياكل الشبكات العصبية شيوعًا، نقوم بالفعل بزيادة بُعد القناة أثناء تعمقنافي الشبكة العصبية، وعادةً ما يتم الاختزال لموازنة الدقة المكانية للحصول على عمق أكبر للقناة. حدسيًا، يمكنك التفكيرفي كل قناة على أنها تستجيب لمجموعة مختلفة من الميزات. الواقع أكثر تعقيداً من هذا بقليل. قد يشير التفسير الساذج إلى أن التمثيلات يتم تعلمها بشكل مستقل لكل بكسل أو لكل قناة. بدلاً من ذلك، يتم تحسين القنوات لتكون مفيدة بشكل مشترك. هذا يعني أنه بدلاً من تعيين قناة واحدة لكاشف الحافة، فقد يعني ذلك ببساطة أن بعض الاتجاهفي مساحة القناة يتوافق مع اكتشاف الحواف.

قم بالإشارة إلى c_i و c_o عدد قنوات الإدخال والإخراج، على التوالي، وليكن k_h و k_w ارتفاع النواة وعرضها. للحصول على مخرجات مع قنوات متعددة، يمكننا إنشاء موتر نواة للشكل لكل قناة إخراج. نجمعها على بُعد قناة الإخراج، بحيث يكون شكل نواة الالتفاف $c_o \times c_i \times k_h \times k_w$.في عمليات الارتباط المتبادل، يتم حساب النتيجة على كل قناة إخراج من نواة الالتفاف المقابلة لقناة الإخراج هذه وتأخذ المدخلات من جميع القنواتفي موتر الإدخال.

نقوم بتنفيذ دالة الارتباط المتبادل cross-correlation function لحساب ناتج قنوات متعددة كما هو موضح أدناه.

```
def corr2d_multi_in_out(X, K):
```

```
    # Iterate through the 0th dimension of `K`, and each
time, perform
    # cross-correlation operations with input `X`. All
of the results are
    # stacked together
    return tf.stack([corr2d_multi_in(X, k) for k in K],
0)
```

نقوم ببناء نواة التفاف عادية مع 3 قنوات إخراج من خلال تسلسل موتر النواة لـ K مع K+1 و K+2.

```
K = tf.stack((K, K + 1, K + 2), 0)
K.shape
```

```
TensorShape([3, 2, 2, 2])
```

أدناه، نقوم بإجراء عمليات الارتباط المتبادل على موتر الإدخال X مع موتر النواة K. الآن يحتوي الإخراج على 3 قنوات. تتوافق نتيجة القناة الأولى مع نتيجة موتر الإدخال السابق X والقناة متعددة المدخلات، نواة قناة الإخراج الأحادي.

```
corr2d_multi_in_out(X, K)
```

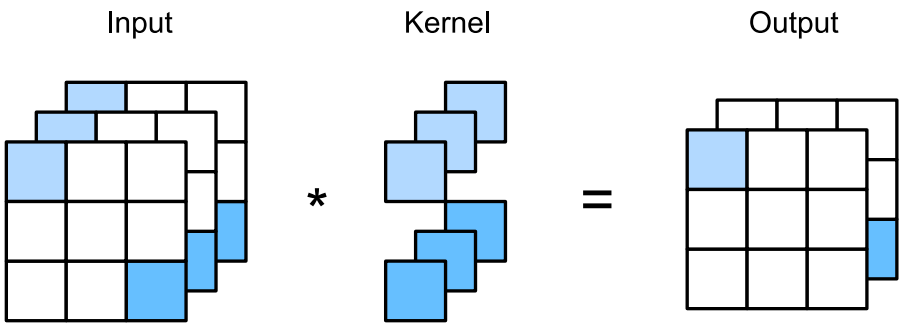
```
<tf.Tensor: shape=(3, 2, 2), dtype=float32, numpy=
array([[[ 56.,  72.],
        [104., 120.]],

       [[ 76., 100.],
        [148., 172.]],

       [[ 96., 128.],
        [192., 224.]]], dtype=float32)>
```

7.4.3. الطبقة التلافيفية 1 × 1 (Convolutional Layer 1 × 1)

في البداية، لا يبدو أن الالتفاف 1 × 1 ، أي $k_h = k_w = 1$، له معنى كبير. بعد كل شيء، الالتفاف يربط بين وحدات البكسل المجاورة. من الواضح أن الالتفاف 1 × 1 لا يفعل ذلك. ومع ذلك، فهي عمليات شائعة يتم تضمينها أحيانًا في تصميمات الشبكات العميقة المعقدة (Lin et al.، 2013، Szegedy et al.، 2017) دعنا نرى بشيء من التفصيل ما تفعله بالفعل.

نظرًا لاستخدام الحد الأدنى من النافذة، يفقد الالتفاف 1 × 1 قدرة الطبقات التلافيفية الأكبر على التعرف على الأنماط المكونة من تفاعلات بين العناصر المجاورةفي أبعاد الارتفاع والعرض. الحساب الوحيد للالتفاف 1 × 1 يحدث على بُعد القناة channel dimension.

يوضح الشكل 7.4.2 حساب الارتباط المتبادل باستخدام نواة الالتفاف 1×1 مع 3 قنوات إدخال وقناتين إخراج. لاحظ أن المدخلات والمخرجات لها نفس الارتفاع والعرض. يتم اشتقاق كل عنصرفي الإخراج من مجموعة خطية من العناصرفي نفس الموضعفي صورة الإدخال. يمكنك التفكيرفي الطبقة التلافيفية 1×1 على أنها تشكل طبقة متصلة بالكامل مطبقةفي كل موقع بكسل فردي لتحويل قيم الإدخال المقابلة c_i إلى قيم إخراج c_o. نظرًا لأن هذه لا تزال طبقة تلافيفية، يتم ربط الأوزان عبر موقع البكسل. وبالتالي فإن الطبقة التلافيفية 1×1 تتطلب أوزانًا (بالإضافة إلى التحيز). لاحظ أيضًا أن الطبقات التلافيفية تتبعها عادةً اللاخطية. هذا يضمن أن التلافيف 1×1 لا يمكن ببساطة طيهافي تلافيف أخرى.



الشكل 7.4.2 يستخدم حساب الارتباط التبادلي نواة الالتفاف 1×1 مع 3 قنوات إدخال وقناتي إخراج. المدخلات والمخرجات لها نفس الارتفاع والعرض.

دعنا نتحقق مما إذا كان هذا يعملفي الممارسة العملية: نقوم بتنفيذ التفاف 1×1 باستخدام طبقة متصلة بالكامل. الشيء الوحيد هو أننا نحتاج إلى إجراء بعض التعديلات على شكل البيانات قبل وبعد عملية ضرب المصفوفة.

```
def corr2d_multi_in_out_1x1(X, K):
    c_i, h, w = X.shape
    c_o = K.shape[0]
    X = tf.reshape(X, (c_i, h * w))
    K = tf.reshape(K, (c_o, c_i))
    # Matrix multiplication in the fully connected layer
    Y = tf.matmul(K, X)
    return tf.reshape(Y, (c_o, h, w))
```

عند إجراء عمليات التلافيف 1×1، تكون الدالة المذكورة أعلاه مكافئة لدالة الارتباط المتبادل التي تم تنفيذها مسبقًا `corr2d_multi_in_out`. دعنا نتحقق من ذلك ببعض نماذج البيانات.

```
X = tf.random.normal((3, 3, 3), 0, 1)
```

```
K = tf.random.normal((2, 3, 1, 1), 0, 1)

Y1 = corr2d_multi_in_out_1x1(X, K)
Y2 = corr2d_multi_in_out(X, K)
assert float(tf.reduce_sum(tf.abs(Y1 - Y2))) < 1e-6
```

7.4.4. المناقشة

تسمح لنا القنوات بدمج أفضل مافي كلا العالمين: MLPs التي تسمح بالتلافيفات غير الخطية والتلافيفات التي تسمح بالتحليل المحلي localized analysis للميزات. على وجه الخصوص، تسمح القنوات لشبكة CNN بالتفكير باستخدام ميزات متعددة، مثل أجهزة الكشف عن الحواف والشكلفي نفس الوقت. كما أنها توفر مفاضلة عملية بين التخفيض الكبير للمعامل الناشئ عن ثبات الترجمة والمحلية، والحاجة إلى نماذج معبرة ومتنوعةفي الرؤية الحاسوبية.

لاحظ، مع ذلك، أن هذه المرونة لها ثمن. بالنظر إلى صورة مع حجم$(h \times w)$، فإن تكلفة حساب الالتفاف $k \times k$ هي $\mathcal{O}(h \cdot w \cdot k^2)$. بالنسبة c_i و c_o قنوات الإدخال والإخراج على التوالي، يزداد هذا إلى $\mathcal{O}(h \cdot w \cdot k^2 \cdot c_i \cdot c_o)$. بالنسبة لصورة 256×256 بكسل التي تحتوي على نواة 5×5 و128 قنوات إدخال وإخراج على التوالي، فإن هذا يصل إلى أكثر من 53مليار عملية (نحسب المضاعفات والإضافات بشكل منفصل).في وقت لاحق سنواجه استراتيجيات فعالة لخفض التكلفة، على سبيل المثال، من خلال اشتراط أن تكون العمليات على مستوى القناة قطرية، مما يؤدي إلى هياكل مثل ResNeXt،(Xie et al.، 2017).

7.4.5. التمارين

1. افترض أن لدينا نواة التفاف بالحجم k_1،k_2 وعلى التوالي (مع عدم وجود نواة غير خطية بينهما).
   1. إثبت أنه يمكن التعبير عن نتيجة العملية من خلال التفاف واحد single convolution.
   2. ما هي أبعاد الالتفاف الواحد المكافئ؟
   3. هل العكس صحيح ، أي هل يمكنك دائمًا تحليل الالتفاف إلى قسمين أصغر؟
2. افترض مدخلاً للشكل $c_i \times h \times w$ ونواة التفاف من الشكل $c_o \times c_i \times k_h \times k_w$ والحشو (p_h, p_w) والخطوة (s_h, s_w).
   1. ما هي التكلفة الحسابية computational cost (الضرب والجمع) للانتشار الأمامي؟
   2. ما هي بصمة الذاكرة memory footprint؟
   3. ما هي بصمة الذاكرة للحساب العكسي backward computation؟
   4. ما هي التكلفة الحسابية للانتشار الخلفي backpropagation؟

3. بأي عامل يزداد عدد الحسابات إذا ضاعفنا عدد قنوات الإدخال c_i وعدد قنوات الإخراج c_o؟ ماذا يحدث إذا ضاعفنا الحشو؟
4. هل المتغيرين Y1 و Y2 في المثال الأخير من هذا القسم متطابقان تمامًا؟ لماذا؟
5. عبر عن التلافيف كضرب مصفوفة ، حتى عندما لا تكون نافذة الالتفاف 1×1؟
6. مهمتك هي تنفيذ التلافيف السريعة باستخدام النواة $k \times k$. أحد الخوارزمية المرشحة هو المسح أفقيًا عبر المصدر، وقراءة kـ خطوة عريضة وحساب 1ـ خطوة عريضة من الإخراج على نطاق واسع بقيمة واحدة في كل مرة. البديل هو قراءة $k + \Delta$ خطوة عريضة وحساب Δ خطوة عريضة من الإخراج. لماذا هذا الأخير هو الأفضل؟ هل هناك حد لمدى الحجم الذي يجب أن تختار Δ ؟
7. افترض أن لدينا مصفوفة $c \times c$.
   1. ما مقدار سرعة الضرب بمصفوفة كتلة قطرية إذا تم تقسيم المصفوفة إلى كتل b؟
   2. ما هو الجانب السلبي من وجود الكتل b؟ كيف يمكنك إصلاحه جزئيًا على الأقل؟

7.5 التجميع Pooling

في كثير من الحالات، تطرح مهمتنا النهائية بعض الأسئلة العالمية حول الصورة، على سبيل المثال، هل تحتوي على قطة؟ وبالتالي، يجب أن تكون وحدات الطبقة النهائية حساسة للمدخلات بالكامل. من خلال تجميع المعلومات تدريجيًا، وإنتاج خرائط خشنة وأكثر خشونة، نحقق هذا الهدف المتمثل في تعلم تمثيل عالمي في النهاية، مع الاحتفاظ بجميع مزايا الطبقات التلافيفية في الطبقات الوسيطة للمعالجة. كلما تعمقنا في الشبكة، زاد المجال التأثيري receptive field (بالنسبة إلى المدخلات) الذي تكون كل عقدة مخفية حساسة له. يؤدي تقليل الدقة المكانية إلى تسريع هذه العملية، حيث تغطي نواة الالتفاف مساحة فعالة أكبر.

علاوة على ذلك، عند اكتشاف ميزات المستوى الأدنى، مثل الحواف (كما تمت مناقشته في القسم 7.2)، غالبًا ما نريد أن تكون تمثيلاتنا ثابتة إلى حد ما للترجمة. على سبيل المثال، إذا أخذنا الصورة `X` بتحديد حاد بين الأسود والأبيض وقمنا بتحويل الصورة بأكملها بمقدار بكسل واحد إلى اليمين، أي `Z[i, j] = X[i, j + 1]` ، فإن الناتج للصورة الجديدة `Z` قد يكون مختلفًا تمامًا. سيتم إزاحة الحافة بمقدار بكسل واحد. في الواقع، نادرًا ما تحدث الأشياء في نفس المكان تمامًا. في الواقع، حتى مع وجود حامل ثلاثي القوائم وجسم ثابت، فإن اهتزاز الكاميرا بسبب حركة مصراع الكاميرا قد يغير كل شيء بمقدار بكسل أو نحو ذلك (يتم تحميل الكاميرات المتطورة بميزات خاصة لمعالجة هذه المشكلة).

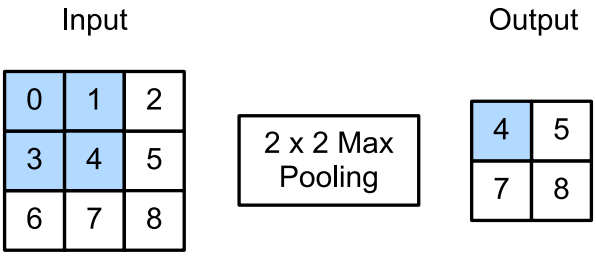
يقدم هذا القسم طبقات التجميع pooling layers، التي تخدم الأغراض المزدوجة للتخفيف من حساسية الطبقات التلافيفية تجاه الموقع وتمثيلات الاختزال المكاني spatially downsampling representations.

7.5.1. تجميع الحد الأقصى وتجميع المتوسط Maximum Pooling and Average Pooling

مثل الطبقات التلافيفية، تتكون عوامل التجميع pooling operators من نافذة ذات شكل ثابت تنزلق فوق جميع المناطقفي الإدخال وفقًا لخطواتها stride، وتحسب ناتجًا واحدًا لكل موقع يتم اجتيازه بواسطة نافذة الشكل الثابت fixed-shape window (تُعرف أحيانًا باسم نافذة التجميع pooling window). ومع ذلك، على عكس حساب الارتباط المتبادل للمدخلات والنواةفي الطبقة التلافيفية، لا تحتوي طبقة التجميع على معلمات (لا توجد نواة). بدلاً من ذلك، تكون عوامل التجميع قطعية deterministic، وتحسب عادةً إما الحد الأقصى maximum أو متوسط average قيمة العناصرفي نافذة التجميع. تسمى هذه العمليات التجميع الأقصى maximum pooling (max-pooling للاختصار) وتجميع المتوسط average pooling، على التوالي.

تجميع المتوسط Average pooling قديم قدم شبكات CNN. الفكرة تشبه اختزال downsampling الصورة. بدلاً من مجرد أخذ قيمة كل ثانية (أو ثالثة) بكسل للصورة ذات الدقة الأقل، يمكننا أن نحصل على متوسط أعلى من البكسلات المجاورة للحصول على صورة ذات نسبة إشارة إلى ضوضاء أفضل نظرًا لأننا نجمع المعلومات من عدة وحدات بكسل متجاورة. تم تقديم تجميع الحد الاقصى Max-poolingفي (Riesenhuber and Poggio، 1999) في سياق علم الأعصاب الإدراكي cognitive neuroscience لوصف كيفية تجميع المعلومات بشكل هرمي لغرض التعرف على الأشياء object recognition، وإصدار سابقفي التعرف على الكلام speech recognition (Yamaguchi et al.، 1990).في جميع الحالات تقريبًا، يُفضل استخدام max-pooling، كما يُشار إليه أيضًا.

في كلتا الحالتين، كما هو الحال مع عامل الارتباط المتبادل، يمكننا التفكيرفي نافذة التجميع pooling window على أنها تبدأ من أعلى يسار موتر الإدخال وتنزلق عبر موتر الإدخال من اليسار إلى اليمين ومن أعلى إلى أسفل.في كل موقع تصل إليه نافذة التجميع، فإنها تحسب الحد الأقصى أو المتوسط لقيمة المستشعر الفرعي للإدخالفي النافذة، اعتمادًا على ما إذا كان التجميع الأقصى max أو المتوسط average مستخدمًا.



7.5.1 تجميع الحد الأقصى Max-pooling مع شكل نافذة تجمع 2×2. الأجزاء المظللة هي أول عنصر إخراج بالإضافة إلى عناصر موتر الإدخال المستخدمة لحساب الإخراج:

$$max(0,1,3,4) = 4.$$

يبلغ ارتفاع موتر الاخراجفي الشكل 7.5.1 2 وعرضه 2. وتُشتق العناصر الأربعة من القيمة القصوىفي كل نافذة تجميع:

$$max(0,1,3,4) = 4,$$
$$max(1,2,4,5) = 5,$$
$$max(3,4,6,7) = 7,$$
$$max(4,5,7,8) = 8.$$

بشكل عام، يمكننا تحديد طبقة تجميع $p \times q$ عن طريق التجميع فوق منطقة بالحجم المذكور. بالعودة إلى مشكلة اكتشاف الحواف، نستخدم ناتج الطبقة التلافيفية كمدخل لتجميع الحد الأقصى 2×2. تشير بواسطة `X` إلى مدخلات الطبقة التلافيفية و`Y` ناتج طبقة التجميع. بغض النظر عما إذا كانت قيم `X[i, j], X[i, j + 1], X[i+1, j]` و `X[i+1, j + 1]` مختلفة ، طبقة التجميع دائمًا لها المخرجات `Y[i, j] = 1`. أي باستخدام طبقة الحد الأقصى من التجميع 2×2 ، لا يزال بإمكاننا اكتشاف ما إذا كان النمط الذي تم التعرف عليه بواسطة الطبقة الالتفافية لا يتحرك أكثر من عنصر واحدفي الارتفاع أو العرض.

في الكود أدناه، نقوم بتنفيذ الانتشار الأمامي لطبقة التجميعفي دالة `pool2d`. هذه الدالة مشابهة للدالة `corr2d` في القسم 7.2. ومع ذلك، ليست هناك حاجة إلى نواة، حيث يتم حساب المخرجات على أنها الحد الأقصى أو المتوسط لكل منطقةفي الإدخال.

```
import tensorflow as tf


def pool2d(X, pool_size, mode='max'):
    p_h, p_w = pool_size
```

```
    Y = tf.Variable(tf.zeros((X.shape[0] - p_h + 1,
X.shape[1] - p_w +1)))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            if mode == 'max':
                Y[i, j].assign(tf.reduce_max(X[i: i +
p_h, j: j + p_w]))
            elif mode =='avg':
                Y[i, j].assign(tf.reduce_mean(X[i: i +
p_h, j: j + p_w]))
    return Y
```

يمكننا إنشاء موتر الإدخال X في الشكل 7.5.1 للتحقق من صحة خرج طبقة التجميع الحد الاقصى ثنائية الأبعاد two-dimensional max-pooling layer.

```
X = tf.constant([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0,
7.0, 8.0]])
pool2d(X, (2, 2))
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32,
numpy=
array([[4., 5.],
       [7., 8.]], dtype=float32)>
```

أيضًا، نجرب طبقة تجميع المتوسط average pooling layer.

```
pool2d(X, (2, 2), 'avg')
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32,
numpy=
array([[2., 3.],
       [5., 6.]], dtype=float32)>
```

7.5.2. الحشو والخطوة Padding and Stride

كما هو الحال مع الطبقات التلافيفية، تغير طبقات التجميع شكل الإخراج. وكمافي السابق، يمكننا ضبط العملية لتحقيق الشكل المطلوب للإخراج عن طريق حشو Padding المدخلات وضبط الخطوة Stride. يمكننا إثبات استخدام الحشو والخطواتفي طبقات التجميع عبر طبقة تجميع الحد الاقصى ثنائية الأبعاد المضمنة من إطار عمل التعلم العميق. نقوم أولاً ببناء موتر الإدخال X الذي يحتوي شكله على أربعة أبعاد، حيث يكون عدد الأمثلة (حجم الدُفعة batch size) وعدد القنوات 1.

لاحظ أنه بخلاف الأطر الأخرى، يفضل TensorFlow ويتم تحسينه لمدخل آخر القنوات channels-last input.

```
X = tf.reshape(tf.range(16, dtype=tf.float32), (1, 4, 4,
1))
X
```

```
<tf.Tensor: shape=(1, 4, 4, 1), dtype=float32, numpy=
array([[[[ 0.],
         [ 1.],
         [ 2.],
         [ 3.]],

        [[ 4.],
         [ 5.],
         [ 6.],
         [ 7.]],

        [[ 8.],
         [ 9.],
         [10.],
         [11.]],

        [[12.],
         [13.],
         [14.],
         [15.]]]], dtype=float32)>
```

نظرًا لأن التجميع يجمع المعلومات من منطقة ما، فإن أطر التعلم العميق تتطابق مع أحجام نوافذ التجميع والخطوات. على سبيل المثال، إذا استخدمنا نافذة تجميع للشكل (3، 3) نحصل على شكل خطوة (3، 3) افتراضيًا.

```
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3])
# Pooling has no model parameters, hence it needs no
initialization
pool2d(X)
```

```
<tf.Tensor: shape=(1, 1, 1, 1), dtype=float32,
numpy=array([[[[10.]]]], dtype=float32)>
```

كما هو متوقع، يمكن تحديد الخطوة والحشو يدويًا لتجاوز الإعدادات الافتراضية لإطار العمل إذا لزم الأمر.

```
paddings = tf.constant([[0, 0], [1,0], [1,0], [0,0]])
X_padded = tf.pad(X, paddings, "CONSTANT")
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3],
padding='valid',
```

```
                                    strides=2)
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 1), dtype=float32, numpy=
array([[[[ 5.],
         [ 7.]],

        [[13.],
         [15.]]]], dtype=float32)>
```

بالطبع، يمكننا تحديد نافذة تجميع عشوائية مستطيلة ذات ارتفاع وعرض تعسفيين على التوالي، كما يوضح المثال أدناه.

```
paddings = tf.constant([[0, 0], [0, 0], [1, 1], [0, 0]])
X_padded = tf.pad(X, paddings, "CONSTANT")

pool2d = tf.keras.layers.MaxPool2D(pool_size=[2, 3],
padding='valid',
                                   strides=(2, 3))
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 1), dtype=float32, numpy=
array([[[[ 5.],
         [ 7.]],

        [[13.],
         [15.]]]], dtype=float32)>
```

7.5.3. قنوات متعددة Multiple Channels

عند معالجة بيانات الإدخال متعددة القنوات multi-channel input data، تقوم طبقة التجميع بتجميع كل قناة إدخال على حدة، بدلاً من جمع المدخلات عبر القنوات كما هو الحال في الطبقة التلافيفية. هذا يعني أن عدد قنوات الإخراج لطبقة التجميع هو نفسه عدد قنوات الإدخال. أدناه، سنقوم بتجميع الموترات `X` و `X + 1` على بُعد القناة لإنشاء إدخال بقناتين.

لاحظ أن هذا سيتطلب تسلسلاً على طول البعد الأخير لـ TensorFlow بسبب بناء جملة القنوات الأخير.

```
X = tf.concat([X, X + 1], 3)  # Concatenate along
`dim=3` due to channels-last syntax
```

كما نرى، لا يزال عدد قنوات الإخراج 2 بعد التجميع.

```
paddings = tf.constant([[0, 0], [1,0], [1,0], [0,0]])
X_padded = tf.pad(X, paddings, "CONSTANT")
```

```
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3],
padding='valid',
                                   strides=2)
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 2), dtype=float32, numpy=
array([[[[ 5.,  6.],
         [ 7.,  8.]],

        [[13., 14.],
         [15., 16.]]]], dtype=float32)>
```

لاحظ أن ناتج تجميع TensorFlow يبدو للوهلة الأولى مختلفًا، ولكن يتم عرض نفس النتائج عدديًا على أنها MXNet و PyTorch. يكمن الاختلاففي الأبعاد، وتؤدي قراءة الإخراج عموديًا إلى نفس الإخراج مثل التطبيقات الأخرى.

7.5.4. الملخص

التجميع Pooling هو عملية بسيطة للغاية. إنه يفعل بالضبط ما يشير إليه اسمه، النتائج الإجمالية عبر نافذة من القيم. كل دلالات الالتفاف convolution semantics، مثل الخطوات strides والحشو padding، تنطبق بنفس الطريقة كما فعلت سابقًا. لاحظ أن التجميع غير مكترث بالقنوات، أي أنه يترك عدد القنوات دون تغيير وينطبق على كل قناة على حدة. أخيرًا، من بين خياري التجميع الشائعين، يُفضل تجميع الحد الأقصى max-pooling على تجميع المتوسط average pooling ، لأنه يمنح درجة معينة من الثبات للإخراج. الاختيار الشائع هو اختيار حجم نافذة التجميع 2×2 إلى ربع الدقة المكانية للإخراج.

لاحظ أن هناك العديد من الطرق لتقليل الدقة بعد التجميع. على سبيل المثال،في التجميع العشوائي stochastic pooling (Zeiler and Fergus، 2013) والتجميع الأقصى الكسري fractional max-pooling (Graham، 2014) يتم الجمع بين التجميع aggregation والعشوائية randomization. يمكن أن يؤدي ذلك إلى تحسين الدقة قليلاًفي بعض الحالات. أخيرًا، كما سنرى لاحقًا مع آلية الانتباه attention mechanism، هناك طرق أكثر دقة لتجميع المخرجات aggregating over outputs، على سبيل المثال، باستخدام المحاذاة بين متجهي الاستعلام query والتمثيل representation vectors.

7.5.5. التمارين

1. نفذ تجميع المتوسط average pooling من خلال الالتفاف.
2. إثبت أن تجميع الحد الأقصى max-pooling لا يمكن تنفيذه من خلال الالتفاف وحده.

3. يمكن تحقيق تجميع الحد الأقصى باستخدام عمليات ReLU ، أي.
   1. عبر عن $max(a, b)$ باستخدام عمليات ReLU فقط.
   2. استخدم هذا لتنفيذ تجميع الحد الأقصى عن طريق التلافيف وطبقات ReLU.
   3. كم عدد القنوات والطبقات التي تحتاجها للالتفاف 2×2 ؟ كم العدد للالتفاف 3×3.
4. ما هي التكلفة الحسابية لطبقة التجميع؟ افترض أن المدخلات إلى طبقة التجميع ذات حجم $c \times h \times w$ ، وأن نافذة التجميع لها شكل $p_h \times p_w$ مع حشوة (p_h, p_w) وخطوة (s_h, s_w).
5. لماذا تتوقع أن يعمل تجميع الحد الأقصى وتجميع المتوسط بشكل مختلف؟
6. هل نحتاج إلى طبقة تجميع صغيرة منفصلة؟ هل يمكنك استبدالها بعملية أخرى؟
7. يمكننا استخدام عملية softmax للتجميع. لماذا قد لا تحظى بشعبية كبيرة؟

7.6. الشبكات العصبية التلافيفية (LeNet)

لدينا الآن جميع المكونات المطلوبة لتجميع شبكة CNN كاملة الوظائف.في مواجهتنا السابقة لبيانات الصورة، طبقنا نموذجًا خطيًا مع انحدار softmax (القسم 4.4) وMLP (القسم 5.2) على صور الملابسفي مجموعة بيانات Fashion-MNIST. لجعل هذه البيانات قابلة للتمكين، قمنا أولاً بتسطيح flattened كل صورة من مصفوفة 28×28 إلى متجه 784 أبعاد ثابت الطول، وبعد ذلك قمنا بمعالجتهافي طبقات متصلة بالكامل. الآن بعد أن أصبح لدينا التعامل مع الطبقات التلافيفية، يمكننا الاحتفاظ بالبنية المكانيةفي صورنا. كميزة إضافية لاستبدال الطبقات المتصلة بالكامل بطبقات تلافيفية، سنستمتع بنماذج شحيحة تتطلب معلمات أقل بكثير.

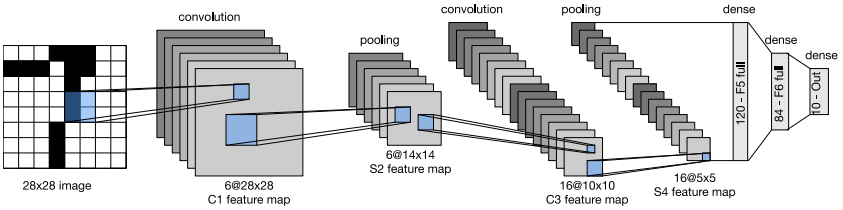
في هذا القسم، سنقدم LeNet، من بين أولى شبكات CNN المنشورة لجذب اهتمام واسع لأدائهافي مهام الرؤية الحاسوبية. تم تقديم النموذج بواسطة (وسمي بإسم) Yann LeCun، ثم باحثفي AT&T Bell Labs، بغرض التعرف على الأرقام المكتوبة بخط اليدفي الصور (LeCun et al.، 1998). يمثل هذا العمل تتويجا لعقد من البحث لتطوير التكنولوجيا.في عام 1989، نشر فريق LeCun أول دراسة لتدريب شبكات CNN بنجاح من خلال الانتشار الخلفي backpropagation (LeCun et al.، 1989).

في ذلك الوقت، حققت LeNet نتائج رائعة مطابقة لأداء آلات المتجهات الداعمة support vector machines، ثم نهجًا مهيمنًافي التعلم الخاضع للإشراف، حيث حقق معدل خطأ أقل من 1٪ لكل رقم. تم تكييف LeNetفي النهاية للتعرف على الأرقام لمعالجة الودائعفي أجهزة

الصراف الآلي. حتى يومنا هذا، لا تزال بعض أجهزة الصراف الآلي تشغل الكود الذي كتبه يان ليكون وزميله ليون بوتوفي التسعينيات!

LeNet .7.6.1

على مستوى عالٍ، تتكون LeNet (LeNet-5) من جزأين: (1) مشفر تلافيفي convolutional encoder يتكون من طبقتين تلافيفيتين؛ و (2) كتلة كثيفة dense block تتكون من ثلاث طبقات متصلة بالكامل؛ تم تلخيص المعماريةفي الشكل 7.6.1.



الشكل 7.6.1 تدفق البياناتفي LeNet. الإدخال عبارة عن رقم مكتوب بخط اليد، والمخرج هو احتمال أكثر من 10 نتائج محتملة.

الوحدات الأساسيةفي كل كتلة تلافيفية convolutional block هي طبقة تلافيفية convolutional layer، ودالة تنشيط sigmoid، وعملية تجميع المتوسط average pooling لاحقة. لاحظ أنه بينما تعمل ReLUs وmax-pooling بشكل أفضل، لم يتم إجراء هذه الاكتشافاتفي ذلك الوقت. تستخدم كل طبقة تلافيفية نواة ودالة تنشيط sigmoid. تطابق هذه الطبقات مدخلات مرتبة مكانيًا إلى عدد من خرائط المعالم ثنائية الأبعاد، مما يؤدي عادةً إلى زيادة عدد القنوات. تحتوي الطبقة التلافيفية الأولى على 6 قنوات إخراج، بينما تحتوي الثانية على 16. كل عملية تجميع (خطوة 2) تقلل الأبعاد بعامل من خلال الاختزال المكاني spatial downsampling. تبعث الكتلة التلافيفية ناتجًا بالشكل المعطى بواسطة (حجم الدُفعة، عدد القنوات، الارتفاع، العرض).

من أجل تمرير الإخراج من الكتلة التلافيفية convolutional block إلى الكتلة الكثيفة dense block، يجب علينا تسطيح كل مثالفي الدفعات الصغيرة minibatch. بمعنى آخر، نأخذ هذا الإدخال رباعي الأبعاد ونحوله إلى مدخلات ثنائية الأبعاد تتوقعها طبقات متصلة تمامًا: كتذكير، يستخدم التمثيل ثنائي الأبعاد الذي نرغب فيه البعد الأول لفهرسة الأمثلةفي minibatch والثانية لإعطاء تمثيل متجه مسطح flat لكل مثال. تحتوي كتلة LeNet الكثيفة على ثلاث طبقات

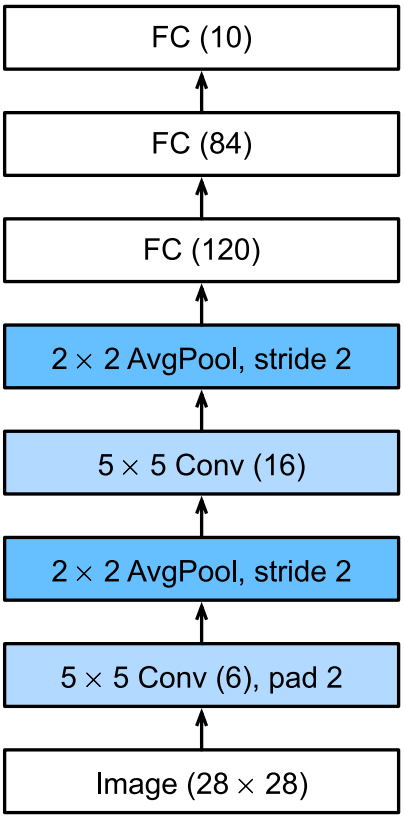
متصلة بالكامل، مع مخرجات 120 و84 و10 على التوالي. نظرًا لأننا ما زلنا نقوم بالتصنيف، فإن طبقة الإخراج ذات 10 أبعاد تتوافق مع عدد فئات الإخراج الممكنة.

أثناء الوصول إلى النقطة التي تفهم فيها حقًا ما يجري داخل LeNet، ربما تكون قد اتخذت القليل من العمل، نأمل أن يقنعك مقتطف الشفرة التالي بأن تنفيذ مثل هذه النماذج باستخدام أطر عمل التعلم العميق الحديثة أمر بسيط بشكل ملحوظ. نحتاج فقط إلى إنشاء كتلة `Sequential` وتربط معًا للطبقات المناسبة ، باستخدام تهيئة Xavier كما هو مقدمفي القسم 5.4.2.2.

```
import tensorflow as tf
from d2l import tensorflow as d2l

class LeNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6,
kernel_size=5,
                                   activation='sigmoid',
padding='same'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Conv2D(filters=16,
kernel_size=5,

activation='sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(120,
activation='sigmoid'),
            tf.keras.layers.Dense(84,
activation='sigmoid'),
            tf.keras.layers.Dense(num_classes)])
```

نأخذ بعض الحريةفي إعادة إنتاج LeNet بقدر ما نستبدل طبقة التنشيط الغاوسية بطبقة softmax. هذا يبسط التنفيذ إلى حد كبير، ليس أقلها بسبب حقيقة أن مفكك شفرة Gaussian نادراً ما يستخدمفي الوقت الحاضر. بخلاف ذلك، تتطابق هذه الشبكة مع بنية LeNet-5 الأصلية.

دعونا نرى ما يحدث داخل الشبكة. من خلال تمرير صورة أحادية القناة 28 × 28 (بالأبيض والأسود) عبر الشبكة وطباعة شكل الإخراج في كل طبقة، يمكننا فحص النموذج للتأكد من أن عملياته تتماشى مع ما نتوقعه من الشكل 7.6.2.



الشكل 7.6.2 التدوين المضغوط Compressed notation لـ LeNet-5.

```
@d2l.add_to_class(d2l.Classifier)  #@save
def layer_summary(self, X_shape):
    X = tf.random.normal(X_shape)
    for layer in self.net.layers:
        X = layer(X)
        print(layer.__class__.__name__, 'output
shape:\t', X.shape)

model = LeNet()
model.layer_summary((1, 28, 28, 1))
```

```
Conv2D output shape:         (1, 28, 28, 6)
AveragePooling2D output shape:       (1, 14, 14, 6)
```

```
Conv2D output shape:         (1, 10, 10, 16)
AveragePooling2D output shape:       (1, 5, 5, 16)
Flatten output shape:        (1, 400)
Dense output shape:  (1, 120)
Dense output shape:  (1, 84)
Dense output shape:  (1, 10)
```

لاحظ أنه يتم تقليل ارتفاع وعرض التمثيل في كل طبقة عبر الكتلة التلافيفية (مقارنة بالطبقة السابقة). تستخدم الطبقة التلافيفية الأولى 2 بكسل من الحشو للتعويض عن الانخفاض في الارتفاع والعرض الذي قد ينتج عن استخدام النواة 5 × 5. بالإضافة إلى ذلك، فإن حجم صورة 28 × 28 بكسل في مجموعة بيانات MNIST OCR الأصلية هو نتيجة لاقتطاع صفوف 2 بكسل (وأعمدة) من عمليات المسح الأصلية التي تم قياس 32 × 32 بكسل. تم القيام بذلك بشكل أساسي لتوفير مساحة (تخفيض بنسبة 30٪) في وقت كانت فيه الميجابايت مهمة.

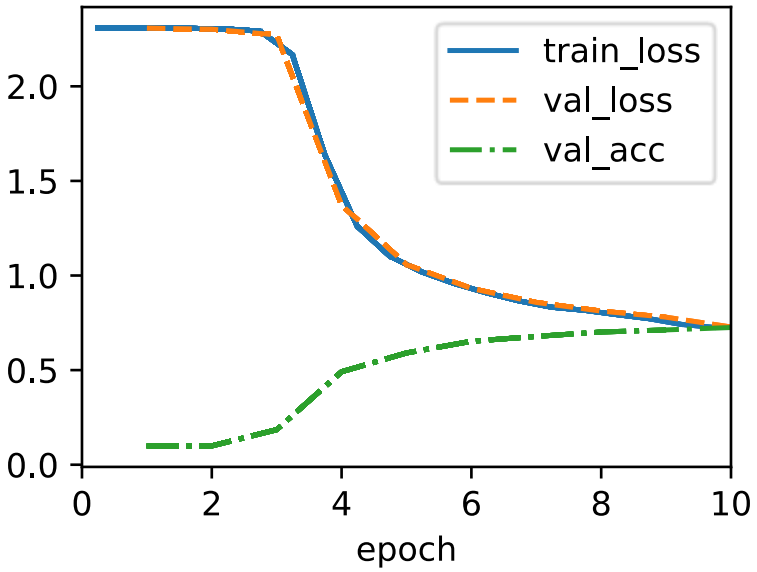
في المقابل، تتجاهل الطبقة التلافيفية الثانية الحشو، وبالتالي يتم تقليل الارتفاع والعرض بمقدار 4 بكسل. مع صعود كومة الطبقات، يزيد عدد القنوات طبقة الطبقة العلوية من 1 في الإدخال إلى 6 بعد الطبقة التلافيفية الأولى و16 بعد الطبقة التلافيفية الثانية. ومع ذلك، فإن كل طبقة تجميع تقسم الارتفاع والعرض إلى النصف. أخيرًا، تقلل كل طبقة متصلة بالكامل من الأبعاد، وتصدر أخيرًا ناتجًا يتطابق بعده مع عدد الفئات.

7.6.2. التدريب Training

الآن بعد أن قمنا بتنفيذ النموذج، دعنا نجري تجربة لنرى كيف يعمل نموذج LeNet-5 على Fashion-MNIST.

بينما تحتوي شبكات CNN على عدد أقل من المعلمات، إلا أنه لا يزال من الممكن حسابها أكثر تكلفة من MLPs العميقة المماثلة لأن كل معلمة تشارك في العديد من عمليات الضرب. إذا كان لديك وصول إلى وحدة معالجة الرسومات GPU، فقد يكون هذا هو الوقت المناسب لوضعها موضع التنفيذ لتسريع التدريب. لاحظ أن فئة `d2l.Trainer` تهتم بكل التفاصيل. بشكل افتراضي، يقوم بتهيئة معلمات النموذج على الأجهزة المتاحة. تمامًا كما هو الحال مع MLPs، فإن دالة الخطأ لدينا هي إنتروبيا متقاطعة cross-entropy، ونقوم بتقليلها عن طريق التدرج الاشتقاقي العشوائي المصغر minibatch stochastic gradient descent.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128)
with d2l.try_gpu():
    model = LeNet(lr=0.1)
    trainer.fit(model, data)
```



7.6.3 الملخص

في هذا الفصل أحرزنا تقدمًا كبيرًا. انتقلنا من MLPs في الثمانينيات إلى شبكات CNN في التسعينيات وأوائل العقد الأول من القرن الحادي والعشرين. تظل البُنى المقترحة، على سبيل المثال،في شكل LeNet-5 ذات مغزى، حتى يومنا هذا. يجدر مقارنة معدلات الخطأفي Fashion-MNISTالتي يمكن تحقيقها مع LeNet-5 بأفضل ما يمكن باستخدام MLPs (القسم 5.2) وتلك ذات البنى الأكثر تقدمًا مثل ResNet (القسم 8.6). يشبه LeNet الأخير أكثر من السابق. أحد الاختلافات الأساسية، كما سنرى، هو أن الكميات الأكبر من الحسابات أتاحت بنى معمارية أكثر تعقيدًا بشكل ملحوظ.

الاختلاف الثاني هو السهولة النسبية التي تمكنا بها من تنفيذ LeNet. ماكان يمثل تحديًا هندسيًا يستحق شهورًا من C ++ وكود التجميع assembly code، والهندسة لتحسين SN، وأداة التعلم العميق القائمة على Lisp، (1988, Bottou and Le Cun)، وأخيراً يمكن الآن إجراء التجارب مع النماذجفي دقائق. هذه الزيادة الهائلةفي الإنتاجية هي التي أدت إلى إضفاء الطابع الديمقراطي على تطوير نموذج التعلم العميق بشكل هائل.في الفصل التالي سوف نتبع هذا الأرنب لنرى أين يأخذنا.

7.6.4. التمارين

1. دعونا نحدث LeNet. نفِّذ واختبر التغييرات التالية:

1. استبدل تجميع المتوسط average pooling بتجميع الحد الاقصى max-pooling.
2. استبدل طبقة softmax بـ ReLU.

2. حاول تغيير حجم شبكة نمط LeNet لتحسين دقتها بالإضافة إلى max-pooling و ReLU.
   1. اضبط حجم نافذة الالتفاف.
   2. اضبط عدد قنوات الإخراج.
   3. اضبط عدد طبقات الالتفاف.
   4. اضبط عدد الطبقات المتصلة بالكامل.
   5. اضبط معدلات التعلم وتفاصيل التدريب الأخرى (على سبيل المثال ، التهيئة وعدد الفترات.)
3. جرب الشبكة المحسنة على مجموعة بيانات MNIST الأصلية.
4. اعرض تنشيط الطبقة الأولى والثانية من LeNet لمدخلات مختلفة (على سبيل المثال، السترات والمعاطف sweaters and coats).
5. ماذا يحدث لعمليات التنشيط عندما تقوم بإدخال صور مختلفة بشكل كبيرفي الشبكة (على سبيل المثال، القطط أو السيارات أو حتى الضوضاء العشوائية)؟

8

الشبكات العصبية التلافيفية الحديثة

8. الشبكات العصبية التلافيفية الحديثة Modern Convolutional Neural Networks

الآن بعد أن فهمنا أساسيات توصيل شبكات CNN معًا، فلنقم بجولةفي هياكل CNN الحديثة. هذه الجولة، بالضرورة، غير مكتملة، وذلك بفضل العدد الكبير من التصاميم الجديدة المثيرة التي تمت إضافتها. تنبع أهميتها من حقيقة أنه لا يمكن استخدامها مباشرة لمهام الرؤية فحسب، بل إنها تعمل أيضًا كمولدات ميزات أساسية للمهام الأكثر تقدمًا مثل التتبع tracking (Zhang et al.، 2021)، والتقطيع segmentation (Long et al.، 2015)، اكتشاف الكائن object detection (Redmon and Farhadi، 2018)، أو تغيير النمط style transformation (Gatys et al.، 2016).في هذا الفصل، تتوافق معظم الأقسام مع بُنية CNN الهامة التي كانت في مرحلة ما (أو حاليًا) النموذج الأساسي الذي تم بناء العديد من مشاريع البحث والأنظمة المنشورة عليه. كانت كل من هذه الشبكات لفترة وجيزة معمارية مهيمنة وكان العديد منهم فائزين أو وصيفينفي مسابقة ImageNet التي كانت بمثابة مقياس للتقدمفي التعلم الخاضع للإشراف في الرؤية الحاسوبية منذ عام 2010. وفي الآونة الأخيرة فقط بدأت المحولات transformers في استبدال شبكات CNN، بدءًا من Dosovitskiy et al. (2021) ويليه محولswin، (Liu et al.، 2021). سنغطي هذا التطور لاحقًافي الفصل الخاص بآليات الانتباه والمحولات Attention Mechanisms and Transformers.

في حين أن فكرة الشبكات العصبية العميقة بسيطة للغاية (تكديس مجموعة من الطبقات معًا)، يمكن أن يختلف الأداء بشكل كبير عبر خيارات البُنى والمعلمات الفائقة. الشبكات العصبية الموصوفةفي هذا الفصل هي نتاج الحدس intuition، وبعض الأفكار الرياضية، والكثير من التجربة والخطأ trial and error. نقدم هذه النماذج بترتيب زمني، جزئيًا لنقل إحساس بالتاريخ بحيث يمكنك تكوين حدسك الخاص حول المكان الذي يتجه إليه المجال وربما تطوير البُنى الخاصة بك. على سبيل المثال، قدّم التسوية بالدُفعات batch normalization والتوصيلات المتبقية الموصوفة residual connections في هذا الفصل فكرتين شائعتين للتدريب وتصميم النماذج العميقة، وكلاهما تم تطبيقهما منذ ذلك الحين على البُنى التي تتجاوز الرؤية الحاسوبية أيضًا.

نبدأ جولتنافي شبكات CNN الحديثة مع AlexNet،(Krizhevsky et al.، 2012)، وهي أول شبكة واسعة النطاق تم نشرها للتغلب على أساليب الرؤية الحاسوبية التقليديةفي تحدي الرؤية واسع النطاق ؛ شبكة VGG،(Simonyan and Zisserman، 2014) ، والتي تستخدم عددًا من الكتل المتكررة للعناصر ؛ الشبكةفي الشبكة (NiN) التي تجمع الشبكات العصبية بالكامل عبر المدخلات (Lin et al.، 2013) ؛ GoogLeNet التي تستخدم شبكات ذات تلافيف

متعددة الفروع multi-branch convolutions (Szegedy وآخرون ، 2015) ؛ الشبكة المتبقية residual network (ResNet)،(He et al.، 2016) ، والتي لا تزال من أشهر البُنى الجاهزة off-the-shelf architectures في الرؤية الحاسوبية ؛ كتل ResNeXt (Xie وآخرون ، 2017) للاتصالات المتفرقة sparser connections؛ و DenseNet، (Huang et al.,2017) لتعميم العمارة المتبقية. بمرور الوقت، تم تطوير العديد من التحسينات الخاصة للشبكات الفعالة، مثل تحويلات الإحداثيات coordinate shifts (ShiftNet)، (Wu et al.، 2018). بلغ هذا ذروتهفي البحث التلقائي عن بنيات فعالة مثل MobileNet v3 (Howard et al.،2019.). ويشمل أيضًا استكشاف التصميم شبه التلقائي semi-automatic design exploration لـ Radosavovic et al. (2020) التي أدت إلى RegNetX / Y والتي سنناقشها لاحقًافي هذا الفصل. العمل مفيد بقدر ما يوفر مسارًا لتزاوج حساب القوة العمياء ببراعة المجربفي البحث عن مساحات تصميم فعالة. وتجدر الإشارة أيضًا إلى عمل Liu et al. (2022) كما يوضح أن تقنيات التدريب (مثل، المحسنون optimizers، وزيادة البيانات data augmentation، والتنظيم regularization) تلعب دورًا محوريًافي تحسين الدقة. كما يوضح أيضًا أن الافتراضات القديمة، مثل حجم نافذة الالتفاف، قد تحتاج إلى إعادة النظر، نظرًا للزيادةفي الحساب والبيانات. سنغطي هذا والعديد من الأسئلة الأخرىفي الوقت المناسب خلال هذا الفصل.

8.1 الشبكات العصبية التلافيفية العميقة (AlexNet)

على الرغم من أن شبكات CNN كانت معروفة جيدًافي مجتمعات الرؤية الحاسوبية والتعلم الآلي بعد تقديم LeNet، (LeCun et al.,1995)، إلا أنها لم تهيمن على هذا المجال على الفور. على الرغم من أن LeNet حققت نتائج جيدةفي مجموعات البيانات الصغيرة المبكرة، إلا أنه لم يتم بعد إنشاء أداء وجدوى تدريب شبكات CNN على مجموعات بيانات أكبر وأكثر واقعية.في الواقع، خلال معظم الوقت الفاصل بين أوائل التسعينيات ونتائج مستجمعات المياه watershed results لعام 2012 (Krizhevsky et al.، 2012)، غالبًا ما تم تجاوز الشبكات العصبية بواسطة طرق التعلم الآلي الأخرى، مثل طرق النواة kernel، (Schölkopf and Smola,2002)، الطرق الجماعية ensemble (Freund et al.، 1996)، والتقدير المهيكل structured estimation (Taskar et al.، 2004).

بالنسبة للرؤية الحاسوبية، ربما لا تكون هذه المقارنة دقيقة تمامًا. أي، على الرغم من أن مدخلات الشبكات التلافيفية CNN تتكون من قيم بكسل خام أو معالجة بخفة lightly-processed (على سبيل المثال، عن طريق التوسيط centering)، فإن الممارسين لن يقوموا أبدًا بتغذية وحدات البكسل الخامفي النماذج التقليدية. بدلاً من ذلك، تتكون خطوط أنابيب الرؤية الحاسوبية

النموذجية من خطوط أنابيب استخراج الميزات الهندسية يدويًا، مثل SIFT، (Lowe,2004)، SURF، (Bay et al.,2006)، وأكياس الكلمات المرئية bags of visual words (Sivic and Zisserman، 2003). بدلاً من تعلم الميزات learning the features، تم تصميم الميزات features were crafted. جاء معظم التقدم من وجود أفكار أكثر ذكاءً لاستخراج الميزات من ناحية ونظرة عميقةفي الهندسة geometry (Hartley and Zisserman، 2000) من ناحية أخرى. غالبًا ما كانت خوارزمية التعلم تعتبر فكرة متأخرة.

على الرغم من توفر بعض مسرعات الشبكة العصبيةفي التسعينيات، إلا أنها لم تكن قوية بما يكفي لإنشاء شبكات CNN عميقة متعددة القنوات ومتعددة الطبقات مع عدد كبير من المعلمات. على سبيل المثال، تمكنت GeForce 256 من NVIDIA من عام 1999 من معالجة 480 مليون عمليةفي الثانية على الأكثر (MFLOPs)، دون أي إطار عمل برمجة مفيد للعمليات خارج الألعاب. تستطيع مسرعات اليوم أداء ما يزيد عن 300 TFLOPs لكل جهاز (NVIDIA's Ampere A100). لاحظ أن FLOPs هي عمليات الفاصلة العائمة مثل عمليات الضرب والإضافات. علاوة على ذلك، كانت مجموعات البيانات datasets لا تزال صغيرة نسبيًا: كان التعرف الضوئي على الحروف على 60.000 صورة بكسل منخفضة الدقة مهمة صعبة للغاية. يضاف إلى هذه العقبات، الحيل الرئيسية لتدريب الشبكات العصبية بمافي ذلك توجيهات تهيئة المعلمات (Glorot and Bengio، 2010)، الانواع الذكية من التدرج الاشتقاقي العشوائي (Kingma and Ba، 2014)، دوال التنشيط غير السحق non-squashing activation functions (Nair and Hinton، 2010)، وتقنيات التنظيم الفعالة effective regularization techniques (Srivastava et al.، 2014) لا تزال مفقودة.

وبالتالي، بدلاً من تدريب أنظمة من طرف إلى طرف end-to-end (بكسل إلى تصنيف pixel to classification)، بدت خطوط الأنابيب الكلاسيكية classical pipelines أكثر مثل هذا:

1. احصل على مجموعة بيانات مثيرة للاهتمام.في الأيام الأولى، كانت مجموعات البيانات هذه تتطلب أجهزة استشعار باهظة الثمن. على سبيل المثال، تتميز Apple QuickTake 100 لعام 1994 بدقة هائلة تبلغ 0.3 ميجابكسل (VGA)، قادرة على تخزين ما يصل إلى 8 صور، وكل ذلك بسعر 1000 دولار.
2. قم بإجراء معالجة مسبقة لمجموعة البيانات باستخدام ميزات مصنوعة يدويًا استنادًا إلى بعض المعرفة بالبصريات والهندسة وأدوات التحليل الأخرى، وأحيانًا على الاكتشافات المصادفة لطلاب الدراسات العليا المحظوظين.
3. قم بتغذية البيانات من خلال مجموعة قياسية من مستخلصات الميزات مثل SIFT (تحويل ميزة مقياس ثابت scale-invariant feature transform)،

(2004,Lowe)، SURF (ميزات قوية مسرعة speeded up robust features) (Bay et al.، 2006)، أو أي عدد من ناحية أخرى ـ خطوط الأنابيب المضبوطة. لا يزال OpenCV يوفر مستخلصات SIFT حتى يومنا هذا!

4. تخلص من التمثيلات representations الناتجةفي المصنف المفضل لديك، من المحتمل أن يكون نموذجًا خطيًا أو طريقة kernel، لتدريب المصنف.

إذا تحدثت إلى باحثي التعلم الآلي، فإنهم يعتقدون أن التعلم الآلي مهم وجميلفي نفس الوقت. أثبتت النظريات الأنيقة خصائص المصنفات المختلفة (Boucheron et al.، 2005) وأصبح التحسين المحدب convex optimization (Boyd and Vandenberghe، 2004) الدعامة الأساسية للحصول عليها. كان مجال التعلم الآلي مزدهرًا وصارمًا ومفيدًا بشكل كبير. ومع ذلك، إذا تحدثت إلى باحثفي الرؤية الحاسوبية، فستسمع قصة مختلفة تمامًا. سيقولون لك إن الحقيقة القذرة للتعرف على الصور هي أن الميزات features والهندسة الرياضية geometry (Hartley and Zisserman، 2000، Hartley and Kahl، 2009)، والهندسة engineering، بدلاً من خوارزميات التعلم الجديدة، هي التي أدت إلى التقدم. يعتقد الباحثون في الرؤية الحاسوبية بشكل مبرر أن مجموعة بيانات أكبر أو أنظف قليلاً أو خط أنابيب محسّن قليلاً لاستخراج الميزات أهم بكثير بالنسبة للدقة النهائية أكثر من أي خوارزمية تعلم.

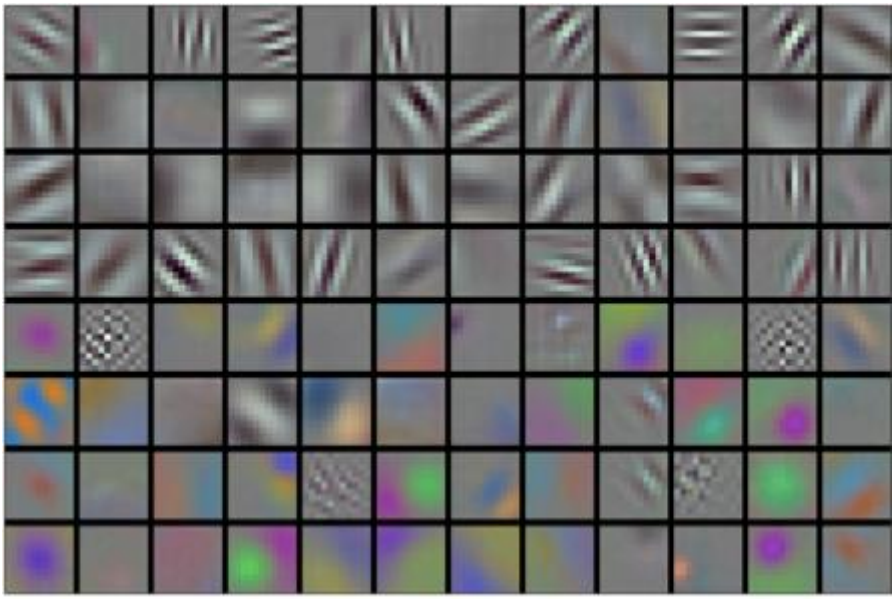
8.1.1. التعلم التمثيلي Representation Learning

هناك طريقة أخرى لتصوير الوضع وهي أن أهم جزء من خط الأنابيب كان التمثيل representation. وحتى عام 2012، تم حساب التمثيلفي الغالب ميكانيكيًا.في الواقع، كانت هندسة مجموعة جديدة من دوال الميزات، وتحسين النتائج، وكتابة الطريقة نوعًا بارزًا من المقالات. SIFT، (2004,Lowe) ، SURF، (2006,.Bay et al) ، HOG (الرسوم البيانية للتدرج الموجه histograms of oriented gradient) (Dalal and Triggs، 2005) ، وأكياس الكلمات المرئية bags of visual words) (Sivic and Zisserman، 2003) ، وما شابه ذلك من مستخلصات الميزة السائدة.

مجموعة أخرى من الباحثين، بمافي ذلك Yann LeCun و Geoff Hinton و Yoshua Bengio و Andrew Ng و Shun-ichi Amari و Juergen Schmidhuber، لديهم خطط مختلفة. لقد اعتقدوا أنه يجب تعلم الميزات نفسها. علاوة على ذلك، اعتقدوا أنه لكي تكون معقدة بشكل معقول، يجب أن تتكون الميزات بشكل هرمي hierarchically من طبقات متعددة تم تعلمها بشكل مشترك، ولكل منها معلمات قابلة للتعلم.في حالة الصورة، قد تكتشف الطبقات الدنيا الحواف والألوان والنسيج textures، على غرار الطريقة التي يعالج بها النظام المرئيفي الحيوانات مدخلاته. على وجه الخصوص، ظل التصميم التلقائي للميزات المرئية مثل

تلك التي تم الحصول عليها عن طريق التشفير المتناثر sparse coding (Olshausen and Field، 1996) يمثل تحديًا مفتوحًا حتى ظهور شبكات CNN الحديثة. لم يكن الأمر كذلك حتى (2012) .Dean et al، Le (2013) أن فكرة إنشاء ميزات من بيانات الصورة اكتسبت تلقائيًا جذباً كبيرًا.

أول شبكة CNN حديثة (.Krizhevsky et al، 2012)، التي سميت AlexNet على اسم أحد مخترعيها، Alex Krizhevsky، هي إلى حد كبير تحسين تطوري على LeNet. لقد حققت أداءً ممتازًافي تحدي ImageNet لعام 2012.



الشكل 8.1.1 تم التعرف على فلاتر الصور بواسطة الطبقة الأولى من AlexNet. إعادة التوليد Reproduction مجاملة من Krizhevsky et al. (2012).

ومن المثير للاهتمامفي الطبقات السفلية للشبكة، أن النموذج تعلم مستخلصات الميزات feature extractors التي تشبه بعض المرشحات التقليدية traditional filters. يوضح الشكل 8.1.1 واصفات الصورة ذات المستوى الأدنى. قد تبني الطبقات العلياني الشبكة على هذه التمثيلات لتمثيل هياكل أكبر، مثل العيون والأنوف وشفرات العشب وما إلى ذلك. قد تمثل الطبقات العليا كائنات كاملة مثل الأشخاص أو الطائرات أو الكلاب أو الأطباق الطائرة.في النهاية، تتعلم الحالة المخفية النهائية تمثيلًا مضغوطًا للصورة يلخص محتوياتها بحيث يمكن فصل البيانات التي تنتمي إلى فئات مختلفة بسهولة.

تشترك AlexNet (2012) وسابقتها LeNet (1995)في العديد من العناصر المعمارية. هذا يطرح السؤال: لماذا استغرق الأمر كل هذا الوقت؟ يتمثل أحد الاختلافات الرئيسيةفي أنه خلال العقدين الماضيين، زادت كمية البيانات وقوة الحوسبة المتاحة بشكل كبير. على هذا النحو، كان AlexNet أكبر بكثير: فقد تم تدريبه على المزيد من البيانات، وعلى وحدات معالجة رسومات GPU أسرع بكثير، مقارنة بوحدات المعالجة المركزية CPU المتوفرةفي عام 1995.

8.1.1.1. المكون المفقود: البيانات Missing Ingredient: Data

تتطلب النماذج العميقة ذات الطبقات المتعددة كميات كبيرة من البيانات من أجل الدخول إلى النظام حيث تتفوق بشكل كبير على الأساليب التقليدية القائمة على التحسينات المحدبة convex optimizations (على سبيل المثال، الأساليب الخطية والنواة). ومع ذلك، نظرًا لقدرة التخزين المحدودة لأجهزة الكمبيوتر، والتكلفة النسبية لأجهزة الاستشعار (التصويرية)، وميزانيات البحث الأكثر تشددًا نسبيًافي التسعينيات، اعتمدت معظم الأبحاث على مجموعات بيانات صغيرة. اعتمدت العديد من الأوراق البحثية على مجموعة قواعد البيانات الخاصة بـ UCI ، والتي احتوى الكثير منها على مئات أو (بضعة آلاف) من الصور الملتقطة بدقة منخفضة وغالبًا بخلفية نظيفة بشكل مصطنع.

في عام 2009، تم إصدار مجموعة بيانات ImageNet، (2009,Deng et al.)، مما يشكل تحديا للباحثين لتعلم نماذج من مليون مثال، 1000 لكل منها من 1000 فئة مميزة من الكائنات. كانت الفئات نفسها مبنية على أكثر عُقد الأسماء شيوعًافي WordNet، (1995,Miller). استخدم فريق ImageNet بحث الصور من Google للترشيح المسبق لمجموعات كبيرة من المرشحين لكل فئة واستخدم خط أنابيب التعهيد الجماعي Amazon Mechanical Turk لتأكيد ما إذا كانت تنتمي إلى الفئة المرتبطة بكل صورة. كان هذا المقياس غير مسبوق، متجاوزًا الآخرين بأكثر من ترتيب من حيث الحجم (على سبيل المثال، يحتوي CIFAR-100 على 60000 صورة). كان الجانب الآخر هو أن الصور كانت بدقة عالية نسبيًا 224 × 224 بكسل، على عكس مجموعة بيانات TinyImages بحجم 80 مليون (Torralba et al.، 2008)، والتي تتكون من صور مصغرة 32 × 32 بكسل. سمح هذا بتشكيل ميزات ذات مستوى أعلى. دفعت المنافسة المصاحبة، والتي أطلق عليها اسم تحدي التعرف البصري على نطاق واسع على ImageNet Large Scale Visual Recognition Challenge،ImageNet (2015,Russakovsky et al.)، الرؤية الحاسوبية وأبحاث التعلم الآلي إلى الأمام، مما أدى إلى تحدي الباحثين لتحديد النماذج الأفضل أداءً على نطاق أكبر مما كان الأكاديميون يعتبرونه سابقًا. تحتوي أكبر مجموعات بيانات الرؤية، مثل LAION-5B، (Schuhmann et al.2022) على مليارات الصور مع بيانات وصفية إضافية.

8.1.1.2. المكون المفقود: الأجهزة Missing Ingredient: Hardware

نماذج التعلم العميق هي مستهلكين شرهين لدورات الحوسبة compute cycles. يمكن أن يستغرق التدريب مئات الفترات epochs، ويتطلب كل تكرار تمرير البيانات عبر طبقات عديدة من عمليات الجبر الخطي المكلفة حسابيًا. هذا هو أحد الأسباب الرئيسية وراء تفضيل الخوارزميات البسيطةفي التسعينيات وأوائل العقد الأول من القرن الحادي والعشرين، بناءً على الأهداف المحدبة المحسّنة optimized convex objectives بشكل أكثر كفاءة.

أثبتت وحدات المعالجة الرسومية (GPUs) أنها غيرت قواعد اللعبةفي جعل التعلم العميق ممكنًا. تم تطوير هذه الرقائق منذ فترة طويلة لتسريع معالجة الرسومات لإفادة ألعاب الكمبيوتر. على وجه الخصوص، تم تحسينها لضرب المصفوفةـالمتجه عالية 4 × 4، والتي تعد مطلوبة للعديد من مهام رسومات الكمبيوتر. لحسن الحظ، الرياضيات مشابهة بشكل لافت للنظر لتلك المطلوبة لحساب الطبقات التلافيفية.في ذلك الوقت تقريبًا، بدأت NVIDIA وATIفي تحسين وحدات معالجة الرسومات لعمليات الحوسبة العامة (Fernando، 2004)، ووصلت إلى حد تسويقها على أنها وحدات معالجة رسومات للأغراض العامة general-purpose GPUs (GPGPUs).

لتوفير بعض الحدس، ضعفي اعتبارك أنوية المعالج الدقيق الحديث (CPU). كل من النوى قوي إلى حد ما يعمل بتردد ساعة عالٍ وذاكرة تخزين مؤقت كبيرة (تصل إلى عدة ميغا بايت من L3). كل نواة مناسبة تمامًا لتنفيذ مجموعة واسعة من التعليمات instructions، مع تنبؤات الفروع branch predictors، وخط أنابيب عميق deep pipeline، ووحدات تنفيذ متخصصة، وتنفيذ تخميني speculative execution، والعديد من الأجراس والصفارات الأخرى التي تمكنه من تشغيل مجموعة كبيرة ومتنوعة من البرامج مع تدفق تحكم متطور sophisticated control flow. ومع ذلك، فإن هذه القوة الواضحة تكمن أيضًافي كعب أخيل: فلبناء الأغراض العامة مكلف للغاية. إنها تتفوقفي كود الأغراض العامة مع الكثير من التحكمفي التدفق. هذا يتطلب الكثير من مساحة الرقاقة chip area، ليس فقط لوحدة الحساب والمنطق arithmetic logical unit (ALU) حيث يحدث الحساب، ولكن أيضًا لجميع الأجراس والصفارات المذكورة أعلاه، بالإضافة إلى واجهات الذاكرة memory interfaces، ومنطق التخزين المؤقت بين النوى caching logic between cores، والوصلات عالية السرعة high-speed interconnects، وما إلى ذلك. تعد وحدات المعالجة المركزية سيئة نسبيًافي أي مهمة واحدة عند مقارنتها بالأجهزة المخصصة. تحتوي أجهزة الكمبيوتر المحمولة الحديثة على 4ـ8 نوى cores، ونادرًا ما تتجاوز الخوادم المتطورة 64 نواة لكل مقبس socket، وذلك ببساطة لأنها ليست فعالة من حيث التكلفة.

وبالمقارنة، يمكن أن تتكون وحدات معالجة الرسومات من آلاف عناصر المعالجة الصغيرة (تحتوي أحدث رقائق Ampere من NIVIDA على ما يصل إلى 6912 نواة CUDA)، وغالبًا ما يتم تجميعهافي مجموعات أكبر (تسميها NVIDIA الاعوجاج warps). تختلف التفاصيل إلى حد ما بين NVIDIA وAMD وARM وبائعي الرقائق الآخرين.في حين أن كل نواة ضعيفة نسبيًا، تعمل بتردد ساعة 1 جيجاهرتز، فإن العدد الإجمالي لهذه النوى هو الذي يجعل أوامر وحدات معالجة الرسومات من حيث الحجم أسرع من وحدات المعالجة المركزية. على سبيل المثال، توفر وحدة معالجة الرسومات Ampere A100 الحديثة من NVIDIA أكثر من 300 TFLOPs لكل شريحة من أجل ضرب المصفوفة ـ المصفوفة متخصصة 16 بت (BFLOAT16)، وما يصل إلى 20 TFLOPs لعمليات النقطة العائمة ذات الأغراض العامة (FP32).في الوقت نفسه، نادراً ما يتجاوز أداء النقطة العائمة لوحدات المعالجة المركزية 1 TFLOPs. على سبيل المثال، يصل Graviton 3 من Amazon إلى 2 TFLOPs لعمليات دقيقة 16 بت، وهو رقم مشابه لأداء GPU لمعالج Apple M1.

هناك العديد من الأسباب التي تجعل وحدات معالجة الرسومات GPU أسرع بكثير من وحدات المعالجة المركزية من حيث FLOPs. أولاً، يميل استهلاك الطاقة إلى النمو بشكل تربيعي مع تردد الساعة. وبالتالي، بالنسبة لميزانية الطاقة لنواة وحدة المعالجة المركزية التي تعمل 4 مرات أسرع (رقم نموذجي)، يمكنك استخدام 16 نواة لوحدة معالجة الرسومات في $\frac{1}{4}$ السرعة التي تنتج $16 \times \frac{1}{4} = 4$ مرات من الأداء. ثانيًا، تعد نوى GPU أبسط بكثير (في الواقع، لم تكن قادرة حتى على تنفيذ التعليمات البرمجية للأغراض العامة لفترة طويلة)، مما يجعلها أكثر كفاءةفي استخدام الطاقة. على سبيل المثال، (1) تميل إلى عدم دعم التقييم التخميني speculative evaluation، (2) عادةً ما يكون من غير الممكن برمجة كل عنصر معالجة على حدة، و(3) ذاكرة التخزين المؤقت caches لكل نواة تميل إلى أن تكون أصغر بكثير. أخيرًا، تتطلب العديد من العملياتفي التعلم العميق نطاقًا ترددياً عاليًا للذاكرة. مرة أخرى، تتألق وحدات معالجة الرسومات هنا مع المسارات buses التي يبلغ عرضها على الأقل 10 أضعاف عدد وحدات المعالجة المركزية.

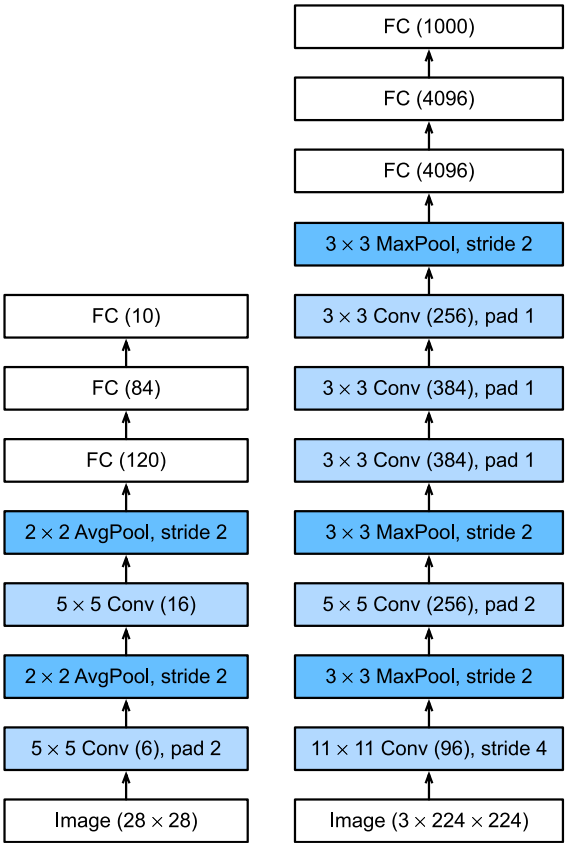
بالعودة إلى عام 2012. حدث تقدم كبير عندما نفذ Alex Krizhevsky وIlya Sutskever شبكة CNN عميقة يمكن تشغيلها على وحدات معالجة الرسومات. لقد أدركوا أن الاختناقات الحسابية computational bottlenecksفي شبكات CNN، والتلافيف ومضاعفات المصفوفات، كلها عمليات يمكن أن تكون متوازيةفي الأجهزة. باستخدام اثنين من NVIDIA GTX 580s مع 3 جيجابايت من الذاكرة، أي منهما كان قادرًا على 1.5 TFLOPs (لا يزال يمثل تحديًا لمعظم وحدات المعالجة المركزية بعد عقد من الزمان)، قاموا بتنفيذ التفافات

سريعة. كان كود cuda-convnet جيدًا بما يكفي لأنه لعدة سنوات كان معيارًا صناعيًا وعمل على تشغيل أول عامين من طفرة التعلم العميق.

AlexNet .8.1.2

AlexNet ، التي استخدمت شبكة CNN ذات 8 طبقات ، فازتفي مسابقة ImageNet Large Scale Visual Recognition Challenge 2012 بهامش كبير (Russakovsky et al.، 2013). أظهرت هذه الشبكة، لأول مرة، أن الميزات التي تم الحصول عليها من خلال التعلم يمكن أن تتجاوز الميزات المصممة يدويًا، مما يكسر النموذج السابقفي الرؤية الحاسوبية.

إن معماريات AlexNet وLeNet متشابهة بشكل لافت للنظر، كما يوضح الشكل 8.1.2. لاحظ أننا نقدم نسخة مبسطة قليلاً من AlexNet لإزالة بعض المراوغات التصميمية design quirks التي كانت مطلوبةفي عام 2012 لجعل النموذج مناسبًا لوحدتي GPU صغيرتين.



الشكل 8.1.2 من LeNet (يسار) إلى AlexNet (يمين).

هناك أيضًا اختلافات كبيرة بين AlexNet وLeNet. أولاً، AlexNet أعمق بكثير من LeNet5 الصغير نسبيًا. تتكون AlexNet من ثماني طبقات: خمس طبقات تلافيفية، وطبقتان مخفيتان متصلتان بالكامل، وطبقة إخراج متصلة بالكامل. ثانيًا، AlexNet تستخدم ReLU بدلاً من sigmoid كدالة التنشيط. دعنا نتعمق في التفاصيل أدناه.

8.1.2.1. المعمارية Architecture

في الطبقة الأولى لـ AlexNet، يكون شكل نافذة الالتفاف convolution window هو 11 × 11. نظرًا لأن الصور في ImageNet أعلى بثماني مرات وأعرض من صور MNIST، تميل الكائنات الموجودة في بيانات ImageNet إلى احتلال المزيد من وحدات البكسل بتفاصيل مرئية أكثر. وبالتالي، هناك حاجة إلى نافذة التفاف أكبر لالتقاط الكائن. يتم تقليل شكل نافذة الالتفاف في الطبقة الثانية إلى 5 × 5، متبوعًا بـ 3 × 3. بالإضافة إلى ذلك، بعد الطبقات التلافيفية الأولى والثانية والخامسة، تضيف الشبكة طبقات تجميع حد اقصى مع شكل نافذة 3 × 3 وخطوة 2. علاوة على ذلك، تمتلك AlexNet قنوات التفاف أكثر بعشر مرات من LeNet.

بعد آخر طبقة تلافيفية، توجد طبقتان كبيرتان متصلتان بالكامل مع 4096 ناتجًا. تتطلب هذه الطبقات معلمات نموذج 1 جيجابايت تقريبًا. نظرًا للذاكرة المحدودة في وحدات معالجة الرسومات المبكرة، استخدم AlexNet الأصلي تصميمًا مزدوجًا لدفق البيانات، بحيث يمكن أن تكون كل واحدة من وحدتي GPU مسؤولة عن تخزين وحساب نصف النموذج فقط. لحسن الحظ، أصبحت ذاكرة GPU وفيرة نسبيًا الآن، لذلك نادرًا ما نحتاج إلى تفكيك النماذج عبر وحدات معالجة الرسومات هذه الأيام (إصدارنا من طراز AlexNet ينحرف عن الورقة الأصلية في هذا الجانب).

8.1.2.2. دوال التنشيط Activation Functions

إلى جانب ذلك، قامت AlexNet بتغيير دالة التنشيط sigmoid إلى دالة تنشيط ReLU أبسط. من ناحية أخرى، يعد حساب دالة تنشيط ReLU أبسط. على سبيل المثال، لا يحتوي على عملية الأُس الموجودة في دالة التنشيط sigmoid. من ناحية أخرى، تجعل دالة التنشيط ReLU تدريب النموذج أسهل عند استخدام طرق تهيئة مختلفة للمعلمات. هذا لأنه، عندما يكون ناتج دالة التنشيط sigmoid قريبًا جدًا من 0 أو 1، يكون التدرج gradient لهذه المناطق تقريبًا 0، لذلك لا يمكن أن يستمر الانتشار الخلفي backpropagation في تحديث بعض معلمات النموذج. في المقابل، يكون التدرج لدالة تنشيط ReLU في الفاصل الزمني الموجب positive interval دائمًا 1 (القسم 5.1.2). لذلك، إذا لم تتم تهيئة معلمات النموذج بشكل صحيح، فقد

تحصل الدالة sigmoid على تدرج يبلغ 0 تقريبًافي الفاصل الزمني الموجب، بحيث لا يمكن تدريب النموذج بشكل فعال.

8.1.2.3. التحكم في القدرات والمعالجة المسبقة Capacity Control and Preprocessing

تتحكم AlexNetفي تعقيد نموذج الطبقة المتصلة بالكامل عن طريق التسرب (الحذف العشوائي) dropout (القسم 5.6)، بينما لا تستخدم LeNet سوى انحلال الوزن weight decay. لزيادة augment البيانات بشكل أكبر، أضافت حلقة التدريب الخاصة بـ AlexNet قدرًا كبيرًا من تكبير الصورة، مثل التقليب flipping والقص clipping وتغيير اللون color changes. هذا يجعل النموذج أكثر قوة ويقلل حجم العينة الأكبر بشكل فعال من الضبط الزائد overfitting. سنناقش زيادة البيانات data augmentation بمزيد من التفصيلفي القسم 14.1. انظر أيضًا (Buslaev et al.، 2020) للحصول على مراجعة متعمقة لخطوات المعالجة المسبقة preprocessing هذه.

```
import tensorflow as tf
from d2l import tensorflow as d2l

class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=96,
kernel_size=11, strides=4,
                                   activation='relu'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            tf.keras.layers.Conv2D(filters=256,
kernel_size=5, padding='same',
                                   activation='relu'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            tf.keras.layers.Conv2D(filters=384,
kernel_size=3, padding='same',
                                   activation='relu'),
            tf.keras.layers.Conv2D(filters=384,
kernel_size=3, padding='same',
                                   activation='relu'),
```

```
            tf.keras.layers.Conv2D(filters=256,
kernel_size=3, padding='same',
                                   activation='relu'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(4096,
activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(4096,
activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(num_classes)])
```

نقوم ببناء مثال بيانات أحادية القناة single-channel data بارتفاع وعرض 224 لملاحظة شكل الإخراج لكل طبقة. إنها تتطابق مع بنية AlexNet في الشكل 8.1.2.

```
AlexNet().layer_summary((1, 224, 224, 1))
```

```
Conv2D output shape:          (1, 54, 54, 96)
MaxPooling2D output shape:    (1, 26, 26, 96)
Conv2D output shape:          (1, 26, 26, 256)
MaxPooling2D output shape:    (1, 12, 12, 256)
Conv2D output shape:          (1, 12, 12, 384)
Conv2D output shape:          (1, 12, 12, 384)
Conv2D output shape:          (1, 12, 12, 256)
MaxPooling2D output shape:    (1, 5, 5, 256)
Flatten output shape:         (1, 6400)
Dense output shape:   (1, 4096)
Dropout output shape:         (1, 4096)
Dense output shape:   (1, 4096)
Dropout output shape:         (1, 4096)
Dense output shape:   (1, 10)
```

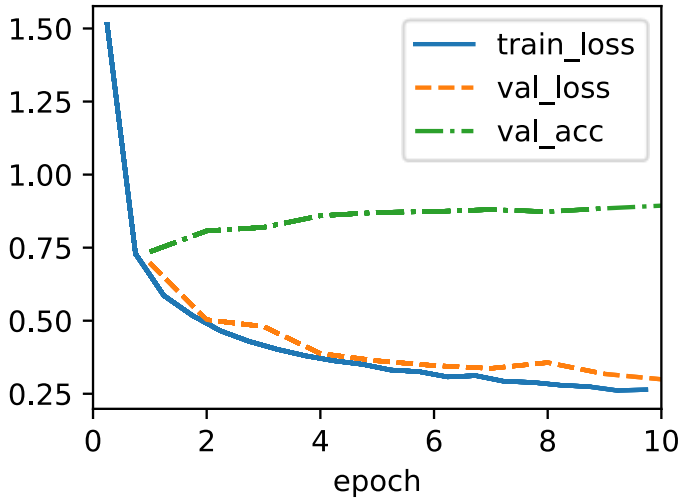
8.1.3. التدريب Training

على الرغم من تدريب AlexNet على ImageNet في (Krizhevsky et al.، 2012)، فإننا نستخدم Fashion-MNIST هنا نظرًا لأن تدريب نموذج ImageNet على التقارب convergence قد يستغرق ساعات أو أيام حتى على وحدة معالجة الرسومات الحديثة. تتمثل إحدى مشكلات تطبيق AlexNet مباشرة على Fashion-MNIST في أن صورها ذات دقة أقل (28 × 28 بكسل) من صور ImageNet. لجعل الأشياء تعمل، نقوم بتجميعها الى 224 × 224. هذه ليست ممارسة ذكية بشكل عام، لأنها ببساطة تزيد من التعقيد الحسابي دون إضافة معلومات. ومع ذلك، فإننا نفعل ذلك هنا لنكون مخلصين لمعمارية AlexNet. نقوم

بإجراء تغيير الحجم `resize` هذا باستخدام وسيطة تغيير الحجمفي مُنشئ `d2l.FashionMNIST`.

الآن، يمكننا البدءفي تدريب AlexNet. بالمقارنة مع LeNetفي القسم 7.6، فإن التغيير الرئيسي هنا هو استخدام معدل تعليمي أصغر وتدريب أبطأ بكثير بسبب الشبكة الأعمق والأوسع، ودقة الصورة الأعلى، والتلافيف الأكثر تكلفة.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(224,
224))
with d2l.try_gpu():
    model = AlexNet(lr=0.01)
    trainer.fit(model, data)
```



8.1.4. المناقشة

يحمل هيكل AlexNet تشابهًا مذهلاً مع LeNet، مع عدد من التحسينات المهمة، من حيث الدقة (dropout) وسهولة التدريب (ReLU). الأمر المذهل بنفس القدر هو مقدار التقدم الذي تم إحرازه فيما يتعلق بأدوات التعلم العميق. ما كان لعدة أشهر من العملفي عام 2012 يمكن الآن إنجازهفي عشرات الأسطر من التعليمات البرمجية باستخدام أي إطار عمل حديث.

بمراجعة المعمارية، نرى أن AlexNet لديها كعب أخيل عندما يتعلق الأمر بالكفاءة: تتطلب آخر طبقتين مخفيتين مصفوفات من الحجم 4096 × 6400 و4096 × 4096 على التوالي. هذا يتوافق مع ذاكرة 164 ميجابايت و81 MFLOPs من الحسابات، وكلاهما نفقات غير

بديهية، خاصة على الأجهزة الأصغر، مثل الهواتف المحمولة. هذا هو أحد أسباب تجاوز AlexNet لهياكل أكثر فاعلية سنغطيهافي الأقسام التالية. ومع ذلك، فهي خطوة أساسية من الشبكات الضحلة إلى الشبكات العميقة المستخدمةفي الوقت الحاضر. لاحظ أنه على الرغم من أن عدد المعلمات يتجاوز إلى حد بعيد كمية بيانات التدريبفي تجاربنا (تحتوي الطبقتان الأخيرتان على أكثر من 40 مليون معلمة، تم تدريبهما على مجموعات بيانات من 60 ألف صورة)، لا يكاد يوجد أي ضبط زائد overfitting: خطأ التدريب والتحقق من الصحة متطابقة تقريبًا خلال التدريب. ويرجع ذلك إلى التنظيم المحسن، مثل dropout المتأصلفي تصميمات الشبكات العميقة الحديثة.

على الرغم من أنه يبدو أن هناك عددًا قليلاً فقط من الخطوطفي تطبيق AlexNet مقارنةً بتطبيق LeNet، فقد استغرق المجتمع الأكاديمي سنوات عديدة لاحتضان هذا التغيير المفاهيمي والاستفادة من نتائجه التجريبية الممتازة. كان هذا أيضًا بسبب عدم وجود أدوات حسابية فعالة. في ذلك الوقت، لم يكن DistBelief (Dean et al., 2012)، ولا Caffe (Jia et al., 2014) موجودًا، ولا يزال Theano (Bergstra et al., 2010) يفتقر إلى العديد من الميزات المميزة. إن توفر TensorFlow (Abadi et al., 2016) هو الوحيد الذي غير هذا الوضع بشكل كبير.

8.1.5. التمارين

1. لمتابعة المناقشة أعلاه، قم بتحليل الخصائص الحسابية لـ AlexNet.
   1. احسب بصمة الذاكرة للتلافيف والطبقات المتصلة بالكامل، على التوالي. أي واحد يهيمن؟
   2. احسب التكلفة الحسابية للالتفافات والطبقات المتصلة بالكامل.
   3. كيف تؤثر الذاكرة (عرض النطاق الترددي bandwidth للقراءة والكتابة ووقت التاخير latency والحجم) على الحساب؟ وهل هناك فرقفي تأثيره على التدريب والاستدلال؟
2. أنت مصمم شريحة chip designer وتحتاج إلى موازنة الحساب وعرض النطاق الترددي للذاكرة. على سبيل المثال، تتطلب الشريحة الأسرع طاقة أكبر وربما مساحة شريحة أكبر. يتطلب المزيد من عرض النطاق الترددي للذاكرة المزيد من pins ومنطق التحكم control logic، وبالتالي مساحة أكبر أيضًا. كيف تقوم بالتحسين؟
3. لماذا لم يعد المهندسون يقدمون تقارير عن معايير الأداء على AlexNet؟
4. حاول زيادة عدد الفترات عند تدريب AlexNet. مقارنة مع LeNet، كيف تختلف النتائج؟ لماذا؟
5. قد تكون AlexNet معقدة للغاية بالنسبة لمجموعة بيانات Fashion-MNIST، خاصة بسبب الدقة المنخفضة للصور الأولية.

1. حاول تبسيط النموذج لجعل التدريب أسرع، مع ضمان عدم انخفاض الدقة بشكل ملحوظ.
2. قم بتعديل حجم الدُفعة batch size، ولاحظ التغييراتفي الإنتاجية throughput (الصور/الثانية) والدقة وذاكرة وحدة معالجة الرسومات.
6. طبق التسرب dropout وReLU على LeNet-5. هل تتحسن؟ هل يمكنك تحسين الأمور أكثر من خلال المعالجة المسبقة للاستفادة من الثوابت invariances الكامنةفي الصور؟
7. هل يمكنك جعل AlexNet تعاني من الضبط الزائد overfitting؟ ما الميزة التي تحتاج إلى إزالتها أو تغييرها لكسر التدريب؟

8.2 الشبكات التي تستخدم الكتل (VGG) Networks Using Blocks

بينما قدمت AlexNet دليلًا تجريبيًا على أن شبكات CNN العميقة يمكنها تحقيق نتائج جيدة، إلا أنها لم تقدم نموذجًا عامًا لتوجيه الباحثين اللاحقينفي تصميم شبكات جديدة.في الأقسام التالية، سوف نقدم عدة مفاهيم إرشادية heuristic concepts شائعة الاستخدام لتصميم شبكات عميقة.

يعكس التقدمفي هذا المجال تقدم VLSI (تكامل واسع النطاق جدًا very large scale integration)في تصميم الرقائق حيث انتقل المهندسون من وضع الترانزستورات إلى العناصر المنطقية logical elements إلى الكتل المنطقية logic blocks (Mead، 1980). وبالمثل، فإن تصميم معماريات الشبكات العصبية قد أصبح أكثر تجريدًا بشكل تدريجي، مع انتقال الباحثين من التفكير من حيث الخلايا العصبية الفردية إلى طبقات كاملة، والآن إلى الكتل blocks، وتكرار أنماط الطبقات. بعد عقد من الزمان، تقدم هذا الآن للباحثين الذين يستخدمون نماذج مدربة بالكامل لإعادة توظيفهم لمهام مختلفة، وإن كانت مرتبطة. عادة ما تسمى هذه النماذج الكبيرة التي تم اختبارها مسبقًا بنماذج الأساس foundation models (Bommasani et al.، 2021).

العودة إلى تصميم الشبكة. ظهرت فكرة استخدام الكتل لأول مرة من مجموعة الهندسة المرئية Visual Geometry Group (VGG)في جامعة أكسفورد،في شبكة VGG التي تحمل الاسم نفسه (Simonyan and Zisserman، 2014). من السهل تنفيذ هذه الهياكل المتكررةفي الكود مع أي إطار عمل تعلم عميق حديث باستخدام الحلقات والروتينات الفرعية.

8.2.1. كتل VGG

كتلة البناء الأساسية لشبكات CNN هي سلسلة مما يلي: (1) طبقة تلافيفية مع حشوة للحفاظ على الدقة، (2) غير خطية مثل ReLU، (3) طبقة تجميع مثل max-pooling لتقليل الدقة.

تتمثل إحدى مشكلات هذا النهج في أن الدقة المكانية تتناقص بسرعة كبيرة. على وجه الخصوص، يفرض هذا حدًا صارمًا للطبقات التلافيفية $\log_2 d$ على الشبكة قبل استخدام جميع الأبعاد (d). على سبيل المثال، في حالة ImageNet، سيكون من المستحيل وجود أكثر من 8 طبقات تلافيفية بهذه الطريقة.

كانت الفكرة الرئيسية لـ Simonyan وZisserman (2014) هي استخدام تلافيف متعددة بين الاختزال downsampling عبر تجميع الحد الأقصى max-pooling في شكل كتلة. كانوا مهتمين في المقام الأول بما إذا كانت الشبكات العميقة أو الواسعة تعمل بشكل أفضل. على سبيل المثال، يلامس التطبيق المتتابع لتلافيفين 3×3 نفس وحدات البكسل 5×5 كما يفعل التفاف واحد. في نفس الوقت، يستخدم الأخير تقريبًا العديد من المعلمات ($25 \cdot c^2$) كما تفعل ثلاثة تلافيف ($3 \cdot 9 \cdot c^2$). في تحليل مفصل إلى حد ما، أظهروا أن الشبكات العميقة والضيقة تتفوق بشكل كبير على نظيراتها الضحلة. وضع هذا التعلم العميق في البحث عن شبكات أعمق من أي وقت مضى مع أكثر من 100 طبقة للتطبيقات النموذجية. أصبح تراص التلافيف 3×3 معيارًا ذهبيًا في الشبكات العميقة اللاحقة (قرار تصميم تمت إعادة النظر فيه مؤخرًا بواسطة Liu et al., 2022). وبالتالي، أصبحت عمليات التنفيذ السريعة للتلافيف الصغيرة عنصرًا أساسيًا في وحدات معالجة الرسومات (Lavin and Gray، 2016).

العودة إلى VGG: تتكون كتلة VGG من سلسلة من التلافيف sequence of convolutions مع 3×3 قنوات مع حشوة من 1 (الحفاظ على الارتفاع والعرض) متبوعة بطبقة تجميع حد أقصى بخطوة 2 (نصف ارتفاع وعرض بعد كل كتلة). في الكود أدناه، نحدد دالة تسمى `vgg_block` لتنفيذ كتلة VGG واحدة.

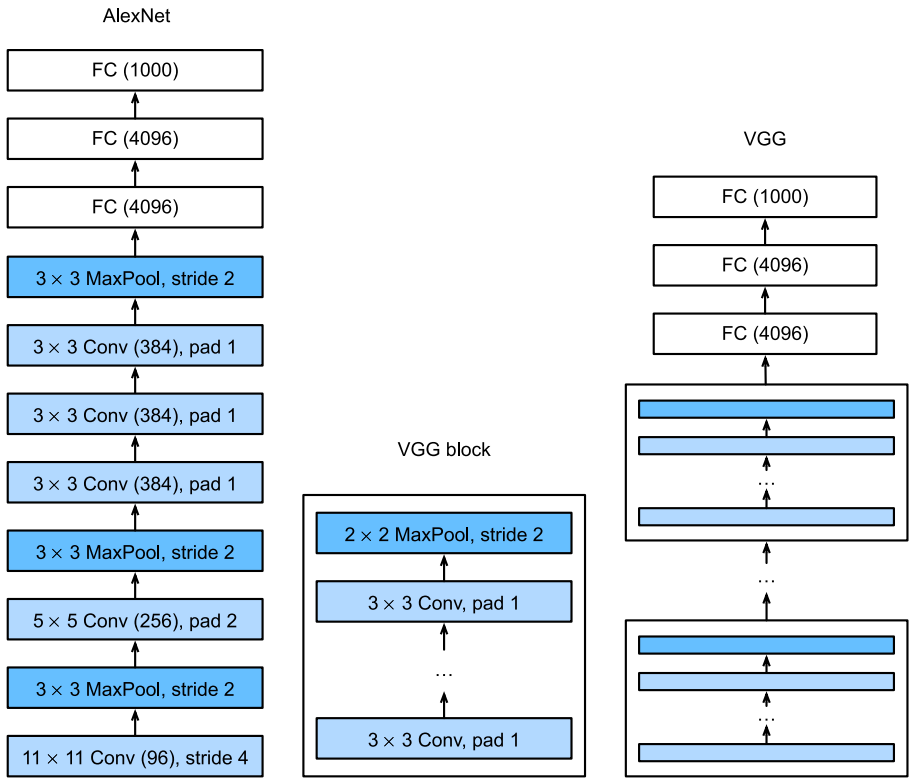
تأخذ الدالة أدناه وسيطتين arguments، تقابلان عدد الطبقات التلافيفية `num_convs` وعدد قنوات الإخراج `num_channels`.

```
import tensorflow as tf
from d2l import tensorflow as d2l


def vgg_block(num_convs, num_channels):
    blk = tf.keras.models.Sequential()
    for _ in range(num_convs):
        blk.add(
            tf.keras.layers.Conv2D(num_channels,
kernel_size=3,
                                   padding='same',
activation='relu'))
```

```
    blk.add(tf.keras.layers.MaxPool2D(pool_size=2,
strides=2))
    return blk
```

8.2.2. شبكة VGG

مثل AlexNet وLeNet، يمكن تقسيم شبكة VGG إلى جزأين: الأول يتكونفي الغالب من طبقات تلافيفية convolutional وتجميعية pooling والثاني يتكون من طبقات متصلة بالكامل مماثلة لتلك الموجودةفي AlexNet. يتمثل الاختلاف الرئيسيفي أن الطبقات التلافيفية يتم تجميعهافي تحويلات غير خطية تترك الأبعاد دون تغيير، متبوعة بخطوة تقليل الدقة resolution-reduction step، كما هو موضحفي الشكل 8.2.1.



الشكل. 8.2.1 من AlexNet إلى VGG. الفرق الرئيسي هو أن VGG يتكون من كتل من الطبقات،في حين أن طبقات AlexNet مصممة بشكل فردي.

يربط الجزء التلافيفي من الشبكة عدة كتل VGG من الشكل 8.2.1 (كما هو محددفي دالة `vgg_block`) على التوالي. هذا التجمع من الالتفافات هو نمط ظل دون تغيير تقريبًا خلال العقد الماضي، على الرغم من أن الاختيار المحدد للعمليات قد خضع لتعديلات كبيرة. يتكون

المتغير conv_arch من قائمة من المجموعات (واحدة لكل كتلة)، حيث تحتوي كل منها على قيمتين: عدد الطبقات التلافيفية وعدد قنوات المخرجات، وهما على وجه التحديد الوسيطات المطلوبة لاستدعاء دالة vgg_block. على هذا النحو، تحدد VGG عائلة من الشبكات وليس مجرد مظهر محدد. لبناء شبكة محددة، نقوم ببساطة بالتكرار عبر القوس arch لتكوين الكتل.

```
class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential()
        for (num_convs, num_channels) in arch:
            self.net.add(vgg_block(num_convs,
num_channels))
        self.net.add(
            tf.keras.models.Sequential([
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(4096,
activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(4096,
activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(num_classes)]))
```

تحتوي شبكة VGG الأصلية على 5 كتل تلافيفية، من بينها الكتل التلافيفية الأولى والثانية تحتوي على طبقة تلافيفية واحدة لكل منها، وتحتوي الثلاثة الأخيرة على طبقتين تلافيفيتين لكل منهما. تحتوي الكتلة الأولى على 64 قناة إخراج وتضاعف كل كتلة لاحقة عدد قنوات الإخراج، حتى يصل هذا الرقم إلى 512. نظرًا لأن هذه الشبكة تستخدم 8 طبقات تلافيفية و3 طبقات متصلة بالكامل، فإنها غالبًا ما تسمى VGG-11.

```
VGG(arch=((1, 64), (1, 128), (2, 256), (2, 512), (2,
512))).layer_summary(
    (1, 224, 224, 1))
```

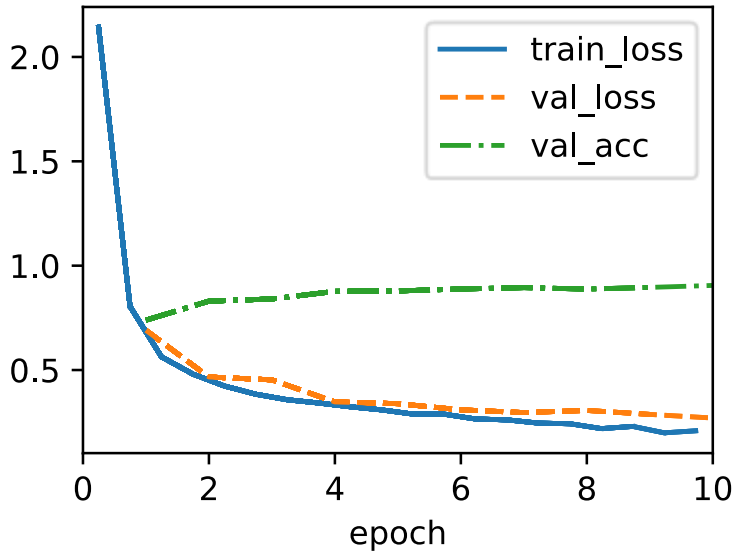
```
Sequential output shape:     (1, 112, 112, 64)
Sequential output shape:     (1, 56, 56, 128)
Sequential output shape:     (1, 28, 28, 256)
Sequential output shape:     (1, 14, 14, 512)
Sequential output shape:     (1, 7, 7, 512)
Sequential output shape:     (1, 10)
```

كما ترون، نقوم بتخفيض الارتفاع والعرض إلى النصف في كل كتلة، ونصل أخيرًا إلى ارتفاع وعرض 7 قبل تسطيح التمثيلات للمعالجة بواسطة الجزء المتصل بالكامل من الشبكة. وصف Simonyan وZisserman (2014) العديد من المتغيرات الأخرى لـ VGG. في الواقع، أصبح من المعتاد اقتراح مجموعات شبكات ذات موازنات سرعة ودقة مختلفة عند تقديم بُنية جديدة.

8.2.3. التدريب Training

نظرًا لأن VGG-11 أكثر تطلبًا من الناحية الحسابية من AlexNet، فإننا نبني شبكة مع عدد أقل من القنوات. هذا أكثر من كافٍ للتدريب على Fashion-MNIST. عملية التدريب النموذجية مماثلة لتلك الخاصة بـ AlexNet في القسم 8.1. لاحظ مرة أخرى التطابق الوثيق بين خطأ التحقق من الصحة وخطأ التدريب، مما يشير إلى كمية صغيرة فقط من الضبط الزائد.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(224,
224))
with d2l.try_gpu():
    model = VGG(arch=((1, 16), (1, 32), (2, 64), (2,
128), (2, 128)), lr=0.01)
    trainer.fit(model, data)
```



8.2.4. الملخص

قد يجادل المرء بأن VGG هي أول شبكة عصبية تلافيفية حديثة حقًا. بينما قدمت AlexNet العديد من المكونات التي تجعل التعلم العميق فعالًا على نطاق واسع، يمكن القول إن VGG هي التي قدمت الخصائص الرئيسية مثل كتل التلافيف المتعددة وتفضيل الشبكات العميقة

والضيقة. إنها أيضًا الشبكة الأولى التي هي في الواقع عائلة كاملة من النماذج ذات المعلمات المتشابهة، مما يمنح الممارس موازنة كبيرة بين التعقيد والسرعة. هذا أيضًا هو المكان الذي تتألق فيه أطر التعلم العميق الحديثة. لم يعد من الضروري إنشاء ملفات تكوين XML لتحديد الشبكة، ولكن بدلاً من ذلك، لتجميع الشبكات المذكورة من خلال كود بايثون البسيط.

أظهر ParNet مؤخرًا (Goyal et al.، 2021) أنه من الممكن تحقيق أداء تنافسي باستخدام بُنية ضحلة أكثر من خلال عدد كبير من الحسابات المتوازية. هذا تطور مثير وهناك أمل في أن يؤثر على تصاميم المعمارية في المستقبل. ومع ذلك، بالنسبة لبقية الفصل، سنتبع مسار التقدم العلمي على مدى العقد الماضي.

8.2.5. التمارين

1. مقارنةً بـ AlexNet، يُعد VGG أبطأ بكثير من حيث الحساب، كما أنه يحتاج إلى المزيد من ذاكرة GPU.
   1. قارن عدد المعلمات المطلوبة لـ AlexNet وVGG.
   2. قارن عدد عمليات الفاصلة العائمة المستخدمة في الطبقات التلافيفية والطبقات المتصلة بالكامل.
   3. كيف يمكنك تقليل التكلفة الحسابية التي أنشأتها الطبقات المتصلة بالكامل؟
4. عند عرض الأبعاد المرتبطة بالطبقات المختلفة للشبكة، لا نرى سوى المعلومات المرتبطة بـ 8 كتل (بالإضافة إلى بعض التحويلات المساعدة)، على الرغم من أن الشبكة بها 11 طبقة. أين ذهبت الطبقات الثلاث المتبقية؟
5. استخدم الجدول 1 في مقالة VGG، (Simonyan and Zisserman,2014) لإنشاء نماذج شائعة أخرى ، مثل VGG-16 أو VGG-19.
6. يعد اختزال Upsampling الدقة في Fashion-MNIST بعامل من 8 إلى 28×28 أبعاد مضيعة للغاية. حاول تعديل بُنية الشبكة وتحويل الدقة، على سبيل المثال، إلى 56 أو 84 بعدًا لإدخالها بدلاً من ذلك. هل يمكنك القيام بذلك دون التقليل من دقة الشبكة؟ ضع في اعتبارك ورقة VGG، (Simonyan and Zisserman,2014) للحصول على أفكار حول إضافة المزيد من العناصر اللاخطية قبل الاختزال downsampling.

8.3 الشبكة في الشبكة Network in Network (NiN)

تشترك LeNet وAlexNet وVGG جميعًا في نمط تصميم مشترك: استخراج الميزات extract features التي تستغل البُنية المكانية spatial structure عبر سلسلة من التلافيف وطبقات التجميع ومعالجة التمثيلات بعد ذلك عبر طبقات متصلة بالكامل. تكمن التحسينات

التي أدخلت على LeNet بواسطة AlexNet وVGG بشكل أساسيفي كيفية توسيع هذه الشبكات اللاحقة وتعميق هاتين الوحدتين.

يطرح هذا التصميم تحديين رئيسيين. أولاً، تستهلك الطبقات المتصلة بالكاملفي نهاية الهيكل عددًا هائلاً من المعلمات. على سبيل المثال، حتى نموذج بسيط مثل VGG-11 يتطلب مصفوفة ضخمة 4096 × 25088، تشغل ما يقرب من 400 ميجابايت من ذاكرة الوصول العشوائي بدقة واحدة (FP32). هذا عائق كبير أمام الحساب، لا سيما على الأجهزة المحمولة والمضمنة. بعد كل شيء، حتى الهواتف المحمولة المتطورة لا تحتوي على أكثر من 8 جيجابايت من ذاكرة الوصول العشوائي.في الوقت الذي تم فيه اختراع VGG، كان هذا ترتيبًا أقل من حيث الحجم (كان لدى ايفون 4S 512 ميجا بايت). على هذا النحو، كان من الصعب تبرير إنفاق غالبية الذاكرة على مصنف الصور.

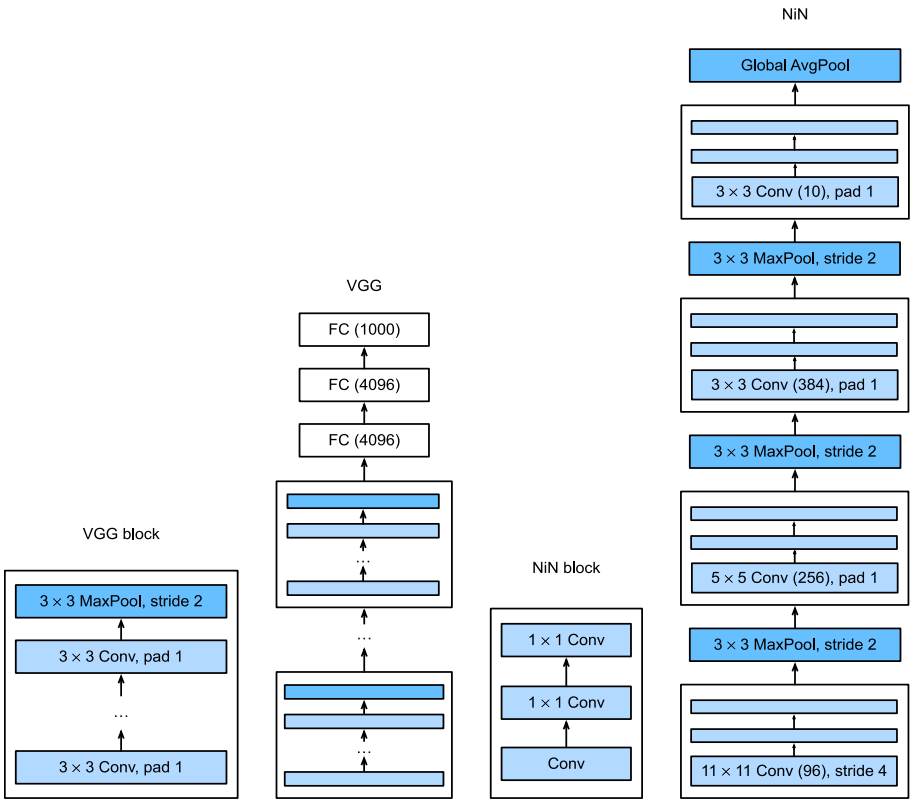
ثانيًا، من المستحيل أيضًا إضافة طبقات متصلة بالكاملفي وقت سابقفي الشبكة لزيادة درجة اللاخطية: سيؤدي القيام بذلك إلى تدمير البنية المكانية وقد يتطلب المزيد من الذاكرة.

تقدم كتل الشبكةفي الشبكة network in network (NiN)، (Lin et al.,2013) بديلاً قادرًا على حل كلتا المشكلتينفي إستراتيجية واحدة بسيطة. تم اقتراحها بناءً على نظرة ثاقبة بسيطة للغاية: (1) استخدام التلافيف 1 × 1 لإضافة عناصر غير خطية محلية عبر عمليات تنشيط القناة و(2) استخدام متوسط التجميع العالمي global average pooling للتكامل عبر جميع المواقعفي طبقة التمثيل الأخيرة. لاحظ أن متوسط التجميع العالمي لن يكون فعالاً، لولا اللاخطية المضافة. دعونا نتعمقفي هذا بالتفصيل.

8.3.1. كتل NiN

راجع القسم 7.4.3. الذي ناقشنا فيه أن مدخلات ومخرجات الطبقات التلافيفية تتكون من موتر رباعي الأبعاد مع محاور مقابلة للمثال والقناة والارتفاع والعرض. تذكر أيضًا أن مدخلات ومخرجات الطبقات المتصلة بالكامل تكون عادةً موترات ثنائية الأبعاد تتوافق مع المثال والميزة. الفكرة وراء NiN هي تطبيق طبقة متصلة بالكاملفي كل موقع بكسل (لكل ارتفاع وعرض). يمكن اعتبار الالتفاف الناتج على أنه طبقة متصلة بالكامل تعمل بشكل مستقل على كل موقع بكسل.

يوضح الشكل 8.3.1 الاختلافات الهيكلية الرئيسية بين VGG وNiN وكتلهما. لاحظ كلاً من الاختلاففي كتل NiN (الالتفاف الأولي يتبعه التلافيف 1 × 1، بينما يحتفظ VGG بالتلافيف 3 × 3) وفي النهاية حيث لم نعد بحاجة إلى طبقة عملاقة متصلة بالكامل.



الشكل 8.3.1 مقارنة معماريات VGG وNiN وكتلها.

```
import tensorflow as tf
from d2l import tensorflow as d2l


def nin_block(out_channels, kernel_size, strides,
padding):
    return tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(out_channels, kernel_size,
strides=strides,
                           padding=padding),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(out_channels, 1),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(out_channels, 1),
    tf.keras.layers.Activation('relu')])
```

8.3.2. نموذج NiN

يستخدم NiN نفس أحجام الالتفاف الأولية مثل AlexNet (تم اقتراحه بعد ذلك بوقت قصير). أحجام النواة هي 11 × 11، 5 × 5 و 3 × 3، على التوالي، وعدد قنوات الإخراج يتطابق مع تلك الموجودةفي AlexNet. يتبع كل كتلة NiN طبقة تجميع حد أقصى بخطوة 2 وشكل نافذة 3 × 3.

يتمثل الاختلاف المهم الثاني بين NiN وAlexNet وVGGفي أن NiN يتجنب الطبقات المتصلة تمامًا معاً. بدلاً من ذلك، يستخدم NiN كتلة NiN مع عدد من قنوات الإخراج مساوٍ لعدد فئات التسميات label classes، متبوعة بطبقة تجميع متوسط عالمي، مما ينتج عنه متجه من السجلات vector of logits. يقلل هذا التصميم بشكل كبير من عدد معلمات النموذج المطلوبة، وإن كان ذلك على حساب زيادة محتملةفي وقت التدريب.

```
class NiN(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            nin_block(96, kernel_size=11, strides=4,
padding='valid'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            nin_block(256, kernel_size=5, strides=1,
padding='same'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            nin_block(384, kernel_size=3, strides=1,
padding='same'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2),
            tf.keras.layers.Dropout(0.5),
            nin_block(num_classes, kernel_size=3,
strides=1, padding='same'),
            tf.keras.layers.GlobalAvgPool2D(),
            tf.keras.layers.Flatten()])
```

نقوم بإنشاء مثال بيانات لمعرفة شكل الإخراج لكل كتلة.

```
model = NiN()
X = tf.random.normal((1, 224, 224, 1))
for layer in model.net.layers:
    X = layer(X)
```

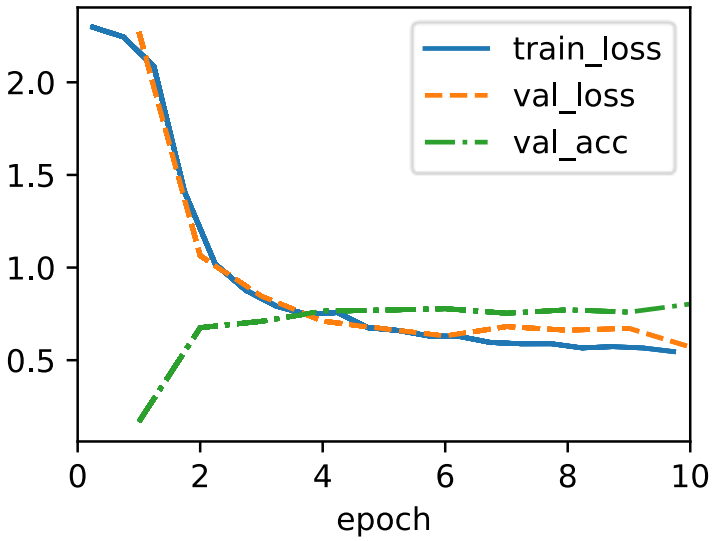
```
    print(layer.__class__.__name__,'output shape:\t',
X.shape)
```

```
Sequential output shape:     (1, 54, 54, 96)
MaxPooling2D output shape:   (1, 26, 26, 96)
Sequential output shape:     (1, 26, 26, 256)
MaxPooling2D output shape:   (1, 12, 12, 256)
Sequential output shape:     (1, 12, 12, 384)
MaxPooling2D output shape:   (1, 5, 5, 384)
Dropout output shape:        (1, 5, 5, 384)
Sequential output shape:     (1, 5, 5, 10)
GlobalAveragePooling2D output shape:         (1, 10)
Flatten output shape:        (1, 10)
```

8.3.3. التدريب Training

كما كان من قبل، نستخدم Fashion-MNIST لتدريب النموذج باستخدام نفس المُحسِّن الذي استخدمناه لـ AlexNet وVGG.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(224,
224))
with d2l.try_gpu():
    model = NiN(lr=0.05)
    trainer.fit(model, data)
```



8.3.4. الملخص

يحتوي NiN على معلمات أقل بشكل كبير من AlexNet وVGG. ينبع هذا في المقام الأول من حقيقة أنه لا يحتاج إلى طبقات عملاقة متصلة بالكامل. بدلاً من ذلك، يستخدم متوسط التجميع العالمي للتجميع عبر جميع مواقع الصور بعد المرحلة الأخيرة من جسم الشبكة. هذا يغني عن الحاجة إلى عمليات تخفيض مكلفة (مكتسبة) ويستبدلها بمتوسط بسيط. ما كان مفاجئًا في ذلك الوقت هو حقيقة أن هذه العملية المتوسطة لم تضر بالدقة. لاحظ أن المتوسط عبر تمثيل منخفض الدقة (مع العديد من القنوات) يضيف أيضًا إلى مقدار ثبات الترجمة الذي يمكن للشبكة التعامل معه.

يساعد اختيار عدد أقل من التلافيفات ذات قنوات العريضة واستبدالها بالتلافيف 1×1 في البحث عن عدد أقل من المعلمات. إنه يوفر قدرًا كبيرًا من اللاخطية عبر القنوات داخل أي موقع معين. أثر كل من التلافيف 1×1 والتجميع العالمي بشكل كبير على تصميمات CNN اللاحقة.

8.3.5. التمارين

1. لماذا توجد طبقتان تلافيفيتان لكل كتلة NiN؟ زد عددهم إلى ثلاثة. قلل عددهم إلى واحد. ما هي التغييرات؟
2. ما الذي يتغير إذا استبدلت التلافيف 1×1 بالتلافيف 3×3؟
3. ماذا يحدث إذا استبدلت متوسط التجميع العالمي بطبقة متصلة بالكامل (السرعة، الدقة، عدد المعلمات)؟
4. احسب استخدام الموارد من أجل NiN.
    1. ما هو عدد المعلمات؟
    2. ما هو مقدار الحساب؟
    3. ما هو مقدار الذاكرة المطلوبة أثناء التدريب؟
    4. ما هو مقدار الذاكرة المطلوبة أثناء التنبؤ؟
5. ما هي المشاكل المحتملة مع تقليص التمثيل $5 \times 5 \times 384$ إلى التمثيل $5 \times 5 \times 10$ في خطوة واحدة؟
6. استخدم قرارات التصميم الهيكلي في VGG التي أدت إلى VGG-11 وVGG-16 وVGG-19 لتصميم عائلة من الشبكات الشبيهة بـ NiN.

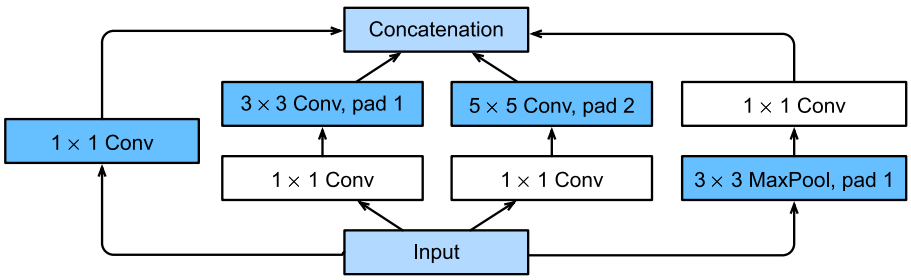
8.4 شبكات متعددة الفروع (GoogLeNet) Multi-Branch Networks

في عام 2014، فازت GoogLeNet بتحدي ImageNet، (2015,Szegedy et al.)، باستخدام هيكل يجمع بين نقاط القوةفي NiN، (2013,Lin et al.)، والكتل المتكررة repeated blocks (Simonyan and Zisserman، 2014)، ومجموعة من قنوات الالتفاف. يمكن القول أيضًا إنها أول شبكة تظهر تمييزًا واضحًا بين الجذع stem (استيعاب البيانات data ingest) والجسم body (معالجة البيانات) والرأس head (التنبؤ)في شبكة CNN. استمر نمط التصميم هذا منذ ذلك الحينفي تصميم الشبكات العميقة: يتم إعطاء الجذع من خلال التلافيف 2ـ3 الأولى التي تعمل على الصورة. يستخرجون ميزات منخفضة المستوى من الصور الأساسية. يتبع ذلك مجموعة من الكتل التلافيفية. أخيرًا، يقوم الرأس بتعيين الميزات التي تم الحصول عليها حتى الآن حسب مشكلة التصنيف أو التجزئة أو الكشف أو التتبع المطلوبة.

كانت المساهمة الرئيسيةفي GoogLeNet هي تصميم هيكل الشبكة. لقد حلت مشكلة اختيار نواة الالتفاف بطريقة بارعة. بينما حاولت أعمال أخرى تحديد الالتفاف الأفضل، بدءًا من 1×1 إلى 11×11، إلا أنها قامت ببساطة بجمع التلافيف متعددة الفروع المتسلسلة. فيما يلي نقدم نسخة مبسطة قليلاً من GoogLeNet: تضمن التصميم الأصلي عددًا من الحيل لتحقيق الاستقرارفي التدريب من خلال دوال الخسارة المتوسطة، المطبقة على طبقات متعددة من الشبكة. لم تعد ضرورية بسبب توفر خوارزميات تدريب محسّنة.

8.4.1. كتل الاستهلال Inception Blocks

تسمى الكتلة التلافيفية الأساسيةفي GoogLeNet بكتل الاستهلال Inception Blocks، وتنبع من الميم "نحن بحاجة إلى التعمق أكثر we need to go deeper"في فيلم Inception.



الشكل 8.4.1 هيكل كتلة الاستهلال.

كما هو موضحفي الشكل 8.4.1، تتكون كتلة الاستهلال inception block من أربعة فروع متوازية. تستخدم الفروع الثلاثة الأولى طبقات تلافيفية ذات أحجام نافذة 1×1، و5×5

لاستخراج المعلومات من الأحجام المكانية المختلفة. يضيف الفرعان الأوسطان أيضًا التفافًا للمدخلات لتقليل عدد القنوات، مما يقلل من تعقيد النموذج. يستخدم الفرع الرابع طبقة تجميع الحد الاقصى، متبوعة بطبقة تلافيفية لتغيير عدد القنوات. تستخدم الفروع الأربعة حشوة مناسبة لإعطاء الإدخال والإخراج نفس الارتفاع والعرض. أخيرًا، يتم ربط النواتج الموجودة على طول كل فرع على طول بُعد القناة وتشكل ناتج الكتلة. المعلمات الفائقة المضبوطة بشكل شائع لكتلة البداية هي عدد قنوات الإخراج لكل طبقة، أي كيفية تخصيص السعة بين التلافيف ذات الحجم المختلف.

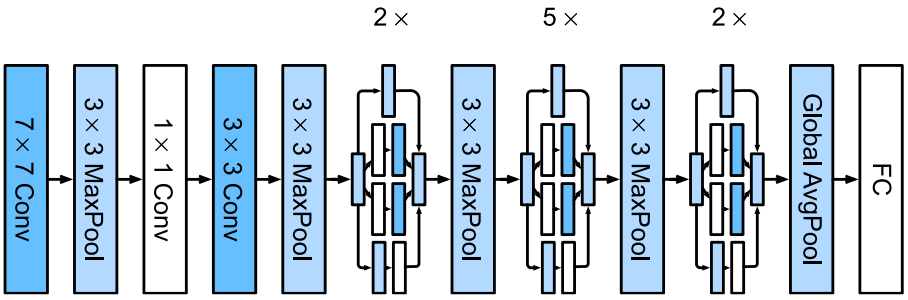
```
import tensorflow as tf
from d2l import tensorflow as d2l


class Inception(tf.keras.Model):
    # `c1`--`c4` are the number of output channels for
each branch
    def __init__(self, c1, c2, c3, c4):
        super().__init__()
        self.b1_1 = tf.keras.layers.Conv2D(c1, 1,
activation='relu')
        self.b2_1 = tf.keras.layers.Conv2D(c2[0], 1,
activation='relu')
        self.b2_2 = tf.keras.layers.Conv2D(c2[1], 3,
padding='same',

activation='relu')
        self.b3_1 = tf.keras.layers.Conv2D(c3[0], 1,
activation='relu')
        self.b3_2 = tf.keras.layers.Conv2D(c3[1], 5,
padding='same',

activation='relu')
        self.b4_1 = tf.keras.layers.MaxPool2D(3, 1,
padding='same')
        self.b4_2 = tf.keras.layers.Conv2D(c4, 1,
activation='relu')

    def call(self, x):
        b1 = self.b1_1(x)
        b2 = self.b2_2(self.b2_1(x))
        b3 = self.b3_2(self.b3_1(x))
```

```
        b4 = self.b4_2(self.b4_1(x))
        return tf.keras.layers.Concatenate()([b1, b2,
b3, b4])
```

للحصول على بعض الحدس عن سبب عمل هذه الشبكة بشكل جيد، ضعفي اعتبارك مجموعة الفلاتر. يستكشفون الصورةفي مجموعة متنوعة من أحجام الفلاتر. هذا يعني أنه يمكن التعرف على التفاصيل ذات النطاقات المختلفة بكفاءة بواسطة فلاتر ذات أحجام مختلفة.في الوقت نفسه، يمكننا تخصيص كميات مختلفة من المعلمات لفلاتر مختلفة.

8.4.2. نموذج GoogLeNet

كما هو مبينفي الشكل 8.4.2، تستخدم GoogLeNet مكدس stack من إجمالي 9 كتل استهلال، مرتبةفي 3 مجموعات مع أقصى تجميع بينهما، ومتوسط التجميع العالميفي رأسها لتوليد تقديراتها. تجميع الحد الأقصى بين كتل البداية يقلل من الأبعاد.في جذعها، تشبه الوحدة النمطية الأولى AlexNet وLeNet.



الشكل 8.4.2 بُنية GoogLeNet.

يمكننا الآن تنفيذ GoogLeNet قطعة قطعة. لنبدأ بالجذع. تستخدم الوحدة الأولى طبقة تلافيفية 7 × 7 ذات 64 قناة.

```
class GoogleNet(d2l.Classifier):
    def b1(self):
        return tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(64, 7, strides=2,
padding='same',
                                   activation='relu'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2,
                                      padding='same')])
```

تستخدم الوحدة الثانية طبقتين تلافيفيتين: الأولى، طبقة تلافيفية 1 × 1 ذات 64 قناة، تليها طبقة تلافيفية 3 × 3 تضاعف عدد القنوات ثلاث مرات. هذا يتوافق مع الفرع الثانيفي كتلة الاستهلال ويختتم تصميم الجسم.في هذه المرحلة لدينا 192قناة.

```
@d2l.add_to_class(GoogleNet)
def b2(self):
    return tf.keras.Sequential([
        tf.keras.layers.Conv2D(64, 1,
activation='relu'),
        tf.keras.layers.Conv2D(192, 3, padding='same',
activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=3,
strides=2, padding='same')])
```

تقوم الوحدة الثالثة بتوصيل كتلتين كاملتين من كتل الاستهلال على التوالي. عدد قنوات الإخراج للكتلة الأولى هو 256 = 32 + 32 + 128 + 64. هذا يرقى إلى نسبة عدد قنوات الإخراج بين الفروع الأربعة لـ 1 :1 :4 :2. لتحقيق ذلك، نقوم أولاً بتقليل أبعاد الإدخال بمقدار $\frac{1}{2}$ وبواسطة $\frac{1}{12}$ في الفرعين الثاني والثالث على التوالي للوصول إلى القنوات 192/12 = 16 والقنوات 192/12 = 16 على التوالي.

يتم زيادة عدد قنوات الإخراج لكتلة الاستهلال الثانية إلى 480 = 64 + 96 + 192 + 128 محققاً نسبة 2 :3 :6 :4 = 64 :96 :192 :128. كما كان من قبل، نحتاج إلى تقليل عدد الأبعاد الوسيطةفي القناة الثانية والثالثة. مقياس $\frac{1}{2}$ و $\frac{1}{8}$ على التوالي يكفي، والعائد القنوات 128 و 32 على التوالي. يتم التقاط هذا من خلال وسيطات المنشئ Inception block التالية.

```
@d2l.add_to_class(GoogleNet)
def b3(self):
    return tf.keras.models.Sequential([
        Inception(64, (96, 128), (16, 32), 32),
        Inception(128, (128, 192), (32, 96), 64),
        tf.keras.layers.MaxPool2D(pool_size=3,
strides=2, padding='same')])
```

الوحدة الرابعة أكثر تعقيدًا. يربط خمس كتل بدايةفي سلسلة، ولديهم 192 + 208 + 48 + 64 = 512، 160 + 224 + 64 + 64 = 512، 128 + 256 + 64 + 64 = 512، و 256 + 320 + 128 + 128 = 832112 + 288 + 64 + 64 = 528 قنوات الإخراج، على التوالي. عدد القنوات المخصصة لهذه الفروع مماثل لتلك الموجودةفي الوحدة الثالثة: الفرع الثاني مع الطبقة التلافيفية 3 × 3 يخرج أكبر عدد من القنوات، يليه الفرع الأول مع الطبقة التلافيفية 1 × 1 فقط، والفرع الثالث مع الطبقة التلافيفية 5 × 5 ، والفرع الرابع مع طبقة

التجميع القصوى 3 × 3. سيقلل الفرعان الثاني والثالث أولاً عدد القنوات وفقًا للنسبة. تختلف هذه النسب اختلافًا طفيفًا في كتل التأسيس المختلفة.

```
@d2l.add_to_class(GoogleNet)
def b4(self):
    return tf.keras.Sequential([
        Inception(192, (96, 208), (16, 48), 64),
        Inception(160, (112, 224), (24, 64), 64),
        Inception(128, (128, 256), (24, 64), 64),
        Inception(112, (144, 288), (32, 64), 64),
        Inception(256, (160, 320), (32, 128), 128),
        tf.keras.layers.MaxPool2D(pool_size=3,
strides=2, padding='same')])
```

تحتوي الوحدة الخامسة على كتلتين ابتدائيتين مع $256 + 320 + 128 + 128 = 832$ و $384 + 384 + 128 + 128 = 1024$ قنوات إخراج. عدد القنوات المخصصة لكل فرع هو نفسه الموجود في الوحدتين الثالثة والرابعة، ولكنه يختلف في القيم المحددة. وتجدر الإشارة إلى أن الكتلة الخامسة تتبعها طبقة الإخراج. تستخدم هذه الكتلة طبقة تجميع المتوسط العالمية لتغيير ارتفاع وعرض كل قناة إلى 1، تمامًا كما هو الحال في NiN. أخيرًا، نقوم بتحويل الإخراج إلى مصفوفة ثنائية الأبعاد متبوعة بطبقة متصلة بالكامل يكون عدد مخرجاتها هو عدد فئات التسمية.

```
@d2l.add_to_class(GoogleNet)
def b5(self):
    return tf.keras.Sequential([
        Inception(256, (160, 320), (32, 128), 128),
        Inception(384, (192, 384), (48, 128), 128),
        tf.keras.layers.GlobalAvgPool2D(),
        tf.keras.layers.Flatten()])
```

الآن بعد أن حددنا جميع الكتل من `b1` إلى `b5`، فإن الأمر يتعلق فقط بتجميعهم جميعًا في شبكة كاملة.

```
@d2l.add_to_class(GoogleNet)
def __init__(self, lr=0.1, num_classes=10):
    super(GoogleNet, self).__init__()
    self.save_hyperparameters()
    self.net = tf.keras.Sequential([
        self.b1(), self.b2(), self.b3(), self.b4(),
self.b5(),
        tf.keras.layers.Dense(num_classes)])
```

يعتبر نموذج GoogLeNet معقدًا من الناحية الحسابية. لاحظ العدد الكبير من المعلمات الفائقة التعسفية نسبيًا من حيث عدد القنوات المختارة، وعدد الكتل قبل تقليل الأبعاد، والتقسيم النسبي للسعة عبر القنوات، وما إلى ذلك. لم تكن الأدوات الآلية لتعريف الشبكة أو استكشاف التصميم متاحة بعد. على سبيل المثال، نحن الآن نعتبر أنه من المسلم به أن إطار عمل التعلم العميق المختص قادر على استنتاج أبعاد موترات الإدخال تلقائيًا.في ذلك الوقت، كان يتعين على المجرب تحديد العديد من هذه التكوينات صراحة، مما يؤديفي كثير من الأحيان إلى إبطاء التجريب النشط. علاوة على ذلك، كانت الأدوات اللازمة للاستكشاف التلقائي لا تزالفي حالة تغير مستمر، وكانت التجارب الأولية تصل إلى حد كبير إلى استكشاف القوة العمياء brute force exploration المكلفة، والخوارزميات الجينية genetic algorithms، والاستراتيجيات المماثلة.

في الوقت الحالي، التعديل الوحيد الذي سنقوم به هو تقليل ارتفاع المدخلات وعرضها من 224 إلى 96 للحصول على وقت تدريب معقول على Fashion-MNIST. هذا يبسط الحساب. دعونا نلقي نظرة على التغييراتفي شكل الإخراج بين الوحدات المختلفة.

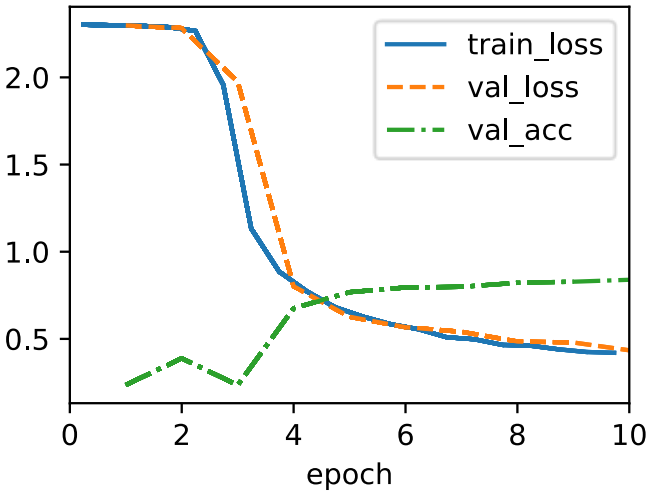
```
model = GoogleNet().layer_summary((1, 96, 96, 1))
```

```
Sequential output shape:     (1, 24, 24, 64)
Sequential output shape:     (1, 12, 12, 192)
Sequential output shape:     (1, 6, 6, 480)
Sequential output shape:     (1, 3, 3, 832)
Sequential output shape:     (1, 1024)
Dense output shape:  (1, 10)
```

8.4.3. التدريب Training

كمافي السابق، نقوم بتدريب نموذجنا باستخدام مجموعة بيانات Fashion-MNIST. نقوم بتحويله إلى دقة 96 × 96 بكسل قبل استدعاء إجراء التدريب.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
with d2l.try_gpu():
    model = GoogleNet(lr=0.01)
    trainer.fit(model, data)
```



8.4.4. المناقشة

الميزة الرئيسية لـ GoogLeNet هي أنها أرخص في الواقع للحوسبة من سابقاتها مع توفير دقة محسنة في نفس الوقت. يمثل هذا بداية تصميم شبكة مدروس بشكل أكبر والذي يتم فيه استبدال تكلفة تقييم الشبكة مع تقليل الأخطاء. كما يمثل بداية التجريب على مستوى الكتلة باستخدام معلمات فائقة لتصميم الشبكة، على الرغم من أنها كانت يدوية بالكامل في ذلك الوقت. سنعيد النظر في هذا الموضوع في القسم 8.8 عند مناقشة استراتيجيات استكشاف بُنية الشبكة.

خلال الأقسام التالية، سنواجه عددًا من خيارات التصميم (على سبيل المثال، تسوية الدُفعات batch normalization، والاتصالات المتبقية residual connections، وتجميع القنوات channel grouping) التي تسمح لنا بتحسين الشبكات بشكل كبير. في الوقت الحالي، يمكنك أن تفخر بتنفيذ ما يمكن القول إنه أول شبكة CNN حديثة حقًا.

8.4.5. التمارين

1. كانت GoogLeNet ناجحة للغاية لدرجة أنها مرت بعدد من التكرارات. هناك العديد من التكرارات لـ GoogLeNet التي تعمل على تحسين السرعة والدقة بشكل تدريجي. حاول تنفيذ وتشغيل بعضها. وهي تشمل ما يلي:
2. أضف طبقة تسوية دفعية (Ioffe and Szegedy، 2015)، كما هو موضح لاحقًا في القسم 8.5.
3. قم بإجراء تعديلات على كتلة الاستهلال (العرض والاختيار وترتيب التلافيف)، كما هو موضح في (Szegedy et al.، 2016).

4. استخدم تجانس التسمية label smoothing لتسوية النموذج، كما هو موضح في (Szegedy et al.، 2016).
5. قم بإجراء مزيد من التعديلات على مجموعة Inception عن طريق إضافة اتصال متبقي residual connection (Szegedy et al.، 2017)، كما هو موضح لاحقًا في القسم 8.6.4.
6. ما هو الحد الأدنى لحجم الصورة لكي يعمل GoogLeNet؟
7. هل يمكنك تصميم متغير من GoogLeNet يعمل على دقة 28×28 بكسل الأصلية لـ Fashion-MNIST؟ كيف ستحتاج إلى تغيير الجذع والجسم ورأس الشبكة، إذا كان هناك أي شيء على الإطلاق؟
8. قارن أحجام معلمات النموذج لـ AlexNet وVGG وNiN وGoogLeNet. كيف تقلل معماريتا الشبكة الأخيرتان بشكل كبير من حجم معلمة النموذج؟
9. قارن مقدار الحساب المطلوب في GoogLeNet وAlexNet. كيف يؤثر ذلك على تصميم شريحة التسريع accelerator chip، على سبيل المثال، من حيث حجم الذاكرة وعرض النطاق الترددي للذاكرة memory bandwidth وحجم ذاكرة التخزين cache size المؤقت ومقدار الحساب وفائدة العمليات المتخصصة؟

8.5 التسوية بالدفعات Batch Normalization

تدريب الشبكات العصبية العميقة أمر صعب. قد يكون جعلهم يتقاربون في فترة زمنية معقولة أمرًا صعبًا. في هذا القسم، نصف التسوية بالدفعات Batch Normalization، وهي تقنية شائعة وفعالة تعمل باستمرار على تسريع تقارب الشبكات العميقة (Ioffe and Szegedy، 2015). جنبًا إلى جنب مع الكتل المتبقية — التي تمت تغطيتها لاحقًا في القسم 8.6 — أتاح التسوية بالدفعات للممارسين تدريب الشبكات بشكل روتيني مع أكثر من 100 طبقة. تكمن فائدة ثانوية (مصادفة) للتسوية بالدُفعات في تنظيمها المتأصل.

8.5.1. تدريب الشبكات العميقة Training Deep Networks

عند التعامل مع البيانات، فإننا غالبًا ما نعالجها مسبقًا قبل التدريب. غالبًا ما تُحدث الخيارات المتعلقة بالمعالجة المسبقة للبيانات فرقًا كبيرًا في النتائج النهائية. تذكر تطبيقنا لـ MLPs للتنبؤ بأسعار المنازل (القسم 5.7). كانت خطوتنا الأولى عند العمل مع البيانات الحقيقية هي توحيد standardize ميزات الإدخال لدينا بحيث يكون لها متوسط صفر $\boldsymbol{\mu} = 0$ وتباين واحد $\boldsymbol{\Sigma} = \mathbf{1}$ عبر العديد من المشاهدات observations (Friedman، 1987). كحد أدنى، كثيرًا ما يقوم المرء بإعادة قياسه بحيث يكون القطر هو الوحدة unity، أي $\Sigma_{ii} = 1$. هناك إستراتيجية أخرى تتمثل في إعادة قياس المتجهات إلى طول الوحدة unit length، وربما صفر متوسط لكل

ملاحظة. يمكن أن يعمل هذا بشكل جيد، على سبيل المثال، لبيانات المستشعر المكاني. تعد تقنيات المعالجة المسبقة هذه وغيرها الكثير مفيدة للحفاظ على التحكمفي مشكلة التقدير بشكل جيد. انظر على سبيل المثال، المقالات التي كتبها Guyon et al. (2008) لمراجعة اختيار الميزات وتقنيات الاستخراج. توحيد المتجهات Standardizing vectors له أيضًا تأثير جانبي لطيف يتمثلفي تقييد دالة التعقيد للدوال التي تعمل على أساسها. على سبيل المثال، الحد المشهور بهامش الشعاع radius-margin bound (Vapnik، 1995)في آلات ناقلات الدعم support vector machines وPerceptron Convergence Theorem، (1962,Novikoff) يعتمدان على مدخلات ذات معيار محدود bounded norm.

حدسيًا، يلعب هذا التوحيد standardization بشكل جيد مع مُحسِّنينا our optimizers لأنه يضع المعلمات مسبقًا على نطاق مماثل. على هذا النحو، من الطبيعي أن نتساءل عما إذا كانت خطوة التسوية normalization المقابلة داخل شبكة عميقة قد لا تكون مفيدة.في حين أن هذا ليس السبب تمامًا الذي أدى إلى اختراع التسوية بالدُفعات (Ioffe and Szegedy، 2015)، إلا أنه طريقة مفيدة لفهمه وتسوية الطبقات layer normalization (Ba et al.، 2016)في إطار موحد.

ثانيًا، بالنسبة إلى MLP أو CNN النموذجي، أثناء تدريبنا، قد تأخذ المتغيراتفي الطبقات المتوسطة (على سبيل المثال، مخرجات التحويل الأفينيفي MLP) قيمًا بأحجام متفاوتة على نطاق واسع: كلاهما على طول الطبقات من الإدخال إلى الإخراج، عبر الوحداتفي نفس الطبقة، وبمرور الوقت بسبب تحديثاتنا لمعلمات النموذج. افترض مخترعو التسوية بالدفعات بشكل غير رسمي أن هذا الانجراف driftفي توزيع مثل هذه المتغيرات يمكن أن يعيق تقارب الشبكة. حدسيًا، قد نخمن أنه إذا كانت إحدى الطبقات بها عمليات تنشيط متغيرة تزيد 100 مرة عن تلك الموجودةفي طبقة أخرى، فقد يتطلب ذلك تعديلات تعويضيةفي معدلات التعلم. تهدف المحللات التكيفية Adaptive solvers مثلAdaGrad، (2011,Duchi et al.) أو Adam، (2014,Kingma and Ba) أو Yogi، (2018,Zaheer et al.) أو Distributed Shampoo، (2020,Anil et al.) إلى معالجة هذا الأمر من وجهة نظر التحسين، على سبيل المثال، عن طريق إضافة جوانب من أساليب الدرجة الثانية. البديل هو منع حدوث المشكلة، ببساطة عن طريق التسوية التكيفية adaptive normalization.

ثالثًا، الشبكات الأعمق معقدة وتميل إلى أن تكون أكثر قدرة على الضبط الزائد overfitting بسهولة. هذا يعني أن التنظيم regularization يصبح أكثر أهمية. تقنية شائعة للتنظيم هي حقن الضوضاء noise injection. كان هذا معروفًا لفترة طويلة، على سبيل المثال، فيما يتعلق بحقن الضوضاء للمدخلات (Bishop، 1995). كما أنه يشكل أساس التسرب dropoutفي القسم

5.6. كما اتضح، مصادفة تمامًا، فإن تسوية الدُفعات ينقل جميع الفوائد الثلاثة: المعالجة المسبقة preprocessing، والاستقرار العددي numerical stability، والتنظيم regularization.

يتم تطبيق تسوية الدُفعات على الطبقات الفردية، أو اختياريًا، لكل منها:في كل تكرار تدريبي، نقوم أولاً بتسوية المدخلات (تسوية الدُفعة) بطرح متوسطها والقسمة على انحرافها المعياري، حيث يتم تقدير كلاهما بناءً على الإحصائيات من الدفعات الصغيرة الحالي. بعد ذلك، نطبق معامل مقياس scale coefficient وتعويض لاستعادة درجات الحرية المفقودة. يرجع هذا تحديدًا إلى هذا التسوية استنادًا إلى إحصائيات الدُفعات، حيث تشتق تسوية الدُفعة اسمها.

لاحظ أنه إذا حاولنا تطبيق تسوية الدُفعات باستخدام دفعات صغيرة من الحجم 1، فلن نتمكن من تعلم أي شيء. هذا لأنه بعد طرح المتوسط، ستأخذ كل وحدة مخفية القيمة 0. كما قد تتخيل، نظرًا لأننا نخصص قسمًا كاملاً لتسوية الدُفعات، مع وجود عدد كافٍ من الدُفعات الصغيرة، فإن الطريقة تثبت فعاليتها وثباتها. أحد الوجبات الجاهزة هنا هو أنه عند تطبيق تسوية الدُفعات، يكون اختيار حجم الدُفعة أكثر أهمية من عدم تسوية الدُفعات، أو على الأقل، هناك حاجة إلى معايرة مناسبة حيث يمكننا تعديلها.

قم بالإشارة بواسطة الدفعات الصغيرة $\mathcal{B}$ وليكن $\mathbf{x} \in \mathcal{B}$ تكون مدخلاً لتسوية الدُفعات (BN). في هذه الحالة، يتم تحديد تسوية الدُفعة على النحو التالي:

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \boldsymbol{\mu}_{\mathcal{B}}}{\boldsymbol{\sigma}_{\mathcal{B}}} + \boldsymbol{\beta}. \qquad (8.5.1)$$

في (8.5.1)، $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ هو متوسط العينة و$\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ هو الانحراف المعياري للعينة من الدفعات الصغيرة. بعد تطبيق التوحيد القياسي standardization، فإن الدفعات الصغيرة الناتج له متوسط صفري وتباين الوحدة. يعد اختيار تباين الوحدة (مقابل بعض الأرقام السحرية الأخرى) اختيارًا عشوائيًا. نستعيد درجة الحرية هذه بتضمين معلمة مقياس عنصري (elementwise) $\boldsymbol{\gamma}$ ومعلمة تحول (shift parameter) $\boldsymbol{\beta}$ لها نفس الشكل مثل $\mathbf{x}$. كلاهما معلمات يجب تعلمها كجزء من تدريب النموذج.

لا يمكن للأحجام المتغيرة للطبقات المتوسطة أن تتباعد أثناء التدريب لأن تسوية الدُفعات يتركز بنشاط ويعيد قياسها مرة أخرى إلى متوسط وحجم معينين (عبر $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ و $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$). تؤكد التجربة العملية أنه، كما أشير إليه عند مناقشة إعادة قياس الميزات feature rescaling، يبدو أن تسوية الدُفعات يسمح بمعدلات تعلم أكثر قوة. نحسب $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ و $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ في (8.5.1) كالتالي:

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} \mathbf{x} \text{ and } \hat{\boldsymbol{\sigma}}^2_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon.$$

لاحظ أننا نضيف ثابتًا صغيرًا لتقدير التباين لضمان عدم محاولة القسمة على الصفر أبدًا، حتى في الحالات التي قد يكون فيها تقدير التباين التجريبي صغيرًا جدًا أو حتى يتلاشى. التقديرات $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$ و $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$ مواجهة مشكلة القياس باستخدام تقديرات صاخبة noisy estimates للمتوسط والتباين. قد تعتقد أن هذه الضوضاء يجب أن تكون مشكلة. على العكس تمامًا، هذا مفيد بالفعل.

تبين أن هذا موضوع متكررفي التعلم العميق. لأسباب لم يتم وصفها جيدًا من الناحية النظرية، غالبًا ما تؤدي المصادر المختلفة للضوضاءفي التحسين إلى تدريب أسرع وضبط زائد أقل: يبدو أن هذا الاختلاف يعمل كشكل من أشكال التنظيم. (Teye et al.، 2018) و (Luo et al.، 2018) يربطان خصائص تسوية الدُفعات بـ Bayesian priors والعقوبات على التوالي. على وجه الخصوص، يلقي هذا بعض الضوء على اللغز لماذا يعمل تسوية الدُفعات بشكل أفضل مع الأحجام الصغيرة المتوسطةفي النطاق $50 \sim 100$. يبدو أن هذا الحجم المعين من الدفعات الصغيرة minibatch يضخ "المقدار المناسب right amount" من الضوضاء لكل طبقة، سواء من حيث الحجم عبر $\hat{\boldsymbol{\sigma}}$، ومن حيث التعويض عن عبر $\hat{\boldsymbol{\mu}}$: الدفعة الصغيرة الأكبر larger minibatch يتم تنظيمها بشكل أقل بسبب التقديرات الأكثر ثباتًا،في حين أن الدفعات الصغيرة الاصغر tiny minibatches تدمر الإشارة المفيدة بسبب التباين العالي. استكشاف هذا الاتجاه بشكل أكبر، والنظرفي الأنواع البديلة للمعالجة المسبقة والتصفية قد يؤدي إلى أنواع فعالة أخرى من التنظيم.

عند إصلاح نموذج مدرب، قد تعتقد أننا نفضل استخدام مجموعة البيانات dataset بأكملها لتقدير المتوسط والتباين. بمجرد اكتمال التدريب، لماذا نرغبفي تصنيف نفس الصورة بشكل مختلف، اعتمادًا على المجموعة التي تتواجد فيها؟ أثناء التدريب، مثل هذا الحساب الدقيق غير ممكن لأن المتغيرات الوسيطة لجميع أمثلة البيانات تتغيرفي كل مرة نقوم فيها بتحديث نموذجنا. ومع ذلك، بمجرد تدريب النموذج، يمكننا حساب متوسطات ومتغيرات كل طبقة بناءً على مجموعة البيانات بأكملها.في الواقع، هذه ممارسة قياسية للنماذج التي تستخدم تسوية الدُفعات، وبالتالي تعمل طبقات تسوية الدفعات بشكل مختلففي وضع التدريب training mode (التسوية عن طريق إحصائيات الدُفعة المصغرة) وفي وضع التنبؤ prediction mode (التسوية بواسطة إحصاءات مجموعة البيانات).في هذا النموذج، تشبه إلى حد كبير سلوك تنظيم التسرب dropout regularization من القسم 5.6، حيث يتم حقن الضوضاء فقط أثناء التدريب.

8.5.2. طبقات تسوية الدفعات Batch Normalization Layers

تختلف تطبيقات تسوية الدفعات للطبقات المتصلة تمامًا والطبقات التلافيفية قليلاً. يتمثل أحد الاختلافات الرئيسية بين تسوية الدُفعات والطبقات الأخرىفي أنه نظرًا لأن تسوية الدُفعات يعمل على الدفعات الصغيرة minibatch كاملفي كل مرة، لا يمكننا تجاهل بُعد الدُفعة batch dimension كما فعلنا من قبل عند تقديم طبقات أخرى.

8.5.2.1. طبقات متصلة بالكامل Fully Connected Layers

عند تطبيق تسوية الدُفعات على طبقات متصلة بالكامل Fully Connected Layers، يتم إدخال تسوية الدُفعة للمقالة الاصلية بعد التحويل الأفيني وقبل دالة التنشيط غير الخطي. جربت التطبيقات اللاحقة إدخال تسوية الدُفعات مباشرة بعد دوال التنشيط (Ioffe and Szegedy، 2015). بالإشارة إلى المدخلات إلى الطبقة المتصلة بالكامل $\mathbf{x}$ عن طريق التحويل الأفيني $\mathbf{Wx} + \mathbf{b}$ (مع معلمة الوزن $\mathbf{W}$ ومعلمة التحيز $\mathbf{b}$)، ودالة التنشيط من خلال ϕ، يمكننا التعبير عن حساب الممكّن لتسوية الدفعات وإخراج الطبقة المتصل بالكامل $\mathbf{h}$ على النحو التالي:

$$\mathbf{h} = \phi(\mathrm{BN}(\mathbf{Wx} + \mathbf{b})).$$

تذكر أن المتوسط والتباين يتم حسابهما على نفس الدفعة الصغيرة minibatch الذي يتم تطبيق التحويل عليه.

8.5.2.2. طبقات تلافيفية Convolutional Layers

وبالمثل، مع الطبقات التلافيفية Convolutional Layers، يمكننا تطبيق تسوية الدُفعات بعد الالتفاف وقبل دالة التنشيط اللاخطي. يتمثل الاختلاف الرئيسي عن تسوية الدُفعاتفي الطبقات المتصلة بالكاملفي أننا نطبق العملية على أساس كل قناة عبر جميع المواقع. يتوافق هذا مع افتراضنا لثبات الترجمة الذي أدى إلى تلافيف: لقد افترضنا أن الموقع المحدد لنمط داخل الصورة لم يكن حاسمًا لغرض الفهم.

افترض أن الدفعات الصغيرة لدينا تحتوي على m أمثلة وأن ناتج الالتفاف لكل قناة له ارتفاع p وعرض q. بالنسبة للطبقات التلافيفية، نقوم بتنفيذ تسوية كل دفعة على العناصر $m \cdot p \cdot q$ لكل قناة إخراجفي وقت واحد. وبالتالي، فإننا نجمع القيم عبر جميع المواقع المكانية عند حساب المتوسط والتباين، وبالتالي نطبق نفس المتوسط والتباين داخل قناة معينة لتسوية القيمةفي كل موقع مكاني. كل قناة لها مقياسها ومعلمات التحول الخاصة بها، وكلاهما قيمة قياسية scalar.

8.5.2.3. تسوية الطبقة Layer Normalization

لاحظ أنهفي سياق التلافيف، يتم تحديد تسوية الدُفعات جيدًا حتى بالنسبة للدفعات الصغيرة ذات الحجم 1: بعد كل شيء، لدينا جميع المواقع عبر الصورة إلى المتوسط. وبالتالي، يتم تحديد

المتوسط والتباين جيدًا، حتى لو كان ذلك ضمن ملاحظة واحدة فقط. قاد هذا الاعتبار Ba et al. (2016) لإدخال مفهوم تسوية الطبقة layer normalization. إنه يعمل تمامًا مثل معيار الدُفعة batch norm، فقط أنه يتم تطبيقه على ملاحظة واحدةفي كل مرة. وبالتالي فإن كلاً من الإزاحة والقيمة القياسية scalars. بالنظر إلى متجه $\mathbf{x}$ ذي الأبعاد n ، يتم إعطاء معايير norms الطبقة بواسطة

$$\mathbf{x} \rightarrow \mathrm{LN}(\mathbf{x}) = \frac{\mathbf{x} - \hat{\mu}}{\hat{\sigma}},$$

حيث يتم تطبيق المقياس scaling والإزاحة offset من حيث المعامل ويتم تقديمهما بواسطة

$$\hat{\mu} := \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \hat{\sigma}^2 := \frac{1}{n}\sum_{i=1}^{n} (x_i - \hat{\mu})^2 + \epsilon.$$

كما كان من قبل، نضيف ازاحة صغيرةً $\epsilon > 0$ لمنع القسمة على الصفر. تتمثل إحدى الفوائد الرئيسية لاستخدام تسوية الطبقةفي أنه يمنع التباعد divergence. بعد كل شيء، تجاهل ϵ، ناتج تسوية الطبقة مستقل عن المقياس. هذا هو، لدينا $\mathrm{LN}(\mathbf{x}) \approx \mathrm{LN}(\alpha\mathbf{x})$ لأي خيار $\alpha \neq 0$. تصبح هذه مساواة لـ $|\alpha| \rightarrow \infty$ (المساواة التقريبية ترجع إلى الازاحة ϵ لتعويض التباين).

ميزة أخرى لتسوية الطبقة هي أنها لا تعتمد على حجم الدفعات الصغيرة. كما أنها مستقلة عما إذاكنافي تدريب أو نظام اختبار. بمعنى آخر، إنه ببساطة تحول حتمي يوحد التنشيطات إلى مقياس معين. يمكن أن يكون هذا مفيدًا جدًافي منع التباعدفي التحسين. نتخطى المزيد من التفاصيل ونوصي القارئ المهتم بالرجوع إلى المقالة الأصلية.

8.5.2.4. تسوية الدُفعات أثناء التنبؤ Batch Normalization During Prediction

كما ذكرنا سابقًا، عادةً ما يتصرف تسوية الدُفعات بشكل مختلففي وضع التدريب ووضع التنبؤ. أولاً، لم يعد الضجيجفي متوسط العينة وتباين العينة الناتج عن تقدير كل منها على الدفعات الصغيرة مرغوبًا بعد أن نقوم بتدريب النموذج. ثانيًا، قد لا نتمتع برفاهية حساب إحصائيات التسوية لكل دفعة. على سبيل المثال، قد نحتاج إلى تطبيق نموذجنا لعمل تنبؤ واحدفي كل مرة.

عادةً، بعد التدريب، نستخدم مجموعة البيانات بأكملها لحساب التقديرات الثابتة للإحصاءات المتغيرة ثم إصلاحهافي وقت التنبؤ. وبالتالي، فإن تسوية الدُفعات يتصرف بشكل مختلف أثناء التدريب وفي وقت الاختبار. تذكر أن التسرب يعرض أيضًا هذه الخاصية.

8.5.3. التنفيذ من البداية Implementation from Scratch

لمعرفة كيفية عمل تسوية الدُفعات عمليًا، نطبق واحدًا من البداية أدناه.

```
import tensorflow as tf
from d2l import tensorflow as d2l


def batch_norm(X, gamma, beta, moving_mean, moving_var,
eps):
    # Compute reciprocal of square root of the moving
variance elementwise
    inv = tf.cast(tf.math.rsqrt(moving_var + eps),
X.dtype)
    # Scale and shift
    inv *= gamma
    Y = X * inv + (beta - moving_mean * inv)
    return Y
```

يمكننا الآن إنشاء طبقة `BatchNorm` مناسبة. ستحتفظ طبقتنا بالمعلمات المناسبة لمقياس `gamma` وتحويل `beta`، وسيتم تحديث كلاهما أثناء التدريب. بالإضافة إلى ذلك، ستحافظ طبقتنا على المتوسطات والانحرافات المعيارية المتحركة للاستخدام اللاحق أثناء تنبؤ النموذج.

بوضع تفاصيل الخوارزمية جانبًا، لاحظ نمط التصميم الذي يقوم عليه تنفيذنا للطبقة. عادةً، نحدد الرياضياتفي دالة منفصلة، على سبيل المثال `batch_norm`. نقوم بعد ذلك بدمج هذه الدالةفي طبقة مخصصة custom layer، والتي يعالج كودهافي الغالب مسائل مسك الدفاتر bookkeeping، مثل نقل البيانات إلى سياق الجهاز الصحيح، وتخصيص أي متغيرات مطلوبة وتهيئتها، وتتبع المتوسطات المتحركة (هنا للمتوسط والتباين)، وما إلى ذلك. يتيح هذا النمط فصلًا نظيفًا للرياضيات عن الكود المعياري. لاحظ أيضًا أنه من أجل السهولة، لم نقلق بشأن الاستنتاج التلقائي لشكل الإدخال هنا، وبالتالي نحتاج إلى تحديد عدد الميزات طوال الوقت.في الوقت الحالي، توفر جميع أطر التعلم العميق الحديثة اكتشافًا تلقائيًا للحجم والشكلفي واجهات برمجة تطبيقات API تسوية الدُفعات عالية المستوى (في الواقع، سنستخدم هذا بدلاً من ذلك).

```
class BatchNorm(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(BatchNorm, self).__init__(**kwargs)

    def build(self, input_shape):
        weight_shape = [input_shape[-1], ]
        # The scale parameter and the shift parameter
(model parameters) are
        # initialized to 1 and 0, respectively
        self.gamma = self.add_weight(name='gamma',
shape=weight_shape,
```

```
            initializer=tf.initializers.ones,
trainable=True)
        self.beta = self.add_weight(name='beta',
shape=weight_shape,
            initializer=tf.initializers.zeros,
trainable=True)
        # The variables that are not model parameters
are initialized to 0
        self.moving_mean =
self.add_weight(name='moving_mean',
            shape=weight_shape,
initializer=tf.initializers.zeros,
            trainable=False)
        self.moving_variance =
self.add_weight(name='moving_variance',
            shape=weight_shape,
initializer=tf.initializers.ones,
            trainable=False)
        super(BatchNorm, self).build(input_shape)

    def assign_moving_average(self, variable, value):
        momentum = 0.1
        delta = (1.0 - momentum) * variable + momentum *
value
        return variable.assign(delta)

    @tf.function
    def call(self, inputs, training):
        if training:
            axes = list(range(len(inputs.shape) - 1))
            batch_mean = tf.reduce_mean(inputs, axes,
keepdims=True)
            batch_variance =
tf.reduce_mean(tf.math.squared_difference(
                inputs, tf.stop_gradient(batch_mean)),
axes, keepdims=True)
            batch_mean = tf.squeeze(batch_mean, axes)
            batch_variance = tf.squeeze(batch_variance,
axes)
            mean_update = self.assign_moving_average(
                self.moving_mean, batch_mean)
```

```
            variance_update =
self.assign_moving_average(
                self.moving_variance, batch_variance)
            self.add_update(mean_update)
            self.add_update(variance_update)
            mean, variance = batch_mean, batch_variance
        else:
            mean, variance = self.moving_mean,
self.moving_variance
        output = batch_norm(inputs, moving_mean=mean,
moving_var=variance,
            beta=self.beta, gamma=self.gamma, eps=1e-5)
        return output
```

استخدمنا الزخم `momentum` للتحكم في التجميع على تقديرات المتوسط والتباين السابقة. هذه تسمية خاطئة misnomer إلى حد ما حيث لا علاقة لها على الإطلاق بمصطلح الزخم momentum term في التحسين في القسم 12.6. ومع ذلك، فهو الاسم المعتمد بشكل شائع لهذا المصطلح، وإحترامًا لاتفاقية تسمية API، نستخدم نفس اسم المتغير في التعليمات البرمجية الخاصة بنا أيضًا.

8.5.4. LeNet مع تسوية الدُفعات

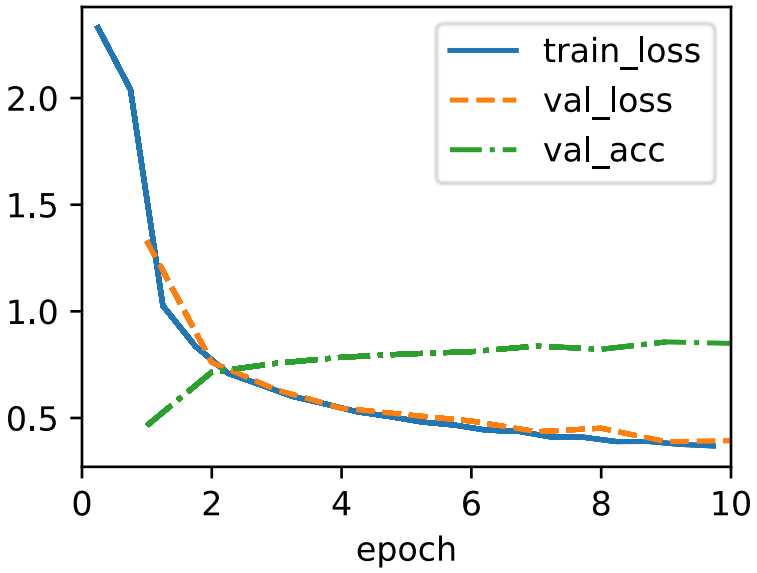
لمعرفة كيفية تطبيق `BatchNorm` في السياق، نطبقه أدناه على نموذج LeNet التقليدي (القسم 7.6). تذكر أنه يتم تطبيق تسوية الدفعات بعد الطبقات التلافيفية أو الطبقات المتصلة بالكامل ولكن قبل دوال التنشيط المقابلة.

```
class BNLeNetScratch(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6,
kernel_size=5,
                                   input_shape=(28, 28,
1)),
            BatchNorm(),
tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Conv2D(filters=16,
kernel_size=5),
```

```
            BatchNorm(),
tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Flatten(),
tf.keras.layers.Dense(120),
            BatchNorm(),
tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.Dense(84), BatchNorm(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.Dense(num_classes)])
```

كمافي السابق، سنقوم بتدريب شبكتنا على مجموعة بيانات Fashion-MNIST. هذا الكود مطابق تقريبًا لذلك عندما قمنا بتدريب LeNet لأول مرة.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128)
with d2l.try_gpu():
    model = BNLeNetScratch(lr=0.5)
    trainer.fit(model, data)
```



دعونا نلقي نظرة على معلمة المقياس (scale parameter) `gamma` ومعامل التحويل (shift parameter) beta الذي تم تعلمه من طبقة تسوية الدفعة الأولى.

```
tf.reshape(model.net.layers[1].gamma, (-1,)),
tf.reshape(
    model.net.layers[1].beta, (-1,))
```

```
(<tf.Tensor: shape=(6,), dtype=float32, numpy=
 array([4.5314302, 2.9945924, 2.1465805, 2.1390676,
3.5647843, 2.0847673],
       dtype=float32)>,
 <tf.Tensor: shape=(6,), dtype=float32, numpy=
 array([ 0.21811652, -0.152415  , -1.0798329 , -
1.2022917 , -0.03723535,
        -0.7957938 ], dtype=float32)>)
```

8.5.5. التنفيذ المختصر Concise Implementation

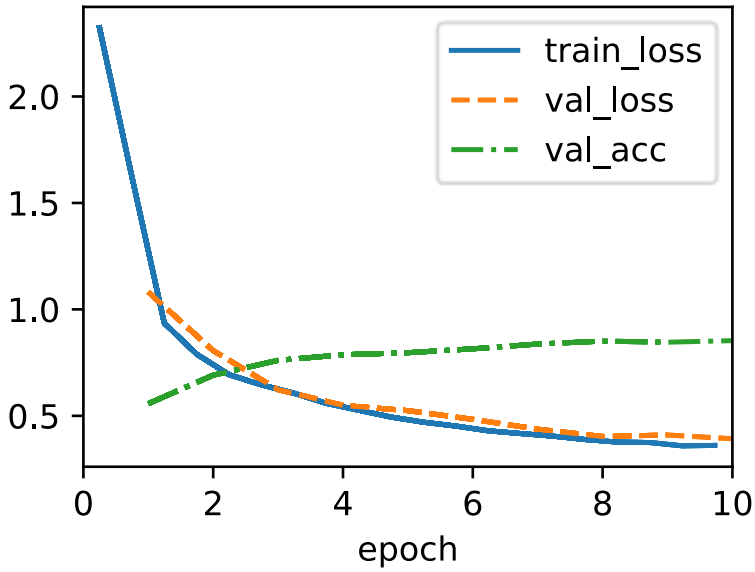
بالمقارنة مع كلاس `BatchNorm`، التي حددناها بأنفسنا للتو، يمكننا استخدام كلاس `BatchNorm`المحددةفي واجهات برمجة التطبيقات عالية المستوى من إطار عمل التعلم العميق مباشرة. يبدو الرمز مطابقًا تقريبًا لتطبيقنا أعلاه، باستثناء أننا لم نعد بحاجة إلى تقديم وسيطات إضافية له للحصول على الأبعاد الصحيحة.

```
class BNLeNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6,
kernel_size=5,
                                   input_shape=(28, 28,
1)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Conv2D(filters=16,
kernel_size=5),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2,
strides=2),
            tf.keras.layers.Flatten(),
tf.keras.layers.Dense(120),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
```

```
            tf.keras.layers.Dense(84),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('sigmoid'),
            tf.keras.layers.Dense(num_classes)])
```

أدناه، نستخدم نفس المعلمات الفائقة لتدريب نموذجنا. لاحظ أنه كالعادة، يعمل متغير واجهة برمجة التطبيقات عالي المستوى بشكل أسرع نظرًا لأن كوده قد تم تجميعه إلى C++ أو CUDA بينما يجب تفسير تطبيقنا المخصص بواسطة بايثون.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128)
with d2l.try_gpu():
    model = BNLeNet(lr=0.5)
    trainer.fit(model, data)
```



8.5.6. المناقشة

بشكل بديهي، يُعتقد أن تسوية الدُفعات batch normalization تجعل مشهد التحسين أكثر سلاسة. ومع ذلك، يجب أن نكون حريصين على التمييز بين الحدس التأملي والتفسيرات الحقيقية للظواهر التي نلاحظها عند تدريب النماذج العميقة. تذكر أننا لا نعرف حتى لماذا تعمم الشبكات العصبية العميقة الأبسط (MLPs وCNN التقليدية) جيدًا في المقام الأول. حتى مع التسرب dropout وتناقص الوزن weight decay، فإنها تظل مرنة للغاية لدرجة أن قدرتها على التعميم على البيانات غير المرئية تحتاج على الأرجح إلى ضمانات تعلم نظرية أكثر دقة.

في المقالة الأصلية التي اقترحت تسوية الدُفعات (Ioffe and Szegedy، 2015) ، بالإضافة إلى تقديم أداة قوية ومفيدة، قدمت شرحًا لسبب نجاحها: عن طريق تقليل التحول المتغير الداخلي internal covariate shift. من المفترض أن المؤلفين قصدوا عن طريق التحول الداخلي المتغير شيئًا مثل الحدس المعبر عنه أعلاه ــ فكرة أن توزيع القيم المتغيرة يتغير على مدار التدريب. ومع ذلك، كانت هناك مشكلتانفي هذا التفسير: 1) هذا الانجراف drift مختلف تمامًا عن التحول المتغير، مما يجعل الاسم تسمية خاطئة misnomer. إذا كان هناك أي شيء، فهو أقرب إلى مفهوم الانجراف. 2) يقدم التفسير حدسًا غير محدد ولكنه يترك السؤال عن سبب نجاح هذه التقنية تحديدًافي سؤال مفتوح يحتاج إلى تفسير دقيق.في هذا الكتاب، نهدف إلى نقل الحدس الذي يستخدمه الممارسون لتوجيه تطورهم للشبكات العصبية العميقة. ومع ذلك، نعتقد أنه من المهم فصل هذه البديهيات التوجيهية عن الحقائق العلمية الراسخة.في النهاية، عندما تتقن هذه المادة وتبدأفي كتابة أوراقك البحثية الخاصة، سترغبفي أن تكون واضحًافي التحديد بين الادعاءات التقنية والحدس.

بعد نجاح تسوية الدفعات، ظهر تفسيره من حيث التحول الداخلي المتغير مرارًا وتكرارًافي المناقشاتفي الأدبيات الفنية والخطاب الأوسع حول كيفية تقديم أبحاث التعلم الآلي.في خطاب لا يُنسى ألقاه أثناء قبوله جائزة اختبار الوقت Test of Time Awardفي مؤتمر NeurIPS 2017، استخدم علي رحيمي التحول الداخلي المشترك internal covariate shift كنقطة محوريةفي حجة تشبه الممارسة الحديثة للتعلم العميق بالكيمياء. بعد ذلك، تمت إعادة النظرفي المثال بالتفصيلفي ورقة موقف تحدد الاتجاهات المثيرة للقلقفي التعلم الآلي (ليبتون وشتاينهاردت، 2018). اقترح مؤلفون آخرون تفسيرات بديلة لنجاح تسوية الدُفعات، حيث زعم البعض أن نجاح التسوية على دفعات يأتي على الرغم من إظهار سلوك يتعارض من بعض النواحي مع تلك التي زُعمتفي المقالة الأصلية (Santurkar et al.، 2018).

نلاحظ أن التحول المتغير الداخلي لا يستحق النقد أكثر من أي من آلاف الادعاءات الغامضة المماثلة التي يتم تقديمها كل عامفي أدبيات التعلم الآلي التقنية. على الأرجح، فإن صدى هذه النقاشات كنقطة محوريةفي هذه المناقشات يرجع إلى التعرف الواسع على الجمهور المستهدف. أثبت تسوية الدُفعات أنه أسلوب لا غنى عنه، تم تطبيقهفي جميع مصنفات الصور المنشورة تقريبًا، مما أكسب المقالة التي قدمت التقنية عشرات الآلاف من الاستشهادات. ومع ذلك، فإننا نخمّن أن المبادئ التوجيهية للتنظيم من خلال حقن الضوضاء والتسريع من خلال إعادة القياس والمعالجة المسبقة أخيرًا قد تؤدي إلى المزيد من الاختراعات للطبقات والتقنياتفي المستقبل.

في ملاحظة أكثر عملية، هناك عدد من الجوانب aspects التي تستحق التذكر حول تسوية الدُفعات:

- أثناء تدريب النموذج، يضبط تسوية الدُفعات باستمرار الإخراج الوسيط للشبكة من خلال استخدام المتوسط والانحراف المعياري للدفعات الصغيرة، بحيث تكون قيم المخرجات الوسيطةفي كل طبقة عبر الشبكة العصبية أكثر استقرارًا.
- تختلف تسوية الدُفعات للطبقات المتصلة تمامًا والطبقات التلافيفية قليلاً.في الواقع، بالنسبة للطبقات التلافيفية، يمكن أحيانًا استخدام تسوية الطبقة كبديل.
- مثل طبقة التسرب dropout layer، فإن طبقات تسوية الدفعات لها سلوكيات مختلفةفي وضع التدريب ووضع التنبؤ.
- يعد تسوية الدُفعات مفيدًا للتنظيم وتحسين التقاربفي التحسين. من ناحية أخرى، يبدو أن الدافع الأصلي لتقليل التحول المتغير الداخلي ليس تفسيرًا صالحًا.

8.5.7. التمارين

1. هل يجب إزالة معلمة التحيز من الطبقة المتصلة بالكامل أو الطبقة التلافيفية قبل تسوية الدُفعة؟ لماذا؟
2. قارن معدلات التعلم لـ LeNet مع أو بدون تسوية الدفعات.
   1. ارسم الزيادةفي دقة التحقق من الصحة validation accuracy.
   2. ما الحجم الذي يمكنك جعل معدل التعلم قبل فشل التحسينفي كلتا الحالتين؟
3. هل نحتاج إلى تسوية الدفعاتفي كل طبقة؟ جربها؟
4. قم بتنفيذ إصدار "خفيف lite" من تسوية الدُفعات الذي يزيل فقط المتوسط ، أو بدلاً من ذلك يزيل التباين فقط. كيف تتصرف؟
5. إصلح معلمات `beta` و `gamma`. راقب وحلل النتائج.
6. هل يمكنك استبدال التسرب dropout بتسوية الدفعات؟ كيف يتغير السلوك؟
7. أفكار البحث: فكرفي تحولات التسوية الأخرى التي يمكنك تطبيقها:
   1. هل يمكنك تطبيق التحويل المتكامل الاحتمالي probability integral transform؟
   2. هل يمكنك استخدام تقدير التغاير covariance estimate الكامل للرتبة؟ لماذا ربما لا تفعل ذلك؟
   3. هل يمكنك استخدام متغيرات مصفوفة مضغوطة compact matrix variants أخرى (كتلة قطرية block-diagonal، رتبة إزاحة منخفضة low-displacement rank، مونارك Monarch، إلخ)؟
   4. هل يعمل ضغط التناثر sparsification compression كمنظم regularizer؟

5. هل هناك توقعات أخرى (على سبيل المثال، مخروط محدب convex cone، وتحويلات خاصة بمجموعة تناظر symmetry group-specific transforms) يمكنك استخدامها؟

8.6 الشبكات المتبقية (ResNet) Residual Networks ResNeXtg

نظرًا لأننا نصمم شبكات أعمق بشكل متزايد، يصبح من الضروري فهم كيف يمكن أن تؤدي إضافة الطبقات إلى زيادة تعقيد الشبكة وتعبيرها. والأهم من ذلك هو القدرة على تصميم الشبكات حيث تؤدي إضافة الطبقات إلى جعل الشبكات أكثر تعبيرًا بشكل صارم بدلاً من الاختلاف فقط. لإحراز بعض التقدم نحتاج إلى القليل من الرياضيات.

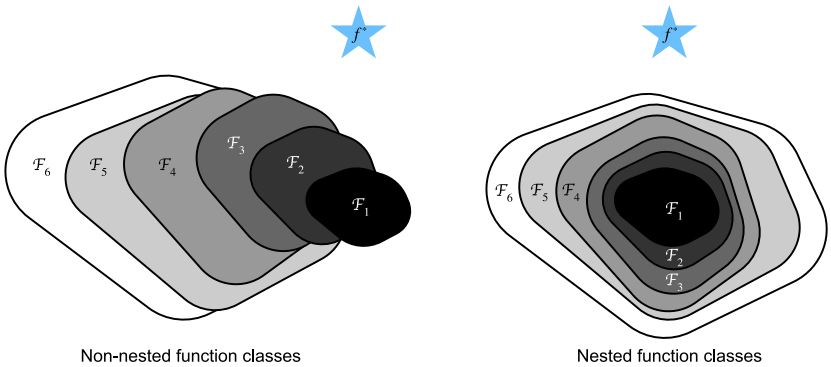
8.6.1. فئات الدوال Function Classes

ضعفي اعتبارك $\mathcal{F}$ فئة الدوال Function Classes التي يمكن أن تصل إليها بُنية شبكة معينة (جنبًا إلى جنب مع معدلات التعلم وإعدادات المعلمات الفائقة الأخرى). وهذا يعني لكل $f \in \mathcal{F}$ أنه توجد مجموعة من المعلمات (على سبيل المثال، الأوزان والتحيزات) التي يمكن الحصول عليها من خلال التدريب على مجموعة بيانات مناسبة. لنفترض أن هذه f^* هي دالة "الحقيقة" truth التي نود حقًا العثور عليها. إذا كانفي $\mathcal{F}$ ، فنحنفي حالة جيدة ولكنفي العادة لن نكون محظوظين جدًا. بدلاً من ذلك، سنحاول العثور على بعض $f_{\mathcal{F}}^*$ وهو أفضل رهانفي $\mathcal{F}$. على سبيل المثال، نظرًا لمجموعة بيانات تحتوي على ميزات $\mathbf{X}$ وتسميات $\mathbf{y}$، فقد نحاول العثور عليها من خلال حل مشكلة التحسين التالية:

$$f_{\mathcal{F}}^* \stackrel{\text{def}}{=} \operatorname*{argmin}_f L(\mathbf{X}, \mathbf{y}, f) \text{ subject to } f \in \mathcal{F}.$$

نحن نعلم أن التنظيم regularization (Morozov، 1984، Tikhonov and Arsenin، 1977) قد يتحكمفي تعقيد $\mathcal{F}$ ويحقق الاتساق consistency، لذا فإن الحجم الأكبر لبيانات التدريب يؤدي عمومًا إلى تحسين $f_{\mathcal{F}}^*$. من المعقول فقط أن نفترض أنه إذا صممنا بُنية مختلفة وأكثر قوة $\mathcal{F}'$، يجب أن نصل إلى نتيجة أفضل. بعبارة أخرى، نتوقع $f_{\mathcal{F}'}^*$ أفضل من $f_{\mathcal{F}}^*$. ومع ذلك، إذا $\mathcal{F} \not\subseteq \mathcal{F}'$ لم يكن هناك ما يضمن حدوث ذلك.في الواقع، $f_{\mathcal{F}'}^*$ قد يكون أسوأ. كما هو موضحفي الشكل 8.6.1، بالنسبة لفئات الدوال غير المتداخلة، لا تقترب فئة الدالة الأكبر دائمًا من دالة "الحقيقة" f^*. على سبيل المثال، على يسار الشكل 8.6.1، على الرغم من أن $\mathcal{F}_3$ أقرب إلى f^* من $\mathcal{F}_1$، فإنه يتحرك بعيدًا وليس هناك ما يضمن أن زيادة التعقيد يمكن أن تقلل المسافة

من f^*. مع فئات الدوال المتداخلة حيث $\mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_6$ على يمين الشكل 8.6.1، يمكننا تجنب المشكلة المذكورة أعلاه من فئات الدوال غير المتداخلة.



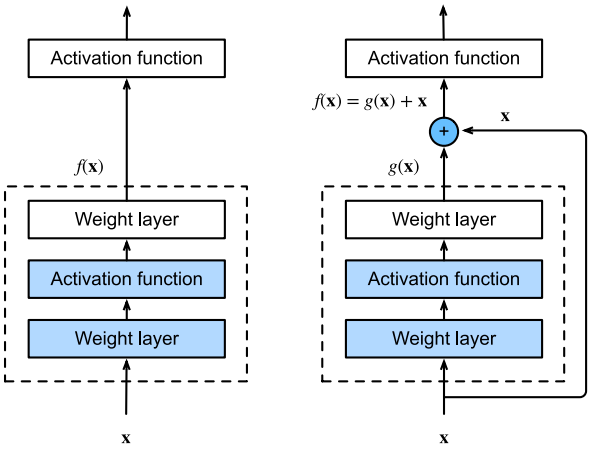
الشكل 8.6.1 بالنسبة لفئات الدوال غير المتداخلة، لا تضمن فئة الدالة الأكبر (المشار إليها بالمنطقة) الاقتراب من دالة "الحقيقة" (f^*). هذا لا يحدث في فئات الدوال المتداخلة.

وبالتالي، فقط إذا كانت فئات الدوال الأكبر تحتوي على الفئات الأصغر، فإننا نضمن أن زيادتها يزيد بشكل صارم من القوة التعبيرية للشبكة. بالنسبة للشبكات العصبية العميقة، إذا تمكنا من تدريب الطبقة المضافة حديثًا على دالة هوية identity function، $f(\mathbf{x}) = \mathbf{x}$, فسيكون النموذج الجديد فعالًا مثل النموذج الأصلي. نظرًا لأن النموذج الجديد قد يحصل على حل أفضل يناسب مجموعة بيانات التدريب، فقد تسهل الطبقة المضافة تقليل أخطاء التدريب.

هذا هو السؤال الذي أخذ في الاعتبار (He et al.، 2016) عند العمل على نماذج رؤية حاسوبية عميقة جدًا. في قلب شبكتهم المتبقية المقترحة (ResNet) هي فكرة أن كل طبقة إضافية يجب أن تحتوي بسهولة أكبر على دالة الهوية كأحد عناصرها. هذه الاعتبارات عميقة إلى حد ما لكنها أدت إلى حل بسيط بشكل مدهش، وهو الكتلة المتبقية residual block. مع ذلك، فازت شبكة ResNet بتحدي التعرف البصري على نطاق واسع على ImageNet في عام 2015. كان للتصميم تأثير عميق على كيفية بناء شبكات عصبية عميقة. على سبيل المثال، تمت إضافة الكتل المتبقية إلى الشبكات المتكررة RNN (Kim et al.، 2017، Prakash et al.، 2016). وبالمثل، فإن المحولات transformers (Vaswani et al.، 2017) تستخدمها لتكديس طبقات عديدة من الشبكات بكفاءة. يتم استخدامه أيضًا في الشبكات العصبية للرسم البياني graph neural networks (Kipf and Welling، 2016) وكمفهوم أساسي، فقد تم استخدامه على نطاق واسع في الرؤية الحاسوبية (Redmon and Farhadi، 2018، Ren et al.، 2015). لاحظ أن الشبكات المتبقية سبقتها شبكات الطرق السريعة highway networks (Srivastava et al.، 2015) التي تشترك في بعض الدوافع، وإن كان ذلك بدون تحديد معلمات أنيقة حول دالة الهوية.

8.6.2. الكتل المتبقية Residual Blocks

دعونا نركز على جزء محلي من الشبكة العصبية، كما هو موضح في الشكل 8.6.2. دلالة على المدخلات $\mathbf{x}$. نفترض أن التعيين الأساسي المطلوب الذي نريد الحصول عليه من خلال التعلم هو $f(\mathbf{x})$، لاستخدامه كمدخل لدالة التنشيط في الأعلى. على اليسار، يجب أن يتعرف الجزء الموجود داخل المربع ذي الخطوط المنقطة على المطابقة (التعيين) $f(\mathbf{x})$ مباشرةً. على اليمين، يحتاج الجزء الموجود داخل المربع ذي الخطوط المنقطة إلى معرفة التعيين المتبقي $g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$، وهو كيفية اشتقاق الكتلة المتبقية اسمها. إذا كان تعيين الهوية $f(\mathbf{x}) = \mathbf{x}$ هو التعيين الأساسي المرغوب فيه، فإن التعيين المتبقي يصل إلى $g(\mathbf{x}) = 0$ وبالتالي يسهل تعلمه: نحتاج فقط إلى دفع الأوزان والتحيزات الخاصة بطبقة الوزن الأعلى (على سبيل المثال، الطبقة المتصلة بالكامل والطبقة التلافيفية) داخل المنطقة المنقطة مربع خط إلى الصفر. يوضح الشكل الأيمن الكتلة المتبقية لـ ResNet، حيث يُطلق على الخط الصلب الذي يحمل إدخال الطبقة إلى عامل الإضافة اتصال متبقي residual connection (أو اتصال اختصار shortcut connection). مع الكتل المتبقية residual blocks، يمكن للمدخلات أن تنتشر بشكل أسرع من خلال الاتصالات المتبقية عبر الطبقات. في الواقع، يمكن اعتبار الكتلة المتبقية كحالة خاصة لكتلة التأسيس متعددة الفروع multi-branch Inception block: لها فرعين أحدهما هو تعيين الهوية identity mapping.



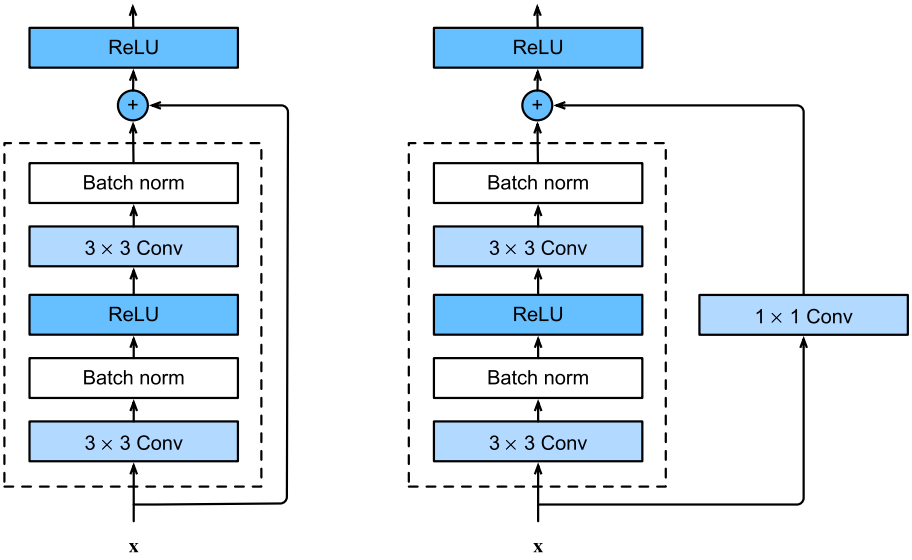
الشكل 8.6.2 في الكتلة العادية (على اليسار)، يجب أن يتعلم الجزء الموجود داخل المربع ذي الخطوط المنقطة تعيين $f(\mathbf{x})$ مباشرةً. في الكتلة المتبقية (على اليمين)، يحتاج الجزء الموجود داخل المربع ذي الخطوط المنقطة إلى معرفة التعيين المتبقي $g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$، مما يسهل تعلم تعيين الهوية $f(\mathbf{x}) = \mathbf{x}$.

تتبع ResNet تصميم الطبقة التلافيفية 3 × 3 الكامل لـ VGG. تحتوي الكتلة المتبقية على طبقتين تلافيفيتين 3 × 3 بنفس عدد قنوات الإخراج. تتبع كل طبقة تلافيفية طبقة تسوية دفعية ودالة تنشيط ReLU. بعد ذلك، نتخطى عمليتي الالتفاف هاتين ونضيف الإدخال مباشرةً قبل دالة تنشيط ReLU النهائية. يتطلب هذا النوع من التصميم أن يكون ناتج الطبقتين التلافيفيتين من نفس شكل المدخلات، بحيث يمكن إضافتهما معًا. إذا أردنا تغيير عدد القنوات، فنحن بحاجة إلى إدخال طبقة تلافيفية 1 × 1 إضافية لتحويل المدخلات إلى الشكل المطلوب لعملية الإضافة. دعونا نلقي نظرة على الكود أدناه.

```
import tensorflow as tf
from d2l import tensorflow as d2l


class Residual(tf.keras.Model):  #@save
    """The Residual block of ResNet."""
    def __init__(self, num_channels, use_1x1conv=False,
strides=1):
        super().__init__()
        self.conv1 =
tf.keras.layers.Conv2D(num_channels, padding='same',

kernel_size=3, strides=strides)
        self.conv2 =
tf.keras.layers.Conv2D(num_channels, kernel_size=3,

padding='same')
        self.conv3 = None
        if use_1x1conv:
            self.conv3 =
tf.keras.layers.Conv2D(num_channels, kernel_size=1,

strides=strides)
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.bn2 = tf.keras.layers.BatchNormalization()

    def call(self, X):
        Y =
tf.keras.activations.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3 is not None:
            X = self.conv3(X)
```

```
        Y += X
        return tf.keras.activations.relu(Y)
```

يولد هذا الرمز نوعين من الشبكات: أحدهما حيث نضيف المدخلات إلى المخرجات قبل تطبيق دالة ReLU اللاخطيةكلما `use_1x1conv=False`، والآخر حيث نعدل القنوات والدقة عن طريق متوسطات الالتفاف 1×1 قبل الإضافة. يوضح الشكل 8.6.3 هذا.



الشكل 8.6.3 فدرة ResNet مع الالتفاف 1×1 وبدونه، والذي يحول الإدخال إلى الشكل المطلوب لعملية الإضافة.

```
blk = Residual(3)
X = tf.random.normal((4, 6, 6, 3))
Y = blk(X)
Y.shape
```

```
TensorShape([4, 6, 6, 3])
```

لدينا أيضًا خيار خفض ارتفاع وعرض الإخراج إلى النصف مع زيادة عدد قنوات الإخراج.في هذه الحالة، نستخدم التلافيف 1×1 عبر `use_1x1conv=True`. يكون هذا مفيدًافي بداية كل كتلة ResNet لتقليل الأبعاد المكانية spatial dimensionality عبر الخطوات = 2.

```
blk = Residual(6, use_1x1conv=True, strides=2)
blk(X).shape
```

```
TensorShape([4, 3, 3, 6])
```

8.6.3. نموذج ResNet

أول طبقتين من ResNet هما نفس طبقات GoogLeNet التي وصفناها من قبل: الطبقة التلافيفية 7 × 7 مع 64 قناة إخراج وخطوة 2 متبوعة بطبقة تجميع الحد الأقصى 3 × 3 بخطوة 2. والفرق هو تسوية الدُفعات تمت إضافة طبقة بعد كل طبقة تلافيفيةفي ResNet.

```
class ResNet(d2l.Classifier):
    def b1(self):
        return tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(64, kernel_size=7,
strides=2,
                                   padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('relu'),
            tf.keras.layers.MaxPool2D(pool_size=3,
strides=2,
                                      padding='same')])
```

تستخدم GoogLeNet أربع وحدات مكونة من كتل الاستهلال Inception. ومع ذلك، تستخدم شبكة ResNet أربع وحدات مكونة من الكتل المتبقية، يستخدم كل منها عدة كتل متبقية مع نفس العدد من قنوات الإخراج. عدد القنواتفي الوحدة الأولى هو نفس عدد قنوات الإدخال. نظرًا لأنه تم بالفعل استخدام طبقة تجميع بحد أقصى بخطوة 2، فليس من الضروري تقليل الارتفاع والعرض.في أول كتلة متبقية لكل من الوحدات اللاحقة، يتم مضاعفة عدد القنوات مقارنةً بالوحدة السابقة، ويتم تقليل الارتفاع والعرض إلى النصف.

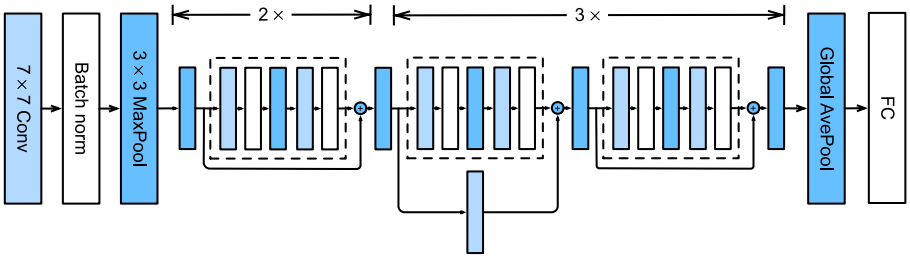
```
@d2l.add_to_class(ResNet)
def block(self, num_residuals, num_channels,
first_block=False):
    blk = tf.keras.models.Sequential()
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.add(Residual(num_channels,
use_1x1conv=True, strides=2))
        else:
            blk.add(Residual(num_channels))
    return blk
```

ثم نضيف جميع الوحدات إلى ResNet. هنا، يتم استخدام كتلتين متبقيتين لكل وحدة. أخيرًا، تمامًا مثل GoogLeNet، نضيف طبقة تجميع المتوسط العالمية، متبوعة بإخراج الطبقة المتصلة بالكامل.

```
@d2l.add_to_class(ResNet)
```

```
def __init__(self, arch, lr=0.1, num_classes=10):
    super(ResNet, self).__init__()
    self.save_hyperparameters()
    self.net = tf.keras.models.Sequential(self.b1())
    for i, b in enumerate(arch):
        self.net.add(self.block(*b, first_block=(i==0)))
    self.net.add(tf.keras.models.Sequential([
        tf.keras.layers.GlobalAvgPool2D(),
        tf.keras.layers.Dense(units=num_classes)]))
```

هناك 4 طبقات تلافيفيةفي كل وحدة (باستثناء الطبقة التلافيفية 1 × 1). جنبا إلى جنب مع الطبقة التلافيفية الأولى والطبقة النهائية المتصلة بالكامل، هناك 18 طبقةفي المجموع. لذلك، يُعرف هذا النموذج باسم ResNet-18. من خلال تكوين أعداد مختلفة من القنوات والكتل المتبقيةفي الوحدة النمطية module، يمكننا إنشاء نماذج مختلفة من ResNet، مثل أعمق 152طبقة ResNet-152. على الرغم من أن البنية الرئيسية لـ ResNet تشبه بنية GoogLeNet، إلا أن بنية ResNet أبسط وأسهلفي التعديل. أدت كل هذه العوامل إلى الاستخدام السريع والواسع النطاق لشبكة ResNet. الشكل 8.6.4 يصور كامل ResNet-18.



الشكل 8.6.4 بُنية ResNet-18.

قبل تدريب ResNet، دعنا نلاحظ كيف يتغير شكل الإدخال عبر وحدات مختلفةفي ResNet. كما هو الحالفي جميع البنى السابقة، تنخفض الدقة بينما يزداد عدد القنوات لأعلى حتى النقطة التي تجمع فيها طبقة تجميع المتوسط العالمي كل المعالم.

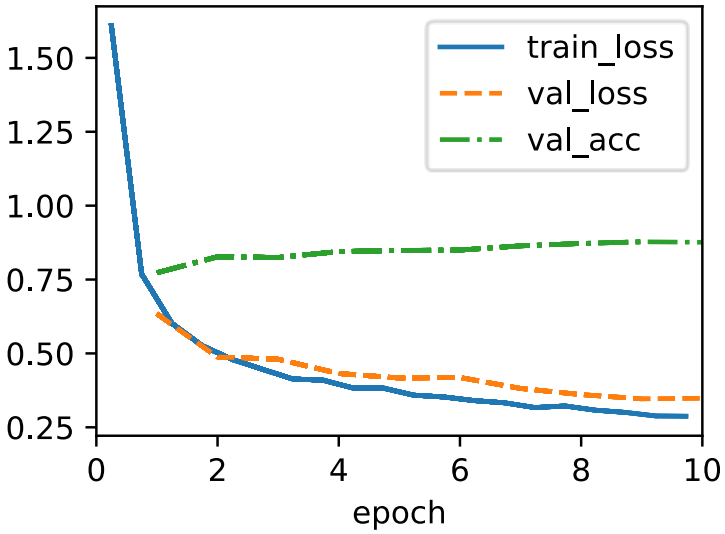
```
class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__(((2, 64), (2, 128), (2, 256),
(2, 512)),
                         lr, num_classes)

ResNet18().layer_summary((1, 96, 96, 1))
```

```
Sequential output shape:     (1, 24, 24, 64)
Sequential output shape:     (1, 24, 24, 64)
Sequential output shape:     (1, 12, 12, 128)
Sequential output shape:     (1, 6, 6, 256)
Sequential output shape:     (1, 3, 3, 512)
Sequential output shape:     (1, 10)
```

8.6.4. التدريب Training

نقوم بتدريب ResNet على مجموعة بيانات Fashion-MNIST، تمامًا كما كان من قبل. ResNet هي بُنية قوية ومرنة. يوضح التدريب على التقاط الرسم وخطأ التحقق من الصحة وجود فجوة كبيرة بين كلا الرسمين البيانيين، حيث يكون خطأ التدريب أقل بكثير. للحصول على شبكة من هذه المرونة، فإن المزيد من بيانات التدريب ستوفر فائدة كبيرةفي سد الفجوة وتحسين الدقة.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
with d2l.try_gpu():
    model = ResNet18(lr=0.01)
    trainer.fit(model, data)
```

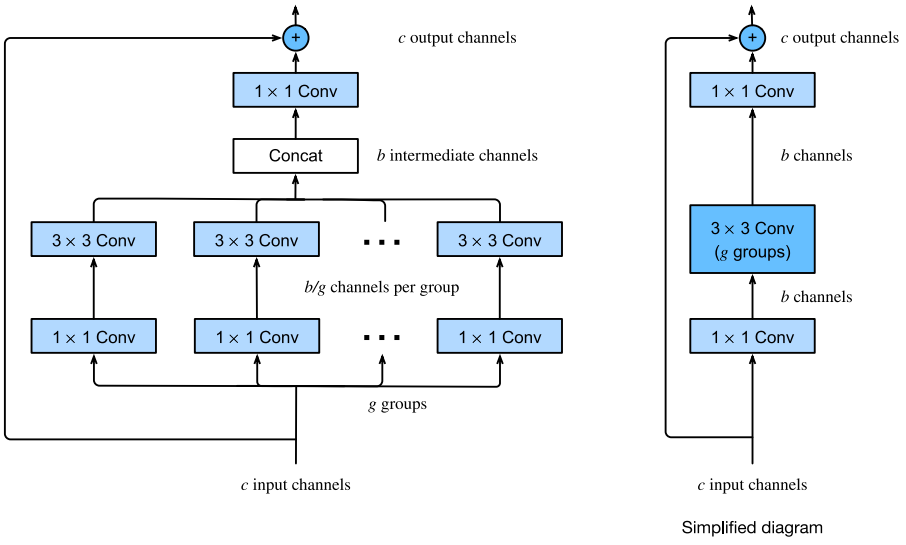


8.6.5. ResNeXt

أحد التحديات التي يواجهها المرءفي تصميم ResNetهو الموازنة trade-off بين اللاخطية والأبعاد داخل كتلة معينة. بمعنى، يمكننا إضافة المزيد من اللاخطية عن طريق زيادة عدد

الطبقات، أو عن طريق زيادة عرض التلافيف. تتمثل الإستراتيجية البديلةفي زيادة عدد القنوات التي يمكنها نقل المعلومات بين الكتل. لسوء الحظ، يأتي هذا الأخير مع عقوبة تربيعية quadratic penalty لأن التكلفة الحسابية لاستيعاب القنوات وقنوات البث تتناسب مع $\mathcal{O}(c_i \cdot c_o)$ (انظر مناقشتنافي القسم 7.4).

يمكننا أخذ بعض الإلهام من كتلة الاستهلال Inception blockفي الشكل 8.4.1 الذي يحتوي على معلومات تتدفق عبر الكتلةفي مجموعات منفصلة. أدى تطبيق فكرة المجموعات المستقلة المتعددة على مجموعة ResNetفي الشكل 8.6.3 إلى تصميم ResNeXt (Xie et al.، 2017). يختلف ResNeXt عن مجموعة التحولات transformationفي البداية، حيث يتبنى نفس التحولفي جميع الفروع، مما يقلل من الحاجة إلى الضبط اليدوي لكل فرع.



الشكل 8.6.5 كتلة ResNeXt. استخدام الالتفاف المجمّع مع المجموعات g أسرع g مرة من الالتفاف الكثيف dense convolution. إنها كتلة بقايا عنق الزجاجة عندما يكون عدد القنوات الوسيطة b أقل من c.

يُطلق على تقسيم الالتفاف من القنوات c_i إلى c_o مجموعة من مجموعات الحجم c_i/g التي تولد مخرجات الحجم c_o/g، بشكل مناسب تمامًا، الالتفاف المجمّع grouped convolution. يتم تقليل التكلفة الحسابية (بالتناسب) من $\mathcal{O}(c_i \cdot c_o)$ إلى $\mathcal{O}(g \cdot (c_i/g) \cdot (c_o/g)) = \mathcal{O}(c_i \cdot c_o/g)$، أي أنها أسرع g مرات. والأفضل من ذلك، يتم أيضًا تقليل عدد المعلمات اللازمة لتوليد الإخراج من مصفوفة $c_i \times c_o$ إلى مصفوفات أصغر حجمًا

$(c_i/g) \times (c_o/g)$، ومرة أخرى يتم تقليلها g مرة أخرى. فيما يلي نفترض أن كلا c_i و c_o يقبل القسمة على g.

التحدي الوحيدفي هذا التصميم هو عدم تبادل أي معلومات بين المجموعات g. تعدل كتلة ResNeXtفي الشكل 8.6.5 هذا بطريقتين: الالتفاف المجمع مع النواة 3×3 يقع بين التفافين 1×1. الثانية تخدم واجب مزدوجفي تغيير عدد القنوات مرة أخرى. وتتمثل الفائدةفي أننا ندفع $\mathcal{O}(c \cdot b)$ تكلفة نواة 1×1 فقط ويمكن أن نتعامل مع تكلفة $\mathcal{O}(b^2/g)$ للنواة 3×3. على غرار تنفيذ الكتلة المتبقيةفي القسم 8.6.2، يتم استبدال الاتصال المتبقي (وبالتالي معممة generalized) بواسطة الالتفاف 1×1.

يوفر الشكل الصحيحفي الشكل 8.6.5 ملخصًا أكثر إيجازًا عن فدرة الشبكة الناتجة. ستلعب أيضًا دورًا رئيسيًافي تصميم شبكات CNN الحديثة العامةفي القسم 8.8. لاحظ أن فكرة التلافيف المجمعة تعود إلى تطبيق AlexNet، (Krizhevsky et al., 2012). عند توزيع الشبكة عبر وحدتي GPU بذاكرة محدودة، تعامل التطبيق مع كل وحدة معالجة رسومات كقناة خاصة بها دون أي آثار سيئة.

يأخذ التنفيذ التالي لفئة `ResNeXtBlock` مدخلات المجموعات (g) ، مع قنوات وسيطة `bot_channels` (b) (عنق الزجاجة). أخيرًا، عندما نحتاج إلى تقليل ارتفاع العرض وعرضه، فإننا نضيف خطوة 2 من خلال تعيين `use_1x1conv=True`، `strides=2`.

```
class ResNeXtBlock(tf.keras.Model):  #@save
    """The ResNeXt block."""
    def __init__(self, num_channels, groups, bot_mul,
use_1x1conv=False,
                 strides=1):
        super().__init__()
        bot_channels = int(round(num_channels *
bot_mul))
        self.conv1 =
tf.keras.layers.Conv2D(bot_channels, 1, strides=1)
        self.conv2 =
tf.keras.layers.Conv2D(bot_channels, 3, strides=strides,

padding="same",

groups=bot_channels//groups)
        self.conv3 =
tf.keras.layers.Conv2D(num_channels, 1, strides=1)
```

```
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.bn2 = tf.keras.layers.BatchNormalization()
        self.bn3 = tf.keras.layers.BatchNormalization()
        if use_1x1conv:
            self.conv4 =
tf.keras.layers.Conv2D(num_channels, 1,

strides=strides)
            self.bn4 =
tf.keras.layers.BatchNormalization()
        else:
            self.conv4 = None

    def call(self, X):
        Y =
tf.keras.activations.relu(self.bn1(self.conv1(X)))
        Y =
tf.keras.activations.relu(self.bn2(self.conv2(Y)))
        Y = self.bn3(self.conv3(Y))
        if self.conv4:
            X = self.bn4(self.conv4(X))
        return tf.keras.activations.relu(Y + X)
```

استخدامه مشابه تمامًا لاستخدام `ResNetBlock` الذي تمت مناقشته سابقًا. على سبيل المثال، عند استخدام (`use_1x1conv=False`، `strides=1`)، يكون المدخلات والمخرجات من نفس الشكل. وبدلاً من ذلك، فإن إعداد `use_1x1conv=True`، `strides=2` ينقص ارتفاع الناتج وعرضه إلى النصف.

```
blk = ResNeXtBlock(32, 16, 1)
X = tf.random.normal((4, 96, 96, 32))
Y = blk(X)
Y.shape
```

```
TensorShape([4, 96, 96, 32])
```

8.6.6. الملخص والمناقشة

تعتبر فئات الدوال المتداخلة Nested function classes مرغوبة لأنها تسمح لنا بالحصول على فئات دوال أكثر قوة بدلاً من فئات دوال مختلفة بمهارة عند إضافة سعة. تتمثل إحدى طرق تحقيق ذلكفي السماح لطبقات إضافية بالمرور عبر المدخلات إلى المخرجات. تسمح الوصلات المتبقية بذلك. نتيجة لذلك، يغير هذا التحيز الاستقرائي من كون الدوال البسيطة من الشكل $f(\mathbf{x}) = 0$ إلى دوال بسيطة تبدو وكأنها $f(\mathbf{x}) = \mathbf{x}$.

يمكن أن يتعلم التعيين المتبقي residual mapping دالة الهوية identity function بسهولة أكبر، مثل دفع المعلماتفي طبقة الوزن إلى الصفر. يمكننا تدريب شبكة عصبية عميقة فعالة من خلال وجود الكتل المتبقية residual blocks. يمكن أن تنتشر المدخلات بشكل أسرع من خلال الوصلات المتبقية عبر الطبقات residual connections across layers. نتيجة لذلك، يمكننا بالتالي تدريب شبكات أعمق بكثير. على سبيل المثال، سمحت ورقة ResNet الأصلية (He et al.، 2016) بما يصل إلى 152 طبقة. فائدة أخرى للشبكات المتبقية هي أنها تسمح لنا بإضافة طبقات، تمت تهيئتها كدالة هوية، أثناء عملية التدريب. بعد كل شيء، السلوك الافتراضي للطبقة هو السماح للبيانات بالمرور دون تغيير. يمكن أن يؤدي ذلك إلى تسريع تدريب الشبكات الكبيرة جدًافي بعض الحالات.

قبل التوصيلات المتبقية residual connections، تم إدخال تجاوز المسارات bypassing paths بوحدات بوابات لتدريب شبكات الطرق السريعة بشكل فعال مع أكثر من 100 طبقة (Srivastava et al.، 2015). باستخدام دوال الهوية كتجاوز المسارات، كان أداء شبكة ResNet جيدًا بشكل ملحوظفي مهام الرؤية الحاسوبية المتعددة. كان للوصلات المتبقية تأثير كبير على تصميم الشبكات العصبية العميقة اللاحقة، لكل من الطبيعة التلافيفية والمتسلسلة. كما سنقدم لاحقًا، تتبنى بنية المحولات (Vaswani et al.، 2017) اتصالات متبقية (جنبًا إلى جنب مع خيارات التصميم الأخرى) وهي منتشرةفي مجالات متنوعة مثل اللغة والرؤية والكلام والتعلم المعزز.

يعد ResNeXt مثالاً على كيفية تطور تصميم الشبكات العصبية التلافيفية بمرور الوقت: من خلال كونه أكثر اقتصادافي الحساب وتداوله مع حجم عمليات التنشيط (عدد القنوات)، فإنه يسمح بشبكات أسرع وأكثر دقة بتكلفة أقل. طريقة بديلة لعرض التلافيف المجمعة grouped convolutions هي التفكيرفي مصفوفة كتلة قطرية block-diagonal matrix للأوزان التلافيفية. لاحظ أن هناك عددًا قليلاً جدًا من هذه "الحيل" التي تؤدي إلى شبكات أكثر كفاءة. على سبيل المثال، يحاكي ShiftNet، 2018(Wu et al.,2018) تأثيرات الالتفاف ، ببساطة عن طريق إضافة عمليات تنشيط متغيرة shifted activations إلى القنوات، مما يوفر تعقيدًا متزايدًا للدوال، هذه المرة دون أي تكلفة حسابية.

من السمات الشائعة للتصميمات التي ناقشناها حتى الآن أن تصميم الشبكة يدوي إلى حد ما، ويعتمد بشكل أساسي على براعة المصممفي العثور على المعلمات الفائقة للشبكة "الصحيحة". في حين أنه من الواضح أنه مجدي، إلا أنه مكلف للغاية من حيث الوقت البشري وليس هناك ما يضمن أن النتيجة مثالية بأي شكل من الأشكال. سنناقشفي القسم 8.8 عددًا من الاستراتيجيات للحصول على شبكات عالية الجودة بطريقة أكثر آلية. على وجه الخصوص، سنراجع فكرة

مساحات تصميم الشبكة network design spaces التي أدت إلى نماذج RegNetX / Y، (2020,Radosavovic et al.).

8.6.7. التمارين

1. ما هي الاختلافات الرئيسية بين كتلة الاستهلال Inception block في الشكل 8.4.1 والكتلة المتبقية residual block؟ كيف يقارنون من حيث الحساب والدقة وفئات الدوال التي يمكنهم وصفها؟
2. ارجع إلى الجدول1 في مقالة ResNet، (2016,He et al.) لتنفيذ انواع مختلفة different variants للشبكة.
3. بالنسبة للشبكات الأعمق، تقدم شبكة ResNet بُنية "عنق الزجاجة bottleneck" لتقليل تعقيد النموذج. حاول تنفيذه.
4. في الإصدارات اللاحقة من ResNet، قام المؤلفون بتغيير بنية "الالتفاف، وتسوية الدفعات، والتفعيل" إلى بُنية "تسوية الدُفعات، والتنشيط، والالتفاف". قم بإجراء هذا التحسين بنفسك. انظر الشكل1 في He et al. (2016) لمزيد من التفاصيل.
5. لماذا لا يمكننا فقط زيادة تعقيد الدوال دون قيود، حتى لو كانت فئات الدوال متداخلة؟

8.7 الشبكات كثيفة الاتصال Densely Connected (DenseNet) Networks

غيّرت ResNet بشكل كبير طريقة عرض كيفية تحديد الدوال في الشبكات العميقة. DenseNet (شبكة تلافيفية كثيفة) هي إلى حد ما الامتداد المنطقي لهذا (Huang et al.، 2017). تتميز DenseNet بكل من نمط الاتصال حيث تتصل كل طبقة بجميع الطبقات السابقة وعملية التسلسل concatenation operation (بدلاً من عامل الإضافة في ResNet) للحفاظ على الميزات من الطبقات السابقة وإعادة استخدامها. لفهم كيفية الوصول إليها، دعنا نأخذ منعطفًا بسيطًا في الرياضيات.

8.7.1. من ResNet إلى DenseNet

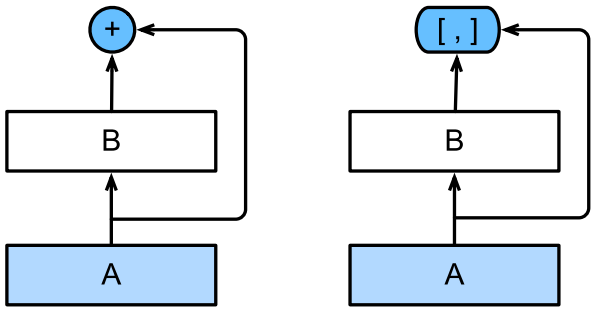
استدعي توسعة تايلور Taylor expansion للدوال. بالنسبة لهذه النقطة $x = 0$ يمكن كتابتها كـ

$$f(x) = f(0) + x \cdot [f'(0) + x \cdot [\frac{f''(0)}{2!} + x \cdot [\frac{f'''(0)}{3!} + \cdots]]].$$

النقطة الأساسية هي أنها تحلل الدالة إلى مصطلحات ذات ترتيب أعلى بشكل متزايد. على نفس المنوال، تحلل ResNet الدوال إلى:

$$f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x}).$$

وهذا يعني أن ResNet تتحلل f إلى مصطلح خطي بسيط ومصطلح غير خطي أكثر تعقيدًا. ماذا لو أردنا التقاط (وليس بالضرورة إضافة) المعلومات التي تتجاوز المصطلحين؟ أحد هذه الحلول هو DenseNet، (Huang et al., 2017).

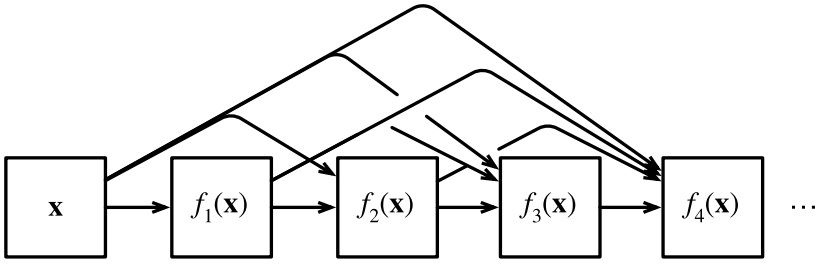


الشكل 8.7.1 الفرق الرئيسي بين ResNet (يسار) وDenseNet (يمين)في التوصيلات عبر الطبقات: استخدام إلاضافة addition واستخدام التسلسل concatenation.

كما هو مبينفي الشكل 8.7.1، فإن الاختلاف الرئيسي بين ResNet وDenseNet هو أنهفي الحالة الأخيرة تكون المخرجات متسلسلة (يُشار إليها بـ [,]) بدلاً من إضافتها. نتيجة لذلك، نقوم بإجراء تعيين من $\mathbf{x}$ الى قيمها بعد تطبيق تسلسل معقد بشكل متزايد من الدوال:

$$\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots].$$

في النهاية، يتم دمج كل هذه الدوالفي MLP لتقليل عدد الميزات مرة أخرى. من حيث التنفيذ، هذا بسيط للغاية: بدلاً من إضافة المصطلحات، نقوم بتجميعها. نشأ اسم DenseNet من حقيقة أن الرسم البياني للتبعية بين المتغيرات يصبح كثيفًا جدًا. ترتبط الطبقة الأخيرة من هذه السلسلة بكثافة بجميع الطبقات السابقة. الوصلات الكثيفة dense connections موضحةفي الشكل 8.7.2.



الشكل 8.7.2 اتصالات كثيفةفي DenseNet. لاحظ كيف تزداد الأبعاد مع العمق.

المكونات الرئيسية التي تتكون منها شبكة DenseNet هي كتل كثيفة dense blocks وطبقات انتقالية transition layers. يحدد الأول كيفية تسلسل المدخلات والمخرجات، بينما يتحكم الأخيرفي عدد القنوات بحيث لا يكون كبيرًا جدًا، نظرًا لأن التوسع $\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), \ldots]$ يمكن أن يكون عالي الأبعاد.

8.7.2. كتل كثيفة Dense Blocks

تستخدم DenseNet بُنية "تسوية الدُفعات، التنشيط، والتفاف" المعدلة لـ ResNet (راجع التمرين في القسم 8.6). أولاً، نقوم بتنفيذ هيكل كتلة الالتفاف.

```
import tensorflow as tf
from d2l import tensorflow as d2l


class ConvBlock(tf.keras.layers.Layer):
    def __init__(self, num_channels):
        super(ConvBlock, self).__init__()
        self.bn = tf.keras.layers.BatchNormalization()
        self.relu = tf.keras.layers.ReLU()
        self.conv = tf.keras.layers.Conv2D(
            filters=num_channels, kernel_size=(3, 3),
padding='same')

        self.listLayers = [self.bn, self.relu,
self.conv]

    def call(self, x):
        y = x
        for layer in self.listLayers.layers:
            y = layer(y)
        y = tf.keras.layers.concatenate([x,y], axis=-1)
        return y
```

تتكون الكتلة الكثيفة من كتل التفاف متعددة، كل منها يستخدم نفس عدد قنوات الإخراج. ومع ذلك،في الانتشار الأمامي، نقوم بربط المدخلات والمخرجات لكل كتلة التفاف على بُعد القناة. يسمح لنا التقييم الكسول Lazy evaluation بضبط الأبعاد تلقائيًا.

```
class DenseBlock(tf.keras.layers.Layer):
    def __init__(self, num_convs, num_channels):
        super(DenseBlock, self).__init__()
        self.listLayers = []
        for _ in range(num_convs):
```

```
self.listLayers.append(ConvBlock(num_channels))

    def call(self, x):
        for layer in self.listLayers.layers:
            x = layer(x)
        return x
```

في المثال التالي، نحدد مثيل `DenseBlock` مع كتلتين التفاف من 10 قنوات إخراج. عند استخدام إدخال بثلاث قنوات، سنحصل على إخراج مع قنوات $3 + 10 + 10 = 23$. يتحكم عدد قنوات كتلة الالتفاففي النموفي عدد قنوات الإخراج بالنسبة إلى عدد قنوات الإدخال. يشار إلى هذا أيضًا بمعدل النمو growth rate.

```
blk = DenseBlock(2, 10)
X = tf.random.uniform((4, 8, 8, 3))
Y = blk(X)
Y.shape
```

```
TensorShape([4, 8, 8, 23])
```

8.7.3. طبقات الانتقال Transition Layers

نظرًا لأن كل كتلة كثيفة ستزيد من عدد القنوات، فإن إضافة الكثير منها سيؤدي إلى نموذج معقد للغاية. يتم استخدام طبقة انتقالية transition layer للتحكمفي مدى تعقيد النموذج. يقلل من عدد القنوات باستخدام الالتفاف. علاوة على ذلك، فإنه يخفض الارتفاع والعرض إلى النصف عبر تجميع متوسط average pooling بخطوة 2.

```
class TransitionBlock(tf.keras.layers.Layer):
    def __init__(self, num_channels, **kwargs):
        super(TransitionBlock, self).__init__(**kwargs)
        self.batch_norm =
tf.keras.layers.BatchNormalization()
        self.relu = tf.keras.layers.ReLU()
        self.conv = tf.keras.layers.Conv2D(num_channels,
kernel_size=1)
        self.avg_pool =
tf.keras.layers.AvgPool2D(pool_size=2, strides=2)

    def call(self, x):
        x = self.batch_norm(x)
        x = self.relu(x)
        x = self.conv(x)
        return self.avg_pool(x)
```

قم بتطبيق طبقة انتقالية بها 10 قنوات على إخراج الكتلة الكثيفةفي المثال السابق. هذا يقلل من عدد قنوات الإخراج إلى 10، ويقلل الارتفاع والعرض إلى النصف.

```
blk = TransitionBlock(10)
blk(Y).shape
```

```
TensorShape([4, 4, 4, 10])
```

8.7.4. نموذج DenseNet

بعد ذلك، سنقوم ببناء نموذج DenseNet. تستخدم DenseNet أولاً نفس الطبقة التلافيفية الفردية وطبقة تجميع الحد الاقصى كمافي ResNet.

```
class DenseNet(d2l.Classifier):
    def b1(self):
        return tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(
                64, kernel_size=7, strides=2,
padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU(),
            tf.keras.layers.MaxPool2D(
                pool_size=3, strides=2,
padding='same')])
```

بعد ذلك، على غرار الوحدات الأربع المكونة من الكتل المتبقية التي تستخدمها ResNet، تستخدم DenseNet أربع كتل كثيفة. على غرار ResNet، يمكننا تعيين عدد الطبقات التلافيفية المستخدمةفي كل كتلة كثيفة. هنا، قمنا بتعيينه على 4، بما يتوافق مع نموذج ResNet-18في القسم 8.6. علاوة على ذلك، قمنا بتعيين عدد القنوات (أي معدل النمو) للطبقات التلافيفيةفي الكتلة الكثيفة إلى 32، لذلك ستتم إضافة 128 قناة إلى كل كتلة كثيفة.

في ResNet، يتم تقليل الارتفاع والعرض بين كل وحدة بواسطة كتلة متبقية بخطوة 2. هنا، نستخدم طبقة الانتقال لخفض الارتفاع والعرض إلى النصف وخفض عدد القنوات إلى النصف. على غرار ResNet، يتم توصيل طبقة تجميع عالمية وطبقة متصلة بالكاملفي النهاية لإنتاج الناتج.

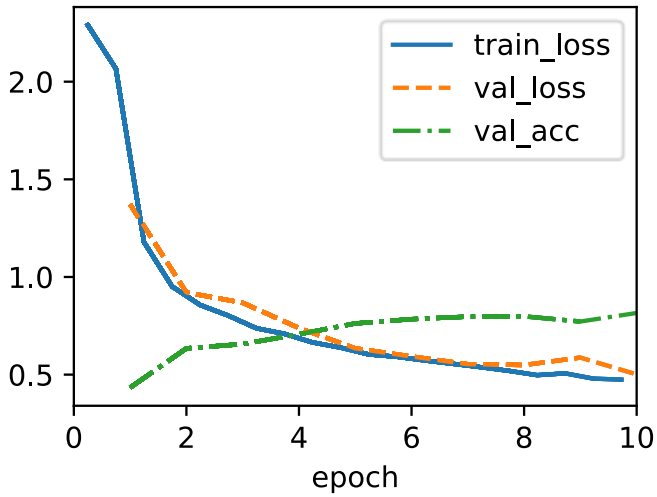
```
@d2l.add_to_class(DenseNet)
def __init__(self, num_channels=64, growth_rate=32,
arch=(4, 4, 4, 4),
             lr=0.1, num_classes=10):
    super(DenseNet, self).__init__()
    self.save_hyperparameters()
```

```
        self.net = tf.keras.models.Sequential(self.b1())
        for i, num_convs in enumerate(arch):
            self.net.add(DenseBlock(num_convs, growth_rate))
            # The number of output channels in the previous
dense block
            num_channels += num_convs * growth_rate
            # A transition layer that halves the number of
channels is added
            # between the dense blocks
            if i != len(arch) - 1:
                num_channels //= 2
                self.net.add(TransitionBlock(num_channels))
        self.net.add(tf.keras.models.Sequential([
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU(),
            tf.keras.layers.GlobalAvgPool2D(),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(num_classes)]))
```

8.7.5. التدريب Training

نظرًا لأننا نستخدم شبكة أعمق هنا،في هذا القسم، سنقوم بتقليل ارتفاع الإدخال وعرضه من 224 إلى 96 لتبسيط الحساب.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
with d2l.try_gpu():
    model = DenseNet(lr=0.01)
    trainer.fit(model, data)
```



8.7.6. الملخص والمناقشة

المكونات الرئيسية التي تتكون منها DenseNet هي كتل كثيفة dense blocks وطبقات انتقالية transition layers. بالنسبة للأخير، نحتاج إلى إبقاء الأبعاد تحت السيطرة عند تكوين الشبكة عن طريق إضافة طبقات انتقالية تقلص عدد القنوات مرة أخرى. فيما يتعلق بالاتصالات عبر الطبقات cross-layer connections، على عكس شبكة ResNet، حيث يتم إضافة المدخلات والمخرجات معًا، تقوم DenseNet بتجميع المدخلات والمخرجات على بُعد القناة. على الرغم من أن عمليات التسلسل هذه تعيد استخدام الميزات لتحقيق كفاءة حسابية، إلا أنها للأسف تؤدي إلى استهلاك كبير لذاكرة وحدة معالجة الرسومات GPU. نتيجة لذلك، قد يتطلب تطبيق DenseNet تطبيقات أكثر كفاءةفي استخدام الذاكرة والتي قد تزيد من وقت التدريب (Pleiss et al.، 2017).

8.7.7. التمارين

1. لماذا نستخدم تجميع المتوسط average pooling بدلاً من تجميع الحد الأقصى max-poolingفي الطبقة الانتقالية؟
2. إحدى المزايا المذكورةفي مقالة DenseNet هي أن معلمات نموذجها أصغر من تلك الخاصة بـ ResNet. لماذا هذا هو الحال؟
3. إحدى المشكلات التي تم انتقاد DenseNet بسببها هي استهلاكها العالي للذاكرة.
   1. هل هذا هو الحال فعلا؟ حاول تغيير شكل الإدخال الى 224×224 لرؤية الاستهلاك الفعلي لذاكرة وحدة معالجة الرسومات بشكل تجريبي.
   2. هل يمكنك التفكيرفي وسيلة بديلة لتقليل استهلاك الذاكرة؟ كيف تريد تغيير أطار العمل؟
4. نفذ إصدارات DenseNet المختلفة الواردةفي الجدول 1 من مقالة DenseNet، (Huang et al.,2017).
5. صمم نموذجًا قائمًا على MLP من خلال تطبيق فكرة DenseNet. قم بتطبيقه على مهمة التنبؤ بسعر السكنفي القسم 5.7.

8.8 تصميم معماريات شبكة الالتفاف Designing Convolution Network Architectures

شهد العقد الأول من القرن الحادي والعشرين تحولًا من هندسة الميزات feature engineering إلى هندسة الشبكات network engineeringفي الرؤية الحاسوبية. نظرًا لأن AlexNet (القسم 8.1) تغلب على نماذج الرؤية الحاسوبية التقليدية على ImageNet، فقد تم نشر شبكات عميقة جدًا عن طريق تكديس نفس الكتل، وخاصة التلافيف 3×3، بواسطة

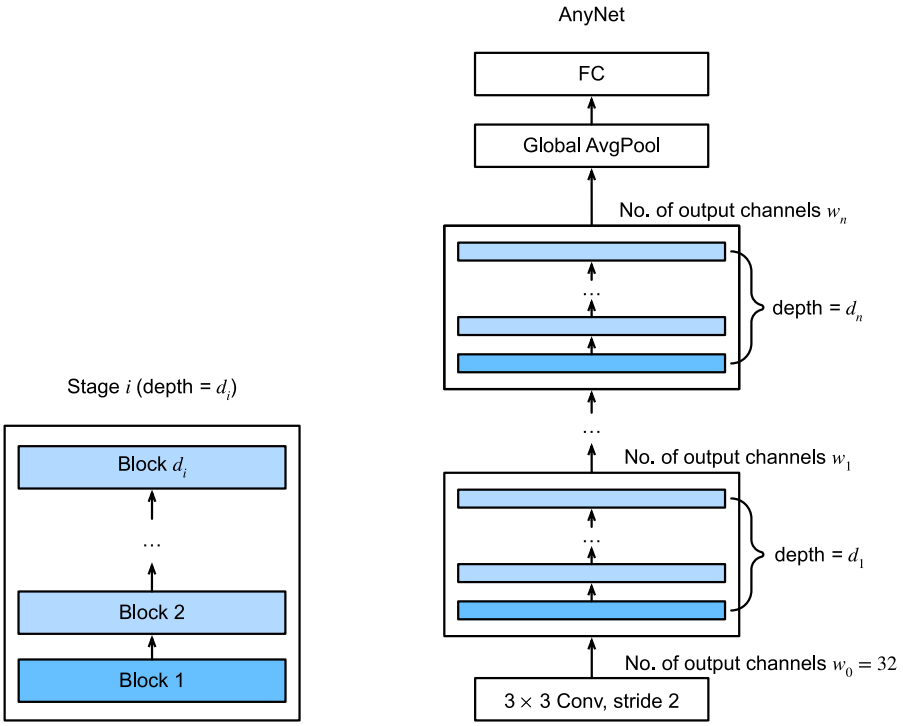
شبكات VGG (القسم 8.2). تضيف الشبكةفي الشبكة (القسم 8.3) العناصر غير الخطية المحلية عبر التلافيف 1 × 1 وتستخدم متوسط التجميع العالمي لتجميع المعلومات عبر جميع المواقع. GoogLeNet (القسم 8.4) عبارة عن شبكة متعددة الفروع تجمع مزايا شبكة VGG والشبكة الموجودةفي الشبكة، حيث تتبنى كتلة الاستهلال Inception block استراتيجية التحويلات المتوازية المتسلسلة. تقوم ResNets (القسم 8.6) بتكديس الكتل المتبقية، وهي عبارة عن شبكات فرعية ثنائية الفروع تستخدم تعيين الهوية identity mappingفي فرع واحد. تعمم DenseNets (القسم 8.7) البنى المتبقية. تشمل البنى الأخرى البارزة MobileNets التي تستخدم التعلم الشبكي network learning لتحقيق دقة عاليةفي الإعدادات محدودة الموارد (Howard et al.، 2019)، وشبكات Squeeze-and-Excitation Networks (SENets) التي تسمح بنقل المعلومات بكفاءة بين القنوات (Hu et al.، 2018) وEfficientNets، (Tan and Le,2019) التي تعمل على توسيع نطاق الشبكات عبر البحث عن العمارة العصبية neural architecture search.

على وجه التحديد، البحث عن العمارة العصبية (NAS)، (Liu et al.,2018)، (Zoph and Le,2016) هو عملية أتمتة بنى الشبكات العصبية. نظرًا لمساحة البحث الثابتة، تستخدم NAS إستراتيجية بحث لتحديد بُنية ضمن مساحة البحث تلقائيًا بناءً على تقدير الأداء الذي تم إرجاعه. نتيجة NAS هي مثيل شبكة واحد.

بدلاً من التركيز على تصميم مثل هذه الحالات الفردية، يتمثل النهج البديلفي تصميم مساحات تصميم الشبكة network design spaces التي تميز مجموعات الشبكات (Radosavovic et al.، 2020). تجمع هذه الطريقة بين قوة التصميم اليدوي وNAS. من خلال الإجراءات شبه الآلية (كما هو الحالفي NAS)، يستكشف تصميم مساحات تصميم الشبكة الجانب الهيكلي لتصميم الشبكة من مساحة تصميم AnyNet الأولية. ثم يشرعفي اكتشاف مبادئ التصميم (كما هو الحالفي التصميم اليدوي) التي تؤدي إلى شبكات بسيطة ومنتظمة: RegNets. قبل إلقاء الضوء على مبادئ التصميم هذه، دعنا نبدأ بمساحة التصميم الأولية.

8.8.1. مساحة تصميم AnyNet

تسمى مساحة التصميم الأولية AnyNet، وهي مساحة تصميم غير مقيدة نسبيًا، حيث يمكننا التركيز على استكشاف بُنية الشبكة بافتراض الكتل القياسية الثابتة مثل ResNeXt (القسم 8.6.5). على وجه التحديد، تتضمن بُنية الشبكة عناصر مثل عدد الكتلnumber of blocks وعدد قنوات الإخراجفي كل مرحلة، وعدد المجموعات number of groups (عرض المجموعة) ونسبة الاختناق bottleneck ratio داخل كل كتلة ResNeXt.



الشكل 8.8.1 مساحة تصميم AnyNet. إلى جانب عدد المجموعات ونسبة الاختناق داخل كل كتلة، تشمل خيارات التصميم: العمق d_i وعدد قنوات الإخراج w_i لأي مرحلة i.

تظهر مساحة تصميم AnyNetفي الشكل 8.8.1. تبدأ هذه الشبكة بجذع stem، يتبعها جسم body بمراحل تحول n ، ورأس head نهائي. بشكل ملموس، فإن جذع الشبكة عبارة عن التفاف 3 × 3 مع الخطوة 2 التي تقسم ارتفاع وعرض صورة الإدخال إلى النصف. رأس الشبكة عبارة عن تجميع متوسط عالمي يتبعه طبقة متصلة بالكامل للتنبؤ بفئة الإخراج. لاحظ أن جذع الشبكة ورأسها يظلان ثابتًا وبسيطًا، بحيث يركز التصميم على هيكل الشبكة الذي يعد مركزيًا للأداء. على وجه التحديد، يتكون جسم الشبكة من مراحل التحويل n (n هو معطى)، حيث تتكون المرحلة i من كتل ResNeXt d_i مع قنوات الإخراج w_i ، وتنخفض بشكل تدريجي الارتفاع والعرض إلى النصف عبر الكتلة الأولى (إعداد `use_1x1conv=True`، `strides=2` في `d2l.ResNeXtBlock` في القسم 8.6.5). دعنا نشير كذلك إلى نسبة الاختناق وعدد المجموعات (عرض المجموعة) داخل كل كتلة ResNeXt للمرحلة i كـ b_i و g_i ، على التوالي. بشكل عام، على الرغم من هيكل الشبكة المباشر، تنوع b_i, g_i و w_i ، و d_i ينتج عنه عدد كبير من الشبكات الممكنةفي مساحة تصميم AnyNet.

لتنفيذ AnyNet، نحدد أولاً جذع شبكتها.

```
import tensorflow as tf
from d2l import tensorflow as d2l


class AnyNet(d2l.Classifier):
    def stem(self, num_channels):
        return tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(num_channels,
kernel_size=3, strides=2,
                                   padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Activation('relu')])
```

تتكون كل مرحلة من العمق لكتل ResNeXt، حيث يحدد `num_channels` عرض الكتلة. لاحظ أن الكتلة الأولى تقسم ارتفاع وعرض صور الإدخال إلى النصف.

```
@d2l.add_to_class(AnyNet)
def stage(self, depth, num_channels, groups, bot_mul):
    net = tf.keras.models.Sequential()
    for i in range(depth):
        if i == 0:
            net.add(d2l.ResNeXtBlock(num_channels,
groups, bot_mul,
                use_1x1conv=True, strides=2))
        else:
            net.add(d2l.ResNeXtBlock(num_channels,
groups, bot_mul))
    return net
```

من خلال وضع جذع الشبكة والجسم والرأس معًا، نكمل تنفيذ AnyNet.

```
@d2l.add_to_class(AnyNet)
def __init__(self, arch, stem_channels, lr=0.1,
num_classes=10):
    super(AnyNet, self).__init__()
    self.save_hyperparameters()
    self.net =
tf.keras.models.Sequential(self.stem(stem_channels))
    for i, s in enumerate(arch):
        self.net.add(self.stage(*s))
    self.net.add(tf.keras.models.Sequential([
        tf.keras.layers.GlobalAvgPool2D(),
```

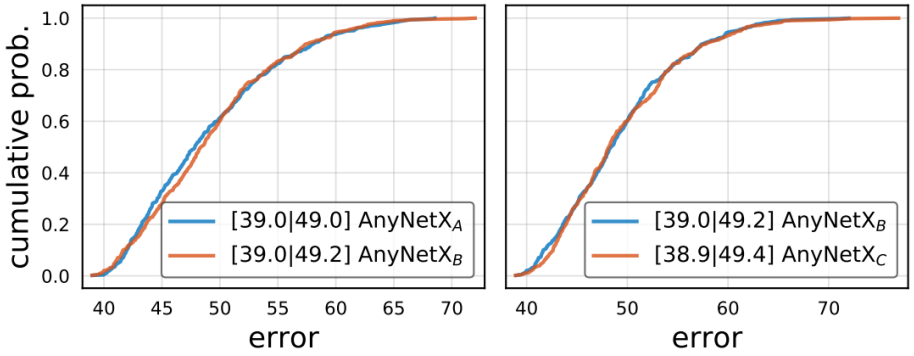
```
tf.keras.layers.Dense(units=num_classes)]))
```

8.8.2. تقييد مساحات التصميم بتوزيعات أخطاء أقل Constraining Design Spaces with Lower Error Distributions

بالنسبة لأي مرحلة من مراحل AnyNet، تكون خيارات التصميم هي نسبة الاختناق b_i وعدد المجموعات g_i داخل كل كتلة وعرض الكتلة w_i والعمق d_i. تبدأ عملية تصميم مساحات الشبكة من بُنية شبكة غير مقيدة نسبيًا تتميز بـ (d_i ، w_i ، g_i، b_i) في مساحة تصميم AnyNet الأولية. ثم تقوم هذه العملية باختبار نماذج تدريجية من مساحة تصميم المدخلات لتقييم توزيع الخطأ (Radosavovic et al.، 2019) كمؤشر للجودة quality indicator لإخراج مساحة تصميم أكثر تقييدًا مع نماذج أبسط قد يكون لها جودة أفضل.

دعونا نوضح بالتفصيل مؤشر الجودة هذا لمساحات التصميم. بالنظر إلى النماذج المأخوذة من بعض مساحات التصميم، تقيس دالة التوزيع التجريبية للخطأ error empirical distribution function، $F(e)$، جزء النماذج التي بها أخطاء e_i أقل من e:

$$F(e) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\,(e_i < e).$$

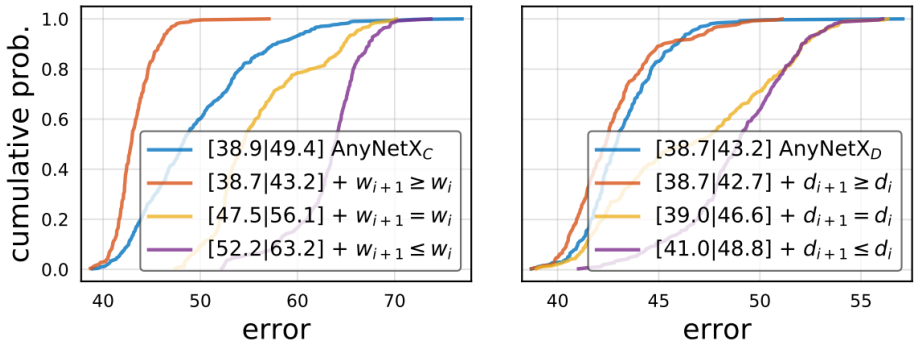


الشكل 8.8.2 مقارنة دوال التوزيع التجريبية للخطأ لمساحات التصميم. تظهر تسميات الشكل خطأ الحد الأدنى والخطأ المتوسط. تؤدي زيادة عرض الشبكة عبر المراحل (AnyNetX$_C$ من إلى AnyNetX$_D$) وزيادة عمق الشبكة عبر المراحل (من AnyNetX$_D$ إلى AnyNetX$_E$) إلى تبسيط مساحة التصميم مع توزيعات أخطاء محسّنة (الشكل مأخوذ من Radosavovic et al.، 2020).

بدءًا من مساحة تصميم AnyNet الأولية غير المقيدة (Radosavovic et al.، 2020)، تؤدي مشاركة نسبة شبكة الاختناق $b_i = b$ لجميع المراحل i إلى مساحة تصميم أكثر تقييدًا. تُظهر نماذج أخذ العينات والتدريب $n = 500$ من كلٍ AnyNetX$_A$ و AnyNetX$_B$, على

يسار الشكل 8.8.2 أن كلا مساحات التصميم لهما نفس الجودة. نظرًا لأن الأبسط هو الأفضل، فإننا نواصل البحث من خلال مشاركة عدد المجموعات $g_i = g$ بشكل إضافي. وهذا يؤدي إلى مزيد من مساحة التصميم المبسطة مع عدم وجود تغيير فعليفي توزيعات الخطأ (يمين الشكل 8.8.2).

يشير التحقيقفي النماذج الجيدة والسيئة من ان AnyNetX_C قد يكون من المفيد زيادة العرض عبر المراحل (Radosavovic et al.، 2020). تجريبيًا، تبسيط AnyNetX_Cو AnyNetX_D مع $w_i \leq w_{i+1}$ لتحسين جودة مساحات التصميم (يسار الشكل 8.8.3). وبالمثل، فإن إضافة المزيد من القيود $d_i \leq d_{i+1}$ لزيادة عمق الشبكة عبر المراحل يعطي أفضل (يمين الشكل 8.8.3).



الشكل 8.8.3 مقارنة دوال التوزيع التجريبية للخطأ لمساحات التصميم. تظهر تسميات الشكل خطأ الحد الأدنى والخطأ المتوسط. تؤدي زيادة عرض الشبكة عبر المراحل (من AnyNetX_C إلى AnyNetX_D) وزيادة عمق الشبكة عبر المراحل (من AnyNetX_D إلى AnyNetX_E) إلى تبسيط مساحة التصميم مع توزيعات أخطاء محسّنة (الشكل مأخوذ من Radosavovic et al.، 2020)).

8.8.3. RegNet

تتكون مساحة التصميم AnyNetX_E الناتجة من شبكات بسيطة تتبع مبادئ التصميم سهلة التفسير:

- مشاركة نسبة شبكة الاختناق $b_i = b$ لجميع المراحل i؛
- مشاركة عدد المجموعات $g_i = g$ لجميع المراحل i؛
- زيادة عرض الشبكة عبر المراحل $w_i \leq w_{i+1}$ ؛
- زيادة عمق الشبكة عبر المراحل: $d_i \leq d_{i+1}$.

باتباع مبادئ التصميم هذه، اقترح (Radosavovic et al.، 2020) قيود خطية كمية للزيادة w_i و d_i ، مما يؤدي إلى RegNetX باستخدام كتل ResNeXt وRegNetY التي تستخدم أيضًا مشغلين من SENets، (Hu et al.,2018). على سبيل المثال، قمنا بتنفيذ متغير RegNetX المكون من 32 طبقة والذي يتميز بـ

- $b_i = 1$;
- $g_i = 16$;
- $w_1 = 32, w_2 = 80$;
- $d_1 = 4, d_2 = 6$.

```
class RegNet32(AnyNet):
    def __init__(self, lr=0.1, num_classes=10):
        stem_channels, groups, bot_mul = 32, 16, 1
        depths, channels = (4, 6), (32, 80)
        super().__init__(
            ((depths[0], channels[0], groups, bot_mul),
             (depths[1], channels[1], groups, bot_mul)),
            stem_channels, lr, num_classes)
```

يمكننا أن نرى أن كل مرحلة من مراحل RegNet تقلل بشكل تدريجي الدقة وتزيد من قنوات الإخراج.

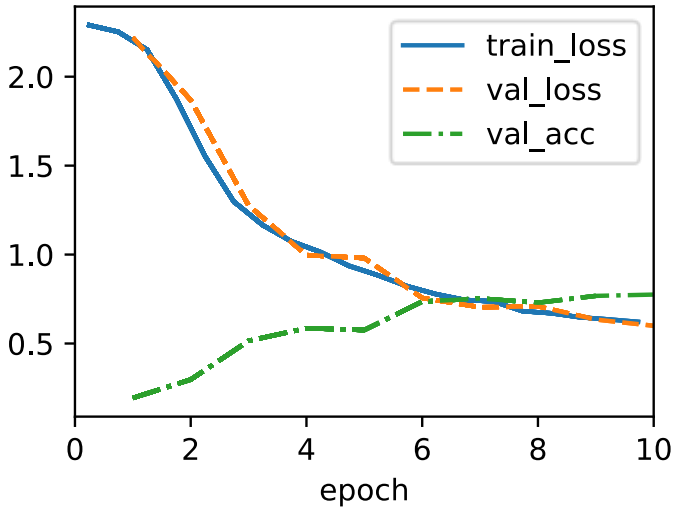
```
RegNet32().layer_summary((1, 96, 96, 1))
```

```
Sequential output shape:     (1, 48, 48, 32)
Sequential output shape:     (1, 24, 24, 32)
Sequential output shape:     (1, 12, 12, 80)
Sequential output shape:     (1, 10)
```

8.8.4. التدريب Training

إن تدريب RegNet المكون من 32 طبقة على مجموعة بيانات Fashion-MNIST هو مثل ما كان عليه من قبل.

```
trainer = d2l.Trainer(max_epochs=10)
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
with d2l.try_gpu():
    model = RegNet32(lr=0.01)
    trainer.fit(model, data)
```



8.8.5. المناقشة

مع الخصائص المرغوبة مثل المحلية locality وثبات الترجمة translation invariance (القسم 7.1) للرؤية، كانت شبكات CNN هي البُنى المهيمنةفي هذا المجال.في الآونة الأخيرة، أثارت المحولات (القسم 11.7) (Dosovitskiy et al.، 2021، Touvron et al.، 2021) و MLPs، (Tolstikhin et al.,2021) أيضًا بحثًا يتجاوز بُنى CNN الراسخة للرؤية. على وجه التحديد، على الرغم من عدم وجود التحيزات الاستقرائية inductive biases المذكورة أعلاه الملازمة لشبكات CNN، فقد حققت محولات الرؤية (القسم 11.8) أداءً متطورًافي تصنيف الصور على نطاق واسعفي أوائل عام 2020، مما يدل على أن قابلية التوسع تتفوق على التحيزات الاستقرائية (Dosovitskiy et al.، 2021). بمعنى آخر، غالبًا ما يكون من الممكن تدريب محولات كبيرة large transformers لتتفوق على شبكات CNN الكبيرةفي مجموعات البيانات الكبيرة. مستوحاة من سلوك التحجيم الفائق superior scaling behavior للمحولات (القسم 11.9) مع الاهتمام الذاتي متعدد الرؤوس multi-head self-attention (القسم 11.5)، تؤدي عملية التحسين التدريجي من بُنية ResNet القياسية نحو تصميم محول الرؤية vision transformer إلى عائلة من شبكات CNN تسمى ConvNeXt النماذج التي تتنافس بشكل إيجابي مع المحولات من أجل الرؤية (Liu et al.، 2022). نحيل القراء المهتمين إلى مناقشات تصميم CNNفي مقالة ConvNeXt، (Liu et al.,2022).

8.8.6. التمارين

1. قم بزيادة عدد المراحل إلى 4. هل يمكنك تصميم RegNet أعمق يعمل بشكل أفضل؟

2. استبدل كتلة ResNeXt بكتلة ResNet. كيف يعمل نموذجك الجديد؟
3. نفذ مثيلات متعددة لعائلة "VioNet" من خلال انتهاك violating مبادئ تصميم RegNet. كيف يؤدون؟ أي من (b_i، g_i، w_i، d_i) هو العامل الأكثر أهمية؟

9

الشبكات العصبية المتكررة

9. الشبكات العصبية المتكررة Recurrent Neural Networks

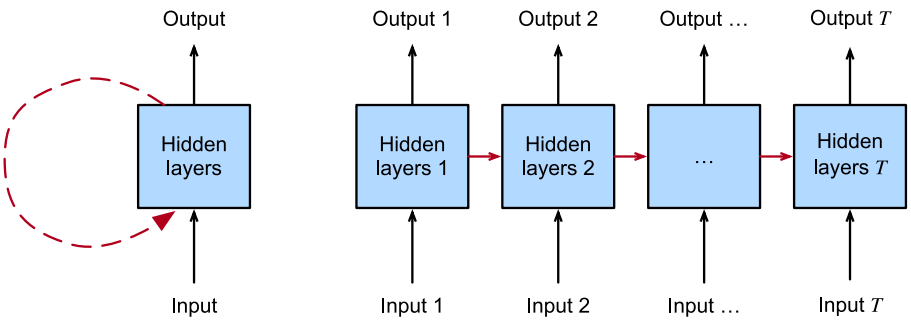
حتى الآن، ركزنا بشكل أساسي على البيانات ذات الطول الثابت fixed-length data. عند تقديم الانحدار الخطي واللوجستيفي القسم 3 والقسم 4 والبيرسيبترون متعددة الطبقاتفي القسم 5، كان من دواعي سرورنا أن نفترض أن كل متجه للميزات يتكون من عدد ثابت من المكونات حيث تتوافق كل ميزة رقمية مع سمة معينة. يطلق على مجموعات البيانات هذه أحيانًا اسم جدولي tabular، لأنه يمكن ترتيبهافي جداول، حيث يحصل كل مثال على صف خاص به، وكل سمة تحصل على عمودها الخاص. بشكل حاسم، مع البيانات المجدولة tabular data، نادرًا ما نفترض أي بُنية معينة فوق الأعمدة.

بعد ذلك،في القسم 7، انتقلنا إلى بيانات الصورة، حيث تتكون المدخلات من قيم البكسل الأولية في كل إحداثيفي الصورة. لا تكاد بيانات الصورة تتناسب مع فاتورة مجموعة البيانات الجدولية النموذجية. هناك، احتجنا إلى استدعاء الشبكات العصبية التلافيفية (CNN) للتعامل مع الهيكل الهرمي والثوابت. ومع ذلك، كانت بياناتنا لا تزال ذات طول ثابت. يتم تمثيل كل صورة من صور Fashion-MNIST كشبكة من قيم البكسل 28 × 28. علاوة على ذلك، كان هدفنا تطوير نموذج ينظر إلى صورة واحدة فقط ثم ينتج تنبؤًا واحدًا. ولكن ما الذي يجب أن نفعله عندما نواجه سلسلة من الصور، كما هو الحالفي مقطع فيديو، أو عند تكليفنا بإنتاج تنبؤ منظم بشكل تسلسلي، كمافي حالة التسميات التوضيحية للصور image captioning؟

تتطلب مهام التعلم التي لا تعد ولا تحصى التعامل مع البيانات المتسلسلة sequential data. يتطلب شرح الصور وتوليف الكلام وتوليد الموسيقى أن تنتج النماذج مخرجات تتكون من تسلسلات.في المجالات الأخرى، مثل توقع السلاسل الزمنية time series prediction وتحليل الفيديو video analysis واسترجاع المعلومات الموسيقية musical information retrieval، يجب أن يتعلم النموذج من المدخلات المتسلسلة. غالبًا ما تنشأ هذه المطالبفي وقت واحد: مهام مثل ترجمة مقاطع نصية من لغة طبيعية إلى أخرى، أو الانخراطفي حوار، أو التحكمفي روبوت، تتطلب نماذج استيعاب وإخراج بيانات منظمة بالتسلسل.

الشبكات العصبية المتكررة (RNNs) هي نماذج التعلم العميق التي تلتقط ديناميكيات التسلسلات عبر الاتصالات المتكررة recurrent connections، والتي يمكن اعتبارها دورات في شبكة العقد. قد يبدو هذا غير منطقيفي البداية. بعد كل شيء، فإن الطبيعة المغذية feedforward nature للشبكات العصبية هي التي تجعل ترتيب الحساب واضحًا. ومع ذلك، يتم تحديد الحواف المتكررة بطريقة دقيقة تضمن عدم حدوث مثل هذا الغموض. يتم إلغاء

التحكمفي الشبكات العصبية المتكررة عبر الخطوات الزمنية time steps (أو خطوات التسلسل sequence steps)، مع تطبيق نفس المعلمات الأساسيةفي كل خطوة. بينما يتم تطبيق الاتصالات القياسية بشكل متزامن لنشر عمليات تنشيط كل طبقة إلى الطبقة اللاحقةفي نفس الخطوة الزمنية، تكون الاتصالات المتكررة ديناميكية، وتمرير المعلومات عبر خطوات الوقت المجاورة. كما يكشف العرض غير المطويunfolded view في الشكل 9.1، يمكن اعتبار شبكات RNN بمثابة شبكات عصبية تلقائية feedforward neural networks حيث تتم مشاركة معلمات كل طبقة (التقليدية والمتكررة conventional and recurrent) عبر خطوات زمنية.



الشكل 9.1 على اليسار يتم تصوير الوصلات المتكررة recurrent connections عبر حواف دورية cyclic edges. على اليمين، نكشف عن RNN بمرور الوقت. هنا، تمتد الحواف المتكررة على خطوات زمنية متجاورة adjacent time steps، بينما يتم حساب الاتصالات التقليدية conventional connections بشكل متزامن.

مثل الشبكات العصبية على نطاق أوسع، تتمتع شبكات RNN بتاريخ طويل يمتد من الانضباط، حيث نشأت كنماذج للدماغ شاعها علماء الإدراك وتم تبنيها لاحقًا كأدوات نمذجة عملية يستخدمها مجتمع التعلم الآلي. كما هو الحال مع التعلم العميق على نطاق أوسع، يتبنى هذا الكتاب منظور التعلم الآلي، مع التركيز على RNNs كأدوات عملية ارتفعت إلى الشعبيةفي 2010 بسبب النتائج المذهلةفي مهام متنوعة مثل التعرف على خط اليد handwriting recognition (Graves et al.، 2008)، الترجمة الآلية machine translation (Sutskever et al.، 2014)، والتعرف على التشخيصات الطبية recognizing medical diagnosis (Lipton et al.، 2016). نوجه القارئ المهتم بمزيد من المواد الأساسية إلى مراجعة شاملة متاحة للجمهور (Lipton et al.، 2015). نلاحظ أيضًا أن التسلسل ليس فريدًا بالنسبة لـ RNNs. على سبيل المثال، يمكن تكييف شبكات CNN التي قدمناها بالفعل للتعامل مع البيانات ذات الطول المتفاوت varying length، على سبيل المثال، الصور ذات الدقة المتفاوتة varying

resolution. علاوة على ذلك، تنازلت RNNs مؤخرًا عن حصة كبيرة من السوق لنماذج المحولات transformer models، والتي سيتم تغطيتهافي القسم 11. ومع ذلك، ارتفعت RNNs إلى الصدارة كنماذج افتراضية للتعامل مع البنية المتسلسلة المعقدةفي التعلم العميق، ولا تزال نماذج أساسية للنمذجة المتسلسلة sequential modeling حتى يومنا هذا. ترتبط قصص RNNs ونمذجة التسلسل ارتباطًا وثيقًا، وهذا فصل عن أبجديات مشاكل نمذجة التسلسل كما هو فصل حول RNNs.

مهدت إحدى الأفكار الرئيسية الطريق لثورةفي نمذجة التسلسل.في حين أن المدخلات والأهداف للعديد من المهام الأساسيةفي التعلم الآلي لا يمكن بسهولة تمثيلها كمتجهات ذات طول ثابت fixed length vectors، إلا أنه يمكن تمثيلهافي كثير من الأحيان على أنها متواليات متغيرة الطول لمتجهات الطول الثابت. على سبيل المثال، يمكن تمثيل المستندات كتسلسل من الكلمات. غالبًا ما يمكن تمثيل السجلات الطبية كتسلسل للأحداث (لقاءات، أدوية، إجراءات، اختبارات معملية، تشخيصات). يمكن تمثيل مقاطع الفيديو كتسلسلات متفاوتة الطول للصور الثابتة.

بينما ظهرت نماذج التسلسلفي مجالات تطبيق لا حصر لها، فإن البحث الأساسيفي المنطقة كان مدفوعًافي الغالب بالتقدمفي المهام الأساسيةفي معالجة اللغة الطبيعية natural language processing. وبالتالي، خلال هذا الفصل، سنركز عرضنا وأمثلة على البيانات النصية text data. إذا فهمت هذه الأمثلة، فإن تطبيق هذه النماذج على أساليب البيانات الأخرى يجب أن يكون بسيطًا نسبيًا.في الأقسام القليلة التالية، نقدم تدوينًا أساسيًا للتسلسلات وبعض مقاييس التقييم لتقييم جودة مخرجات النموذج المنظم بشكل تسلسلي. بعد ذلك، نناقش المفاهيم الأساسية لنموذج اللغة ونستخدم هذه المناقشة لتحفيز نماذج RNN الأولى لدينا. أخيرًا، نصف طريقة حساب التدرجات عند الانتشار الخلفي من خلال شبكات RNN واستكشاف بعض التحديات التي غالبًا ما تتم مواجهتها عند تدريب مثل هذه الشبكات، وتحفيز بُنى RNN الحديثة التي ستتبعفي القسم 10.

9.1 العمل مع التسلسلات Working with Sequences

حتى الآن، ركزنا على النماذج التي تتكون مدخلاتها من متجه ميزة feature vector واحد $\mathbf{x} \in \mathbb{R}^d$. يتمثل التغيير الرئيسي للمنظور عند تطوير النماذج القادرة على معالجة التسلسلاتفي أننا نركز الآن على المدخلات التي تتكون من قائمة مرتبة من متجهات الميزات $\mathbf{x}_1, \ldots, \mathbf{x}_T$، حيث يقع كل متجه ميزة $\mathbf{x}_t$ مفهرسة بخطوة زمنية $t \in \mathbb{Z}^+$ في $\mathbb{R}^d$.

تتكون بعض مجموعات البيانات من تسلسل ضخم واحد. ضعفي اعتبارك، على سبيل المثال، التدفقات الطويلة للغاية لقراءات أجهزة الاستشعار التي قد تكون متاحة لعلماء المناخ.في مثل

هذه الحالات، قد نقوم بإنشاء مجموعات بيانات تدريبية عن طريق أخذ عينات عشوائية لاحقة من بعض الطول المحدد مسبقًا.في كثير من الأحيان، تصل بياناتنا كمجموعة من التسلسلات sequences. خذ بعين الاعتبار الأمثلة التالية: (1) مجموعة من الوثائق، كل منها يمثل على أنه تسلسل كلمات خاص به، ولكل منها طوله الخاص T_i؛ (2) التمثيل المتسلسل للمريض الذي يبقىفي المستشفى، حيث تتكون كل إقامة من عدد من الأحداث ويعتمد طول التسلسل تقريبًا على طول مدة الإقامة.

في السابق، عند التعامل مع المدخلات الفردية، افترضنا أنه تم أخذ عينات منها بشكل مستقل عن نفس التوزيع الأساسي $P(X)$. بينما لا نزال نفترض أن التسلسلات بأكملها (على سبيل المثال، المستندات الكاملة أو مسارات المريض) يتم أخذ عينات منها بشكل مستقل، لا يمكننا افتراض أن البيانات التي تصلفي كل خطوة زمنية مستقلة عن بعضها البعض. على سبيل المثال، تعتمد الكلمات التي من المحتمل أن تظهر لاحقًافي المستند بشكل كبير على الكلمات التي وردت سابقًافي المستند. يعتمد الدواء الذي من المحتمل أن يتلقاها المريضفي اليوم العاشر من زيارة المستشفى بشكل كبير على ما حدثفي الأيام التسعة السابقة.

وهذا ينبغي أن يكون مفاجئا. إذا لم نكن نعتقد أن العناصرفي التسلسل كانت مرتبطة ببعضنا البعض، فلن نتكبد عناء تصميمها كسلسلةفي المقام الأول. ضعفي اعتبارك فائدة ميزات الملء التلقائي auto-fill الشائعةفي أدوات البحث وعملاء البريد الإلكتروني الحديث. إنها مفيدة على وجه التحديد لأنه غالبًا ما يكون من الممكن التنبؤ (بشكل ناقص، ولكن أفضل من التخمين العشوائي) ما قد تكون الاستمرارية المحتملة للتسلسل، مع إعطاء بعض البادئة الأولية. بالنسبة لمعظم نماذج التسلسل، لا نطلب الاستقلال، أو حتى الثبات، لتسلسلاتنا. بدلاً من ذلك، نطلب فقط أن يتم أخذ عينات من التسلسلات نفسها من بعض التوزيع الأساسي الثابت على التسلسلات بأكملها.

يسمح هذا النهج المرن بظواهر مثل (1) المستندات التي تبدو مختلفة بشكل كبيرفي البداية عما كانت عليهفي النهاية، أو (2) تطور حالة المريض إما نحو الشفاء أو نحو الوفاة خلال فترة الإقامة في المستشفى؛ و (3) تطور ذوق العميل بطرق يمكن التنبؤ بها على مدار التفاعل المستمر مع نظام التوصية recommender system.

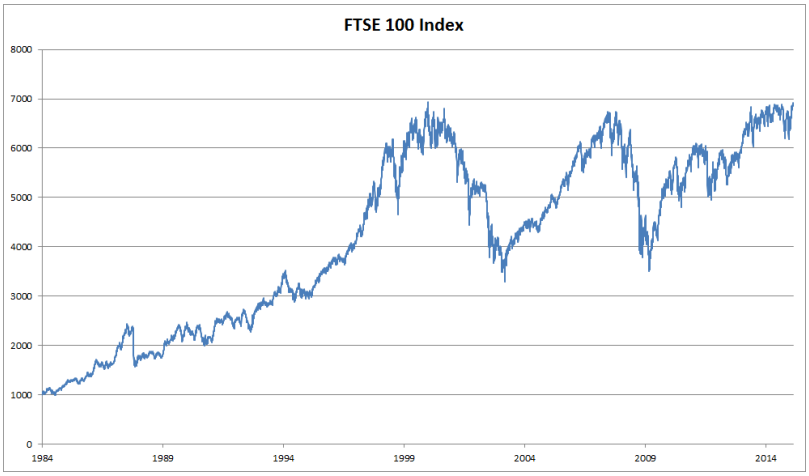
نرغب أحيانًافي توقع هدف ثابت y بالنظر إلى إدخال منظم بشكل تسلسلي (على سبيل المثال، تصنيف المشاعر sentiment classification بناءً على مراجعة فيلم).في أوقات أخرى، نرغب في توقع هدف منظم بشكل تسلسلي ($y_1, \ldots, y_T$) مع الأخذفي الاعتبار إدخال ثابت (على سبيل المثال، تسمية توضيحية للصورة image captioning). لا يزال هدفنافي أوقات أخرى هو التنبؤ بالأهداف المنظمة بشكل تسلسلي بناءً على المدخلات المنظمة بالتسلسل (على سبيل المثال،

الترجمة الآلية أو التسميات التوضيحية للفيديو video captioning). تأخذ مهام التسلسل إلى التسلسل شكلين: (1) محاذاة aligned: حيث تتم محاذاة الإدخال في كل خطوة زمنية مع هدف مطابق (على سبيل المثال، جزء من علامات الكلام)؛ (2) غير محاذي unaligned: حيث لا يُظهر المدخل والهدف بالضرورة استجابات خطوة بخطوة (على سبيل المثال، الترجمة الآلية).

قبل أن نقلق بشأن التعامل مع الأهداف من أي نوع، يمكننا معالجة المشكلة الأكثر وضوحًا: نمذجة الكثافة غير الخاضعة للإشراف unsupervised density modeling (وتسمى أيضًا نمذجة التسلسل sequence modeling). هنا، بالنظر إلى مجموعة من المتسلسلات sequences، هدفنا هو تقدير دالة الكتلة الاحتمالية probability mass function التي تخبرنا بمدى احتمالية رؤية أي تسلسل معين، أي $p(\mathbf{x}_1, \ldots, \mathbf{x}_T)$.

9.1.1. نماذج الانحدار الذاتي Autoregressive Models

قبل تقديم الشبكات العصبية المتخصصة المصممة للتعامل مع البيانات المنظمة بشكل تسلسلي، دعنا نلقي نظرة على بعض بيانات التسلسل الفعلي وبناء بعض الحدس الأساسي والأدوات الإحصائية. على وجه الخصوص، سوف نركز على بيانات أسعار الأسهم من مؤشر FTSE 100 (الشكل 9.1.1). في كل خطوة زمنية $t \in \mathbb{Z}^+$، نلاحظ سعر المؤشر في ذلك الوقت، يُرمز إليه بـ x_t.



الشكل 9.1.1 مؤشر FTSE 100 على مدار حوالي 30 عامًا.

افترض الآن أن المتداول يرغب في إجراء صفقات قصيرة الأجل، والدخول بشكل استراتيجي إلى المؤشر أو الخروج منه، اعتمادًا على ما إذا كان يعتقد أنه سيرتفع أو ينخفض في الخطوة الزمنية اللاحقة. في غياب أي ميزات أخرى (أخبار، بيانات التقارير المالية، إلخ)، فإن الإشارة الوحيدة

المتاحة للتنبؤ بالقيمة اللاحقة هي تاريخ الأسعار حتى الآن. وبالتالي فإن المتداول مهتم بمعرفة توزيع الاحتمالات

$$P(x_t \mid x_{t-1}, \dots, x_1)$$

على الأسعار التي قد يتخذها المؤشرفي الخطوة الزمنية اللاحقة. بينما قد يكون تقدير التوزيع بالكامل على متغير عشوائي ذي قيمة مستمرة أمرًا صعبًا، سيكون من دواعي سرور المتداول التركيز على بعض الإحصائيات الرئيسية للتوزيع، لا سيما القيمة المتوقعة والتباين. استراتيجية واحدة بسيطة لتقدير التوقع المشروط

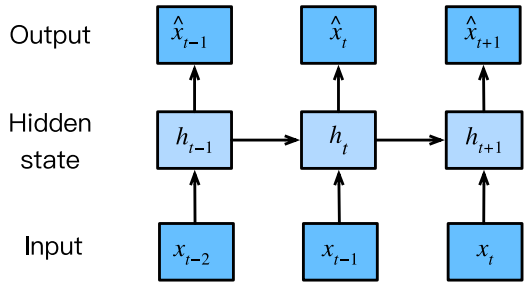
$$\mathbb{E}[(x_t \mid x_{t-1}, \dots, x_1)],$$

سيكون تطبيق نموذج الانحدار الخطي linear regression (راجع القسم 3.1). مثل هذه النماذج التي تتراجع عن قيمة الإشارة على القيم السابقة لتلك الإشارة نفسها تسمى بشكل طبيعي نماذج الانحدار الذاتي autoregressive models. هناك مشكلة رئيسية واحدة فقط: يختلف عدد المدخلات $x_{t-1}, \dots, x_1$ حسب t. أي أن عدد المدخلات يزداد مع كمية البيانات التي نواجهها. وبالتالي، إذا أردنا التعامل مع بياناتنا التاريخية كمجموعة تدريب، فإننا نواجه مشكلة أن كل مثال يحتوي على عدد مختلف من الميزات. سوف يدور الكثير مما يليفي هذا الفصل حول تقنيات التغلب على هذه التحديات عند الانخراطفي مشاكل نمذجة الانحدار الذاتي حيث يكون موضوع الاهتمام هو $P(x_t \mid x_{t-1}, \dots, x_1)$أو بعض الإحصائيات الخاصة بهذا التوزيع.

تتكرر بعض الاستراتيجيات بشكل متكرر. أولاً، قد نعتقد أنه على الرغم من توفر التسلسلات الطويلة $x_{t-1}, \dots, x_1$، فقد لا يكون من الضروري الرجوع إلى الوراء حتى الآنفي التاريخ عند التنبؤ بالمستقبل القريب.في هذه الحالة، قد نكتفي بشرط بعض النوافذ الطويلة τ واستخدام الملاحظات $x_{t-1}, \dots, x_{t-\tau}$ فقط. الفائدة المباشرة هي أن عدد المدخلات الآن هو نفسه دائمًا، على الأقل بالنسبة لـ $t > \tau$. هذا يسمح لنا بتدريب أي نموذج خطي أو شبكة عميقة تتطلب متجهات ذات طول ثابت كمدخلات. ثانيًا، قد نقوم بتطوير نماذج تحافظ على بعض الملخصات h_t للمشاهدات السابقة (انظر الشكل 9.1.2) وفي نفس الوقت يتم تحديث h_t بالإضافة إلى التنبؤ $\hat{x}_t$. هذا يؤدي إلى النماذج التي تقدر مع $\hat{x}_t = P(x_t \mid h_t)$ بالإضافة إلى تحديثات النموذج $h_t = g(h_{t-1}, x_{t-1})$. نظرًا لعدم ملاحظة h_t مطلقًا، فإن هذه النماذج تسمى أيضًا نماذج الانحدار الذاتي الكامنة latent autoregressive models.

لإنشاء بيانات التدريب من البيانات التاريخية، يقوم المرء عادة بإنشاء أمثلة عن طريق أخذ العينات بشكل عشوائي. بشكل عام، لا نتوقع وقتًا للوقوف بلا حراك. ومع ذلك، فإننا نفترض غالبًا أنهفي حين أن القيم المحددة لـ x_t قد تتغير، فإن الديناميكيات التي يتم وفقًا لها إنشاء كل

ملاحظة لاحقة بالنظر إلى الملاحظات السابقة لا تتغير. يسمي الإحصائيون الديناميكيات التي لا تتغير بالثبات stationary.



الشكل 9.1.2 نموذج الانحدار الذاتي الكامن latent autoregressive model.

9.1.2. نماذج التسلسل Sequence Models

في بعض الأحيان، خاصة عند العمل مع اللغة، نرغب في تقدير الاحتمال المشترك لتسلسل كامل. هذه مهمة شائعة عند العمل مع التسلسلات المكونة من رموز منفصلة discrete tokens، مثل الكلمات. بشكل عام، تسمى هذه الدوال المقدرة نماذج التسلسل sequence models وبالنسبة لبيانات اللغة الطبيعية، يطلق عليها نماذج اللغة language models. لقد كان مجال نمذجة التسلسل مدفوعًا إلى حد كبير بمعالجة اللغة الطبيعية، لدرجة أننا غالبًا ما نصف نماذج التسلسل بأنها "نماذج لغة"، حتى عند التعامل مع البيانات غير اللغوية. أثبتت النماذج اللغوية أنها مفيدة لجميع أنواع الأسباب.في بعض الأحيان نريد تقييم احتمالية الجمل. على سبيل المثال، قد نرغب في مقارنة الطبيعة الطبيعية لمخرجين مرشحين تم إنشاؤهما بواسطة نظام الترجمة الآلية أو عن طريق نظام التعرف على الكلام. لكن نمذجة اللغة لا تمنحنا فقط القدرة على تقييم الاحتمالية، ولكن أيضًا القدرة على أخذ عينات من التسلسلات، وحتى تحسين التسلسلات الأكثر احتمالية.

في حين أن نمذجة اللغة قد لا تبدو، للوهلة الأولى، كمشكلة الانحدار الذاتي، يمكننا تقليل النمذجة اللغوية إلى التنبؤ الانحدار الذاتي عن طريق تحليل كثافة مفصل تسلسل $p(x_t \mid x_1, \ldots, x_T)$ إلى ضرب كثافات شرطية بطريقة من اليسار إلى اليمين عن طريق تطبيق قاعدة السلسلة للاحتمالية:

$$P(x_1, \ldots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t \mid x_{t-1}, \ldots, x_1).$$

لاحظ أنه إذا كنا نعمل مع إشارات منفصلة مثل الكلمات، فيجب أن يكون نموذج الانحدار الذاتي مصنفًا احتماليًا، مما ينتج عنه توزيع احتمالي كامل على المفردات الخاصة بالكلمات التي ستأتي بعد ذلك، بالنظر إلى السياق الأيسر.

9.1.2.1. نماذج ماركوف Markov Models

لنفترض الآن أننا نرغب في استخدام الاستراتيجية المذكورة أعلاه، حيث نشترط فقط على الخطوات الزمنية السابقة τ، أي $x_{t-\tau}, \ldots, x_{t-1}$، بدلاً من سجل التسلسل $x_1, \ldots, x_{t-1}$ بأكمله. كلما استطعنا التخلص من التاريخ بعيدًا عن الخطوات الثمينة τ دون أي خسارة في القدرة التنبؤية، نقول إن التسلسل يلبي شرط ماركوف Markov condition، أي أن المستقبل مستقل مشروطًا عن الماضي، بالنظر إلى التاريخ الحديث. عندما $\tau = 1$ نقول إن البيانات تتميز بنموذج ماركوف من الدرجة الأولى، و$\tau = k$ عندما نقول إن البيانات تتميز بنموذج ماركوف من الدرجة k. عندما $\tau = k$ يكون شرط ماركوف من الدرجة الأولى ($\tau = 1$)، يصبح عامل احتمالنا المشترك نتاجًا لاحتمالات كل كلمة بالنظر إلى الكلمة السابقة:

$$P(x_1, \ldots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t \mid x_{t-1}).$$

غالبًا ما نجد أنه من المفيد العمل مع النماذج التي تستمر كما لو أن شرط ماركوف قد تم استيفائه، حتى عندما نعلم أن هذا صحيح تقريبًا. مع المستندات النصية الحقيقية، نستمر في الحصول على المعلومات حيث نقوم بتضمين المزيد والمزيد من سياق اليسار. لكن هذه المكاسب تتضاءل بسرعة. وبالتالي، فإننا في بعض الأحيان نتنازل عن الصعوبات الحسابية والإحصائية ونتجنبها من خلال نماذج التدريب التي تعتمد صلاحيتها على شرط ماركوف من الدرجة k. حتى النماذج اللغوية الضخمة القائمة على RNN والتي تعتمد على المحولات في الوقت الحاضر نادرًا ما تتضمن أكثر من آلاف كلمات السياق.

باستخدام البيانات المنفصلة discrete data، يحسب نموذج ماركوف الحقيقي ببساطة عدد المرات التي حدثت فيها كل كلمة في كل سياق، مما ينتج عنه تقدير التردد النسبي لـ $P(x_t \mid x_{t-1})$. عندما تفترض البيانات قيمًا منفصلة فقط (كما في اللغة)، يمكن حساب التسلسل الأكثر احتمالاً للكلمات بكفاءة باستخدام البرمجة الديناميكية dynamic programming.

9.1.2.2. ترتيب فك التشفير The Order of Decoding

قد تتساءل، لماذا يتعين علينا تمثيل تحليل تسلسل نصي $P(x_1, \ldots, x_T)$ إلى عوامل كسلسلة من الاحتمالات الشرطية من اليسار إلى اليمين. لماذا لا يكون ترتيبًا من اليمين إلى اليسار أو ترتيبًا

عشوائيًا آخر على ما يبدو؟ من حيث المبدأ، لا حرجفي ظهور $P(x_1, \ldots, x_T)$ بترتيب عكسي. النتيجة هي التفكيك الى العوامل صالح valid factorization:

$$P(x_1, \ldots, x_T) = \prod_{t=T}^{1} P(x_t \mid x_{t+1}, \ldots, x_T).$$

ومع ذلك، هناك العديد من الأسباب التي تجعل تحليل النص إلى عوامل في نفس الاتجاهات التي نقرأها (من اليسار إلى اليمين لمعظم اللغات، ولكن من اليمين إلى اليسار للعربية والعبرية) مفضلًا لمهمة نمذجة اللغة. أولاً، هذا مجرد اتجاه طبيعي أكثر بالنسبة لنا للتفكير فيه. بعد كل شيء نقرأ النص كل يوم، وتسترشد هذه العملية بقدرتنا على توقع الكلمات والعبارات التي من المحتمل أن تأتي بعد ذلك. فكر فقطفي عدد المرات التي أكملت فيها جملة شخص آخر. وبالتالي، حتى لو لم يكن لدينا سبب آخر لتفضيل مثل هذه فك التشفير بالترتيب، فإنها ستكون مفيدة فقط إذا كان لدينا حدس أفضل لما يجب أن يكون مرجحًا عند التنبؤ بهذا الترتيب.

ثانيًا، من خلال التحليل بالترتيب factorizing in order، يمكننا تعيين احتمالات للتسلسلات الطويلة بشكل تعسفي باستخدام نفس نموذج اللغة. لتحويل الاحتمالية عبر الخطوات 1 من خلال t إلى احتمال يمتد إلى كلمة $t + 1$، فإننا ببساطة نضربفي الاحتمال الشرطي للرمز token المعطى الإضافي بالنظر إلى الاحتمالية السابقة: $P(x_{t+1}, \ldots, x_1) = P(x_t, \ldots, x_1) \cdot P(x_{t+1} \mid x_t, \ldots, x_1)$.

ثالثًا، لدينا نماذج تنبؤية أقوى للتنبؤ بالكلمات المجاورة مقابل الكلمات الموجودةفي مواقع أخرى عشوائية.في حين أن جميع أوامر التحليل إلى العوامل صحيحة، فإنها لا تمثل بالضرورة جميعًا مشاكل النمذجة التنبؤية السهلة على حد سواء. لا ينطبق هذا على اللغة فحسب، بل ينطبق أيضًا على أنواع البيانات الأخرى، على سبيل المثال، عندما تكون البيانات منظمة سببيًا. نعتقد أن الأحداث المستقبلية لا يمكن أن تؤثر على الماضي. ومن ثم، إذا غيرنا x_t، فقد نتمكن من التأثير على ما يحدث لـ x_{t+1} للمضي قدمًا ولكن ليس العكس. أي، إذا غيرنا x_t، فإن التوزيع على الأحداث الماضية لن يتغير.في بعض السياقات، يجعل هذا من السهل التنبؤ بـ $P(x_{t+1} \mid x_t)$ بدلاً من التنبؤ بـ $P(x_t \mid x_{t+1})$. على سبيل المثال،في بعض الحالات، يمكننا أن نجد $x_{t+1} = f(x_t) + \epsilon$ لبعض الضوضاء المضافة ϵ، بينما العكس ليس صحيحًا (Hoyer et al.، 2009). هذه أخبار رائعة، نظرًا لأنه عادةً ما يكون الاتجاه الأمامي هو الذي نهتم بتقديره. كتاب بيترز وآخرون (Peters et al). (2017) أوضح المزيد حول هذا الموضوع. نحن بالكاد نخدش سطحه.

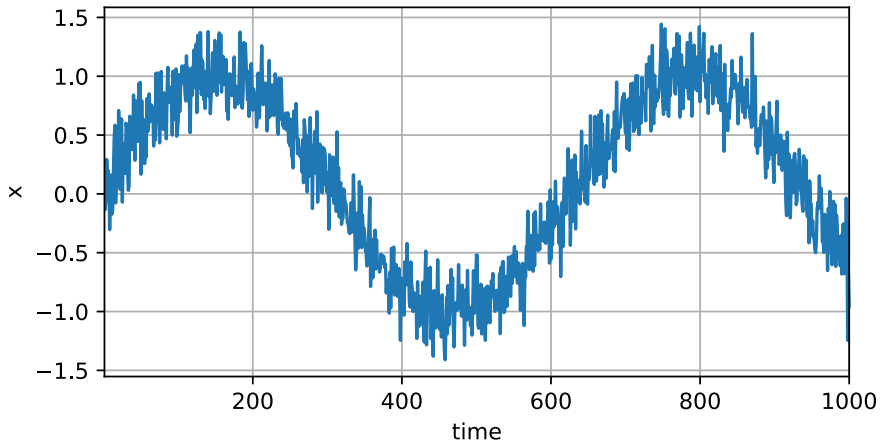
9.1.3. التدريب Training

قبل أن نركز اهتمامنا على البيانات النصية، دعنا نجرب ذلك أولاً باستخدام بعض البيانات التركيبية ذات القيمة المستمرة continuous-valued synthetic data.

```
%matplotlib inline
import tensorflow as tf
from d2l import tensorflow as d2l
```

هنا، ستتبع 1000 بيانات تركيبية دالة المثلثية `sin`، مطبقة على 0.01 مرة من الخطوة الزمنية. لجعل المشكلة أكثر إثارة للاهتمام، قمنا بإفساد كل عينة بضوضاء مضافة. من هذا التسلسل نستخرج أمثلة تدريبية، كل منها يتكون من ميزات وتسمية.

```
class Data(d2l.DataModule):
    def __init__(self, batch_size=16, T=1000,
num_train=600, tau=4):
        self.save_hyperparameters()
        self.time = tf.range(1, T + 1, dtype=tf.float32)
        self.x = tf.sin(0.01 * self.time) +
tf.random.normal([T]) * 0.2

data = Data()
d2l.plot(data.time, data.x, 'time', 'x', xlim=[1, 1000],
figsize=(6, 3))
```



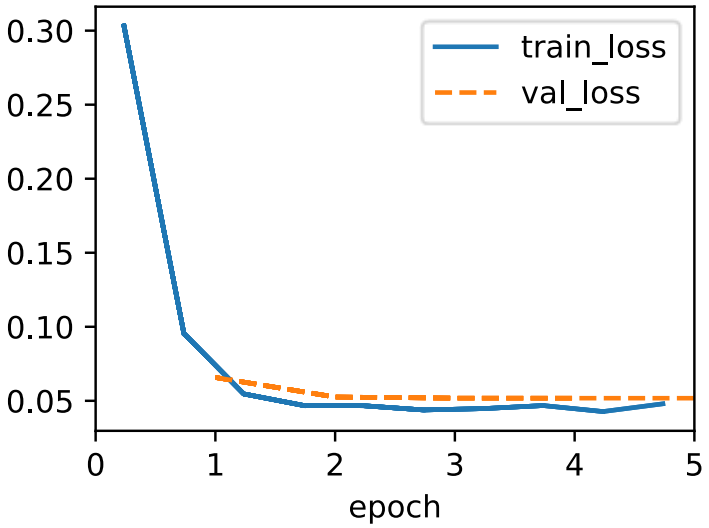
للبدء، نجرب نموذجًا يعمل كما لو أن البيانات استوفت شرط ماركوف ذات الرتبة τ ، وبالتالي يتنبأ باستخدام الملاحظات السابقة τ فقط. وبالتالي لكل خطوة زمنية لدينا مثال مع التسمية $y = x_t$ والميزات $\mathbf{x}_t = [x_{t-\tau}, \ldots, x_{t-1}]$. قد يكون القارئ الذكي قد لاحظ أن هذه النتائج في أمثلة

$\tau - 1000$، لأننا نفتقر إلى التاريخ الكافي $y_\tau, \ldots, y_1$ لها. بينما يمكننا وضع المتتاليات الأولى τ بالأصفار، لإبقاء الأمور بسيطة، نقوم بإسقاطهافي الوقت الحالي. تحتوي مجموعة البيانات الناتجة على أمثلة $T - \tau$ ، حيث يكون لكل إدخالفي النموذج طول تسلسل τ. نقوم بإنشاء مكرر بياناتفي أول 600 مثال، يغطي فترة دالة sin.

```
@d2l.add_to_class(Data)
def get_dataloader(self, train):
    features = [self.x[i : self.T-self.tau+i] for i in
range(self.tau)]
    self.features = tf.stack(features, 1)
    self.labels = tf.reshape(self.x[self.tau:], (-1, 1))
    i = slice(0, self.num_train) if train else
slice(self.num_train, None)
    return self.get_tensorloader([self.features,
self.labels], train, i)
```

في هذا المثال سيكون نموذجنا انحدارًا خطيًا قياسيًا.

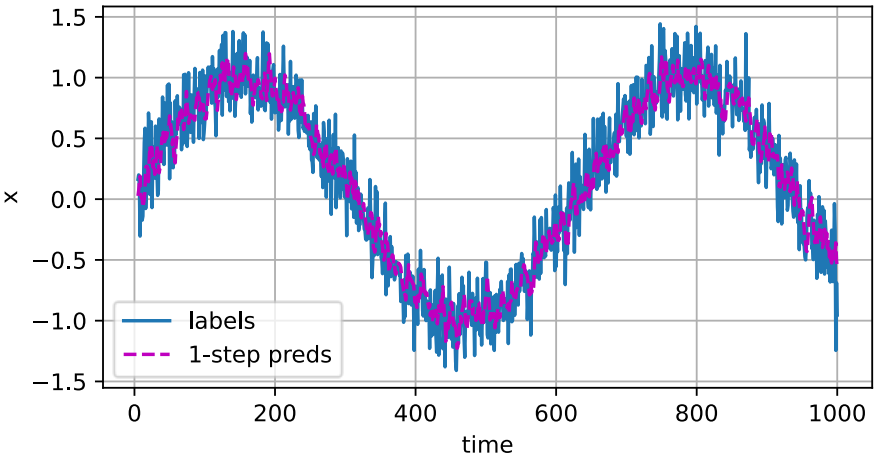
```
model = d2l.LinearRegression(lr=0.01)
trainer = d2l.Trainer(max_epochs=5)
trainer.fit(model, data)
```



9.1.4. التنبؤ Prediction

لتقييم نموذجنا، نتحقق أولاً من مدى جودة أداء نموذجنا عند التنبؤ بخطوة واحدة للأمام one-step-ahead prediction.

```
onestep_preds = model(data.features).numpy()
d2l.plot(data.time[data.tau:], [data.labels,
onestep_preds], 'time', 'x',
         legend=['labels', '1-step preds'], figsize=(6,
3))
```



تبدو التوقعات بخطوة واحدة جيدة، حتى قرب النهاية $t = 1000$.

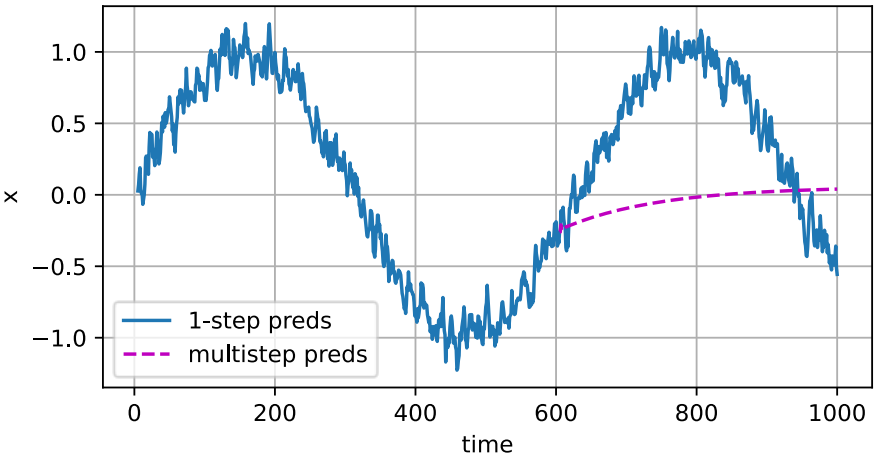
فكر الآن، ماذا لو لاحظنا فقط بيانات التسلسل حتى الخطوة الزمنية 604 (`n_train + tau`) ولكننا نرغب في عمل تنبؤات عدة خطوات في المستقبل. لسوء الحظ، لا يمكننا حساب التنبؤ بخطوة واحدة للخطوة الزمنية 609 مباشرة، لأننا لا نعرف المدخلات المقابلة، بعد أن رأينا فقط حتى x_{604}. يمكننا معالجة هذه المشكلة عن طريق إدخال توقعاتنا السابقة كمدخلات لنموذجنا لعمل تنبؤات لاحقة، والتوقع للأمام، خطوة واحدة في كل مرة، حتى الوصول إلى الخطوة الزمنية المطلوبة:

$$\begin{aligned}
\hat{x}_{605} &= f(x_{601}, x_{602}, x_{603}, x_{604}),\\
\hat{x}_{606} &= f(x_{602}, x_{603}, x_{604}, \hat{x}_{605}),\\
\hat{x}_{607} &= f(x_{603}, x_{604}, \hat{x}_{605}, \hat{x}_{606}),\\
\hat{x}_{608} &= f(x_{604}, \hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}),\\
\hat{x}_{609} &= f(\hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}, \hat{x}_{608}),\\
&\dots
\end{aligned}$$

بشكل عام، بالنسبة للتسلسل المرصود $x_1, \dots, x_t$، خرجه المتوقع $\hat{x}_{t+k}$ في الخطوة الزمنية $t + k$ يسمى بالتنبؤ بـ k خطوة إلى الأمام k-step-ahead prediction. منذ أن لاحظنا حتى x_{604}،

التنبؤ بخطوة k للأمام هو $\hat{x}_{604+k}$. بمعنى آخر، سيتعين علينا الاستمرارفي استخدام تنبؤاتنا الخاصة لعمل تنبؤات متعددة الخطوات. دعونا نرى كيف تسير الأمور على ما يرام.

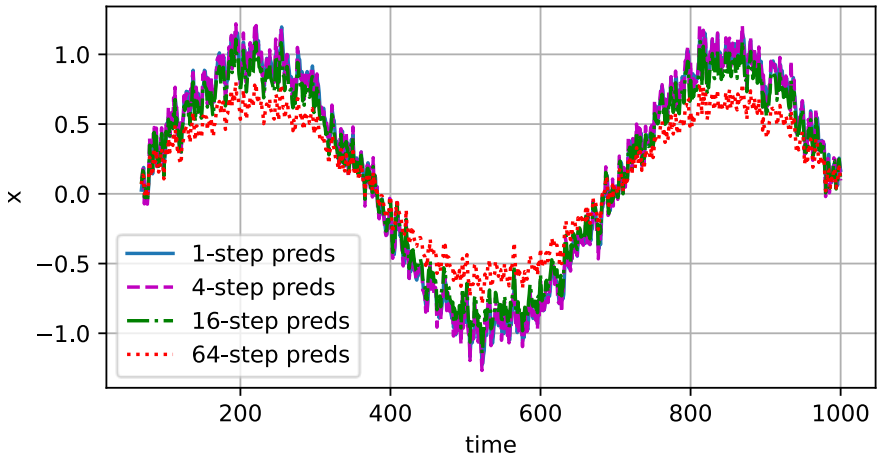
```
multistep_preds = tf.Variable(tf.zeros(data.T))
multistep_preds[:].assign(data.x)
for i in range(data.num_train + data.tau, data.T):
    multistep_preds[i].assign(tf.reshape(model(
        tf.reshape(multistep_preds[i-data.tau : i], (1,
-1))), ()))

d2l.plot([data.time[data.tau:],
data.time[data.num_train+data.tau:]],
         [onestep_preds,
multistep_preds[data.num_train+data.tau:]], 'time',
         'x', legend=['1-step preds', 'multistep
preds'], figsize=(6, 3))
```



لسوء الحظ،في هذه الحالة نفشل بشكل مذهل. تتحلل التنبؤات إلى ثابت بسرعة كبيرة بعد بضع خطوات تنبؤ. لماذاكان أداء الخوارزمية أسوأ بكثير عند توقع المزيدفي المستقبل؟في النهاية، يرجع هذا إلى حقيقة أن الأخطاء تتراكم. لنفترض أنه بعد الخطوة 1 لدينا بعض الخطأ $\epsilon_1 = \overline{\epsilon}$. الآن مدخلات الخطوة 2 مضطربة بواسطة ϵ_1، وبالتالي نعاني من بعض الخطأفي ترتيب $\epsilon_2 = \overline{\epsilon} + c\epsilon_1$ لبعض الثابت c، وما إلى ذلك. يمكن أن تتباعد التنبؤات بسرعة عن الملاحظات الحقيقية. قد تكون بالفعل على دراية بهذه الظاهرة الشائعة. على سبيل المثال، تنبؤات الطقس خلال الـ 24 ساعة القادمة تميل إلى أن تكون دقيقة جدًا ولكن بعد ذلك، تنخفض الدقة بسرعة. سنناقش طرق تحسين هذا خلال هذا الفصل وما بعده.

دعونا نلقي نظرة فاحصة على الصعوباتفي بالتنبؤات بـ k خطوة إلى الأمام عن طريق حساب التنبؤات على التسلسل الكامل لـ $k = 1,4,16,64$

```
def k_step_pred(k):
    features = []
    for i in range(data.tau):
        features.append(data.x[i : i+data.T-data.tau-
k+1])
    # The (i+tau)-th element stores the (i+1)-step-ahead
predictions
    for i in range(k):
        preds = model(tf.stack(features[i : i+data.tau],
1))
        features.append(tf.reshape(preds, -1))
    return features[data.tau:]

steps = (1, 4, 16, 64)
preds = k_step_pred(steps[-1])
d2l.plot(data.time[data.tau+steps[-1]-1:],
         [preds[k - 1].numpy() for k in steps], 'time',
'x',
         legend=[f'{k}-step preds' for k in steps],
figsize=(6, 3))
```



يوضح هذا بوضوح كيف تتغير جودة التنبؤ بينما نحاول التنبؤ أكثرفي المستقبل.في حين أن التنبؤات ذات الأربع خطوات 4-step-ahead predictions لا تزال تبدو جيدة، فإن أي شيء يتجاوز ذلك يكون عديم الفائدة تقريبًا.

9.1.5. الملخص

هناك اختلاف كبيرفي الصعوبة بين الاستيفاء interpolationوالاستقراء interpolation. وبالتالي، إذاكان لديك تسلسل sequence، فاحترم دائمًا الترتيب الزمني للبيانات عند التدريب، أي لا تتدرب أبدًا على البيانات المستقبلية. بالنظر إلى هذا النوع من البيانات، تتطلب نماذج التسلسل أدوات إحصائية متخصصة للتقدير. هناك خياران شائعان هما نماذج الانحدار الذاتي autoregressive models ونماذج الانحدار التلقائي المتغيرة الكامنة latent-variable autoregressive models. بالنسبة للنماذج السببية causal models (على سبيل المثال، الوقتفي المضي قدمًا time going forward)، يكون تقدير الاتجاه الأمامي أسهل كثيرًا من الاتجاه العكسي. بالنسبة للتسلسل المرصود حتى الخطوة الزمنية، فإن ناتجها المتوقعفي الخطوة الزمنية هو التنبؤ بالخطوة إلى الأمام. كلما توقعنا مزيدًا من الوقت من خلال الزيادة، تتراكم الأخطاء وتتدهور جودة التنبؤ، غالبًا بشكل كبير.

9.1.6. التمارين

1. قم بتحسين النموذج في تجربة هذا القسم.
1. هل تدمج أكثر من المشاهدات observations الأربع الماضية؟كم تحتاج حقا؟
2. كم عدد المشاهدات السابقة التي ستحتاجها إذا لم يكن هناك ضوضاء؟ تلميح: يمكنك الكتابة كمعادلة تفاضلية cos.
3. هل يمكنك دمج المشاهدات القديمة مع الحفاظ على العدد الإجمالي للسمات ثابتًا؟ هل هذا يحسن الدقة؟ لماذا؟
4. قم بتغيير بُنية الشبكة العصبية وتقييم الأداء. يمكنك تدريب النموذج الجديد بمزيد من الفترات epochs. ماذا تلاحظ؟
2. يرغب المستثمر في العثور على ورقة مالية جيدة للشراء. ينظر إلى العوائد السابقة ليقرر أيها من المرجح أن يكون جيدًا. ما الذي يمكن أن يحدث خطأ في هذه الاستراتيجية؟
3. هل السببية causality تنطبق أيضا على النص؟ إلى أي مدى؟
4. أعط مثالاً عندما قد تكون هناك حاجة إلى نموذج الانحدار التلقائي الكامن لالتقاط ديناميكية البيانات.

9.2. تحويل النص الخام إلى بيانات التسلسل Converting Raw Text into Sequence Data

خلال هذا الكتاب، سنعمل غالبًا مع بيانات نصية ممثلةفي شكل تسلسلات من الكلمات أو الأحرف أو مقاطع الكلمات. للبدء، سنحتاج إلى بعض الأدوات الأساسية لتحويل النص الخام

raw text إلى تسلسلات sequences بالشكل المناسب. تنفذ خطوط الأنابيب النموذجية المعالجة المسبقة Typical preprocessing pipelines الخطوات التالية:

1. تحميل النص كسلاسل strings في الذاكرة.
2. قسّم السلاسل إلى رموز tokens (مثل الكلمات أو الأحرف).
3. قم ببناء قاموس مفردات لربط كل عنصر من عناصر المفردات بفهرس رقمي.
4. تحويل النص إلى متسلسلات sequences من المؤشرات الرقمية.

```
import collections
import random
import re
import tensorflow as tf
from d2l import tensorflow as d2l
```

9.2.1. قراءة مجموعة البيانات Reading the Dataset

هنا ، سنعمل مع كتاب The Time Machine لـ H.G Wells، وهو كتاب يحتوي على ما يزيد قليلاً عن 30000 كلمة. في حين أن التطبيقات الحقيقية ستشمل عادةً مجموعات بيانات أكبر بكثير، فإن هذا يكفي لإثبات خط أنابيب المعالجة المسبقة. تقرأ طريقة `_download` التالية النص الخام في سلسلة نصية.

```
class TimeMachine(d2l.DataModule): #@save
    def _download(self):
        fname = d2l.download(d2l.DATA_URL +
'timemachine.txt', self.root,

'090b5e7e70c295757f55df93cb0a180b9691891a')
        with open(fname) as f:
            return f.read()

data = TimeMachine()
raw_text = data._download()
raw_text[:60]
```

```
'The Time Machine, by H. G. Wells
[1898]\n\n\nInnnThe Time Tra'
```

للتبسيط ، نتجاهل علامات الترقيم والكتابة بالأحرف الكبيرة عند المعالجة المسبقة للنص الخام raw text.

```
@d2l.add_to_class(TimeMachine)  #@save
def _preprocess(self, text):
    return re.sub('[^A-Za-z]+', ' ', text).lower()
```

```
text = data._preprocess(raw_text)
text[:60]
```

```
'the time machine by h g wells i the time traveller for
so it'
```

9.2.2. الترميز Tokenization

الرموز Tokens هي الوحدات الذرية (غير القابلة للتجزئة) للنص. تتوافق كل خطوة زمنية مع رمز Token واحد، ولكن ما يشكل رمزًا على وجه التحديد هو اختيار التصميم. على سبيل المثال ، يمكننا تمثيل الجملة "Baby needs a new pair of shoes " على شكل سلسلة من 7 كلمات ، حيث تشتمل مجموعة كل الكلمات على مفردات كبيرة (عادةً عشرات أو مئات الآلاف من الكلمات). أو قد نمثل نفس الجملة كتسلسل أطول بكثير من 30 حرفًا ، باستخدام مفردات أصغر بكثير) لا يوجد سوى 256 حرف ASCII مميز .(أدناه ، نقوم بترميز نصنا المعالج مسبقًا إلى سلسلة من الأحرف.

```
@d2l.add_to_class(TimeMachine)  #@save
def _tokenize(self, text):
    return list(text)

tokens = data._tokenize(text)
','.join(tokens[:30])
```

```
't,h,e, ,t,i,m,e, ,m,a,c,h,i,n,e, ,b,y, ,h, ,g,
,w,e,l,l,s, '
```

9.2.3. المفردات Vocabulary

هذه الرموز tokens لا تزال سلاسل strings. ومع ذلك ، يجب أن تتكون مدخلات نماذجنا في النهاية من مدخلات رقمية. بعد ذلك ، نقدم فئة لإنشاء المفردات vocabularies، أي الكائنات التي تربط كل قيمة رمزية مميزة بفهرس فريد unique index. أولاً، نحدد مجموعة الرموز الفريدة في مجموعة التدريب الخاصة بنا. ثم نقوم بتعيين فهرس رقمي لكل رمز فريد unique token. غالبًا ما يتم حذف عناصر المفردات النادرة للراحة. عندما نواجه رمزًا مميزًا في وقت التدريب أو الاختبار لم يسبق رؤيته أو تم إسقاطه من المفردات، فإننا نقوم بتمثيله برمز "<unk>"، مما يدل على أن هذه قيمة غير معروفة unknown value.

```
class Vocab:  #@save
    """Vocabulary for text."""
    def __init__(self, tokens=[], min_freq=0,
reserved_tokens=[]):
        # Flatten a 2D list if needed
        if tokens and isinstance(tokens[0], list):
```

```
            tokens = [token for line in tokens for token
in line]
        # Count token frequencies
        counter = collections.Counter(tokens)
        self.token_freqs = sorted(counter.items(),
key=lambda x: x[1],
                                  reverse=True)
        # The list of unique tokens
        self.idx_to_token = list(sorted(set(['<unk>'] +
reserved_tokens + [
            token for token, freq in self.token_freqs if
freq >= min_freq])))
        self.token_to_idx = {token: idx
                             for idx, token in
enumerate(self.idx_to_token)}

    def __len__(self):
        return len(self.idx_to_token)

    def __getitem__(self, tokens):
        if not isinstance(tokens, (list, tuple)):
            return self.token_to_idx.get(tokens,
self.unk)
        return [self.__getitem__(token) for token in
tokens]

    def to_tokens(self, indices):
        if hasattr(indices, '__len__') and len(indices)
> 1:
            return [self.idx_to_token[int(index)] for
index in indices]
        return self.idx_to_token[indices]

    @property
    def unk(self):  # Index for the unknown token
        return self.token_to_idx['<unk>']
```

نقوم الآن ببناء مفردات لمجموعة البيانات الخاصة بنا ، وتحويل تسلسل السلاسل إلى قائمة من المؤشرات الرقمية numerical indices. لاحظ أننا لم نفقد أي معلومات ويمكننا بسهولة تحويل مجموعة البيانات الخاصة بنا إلى تمثيلها الأصلي (سلسلة string).

```
vocab = Vocab(tokens)
```

```
indices = vocab[tokens[:10]]
print('indices:', indices)
print('words:', vocab.to_tokens(indices))
```

```
indices: [21, 9, 6, 0, 21, 10, 14, 6, 0, 14]
words: ['t', 'h', 'e', ' ', 't', 'i', 'm', 'e', ' ',
'm']
```

9.2.4. وضع كل شيء معا Putting It All Together

باستخدام الفئات والطرق المذكورة أعلاه ، نقوم بتجميع كل شيء في طريقة الإنشاء التالية لفئة `TimeMachine` ، والتي تُرجع المجموعة `corpus` ، وقائمة من المؤشرات المرمزة token indices ، والمفردات `vocab` ، وهي مفردات مجموعة The Time Machine. التعديلات التي أجريناها هنا هي: (1) نقوم بترميز النص إلى أحرف ، وليس كلمات ، لتبسيط التدريب في أقسام لاحقة ؛ (2) المجموعة عبارة عن قائمة واحدة، وليست قائمة بقوائم الرموز ، نظرًا لأن كل سطر نص في مجموعة بيانات The Time Machine ليس بالضرورة جملة أو فقرة.

```
@d2l.add_to_class(TimeMachine)  #@save
def build(self, raw_text, vocab=None):
    tokens = self._tokenize(self._preprocess(raw_text))
    if vocab is None: vocab = Vocab(tokens)
    corpus = [vocab[token] for token in tokens]
    return corpus, vocab

corpus, vocab = data.build(raw_text)
len(corpus), len(vocab)
```

```
(173428, 28)
```

9.2.5. إحصائيات اللغة الاستكشافية Exploratory Language Statistics

باستخدام المجموعة الحقيقية وكلاس `Vocab` المحدد عبر الكلمات، يمكننا فحص الإحصائيات الأساسية المتعلقة باستخدام الكلمات في مجموعتنا. أدناه ، نبني مفردات من الكلمات المستخدمة في The Time Machine ونطبع أكثر 10 كلمات تكرارا.

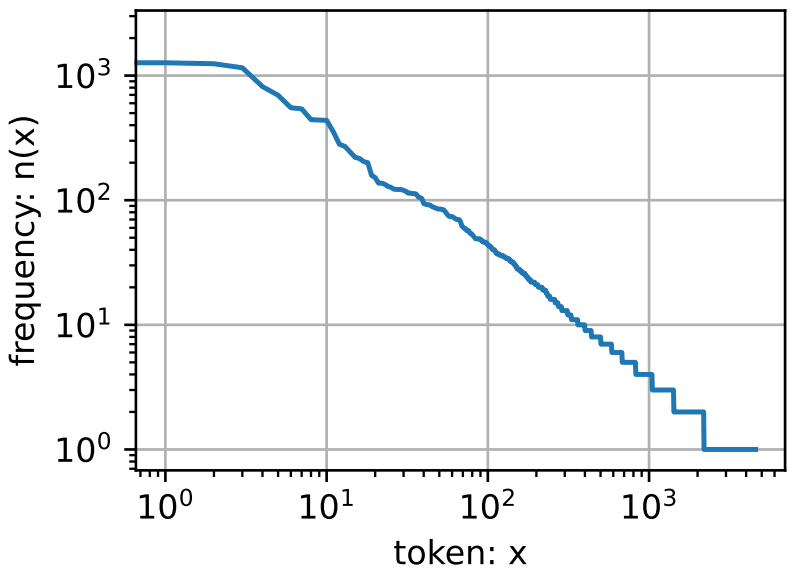
```
words = text.split()
vocab = Vocab(words)
vocab.token_freqs[:10]
```

```
[('the', 2261),
 ('i', 1267),
 ('and', 1245),
 ('of', 1155),
 ('a', 816),
 ('to', 695),
```

```
('was', 552),
('in', 541),
('that', 443),
('my', 440)]
```

لاحظ أن الكلمات العشر الأكثر شيوعًا ليست كلها وصفية descriptive. قد تتخيل أننا قد نرى قائمة مشابهة جدًا إذا اخترنا أي كتاب عشوائيًا. مقالات مثل "the" و "a" ، ضمائر مثل "i" و "my" ، وحروف الجر مثل "of" و "to" و "in" تحدث غالبًا لأنها تخدم أدوارًا نحوية مشتركة. غالبًا ما تسمى هذه الكلمات الشائعة في آن واحد ولكنها وصفية بشكل خاص كلمات التوقف stop words ، وفي الأجيال السابقة من مصنفات النص التي تستند إلى تمثيلات كيس من الكلمات bag-of-words ، غالبًا ما يتم تصفيتها. ومع ذلك ، فهي تحمل معنى وليس من الضروري تصفيتها عند العمل مع النماذج العصبية الحديثة القائمة على RNN والمحول. إذا نظرت إلى أسفل القائمة، ستلاحظ أن تردد الكلمات يتلاشى بسرعة. الكلمة العاشرة الأكثر شيوعًا هي أقل من 1/5 مثل الكلمة الأكثر شيوعًا. يميل تردد الكلمات إلى اتباع توزيع قانون السلطة power law distribution (تحديدًا Zipfian) أثناء نزولنا في المراتب ranks. للحصول على فكرة أفضل، نرسم رقم تكرار الكلمة word frequency.

```
freqs = [freq for token, freq in vocab.token_freqs]
d2l.plot(freqs, xlabel='token: x', ylabel='frequency:
n(x)',
         xscale='log', yscale='log')
```



بعد التعامل مع الكلمات القليلة الأولى كاستثناءات، تتبع جميع الكلمات المتبقية تقريبًا خطًا مستقيمًا في رسم log-log. يلتقط قانون Zipf هذه الظاهرة ، والذي ينص على أن تكرار n_i الكلمة الأكثر شيوعًا هو:

$$n_i \propto \frac{1}{i^\alpha},$$

وهو ما يعادل:

$$\log n_i = -\alpha \log i + c,$$

حيث α هو الأس الذي يميز التوزيع وهو ثابت. يجب أن يمنحنا هذا بالفعل وقفة إذا أردنا نمذجة الكلمات عن طريق حساب الإحصائيات. بعد كل شيء، سنبالغ في تقدير تكرار الذيل، والمعروف أيضًا باسم الكلمات النادرة infrequent words. ولكن ماذا عن تركيبات الكلمات الأخرى ، مثل كلمتين متتاليتين (bigrams) ، وثلاث كلمات متتالية (trigrams) ، وما بعدها؟ دعونا نرى ما إذا كان تردد bigram يتصرف بنفس طريقة تكرار الكلمة المفردة (unigram).

```
bigram_tokens = ['--'.join(pair) for pair in
zip(words[:-1], words[1:])]
bigram_vocab = Vocab(bigram_tokens)
bigram_vocab.token_freqs[:10]
```

```
[('of--the', 309),
 ('in--the', 169),
 ('i--had', 130),
 ('i--was', 112),
 ('and--the', 109),
 ('the--time', 102),
 ('it--was', 99),
 ('to--the', 85),
 ('as--i', 78),
 ('of--a', 73)]
```

شيء واحد ملحوظ هنا. من بين أزواج الكلمات العشرة الأكثر شيوعًا ، تتكون تسعة من كلتا كلمات التوقف وواحدة فقط ذات صلة بالكتاب الفعلي – "the time". علاوة على ذلك ، دعنا نرى ما إذا كان تردد الشكل الثلاثي trigram يتصرف بنفس الطريقة.

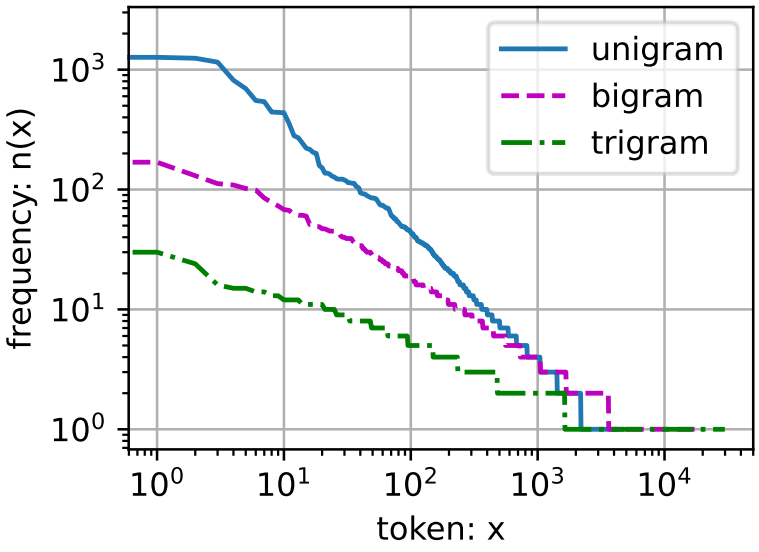
```
trigram_tokens = ['--'.join(triple) for triple in zip(
    words[:-2], words[1:-1], words[2:])]
trigram_vocab = d2l.Vocab(trigram_tokens)
trigram_vocab.token_freqs[:10]
```

```
[('the--time--traveller', 59),
```

```
('the--time--machine', 30),
('the--medical--man', 24),
('it--seemed--to', 16),
('it--was--a', 15),
('here--and--there', 15),
('seemed--to--me', 14),
('i--did--not', 14),
('i--saw--the', 13),
('i--began--to', 13)]
```

أخيرًا ، دعنا نتخيل معدل تكرار الرمز token frequency بين هذه النماذج الثلاثة: الأحادي unigrams ، وbigrams ، وtrigrams.

```
bigram_freqs = [freq for token, freq in
bigram_vocab.token_freqs]
trigram_freqs = [freq for token, freq in
trigram_vocab.token_freqs]
d2l.plot([freqs, bigram_freqs, trigram_freqs],
xlabel='token: x',
         ylabel='frequency: n(x)', xscale='log',
yscale='log',
         legend=['unigram', 'bigram', 'trigram'])
```



هذا الرقم مثير للغاية. أولاً ، بخلاف كلمات unigram ، يبدو أيضًا أن تسلسل الكلمات يتبع قانون Zipf ، وإن كان بأس أصغر α في (9.2.1) ، اعتمادًا على طول التسلسل. ثانيًا ، عدد n- grams المميزة ليس بهذه الضخامة. هذا يعطينا الأمل في أن هناك قدرًا كبيرًا من البُنية في اللغة.

ثالثًا ، نادرًا ما تحدث العديد من n-grams. هذا يجعل بعض الأساليب غير مناسبة لنمذجة اللغة ويحفز استخدام نماذج التعلم العميق. سنناقش هذا في القسم التالي.

9.2.6. الملخص

يعتبر النص من أكثر أشكال بيانات التسلسل شيوعًافي التعلم العميق. الخيارات الشائعة لما يشكل رمزًا token هي الأحرف والكلمات وقطع الكلمات. لمعالجة النص، نقوم عادةً (1) بتقسيم النص إلى رموز tokens؛ (2) بناء مفردات لتعيين سلاسل الرموز token strings إلى مؤشرات رقمية numerical indices؛ و (3) تحويل البيانات النصية إلى مؤشرات رمزية token indices للنماذج للتلاعب بها. من الناحية العملية، فإن تكرار الكلمات يميل إلى اتباع قانون Zipf. هذا صحيح ليس فقط للكلمات الفردية (unigrams)، ولكن أيضًا بالنسبة إلى n-grams.

9.2.7. التمارين

1. في تجربة هذا القسم، قم بترميز tokenize النص إلى كلمات وقم بتغيير قيمة وسيطة `min_freq` لمثيل `Vocab`. صف نوعياً كيف تؤثر التغييراتفي `min_freq` على حجم المفردات الناتجة.
2. قدر أُس توزيع Zipfian لـ unigrams وbigrams وtrigramsفي هذه المجموعة corpus.
3. ابحث عن بعض مصادر البيانات الأخرى (قم بتنزيل مجموعة بيانات قياسية للتعلم الآلي، واختر كتابًا آخرفي مجال عام اخر، واكتشف موقعًا إلكترونيًا، وما إلى ذلك). لكل منها، قم بترميز البيانات على مستوى الكلمة والحرف. كيف تقارن أحجام المفردات مع مجموعة The Time Machine بقيم مكافئة لـ `min_freq`. قدر الأس لتوزيع Zipfian المقابل لتوزيعات unigram وbigram لهذه المجموعات. كيف تتم مقارنتها بالقيم التي لاحظتها لمجموعة The Time Machine؟

9.3 نماذج اللغة Language Models

في القسم 9.2، رأينا كيفية تعيين تسلسلات نصية text sequences إلى رموز tokens، حيث يمكن عرض هذه الرموز على أنها سلسلة من المشاهدات المنفصلة، مثل الكلمات أو الأحرف. افترض أن الرموزفي تسلسل نصي للطول T هي بدورها $x_1, x_2, \ldots, x_T$. الهدف من النماذج اللغوية هو تقدير الاحتمال المشترك للتسلسل بأكمله:

$$P(x_1, x_2, \ldots, x_T),$$

حيث يمكن تطبيق الأدوات الإحصائية في القسم 9.1.

النماذج اللغوية Language models مفيدة بشكل لا يصدق. على سبيل المثال ، سيكون نموذج اللغة المثالي قادرًا على إنشاء نص طبيعي بمفرده ، وذلك ببساطة عن طريق رسم رمز واحد في كل مرة $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1)$. على عكس القرد الذي يستخدم آلة كاتبة ، فإن كل النص الخارج من هذا النموذج سيمر كلغة طبيعية ، على سبيل المثال ، نص إنجليزي. علاوة على ذلك، سيكون كافيًا لإنشاء حوار هادف ، ببساطة عن طريق تكييف النص على أجزاء الحوار السابقة. من الواضح أننا ما زلنا بعيدين جدًا عن تصميم مثل هذا النظام ، لأنه سيحتاج إلى فهم النص بدلاً من مجرد إنشاء محتوى معقول نحويًا.

ومع ذلك ، فإن النماذج اللغوية تقدم خدمة رائعة حتى في شكلها المحدود. على سبيل المثال ، الجملتان "to recognize speech" و "to wreck a nice beach" تبدو متشابهة جدًا. يمكن أن يتسبب هذا في الغموض في التعرف على الكلام ، والذي يمكن حله بسهولة من خلال نموذج لغوي يرفض الترجمة الثانية باعتبارها غريبة. وبالمثل ، في خوارزمية تلخيص المستندات document summarization، من المفيد معرفة أن "dog bites man" أكثر تكرارًا من "man bites dog" ، أو أن عبارة "I want to eat grandma" هي عبارة مزعجة إلى حد ما ، بينما "I want to eat, grandma" أكثر اعتدالًا.

9.3.1. تعلم نماذج اللغة Learning Language Models

السؤال الواضح هو كيف يجب أن نمثل مستندًا ، أو حتى سلسلة من الرموز tokens. افترض أننا نقوم بترميز tokenize البيانات النصية على مستوى الكلمة. لنبدأ بتطبيق قواعد الاحتمال الأساسية:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1}).$$

على سبيل المثال ، يمكن إعطاء احتمال وجود تسلسل نصي يحتوي على أربع كلمات على النحو التالي:

$$P(\text{deep,learning,is,fun}) = P(\text{deep})P(\text{learning} \mid \text{deep})P(\text{is} \mid \text{deep,learning})P(\text{fun} \mid \text{deep,learning,is}).$$

9.3.1.1. نماذج ماركوف و n-غرام Markov Models and n-grams

من بين تحليل نموذج التسلسلفي القسم 9.1، دعنا نطبق نماذج ماركوف على نمذجة اللغة. التوزيع على التسلسلات يلبي خاصية ماركوف من الدرجة الأولى إذا $P(x_{t+1} \mid x_t, \ldots, x_1) = P(x_{t+1} \mid x_t)$. تتوافق الطلبات الأعلى مع التبعيات الأطول longer dependencies. يؤدي هذا إلى عدد من التقديرات التقريبية التي يمكن أن نطبقها لنمذجة تسلسل:

$$
\begin{aligned}
P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2)P(x_3)P(x_4),\\
P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)P(x_4 \mid x_3),\\
P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_4 \mid x_2, x_3).
\end{aligned}
$$

يُشار عادةً إلى صيغ الاحتمال التي تتضمن متغير واحد، ومتغيرين، وثلاثة متغيرات على أنها نماذج unigram، وbigram، وtrigram، على التوالي. من أجل حساب نموذج اللغة، نحتاج إلى حساب احتمال الكلمات والاحتمال الشرطي للكلمة بالنظر إلى الكلمات القليلة السابقة. لاحظ أن هذه الاحتمالات هي معلمات نموذج اللغة language model parameters.

9.3.1.2. تردد الكلمات Word Frequency

هنا، نفترض أن مجموعة بيانات التدريب عبارة عن مجموعة نصية كبيرة، مثل جميع إدخالات Wikipedia و Project Gutenberg وجميع النصوص المنشورة على الويب. يمكن حساب احتمال الكلمات من تكرار الكلمات النسبي relative word frequency لكلمة معينةفي مجموعة بيانات التدريب. على سبيل المثال، التقدير $\hat{P}(\text{deep})$يمكن حسابه على أنه احتمال أي جملة تبدأ بكلمة "deep". قد يكون النهج الأقل دقة هو حساب جميع تكرارات كلمة "deep" وتقسيمها على العدد الإجمالي للكلماتفي المجموعة. هذا يعمل بشكل جيد إلى حد ما، خاصة مع الكلمات المتكررة. للمضي قدمًا، يمكننا محاولة التقدير:

$$\hat{P}(\text{learning} \mid \text{deep}) = \frac{n(\text{deep, learning})}{n(\text{deep})},$$

حيث $n(x)$ و$n(x, x')$ عدد تكرارات المفردات وأزواج الكلمات المتتالية، على التوالي. لسوء الحظ، يعد تقدير احتمال وجود زوج من الكلمات أكثر صعوبة إلى حد ما ، نظرًا لأن حدوث "deep learning" أقل تكرارًا. على وجه الخصوص ، بالنسبة لبعض تركيبات الكلمات غير المعتادة ، قد يكون من الصعب العثور على تكرارات كافية للحصول على تقديرات دقيقة. كما هو مقترح من النتائج التجريبية في القسم 9.2.5 ، تأخذ الأمور منعطفًا نحو الأسوأ بالنسبة لتركيبات مكونة من ثلاث كلمات وما بعدها. سيكون هناك العديد من التركيبات المعقولة المكونة من ثلاث كلمات والتي من المحتمل ألا نراها في مجموعة البيانات الخاصة بنا. ما لم نقدم بعض الحلول لتعيين مثل هذه المجموعات من الكلمات عددًا غير صفري ، فلن نتمكن من استخدامها في نموذج اللغة. إذا كانت مجموعة البيانات صغيرة أو كانت الكلمات نادرة جدًا ، فقد لا نعثر حتى على واحدة منها.

9.3.1.3. تجانس لابلاس Laplace Smoothing

تتمثل الإستراتيجية الشائعة في إجراء بعض أشكال تجانس لابلاس Laplace Smoothing. الحل هو إضافة ثابت صغير لجميع الأعداد. قم بالإشارة إلى

n كعدد إجمالي للكلمات في مجموعة التدريب وعدد الكلمات الفريدة. يساعد هذا الحل مع الفردي ، على سبيل المثال ، عبر

$$\hat{P}(x) = \frac{n(x) + \epsilon_1/m}{n + \epsilon_1},$$
$$\hat{P}(x' \mid x) = \frac{n(x, x') + \epsilon_2 \hat{P}(x')}{n(x) + \epsilon_2},$$
$$\hat{P}(x'' \mid x, x') = \frac{n(x, x', x'') + \epsilon_3 \hat{P}(x'')}{n(x, x') + \epsilon_3}.$$

هنا ϵ_1, ϵ_2، وϵ_3 المعلمات الفائقة. خذ ϵ_1 كمثال: عندما $\epsilon_1 = 0$ لا يتم تطبيق أي تجانس no smoothing؛ عندما ϵ_1 تقترب من اللانهاية الموجبة، $\hat{P}(x)$ يقترب من الاحتمال الموحد $1/m$. ما ورد أعلاه هو البديل البدائي إلى حد ما لما يمكن أن تحققه التقنيات الأخرى (Wood et al.، 2011).

لسوء الحظ، تصبح مثل هذه النماذج غير عملية إلى حد ما بسرعة للأسباب التالية. أولاً، كما تمت مناقشتهفي القسم 9.2.5، نادرًا ما تحدث العديد من القواعد اللغوية، مما يجعل تجانس لابلاس غير مناسب إلى حد ما لنمذجة اللغة. ثانيًا، نحتاج إلى تخزين كل التواتر counts. ثالثًا، هذا يتجاهل تمامًا معنى الكلمات. على سبيل المثال، يجب أن تحدث "القط cat" و" القطط feline"في السياقات ذات الصلة. من الصعب للغاية تعديل مثل هذه النماذج لسياقات إضافية، في حين أن نماذج اللغة القائمة على التعلم العميق مناسبة تمامًا لأخذ ذلكفي الاعتبار. يكاد يكون من المؤكد أن تكون تسلسلات الكلمات الطويلة الأخيرة جديدة، ومن ثم فإن النموذج الذي يحسب ببساطة تواتر تسلسلات الكلمات التي شوهدت سابقًا لا بد أن يؤدي أداءً ضعيفًا هناك. لذلك، فإننا نركز على استخدام الشبكات العصبية لنمذجة اللغةفي بقية الفصل.

9.3.2. ارتباك Perplexity

بعد ذلك، دعنا نناقش كيفية قياس جودة نموذج اللغة، والتي سيتم استخدامها لتقييم نماذجنافي الأقسام التالية. إحدى الطرق هي التحقق من مدى دهشة النص surprising the text. نموذج اللغة الجيد قادر على التنبؤ برموز عالية الدقة بما سنراه بعد ذلك. تأمل الاستمرارية التالية لعبارة "إنها تمطر It is raining"، على النحو الذي اقترحته نماذج لغوية مختلفة:

1. "It is raining outside"
2. " It is raining banana tree"
3. "It is raining piouw;kcj pwepoiut"

من حيث الجودة، من الواضح أن المثال 1 هو الأفضل. الكلمات منطقية ومتماسكة منطقيًا.في حين أنه قد لا يعكس بدقة الكلمة التي تليها معنوية (كان من الممكن أن تكون " in San Francisco" و "in winter" امتدادات معقولة تمامًا)، إلا أن النموذج قادر على التقاط أي نوع من الكلمات التالية. المثال 2 أسوأ بكثير من خلال إنتاج امتداد غير منطقي. ومع ذلك، فقد تعلم النموذج على الأقل كيفية تهجئة الكلمات ودرجة معينة من الارتباط بين الكلمات. أخيرًا، يشير المثال 3 إلى نموذج سيئ التدريب لا يناسب البيانات بشكل صحيح.

قد نقيس جودة النموذج عن طريق حساب احتمالية التسلسل likelihood of the sequence. لسوء الحظ، هذا رقم يصعب فهمه ويصعب مقارنته. بعد كل شيء، من المرجح أن تحدث التسلسلات الأقصر بكثير من التسلسلات الأطول، وبالتالي فإن تقييم النموذج على ماغنوم أوبس تولستوي الحرب والسلام سينتج حتمًا احتمالية أقل بكثير من، على سبيل المثال،في رواية الأمير الصغير لسانت إكزوبيري. ما ينقص هو ما يعادل المتوسط.

نظرية المعلومات Information theory تأتيفي متناول اليد هنا. لقد حددنا الانتروبيا entropy، والمفاجأة surprisal، والانتروبيا المتقاطعة cross-entropy عندما قدمنا انحدار softmax (القسم 4.1.3). إذا أردنا ضغط النص، فيمكننا أن نسأل عن توقع الرمز token التالي في ضوء المجموعة الحالية من الرموز. يجب أن يسمح لنا نموذج اللغة الأفضل بالتنبؤ بالرمز التالي بشكل أكثر دقة. وبالتالي، يجب أن يسمح لنا بإنفاق عدد أقل من البتاتفي ضغط التسلسل. لذلك يمكننا قياسه من خلال متوسط خسارة الانتروبيا على جميع الرموز للتسلسل:

$$\frac{1}{n}\sum_{t=1}^{n} -\log P(x_t \mid x_{t-1}, \ldots, x_1), \quad (9.3.7)$$

حيث P يتم تقديمه بواسطة نموذج اللغة وهو الرمز الفعلي الذي يتم ملاحظتهفي الخطوة الزمنية tمن التسلسل. هذا يجعل الأداءفي المستندات ذات الأطوال المختلفة قابلاً للمقارنة. لأسباب تاريخية، يفضل العلماءفي معالجة اللغة الطبيعية استخدام كمية تسمى الارتباك perplexity. باختصار، هو أُس لـ (9.3.7):

$$\exp\left(-\frac{1}{n}\sum_{t=1}^{n} \log P(x_t \mid x_{t-1}, \ldots, x_1)\right).$$

يمكن فهم الارتباك Perplexity بشكل أفضل على أنها المتوسط الهندسي لعدد الخيارات الحقيقية التي لدينا عند تحديد الرمز token الذي نختاره بعد ذلك. دعونا نلقي نظرة على عدد من الحالات:

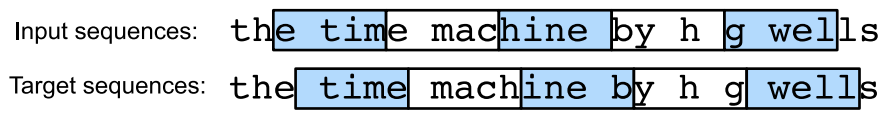
- في أفضل سيناريو، يقوم النموذج دائمًا بتقدير احتمالية الرمز المستهدف تمامًا مثل 1. في هذه الحالة، يكون ارتباك النموذج هو 1.
- في أسوأ السيناريوهات، يتنبأ النموذج دائمًا باحتمالية الرمز على أنه 0.في هذه الحالة، يكون الارتباك هو اللانهاية الموجبة positive infinity.
- في الأساس، يتنبأ النموذج بتوزيع موحد على جميع الرموز المتاحة للمفردات.في هذه الحالة، الارتباك تساوي عدد الرموز الفريدة unique tokens للمفردات.في الواقع، إذا قمنا بتخزين التسلسل دون أي ضغط، فسيكون هذا أفضل ما يمكننا فعله لتشفيره. ومن ثم، فإن هذا يوفر حدًا أعلى غير بديهي يجب أن يتغلب عليه أي نموذج مفيد.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

9.3.3. تسلسل التقسيم Partitioning Sequences

سنصمم نماذج لغوية باستخدام الشبكات العصبية ونستخدم الارتباك perplexity لتقييم مدى جودة النموذجفي توقع الرمز التالي بالنظر إلى المجموعة الحالية من الرموزفي التسلسلات النصية. قبل تقديم النموذج، لنفترض أنه يعالج دفعة صغيرة من التسلسلات بطول محدد مسبقًافي كل مرة. السؤال الآن هو كيف تقرأ الدفعات الصغيرة من تسلسلات الإدخال والتسلسلات المستهدفة بشكل عشوائي.

افترض أن مجموعة البيانات dataset تتخذ شكل سلسلة من مؤشرات الرمز T في corpus. سنقوم بتقسيمها إلى تكرارات لاحقة، حيث يكون لكل منها n رموز (خطوات زمنية). لتكرار (تقريبًا) جميع الرموز لمجموعة البيانات بأكملها لكل فترة epoch والحصول على جميع الأطوالn الممكنة، يمكننا تقديم العشوائية randomness. بشكل أكثر تحديداً،في بداية كل فترة، تجاهل الرموز الأولى d، حيث $d \in [0, n)$ يتم أخذ عينات بشكل موحد عشوائيًا. ثم يتم تقسيم بقية التسلسل إلى تسلسلات لاحقة $m = \lfloor (T - d)/n \rfloor$. قم بالإشارة إلى $\mathbf{x}_t = [x_t, \ldots, x_{t+n-1}]$ كطولnللتسلسلات التي تبدأ من الرمز في الخطوة الزمنية t. النتائج m اللاحقة المقسمة الناتجة هي $\mathbf{x}_d, \mathbf{x}_{d+n}, \ldots, \mathbf{x}_{d+n(m-1)}$. كل نتيجة لاحقة سيتم استخدامها كتسلسل إدخالفي نموذج اللغة.



الشكل 9.3.1 الحصول على 5 أزواج من متواليات الإدخال والتسلسلات المستهدفة من الطول المقسم ـ 5 المتتاليات اللاحقة.

بالنسبة لنمذجة اللغة، يتمثل الهدففي التنبؤ بالرمز التالي بناءً على الرموز التي رأيناها حتى الآن، ومن ثم فإن الأهداف (التسميات) هي التسلسل الأصلي، والتي تم إزاحتها بواسطة رمز واحد. التسلسل المستهدف لأي تسلسل $\mathbf{x}_t$ إدخال يكون $\mathbf{x}_{t+1}$ بطول n.

يوضح الشكل 9.3.1 مثالاً للحصول على 5 أزواج من متواليات الإدخال والتسلسلات المستهدفة مع $n = 5$ و $d = 2$.

```
@d2l.add_to_class(d2l.TimeMachine)  #@save
def __init__(self, batch_size, num_steps,
num_train=10000, num_val=5000):
    super(d2l.TimeMachine, self).__init__()
    self.save_hyperparameters()
    corpus, self.vocab = self.build(self._download())
    array = tf.constant([corpus[i:i+num_steps+1]
                         for i in range(0, len(corpus)-
num_steps-1)])
    self.X, self.Y = array[:,:-1], array[:,1:]
```

لتدريب نماذج اللغة، سنقوم بأخذ عينات عشوائية من أزواج من تسلسلات الإدخال والتسلسلات المستهدفةفي الدفعات الصغيرة. يُنشئ مُحمل البيانات data loader التالي بشكل عشوائي دفعة صغيرة من مجموعة البياناتفي كل مرة. تحدد الوسيطة `batch_size` عدد الأمثلة اللاحقة (`self.b`)في كل الدفعة الصغيرة و `num_steps` هو الطول التاليفي الرموز (`self.n`).

```
@d2l.add_to_class(d2l.TimeMachine)  #@save
def get_dataloader(self, train):
    idx = slice(0, self.num_train) if train else slice(
        self.num_train, self.num_train + self.num_val)
    return self.get_tensorloader([self.X, self.Y],
train, idx)
```

كما نرىفي ما يلي ، يمكن الحصول على دفعة صغيرة من التسلسلات المستهدفة عن طريق تحويل تسلسلات الإدخال بواسطة رمز واحد.

```
data = d2l.TimeMachine(batch_size=2, num_steps=10)
for X, Y in data.train_dataloader():
    print('X:', X, '\nY:', Y)
    break
```

```
X: tf.Tensor(
[[26  0  2  5 14 10 19  6  5  0]
 [14  0 21  9  6  0  4 19  2  5]], shape=(2, 10),
dtype=int32)
Y: tf.Tensor(
```

```
[[ 0  2  5 14 10 19  6  5  0  9]
 [ 0 21  9  6  0  4 19  2  5 13]], shape=(2, 10),
dtype=int32)
```

9.3.4. الملخص

نماذج اللغة تقدر الاحتمال المشترك لتسلسل نصي. بالنسبة للتسلسلات الطويلة، توفر n-gram نموذجًا مناسبًا عن طريق اقتطاع التبعية. ومع ذلك، هناك الكثير من المعمارية ولكن ليس هناك تردد كافٍ للتعامل مع مجموعات الكلمات النادرة بكفاءة عبر تجانس لابلاس. وبالتالي، سوف نركز على نمذجة اللغة العصبيةفي الأقسام اللاحقة. لتدريب نماذج اللغة، يمكننا عشوائيًا أخذ عينات من أزواج من تسلسلات الإدخال والتسلسلات المستهدفةفي الدفعات الصغيرة. بعد التدريب، سوف نستخدم الارتباك perplexity لقياس جودة النموذج اللغوي.

9.3.5. التمارين

1. افترض أن هناك 100,000 كلمات في مجموعة بيانات التدريب. ما مقدار تكرار الكلمات والتكرار المجاور متعدد الكلمات التي يحتاجها أربعة غرام للتخزين؟
2. كيف يمكنك أن تكون نموذجا للحوار dialogue؟
3. ما الطرق الأخرى التي يمكنك التفكير بها لقراءة بيانات التسلسل الطويل long sequence data؟
4. ضع في اعتبارك طريقتنا للتخلص من عدد عشوائي منتظم من الرموز tokens القليلة الأولى في بداية كل فترة.
   1. هل يؤدي حقًا إلى توزيع منتظم uniform distribution تمامًا على التسلسلات الموجودة في المستند؟
   2. ما الذي يجب عليك فعله لجعل الأشياء أكثر انتظاماً uniform؟
5. إذا أردنا أن يكون المثال المتسلسل جملة كاملة، فما نوع المشكلة التي يقدمها هذا في أخذ الدفعات الصغيرة؟ كيف يمكننا حل المشكلة؟

9.4 الشبكات العصبية المتكررة Recurrent Neural Networks

وصفنافي القسم 9.3 نماذج ماركوف و nـ غرام لنمذجة اللغة، حيث يعتمد الاحتمال الشرطي للرمز x_tفي الخطوة الزمنية t فقط على الرموز السابقة $n-1$. إذا أردنا دمج التأثير المحتمل للرموزفي وقت أبكر من الوقت $t-(n-1)$، فنحن بحاجة إلى زيادة n. ومع ذلك، فإن عدد معلمات النموذج سيزداد أيضًا بشكل كبير معه، حيث نحتاج إلى تخزين $|\mathcal{V}|^n$ أرقام لمجموعة مفردات $\mathcal{V}$. وبالتالي، بدلاً من نمذجة $P(x_t \mid x_{t-1}, \ldots, x_{t-n+1})$، من الأفضل استخدام نموذج متغير كامن latent variable model:

$$P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid h_{t-1}), \qquad (9.4.1)$$

حيث h_{t-1} هي حالة مخفية hidden state تخزن معلومات التسلسل حتى خطوة زمنية $t-1$. بشكل عام، يمكن حساب الحالة المخفيةفي أي خطوة زمنية t بناءً على كل من الإدخال الحالي x_t والحالة المخفية السابقة h_{t-1}:

$$h_t = f(x_t, h_{t-1}). \qquad (9.4.2)$$

بالنسبة لدالة قوية بدرجة كافيةفي (9.4.2)، فإن نموذج المتغير الكامن ليس تقريبيًا. بعد كل هذا، يمكن h_t ببساطة تخزين جميع البيانات التي لاحظتها حتى الآن. ومع ذلك، فمن المحتمل أن يجعل كلاً من الحساب والتخزين باهظ التكلفة.

تذكر أننا ناقشنا الطبقات المخفية مع الوحدات المخفيةفي القسم 5. ومن الجدير بالذكر أن الطبقات المخفية والحالات المخفية تشير إلى مفهومين مختلفين تمامًا. الطبقات المخفية، كما هو موضح، هي طبقات مخفية عن العرض على المسار من الإدخال إلى الإخراج. الحالات المخفية هي مدخلات تحدث تقنيًا لكل ما نقوم بهفي خطوة معينة، ولا يمكن حسابها إلا من خلال النظر إلى البياناتفي خطوات زمنية سابقة.

الشبكات العصبية المتكررة (RNNs) هي شبكات عصبية ذات حالات مخفية. قبل تقديم نموذجRNN ، نعيد أولاً زيارة نموذج MLP المقدمفي القسم 5.1.

9.4.1. الشبكات العصبية بدون الحالات المخفية Neural Networks without Hidden States

دعونا نلقي نظرة على MLP بطبقة واحدة مخفية. دع دالة تنشيط الطبقة المخفية تكون ϕ. بالنظر إلى دفعة صغيرة من الأمثلة $\mathbf{X} \in \mathbb{R}^{n \times d}$ مع حجم الدُفعة n والمدخلات d، يتم حساب ناتج الطبقة المخفية كـ

$$\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h). \qquad (9.4.3)$$

في (9.4.3) لدينا معامل الوزن $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ ومعامل التحيز $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ وعدد الوحدات المخفية h للطبقة المخفية. وبالتالي، يتم تطبيق البث broadcasting (انظر القسم 2.1.4) أثناء عملية الجمع summation. بعد ذلك، يتم استخدام إخراج الطبقة المخفية $\mathbf{H}$ كمدخل لطبقة الإخراج. يتم إعطاء طبقة الإخراج بواسطة

$$\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q,$$

حيث $\mathbf{O} \in \mathbb{R}^{n \times q}$ هو متغير الإخراج، $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ هو معلمة الوزن، و $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ هو معلمة التحيز لطبقة الإخراج. إذا كانت مشكلة تصنيف، فيمكننا استخدام $\text{softmax}(\mathbf{O})$ لحساب التوزيع الاحتمالي لفئات المخرجات.

هذا مشابه تمامًا لمشكلة الانحدار التي حللناها سابقًا في القسم 9.1، ومن ثم قمنا بحذف التفاصيل. يكفي أن نقول إنه يمكننا اختيار أزواج تسمية الميزات عشوائيًا وتعلم معلمات شبكتنا عبر التمايز التلقائي automatic differentiation والتدرج الاشتقاقي العشوائي SGD.

9.4.2. الشبكات العصبية المتكررة مع الحالات المخفية Recurrent Neural Networks with Hidden States

تختلف الأمور تمامًا عندما تكون لدينا حالات مخفية. دعونا نلقي نظرة على الهيكل ببعض التفاصيل.

افترض أن لدينا دفعة صغيرة من المدخلات $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ في الخطوة الزمنية t. بمعنى آخر، بالنسبة لدفعة صغيرة من أمثلة التسلسل n ، يتوافق كل صف مع مثال واحد في خطوة زمنية t من التسلسل. بعد ذلك، قم بالإشارة إلى $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ كإخراج الطبقة المخفية للخطوة الزمنية t. على عكس MLP، نحفظ هنا إخراج الطبقة المخفية من الخطوة الزمنية السابقة ونقدم معلمة وزن جديدة $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ لوصف كيفية استخدام إخراج الطبقة المخفية لخطوة الوقت السابقة في الخطوة الزمنية الحالية. على وجه التحديد، يتم تحديد حساب ناتج الطبقة المخفية لخطوة الوقت الحالية من خلال إدخال خطوة الوقت الحالي مع إخراج الطبقة المخفية لخطوة الوقت السابقة:

$$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h). \quad (9.5.4)$$

بالمقارنة مع (9.4.3)، يضيف (9.4.5) مصطلحًا آخر $\mathbf{H}_{t-1} \mathbf{W}_{hh}$، وبالتالي يُنشئ (9.4.2). من العلاقة بين مخرجات الطبقة المخفية $\mathbf{H}_t$ و $\mathbf{H}_{t-1}$ للخطوات الزمنية المجاورة، نعلم أن هذه المتغيرات تلتقط وتحتفظ بالمعلومات التاريخية للتسلسل حتى الخطوة الزمنية الحالية، تمامًا مثل حالة أو ذاكرة الخطوة الزمنية الحالية للشبكة العصبية. لذلك، يُطلق على إخراج الطبقة المخفية هذا الحالة المخفية hidden state. نظرًا لأن الحالة المخفية تستخدم نفس تعريف الخطوة الزمنية السابقة في الخطوة الزمنية الحالية، فإن حساب (9.4.5) متكرر recurrent. ومن ثم، كما قلنا، الشبكات العصبية ذات الحالات المخفية القائمة على الحساب المتكرر recurrent computation تسمى الشبكات العصبية المتكررة recurrent neural networks. تسمى الطبقات التي تقوم بحساب (9.4.5) في RNNs الطبقات المتكررة recurrent layers.

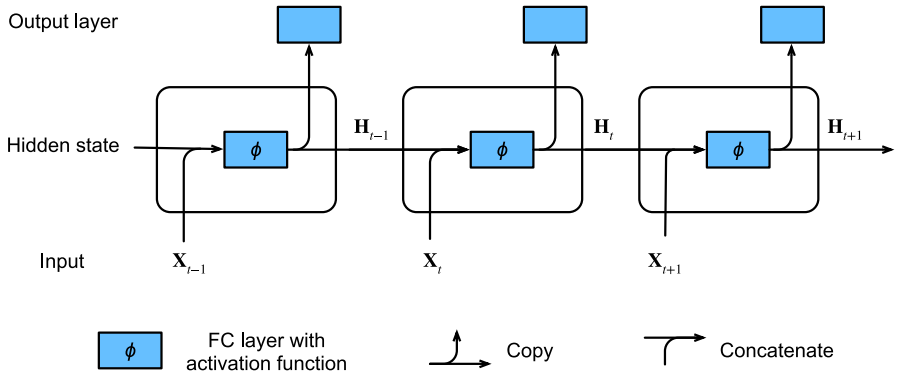
هناك العديد من الطرق المختلفة لبناء RNNs. RNNs ذات الحالة المخفية المحددة بواسطة (9.4.5) شائعة جداً. بالنسبة للخطوة الزمنية t، يكون إخراج طبقة الإخراج مشابهًا للحساب في MLP:

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$

تتضمن معلمات RNN الأوزان $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}, \mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ وانحياز $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ الطبقة المخفية، جنبًا إلى جنب مع الأوزان $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ وانحياز $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ طبقة الإخراج. من الجدير بالذكر أنه حتى في الخطوات الزمنية المختلفة، تستخدم RNN دائمًا معلمات النموذج هذه. لذلك، لا تنمو تكلفة المعلمات لـ RNN مع زيادة عدد الخطوات الزمنية.

يوضح الشكل 9.4.1 المنطق الحسابي computational logic لشبكة RNN في ثلاث خطوات زمنية متجاورة. في أي خطوة زمنية t، يمكن التعامل مع حساب الحالة المخفية $\mathbf{H}_{t-1}$ على النحو التالي: (1) ربط الإدخال $\mathbf{X}_t$ في الخطوة الزمنية الحالية t والحالة المخفية $\mathbf{H}_{t-1}$ في الخطوة الزمنية السابقة $t-1$؛ (2) تغذية نتيجة التسلسل في طبقة متصلة بالكامل بدالة التنشيط ϕ. ناتج هذه الطبقة المتصلة بالكامل هو الحالة المخفية $\mathbf{H}_t$ لخطوة الوقت الحالية t. في هذه الحالة، معلمات النموذج هي تسلسل $\mathbf{W}_{xh}$ و $\mathbf{W}_{hh}$، وانحياز $\mathbf{b}_h$، الكل من (9.4.5). ستشارك الحالة المخفية $\mathbf{H}_t$ للخطوة الزمنية الحالية t في حساب الحالة المخفية $\mathbf{H}_{t+1}$ للخطوة الزمنية التالية $t+1$. علاوة على ذلك، $\mathbf{H}_t$ سيتم إدخالها أيضًا في طبقة الإخراج المتصلة بالكامل لحساب ناتج $\mathbf{O}_t$ الخطوة الزمنية الحالية t.

لقد ذكرنا للتو أن حساب $\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh}$ للحالة المخفية يعادل ضرب المصفوفة لتسلسل $\mathbf{X}_t$ و $\mathbf{H}_{t-1}$ وتسلسل $\mathbf{W}_{xh}$ و $\mathbf{W}_{hh}$. على الرغم من أنه يمكن إثبات ذلك في الرياضيات، إلا أننا فيما يلي نستخدم مقتطفًا بسيطًا من التعليمات البرمجية لإظهار ذلك. بادئ ذي بدء، نحدد المصفوفات X و W_xh و H و W_hh ، والتي تكون أشكالها (3, 1)، (1, 4) ، (3, 4) ، (4, 4) على التوالي. بضرب X في W_xh ، و H في W_hh ، على التوالي ، ثم إضافة هذين المضاعفين، نحصل على مصفوفة الشكل (3, 4).



شكل 9.4.1 RNN مع حالة مخفية.

```
import tensorflow as tf
from d2l import tensorflow as d2l

X, W_xh = tf.random.normal((3, 1)), tf.random.normal((1,
4))
H, W_hh = tf.random.normal((3, 4)), tf.random.normal((4,
4))
tf.matmul(X, W_xh) + tf.matmul(H, W_hh)
```

```
<tf.Tensor: shape=(3, 4), dtype=float32, numpy=
array([[-2.7432978 , -2.2042181 , -4.0852065 ,  2.702191
],
       [ 3.1250827 ,  0.27091736,  1.192738  ,
0.39742705],
       [ 3.429378  ,  1.6246774 ,  2.52065   , -
0.16307715]],
      dtype=float32)>
```

الآن نقوم بتجميع المصفوفات `X` و `H` على طول الأعمدة (المحور 1)، والمصفوفات `W_xh` و `W_hh` على طول الصفوف (المحور 0). ينتج عن هاتين السلسلتين مصفوفات الشكل (3, 5) والشكل (5, 4)، على التوالي. بضرب هاتين المصفوفتين المتسلسلتين، نحصل على نفس مصفوفة الإخراج للشكل (3, 4) على النحو الوارد أعلاه.

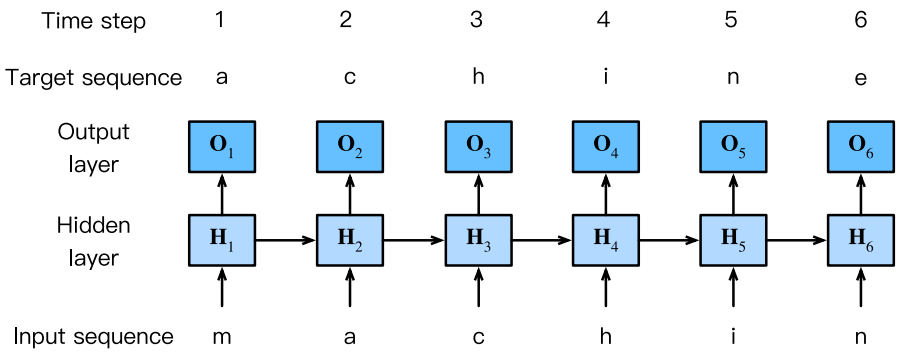
```
tf.matmul(tf.concat((X, H), 1), tf.concat((W_xh, W_hh),
0))
```

```
<tf.Tensor: shape=(3, 4), dtype=float32, numpy=
array([[-2.7432978 , -2.2042184 , -4.0852065 ,
2.7021914 ],
```

```
       [ 3.1250825 ,  0.27091733,  1.192738  ,
0.39742705],
       [ 3.429378  ,  1.6246774 ,  2.52065   , -
0.16307715]],
      dtype=float32)>
```

9.4.3. نماذج اللغة على مستوى الأحرف المستندة إلى RNN RNN-based Character-Level Language Models

تذكر أنه بالنسبة لنمذجة اللغةفي القسم 9.3، فإننا نهدف إلى التنبؤ بالرمز التالي بناءً على الرموز الحالية والسابقة، وبالتالي نقوم بتحويل التسلسل الأصلي برمز واحد كأهداف (تسميات). بنجيو وآخرون. اقترح أولاً استخدام شبكة عصبية لنمذجة اللغة (Bengio et al.، 2003). فيما يلي نوضح كيف يمكن استخدام RNNs لبناء نموذج لغوي. دع حجم الدفعات الصغيرة يكون واحدًا، وتسلسل النص يكون "machine". لتبسيط التدريبفي الأقسام اللاحقة، نقوم بترميز tokenize النص إلى أحرف بدلاً من الكلمات والنظرفي نموذج لغة على مستوى الحرف. يوضح الشكل 9.4.2 كيفية التنبؤ بالحرف التالي بناءً على الأحرف الحالية والسابقة عبر RNN لنمذجة اللغة على مستوى الحرف RNN for character-level language modeling.



الشكل 9.4.2 نموذج لغة على مستوى الحرف يعتمد على RNN. تسلسل المدخلات والهدف هما "machin" و "achine"، على التوالي.

أثناء عملية التدريب، نقوم بتشغيل عملية softmax على الإخراج من طبقة الإخراج لكل خطوة زمنية، ثم نستخدم خطأ الانتروبيا المتقاطعة لحساب الخطأ بين إخراج النموذج والهدف. بسبب الحساب المتكرر للحالة المخفيةفي الطبقة المخفية، يتم تحديد ناتج الخطوة الزمنية 3في الشكل 9.4.2، من خلال تسلسل النص "m" و "a" و "c". نظرًا لأن الحرف التالي للتسلسلفي بيانات التدريب هو "h"، فإن خطأ الوقت الخطوة 3 ستعتمد على توزيع الاحتمالية للحرف التالي الذي تم إنشاؤه بناءً على تسلسل الميزات "m" و "a" و "c" والهدف "h" لهذه الخطوة الزمنية.

في الممارسة العملية، يتم تمثيل كل رمز من خلال متجه ذو الأبعاد d، ونستخدم حجم الدُفعة $n > 1$. لذلك، ستكون المدخلات $\mathbf{X}_t$ في الخطوة الزمنية t عبارة عن مصفوفة $n \times d$ مطابقة لما ناقشناه في القسم 9.4.2.

في الأقسام التالية، سنقوم بتنفيذ RNNs لنماذج اللغة على مستوى الأحرف.

9.4.4. الملخص

تسمى الشبكة العصبية التي تستخدم الحساب المتكرر recurrent computation للحالات المخفية hidden states الشبكة العصبية المتكررة (RNN). يمكن للحالة المخفية لـ RNN التقاط المعلومات التاريخية للتسلسل حتى الخطوة الزمنية الحالية. مع الحساب المتكرر، لا يزداد عدد معلمات نموذج RNN مع زيادة عدد الخطوات الزمنية. بالنسبة للتطبيقات، يمكن استخدام RNN لإنشاء نماذج لغة على مستوى الأحرف character-level language models.

9.4.5. التمارين

1. إذا استخدمنا RNN للتنبؤ بالحرف التالي في تسلسل نصي، فما هو البُعد المطلوب لأي إخراج؟
2. لماذا يمكن لـ RNN التعبير عن الاحتمال الشرطي لرمز في خطوة زمنية معينة بناءً على جميع الرموز السابقة في تسلسل النص؟
3. ماذا يحدث للتدرج gradient إذا قمت بالنشر الخلفي من خلال تسلسل طويل long sequence؟
4. ما هي بعض المشاكل المرتبطة بنموذج اللغة الموصوف في هذا القسم؟

9.5 تنفيذ الشبكة العصبية المتكررة من الصفر Recurrent Neural Network Implementation from Scratch

نحن الآن جاهزون لتنفيذ RNN من البداية. على وجه الخصوص، سنقوم بتدريب RNN هذا للعمل كنموذج لغة على مستوى الأحرف (انظر القسم 9.4) وتدريبه على مجموعة تتكون من النص الكامل لـ ""The Time Machine"" لـ H.G Wells، باتباع خطوات معالجة البيانات الموضحة في القسم 9.2. نبدأ بتحميل مجموعة البيانات.

```
%matplotlib inline
import math
import tensorflow as tf
from d2l import tensorflow as d2l
```

9.5.1. نموذج RNN

نبدأ بتحديد فئة لتنفيذ نموذج RNN (القسم 9.4.2). لاحظ أن عدد الوحدات المخفية `num_hiddens` عبارة عن معلمة فائقة قابلة للضبط tunable hyperparameter.

```
class RNNScratch(d2l.Module):  #@save
    def __init__(self, num_inputs, num_hiddens,
sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W_xh = tf.Variable(tf.random.normal(
            (num_inputs, num_hiddens)) * sigma)
        self.W_hh = tf.Variable(tf.random.normal(
            (num_hiddens, num_hiddens)) * sigma)
        self.b_h = tf.Variable(tf.zeros(num_hiddens))
```

تحدد طريقة `forward` أدناه كيفية حساب المخرجات والحالة المخفيةفي أي خطوة زمنية، بالنظر إلى الإدخال الحالي وحالة النموذجفي الخطوة الزمنية السابقة. لاحظ أن نموذج RNN يمر عبر البُعد الخارجي للمدخلات `inputs`، محدثًا الحالة المخفية خطوة واحدةفي كل مرة. النموذج هنا يستخدم دالة التنشيط tanh (القسم 5.1.2.3).

```
@d2l.add_to_class(RNNScratch)  #@save
def forward(self, inputs, state=None):
    if state is not None:
        state, = state
        state = tf.reshape(state, (-1,
self.W_hh.shape[0]))
    outputs = []
    for X in inputs:  # Shape of inputs: (num_steps,
batch_size, num_inputs)
        state = tf.tanh(tf.matmul(X, self.W_xh) + (
            tf.matmul(state, self.W_hh) if state is not
None else 0)
                        + self.b_h)
        outputs.append(state)
    return outputs, state
```

يمكننا تغذية الدفعة الصغيرة minibatch من تسلسل الإدخالفي نموذج RNN على النحو التالي.

```
batch_size, num_inputs, num_hiddens, num_steps = 2, 16,
32, 100
rnn = RNNScratch(num_inputs, num_hiddens)
```

```
X = tf.ones((num_steps, batch_size, num_inputs))
outputs, state = rnn(X)
```

دعنا نتحقق مما إذا كان نموذج RNN ينتج نتائج الأشكال الصحيحة للتأكد من بقاء أبعاد الحالة المخفية دون تغيير.

```
def check_len(a, n):  #@save
    assert len(a) == n, f'list\'s len {len(a)} !=
expected length {n}'

def check_shape(a, shape):  #@save
    assert a.shape == shape, \
            f'tensor\'s shape {a.shape} != expected
shape {shape}'

d2l.check_len(outputs, num_steps)
d2l.check_shape(outputs[0], (batch_size, num_hiddens))
d2l.check_shape(state, (batch_size, num_hiddens))
```

9.5.2. نموذج اللغة القائم على RNN RNN-based Language Model

تحدد فئة `RNNLMScratch` التالية نموذج لغة قائم على RNN ، حيث نمررفي RNN عبر الوسيطة `rnn` للطريقة `__init__`. عند تدريب النماذج اللغوية، تكون المدخلات والمخرجات من نفس المفردات. ومن ثم، فإنهما لهما نفس البعد، والذي يساوي حجم المفردات vocabulary size. لاحظ أننا نستخدم الارتباك perplexity لتقييم النموذج. كما تمت مناقشتهفي القسم 9.3.2، يضمن هذا إمكانية مقارنة التسلسلات ذات الأطوال المختلفة.

```
class RNNLMScratch(d2l.Classifier):  #@save
    def __init__(self, rnn, vocab_size, lr=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.init_params()

    def init_params(self):
        self.W_hq = tf.Variable(tf.random.normal(
            (self.rnn.num_hiddens, self.vocab_size)) *
self.rnn.sigma)
        self.b_q =
tf.Variable(tf.zeros(self.vocab_size))

    def training_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('ppl', tf.exp(l), train=True)
```

```
        return l

    def validation_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('ppl', tf.exp(l), train=False)
```

9.5.2.1. ترميز واحد ساخن One-Hot Encoding

تذكر أن كل رمز يتم تمثيله بمؤشر رقمي يشير إلى الموضعفي مفردات الكلمة / الحرف / قطعة الكلمة المقابلة. قد تميل إلى بناء شبكة عصبية مع عقدة إدخال واحدة (في كل خطوة زمنية)، حيث يمكن تغذية الفهرس كقيمة عددية. يعمل هذا عندما نتعامل مع مدخلات عددية مثل السعر أو درجة الحرارة، حيث يجب التعامل مع أي قيمتين قريبتين بشكل كافٍ من بعضهما البعض بالمثل. لكن هذا ليس منطقيًا تمامًا. تصادف أن تكون الكلمات 45^{th} والكلمات 46^{th}في مفرداتنا "their " و "said "، والتي لا تتشابه معانيها عن بُعد.

عند التعامل مع مثل هذه البيانات الفئوية categorical data، فإن الإستراتيجية الأكثر شيوعًا هي تمثيل كل عنصر بترميز واحد ساخن ne-hot encoding (استدعاء من القسم 4.1.1). ترميز واحد ساخن هو متجه يتم تحديد طوله من خلال حجم المفردات N ، حيث يتم تعيين جميع الإدخالات على 0 ، باستثناء الإدخال المقابل للرمز المميز الخاص بنا، والذي تم تعيينه على 1. على سبيل المثال، إذا كانت المفردات تحتوي على 5 عناصر، فإن المتجهات الساخنة الواحدة المقابلة للمؤشرات 0 و 2 ستكون على النحو التالي.

```
tf.one_hot(tf.constant([0, 2]), 5)
```

```
<tf.Tensor: shape=(2, 5), dtype=float32, numpy=
array([[1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.]], dtype=float32)>
```

سوف تأخذ الدفعات الصغيرة التي نختبرهافي كل تكرار الشكل (حجم الدُفعة batch size، عدد الخطوات الزمنية number of time steps). بمجرد تمثيل كل إدخال على أنه متجه واحد ساخن، يمكننا التفكيرفي كل دفعة صغيرة على أنه موتر ثلاثي الأبعاد، حيث يتم إعطاء الطول على طول المحور الثالث من خلال حجم المفردات (`len(vocab)`). غالبًا ما نحول المدخلات حتى نحصل على ناتج من الشكل (عدد الخطوات الزمنية، حجم الدفعة، حجم المفردات vocabulary size). سيتيح لنا ذلك إجراء حلقة أكثر ملاءمة عبر البعد الخارجي لتحديث الحالات المخفية للدفعة الصغيرة، والوقت خطوة بخطوة (على سبيل المثال،في طريقة `forward` أعلاه).

```
@d2l.add_to_class(RNNLMScratch)  #@save
def one_hot(self, X):
    # Output shape: (num_steps, batch_size, vocab_size)
```

```
    return tf.one_hot(tf.transpose(X), self.vocab_size)
```

9.5.2.2. تحويل مخرجات RNN Transforming RNN Outputs

يستخدم نموذج اللغة طبقة إخراج متصلة بالكامل لتحويل مخرجات RNN إلى تنبؤات رمزية token predictions في كل خطوة زمنية.

```
@d2l.add_to_class(RNNLMScratch)  #@save
def output_layer(self, rnn_outputs):
    outputs = [tf.matmul(H, self.W_hq) + self.b_q for H
in rnn_outputs]
    return tf.stack(outputs, 1)

@d2l.add_to_class(RNNLMScratch)  #@save
def forward(self, X, state=None):
    embs = self.one_hot(X)
    rnn_outputs, _ = self.rnn(embs, state)
    return self.output_layer(rnn_outputs)
```

دعنا نتحقق مما إذا كان الحساب الأمامي forward computation ينتج مخرجات بالشكل الصحيح.

```
model = RNNLMScratch(rnn, num_inputs)
outputs = model(tf.ones((batch_size, num_steps),
dtype=tf.int64))
d2l.check_shape(outputs, (batch_size, num_steps,
num_inputs))
```

9.5.3. قص التدرج Gradient Clipping

بينما كنت معتادًا بالفعل على التفكير في الشبكات العصبية على أنها "عميقة" بمعنى أن العديد من الطبقات تفصل بين المدخلات والمخرجات حتى في غضون خطوة زمنية واحدة، فإن طول التسلسل يقدم مفهومًا جديدًا للعمق. بالإضافة إلى المرور عبر الشبكة في اتجاه الإدخال إلى الإخراج، يجب أن تمر المدخلات في الخطوة الأولى عبر سلسلة من الطبقات T على طول الخطوات الزمنية للتأثير على إخراج النموذج في الخطوة الزمنية النهائية. بأخذ وجهة النظر العكسية، في كل تكرار، نقوم بإعادة نشر التدرجات عبر الزمن، مما ينتج عنه سلسلة من ضرب المصفوفة ذات الطول $\mathcal{O}(T)$. كما هو مذكور في القسم 5.4، يمكن أن يؤدي هذا إلى عدم الاستقرار العددي، مما يتسبب في انفجار التدرجات أو اختفائها اعتمادًا على خصائص مصفوفات الوزن.

يعد التعامل مع التدرجات المتلاشية والمتفجرة مشكلة أساسية عند تصميم شبكات RNN وقد ألهمت بعضًا من أكبر التطورات في هياكل الشبكات العصبية الحديثة. في الفصل التالي،

سنتحدث عن البُنى المتخصصة التي تم تصميمها على أمل التخفيف من مشكلة التدرج المتلاشي vanishing gradient. ومع ذلك، حتى RNNs الحديثة لا تزال تعاني في كثير من الأحيان من انفجار التدرجات exploding gradients. أحد الحلول غير الأنيقة ولكنه موجود في كل مكان هو قص التدرجات Gradients Clipping ببساطة لإجبار التدرجات "المقطوعة clipped " الناتجة على أخذ قيم أصغر.

بشكل عام، عند تحسين بعض الأهداف عن طريق التدرج الاشتقاقي gradient descent، نقوم بتحديث متكرر للمعامل موضع الاهتمام، على سبيل المثال متجه $\mathbf{x}$ ، ولكن ندفعه في اتجاه التدرج السلبي $\mathbf{g}$ (في التدرج الاشتقاقي العشوائي SGD، نحسب هذا التدرج على عينة عشوائية من الدفعات الصغيرة). على سبيل المثال، مع معدل التعلم $\eta > 0$ ، يأخذ كل تحديث النموذج الشكل $\mathbf{x} \leftarrow \mathbf{x} - \eta \mathbf{g}$. لنفترض كذلك أن الدالة الموضوعية f سلسة بدرجة كافية. رسميًا، نقول إن الهدف هو Lipschitz مستمر مع ثابت L ، وهذا يعني أنه بالنسبة لأي $\mathbf{x}$ و $\mathbf{y}$ ، لدينا

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\|.$$

كما ترى، عندما نقوم بتحديث متجه المعلمة عن طريق طرح $\eta \mathbf{g}$ ، فإن التغيير في قيمة الهدف يعتمد على معدل التعلم ومعيار التدرج وL على النحو التالي:

$$|f(\mathbf{x}) - f(\mathbf{x} - \eta \mathbf{g})| \leq L\eta\|\mathbf{g}\|.$$

بمعنى آخر، لا يمكن أن يتغير الهدف بأكثر من $L\eta\|\mathbf{g}\|$. قد يُنظر إلى وجود قيمة صغيرة لهذا الحد الأعلى على أنه أمر جيد أو سيء. على الجانب السلبي، نحن نحد من السرعة التي يمكننا بها تقليل قيمة الهدف. على الجانب المشرق، يحد هذا من مقدار الخطأ الذي يمكن أن نخطئ فيه في أي خطوة من خطوات الانحدار.

عندما نقول إن التدرجات تنفجر، فإننا نعني أن $\|\mathbf{g}\|$ يصبح كبيرًا للغاية. في هذه الحالة الأسوأ، قد نحدث الكثير من الضرر في خطوة متدرجة واحدة بحيث يمكننا التراجع عن كل التقدم الذي تم إحرازه على مدار آلاف التكرارات التدريبية. عندما يمكن أن تكون التدرجات كبيرة جدًا، غالبًا ما يتباعد تدريب الشبكة العصبية، ويفشل في تقليل قيمة الهدف. في أوقات أخرى، يتقارب التدريب في النهاية ولكنه غير مستقر بسبب الارتفاع الهائل في الخطأ.

تتمثل إحدى طرق الحد من حجم $L\eta\|\mathbf{g}\|$ في تقليص معدل التعلم η إلى قيم صغيرة. تتمثل إحدى الميزات هنا في أننا لا نحيز التحديثات. لكن ماذا لو نادرًا ما نحصل على تدرجات كبيرة؟ هذه الخطوة الجذرية تبطئ تقدمنا في جميع الخطوات، فقط للتعامل مع أحداث التدرج المتفجر النادرة. البديل الشائع هو تبني قص التدرج الإرشادي gradient clipping heuristic لإسقاط التدرجات $\mathbf{g}$ على كرة من نصف قطر θ معين على النحو التالي:

$$\mathbf{g} \leftarrow min(1, \frac{\theta}{\|\mathbf{g}\|})\mathbf{g}.$$

هذا يضمن ان معيار التدرج لا يتجاوز θ أبدًا وأن التدرج المحدث يتماشى تمامًا مع الاتجاه الأصلي لـ $\mathbf{g}$. كما أن لها أيضًا تأثير جانبي مرغوب فيه للحد من التأثير الذي يمكن أن تمارسه أي دفعة صغيرة (وداخله أي عينة معينة) على متجه المعلمة. هذا يمنح درجة معينة من المتانة للنموذج. لنكون واضحين، إنه اختراق. يعني قص التدرج Gradient clipping أننا لا نتبع دائمًا التدرج الحقيقي ومن الصعب التفكير بشكل تحليليفي الآثار الجانبية المحتملة. ومع ذلك، فهو اختراق مفيد للغاية، ويتم اعتماده على نطاق واسعفي تطبيقات RNNفي معظم أطر التعلم العميق.

نحدد أدناه طريقة لقص التدرجات، والتي يتم استدعاؤها بواسطة طريقة `fit_epoch` لفئة `d2l.Trainer` (انظر القسم 3.4). لاحظ أنه عند حساب معيار التدرج gradient norm، فإننا نجمع جميع معلمات النموذج، ونتعامل معها على أنها متجه معلمة عملاق واحد.

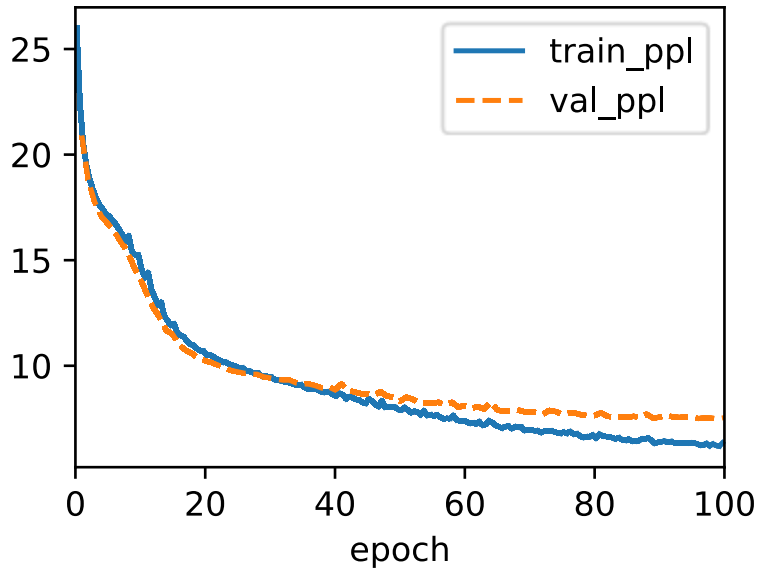
```
@d2l.add_to_class(d2l.Trainer)  #@save
def clip_gradients(self, grad_clip_val, grads):
    grad_clip_val = tf.constant(grad_clip_val,
dtype=tf.float32)
    new_grads = [tf.convert_to_tensor(grad) if
isinstance(
        grad, tf.IndexedSlices) else grad for grad in
grads]
    norm = tf.math.sqrt(sum((tf.reduce_sum(grad ** 2))
for grad in new_grads))
    if tf.greater(norm, grad_clip_val):
        for i, grad in enumerate(new_grads):
            new_grads[i] = grad * grad_clip_val / norm
        return new_grads
    return grads
```

9.5.4. التدريب Training

باستخدام مجموعة بيانات (بيانات) The Time Machine، نقوم بتدريب نموذج لغة على مستوى الأحرف (`model`) بناءً على (`rnn`) RNN المنفذ من البداية. لاحظ أننا نحسب التدرجات أولاً، ثم نقصها (نقطعها)، ونقومفي النهاية بتحديث معلمات النموذج باستخدام التدرجات المقطوعة clipped gradients.

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
with d2l.try_gpu():
```

```
    rnn = RNNScratch(num_inputs=len(data.vocab),
num_hiddens=32)
    model = RNNLMScratch(rnn,
vocab_size=len(data.vocab), lr=1)
trainer = d2l.Trainer(max_epochs=100,
gradient_clip_val=1)
trainer.fit(model, data)
```



9.5.5. فك الترميز Decoding

بمجرد تعلم نموذج اللغة، يمكننا استخدامه ليس فقط للتنبؤ بالرمز التالي ولكن لمواصلة التنبؤ بكل رمز لاحق، والتعامل مع الرمز المتوقع مسبقًا كما لو كان الرمز التاليفي الإدخال.في بعض الأحيان نرغب فقطفي إنشاء نص كما لو كنا نبدأفي بداية المستند. ومع ذلك، غالبًا ما يكون من المفيد شرط نموذج اللغة على بادئة يوفرها المستخدم. على سبيل المثال، إذا كنا نطور ميزة الإكمال التلقائي لمحرك البحث أو لمساعدة المستخدمينفي كتابة رسائل البريد الإلكتروني، فنحن نرغبفي تغذية ما كتبوه حتى الآن (البادئة the prefix)، ثم إنشاء استمرار محتمل.

تولد دالة التنبؤ `predict` التالية استمرارًا ، حرفًا واحدًافي كل مرة ، بعد إدخال بادئة `prefix` مقدمة من المستخدم ، عند المرور عبر الأحرف الموجودةفي البادئة `prefix`، نستمرفي تمرير الحالة المخفية إلى الخطوة التالية ولكن لا يتم إنشاء أي ناتج. وهذا ما يسمى بفترة الإحماء warm-up period. بعد استيعاب البادئة، نحن الآن على استعداد لبدء إرسال الأحرف التالية، والتي سيتم إعادة إدخال كل منهافي النموذج كمدخلفي الخطوة الزمنية اللاحقة.

```
@d2l.add_to_class(RNNLMScratch)  #@save
def predict(self, prefix, num_preds, vocab,
device=None):
    state, outputs = None, [vocab[prefix[0]]]
    for i in range(len(prefix) + num_preds - 1):
        X = tf.constant([[outputs[-1]]])
        embs = self.one_hot(X)
        rnn_outputs, state = self.rnn(embs, state)
        if i < len(prefix) - 1:  # Warm-up period
            outputs.append(vocab[prefix[i + 1]])
        else:  # Predict `num_preds` steps
            Y = self.output_layer(rnn_outputs)
            outputs.append(int(tf.reshape(tf.argmax(Y,
axis=2), 1)))
    return ''.join([vocab.idx_to_token[i] for i in
outputs])
```

فيما يلي، نحدد البادئة ونجعلها تولد 20 حرفًا إضافيًا.

```
model.predict('it has', 20, data.vocab)
```

```
'it has hathe the peat he a'
```

في حين أن تنفيذ نموذج RNN أعلاه من البداية مفيد، إلا أنه ليس مناسبًا.في القسم التالي، سنرى كيفية الاستفادة من أطر التعلم العميق لتوجيه RNNs باستخدام البنى القياسية، وجني مكاسبفي الأداء من خلال الاعتماد على دوال المكتبة المحسنة للغاية.

9.5.6. الملخص

يمكننا تدريب نماذج اللغة المستندة إلى RNN لإنشاء نص يتبع بادئة النص المقدمة من المستخدم user-provided text prefix. يتكون نموذج لغة RNN البسيط من ترميز الإدخال ونمذجة RNN وتوليد الإخراج. أثناء التدريب، يمكن لقص التدرج gradient clipping أن يخفف من مشكلة انفجار التدرجات ولكنه لا يعالج مشكلة تلاشي التدرجات.في التجربة، قمنا بتنفيذ نموذج بسيط للغة RNN وقمنا بتدريبه باستخدام قص التدرج على تسلسلات نصية، تم ترميزها على مستوى الحرف. من خلال التكييف على بادئة، يمكننا استخدام نموذج اللغة لإنشاء عمليات استمرارية محتملة، والتي تثبت فائدتهافي العديد من التطبيقات، على سبيل المثال، ميزات الإكمال التلقائي.

9.5.7. التمارين

1. هل يتنبأ نموذج اللغة المطبق بالرمز التالي بناءً على جميع الرموز السابقة حتى أول رمز في The Time Machine؟

2. ما هي المعلمة الفائقة التي تتحكمفي طول السجل المستخدم للتنبؤ؟
3. أظهر أن الترميز الساخن يعادل اختيار تضمين مختلف لكل كائن.
4. اضبط المعلمات الفائقة (على سبيل المثال، عدد الفترات، عدد الوحدات المخفية، عدد الخطوات الزمنيةفي الدفعات الصغيرة، ومعدل التعلم) لتحسين الارتباك perplexity. إلى أي مدى يمكن أن تنخفض بينما تتمسك بهذه البُنية البسيطة؟
5. استبدل الترميز الواحد الساخن one-hot encoding بالتضمينات القابل للتعلم learnable embeddings. هل هذا يؤدي إلى أداء أفضل؟
6. قم بإجراء تجربة لتحديد مدى جودة عمل نموذج اللغة هذا الذي تم تدريبه على The Time Machine على كتب أخرى من تأليف H.G Wells، على سبيل المثال، The War of the Worlds.
7. قم بإجراء تجربة أخرى لتقييم مدى ارتباك هذا النموذجفي الكتب التي كتبها مؤلفون آخرون.
8. قم بتعديل دالة التنبؤ مثل استخدام أخذ العينات بدلاً من اختيار الحرف التالي الأكثر احتمالاً.
   - ماذا يحدث؟
   - قم بتحيز النموذج نحو المخرجات الأكثر احتمالية، على سبيل المثال، عن طريق أخذ العينات من $q(x_t \mid x_{t-1}, \ldots, x_1) \propto P(x_t \mid x_{t-1}, \ldots, x_1)^{\alpha}$.
9. قم بتشغيل الكودفي هذا القسم دون قص التدرج. ماذا يحدث؟
10. استبدل دالة التنشيط المستخدمةفي هذا القسم بـ ReLU وكرر التجاربفي هذا القسم. هل ما زلنا بحاجة إلى قص التدرج؟ لماذا؟

9.6 التنفيذ المختصر للشبكات العصبية المتكررة Concise Implementation of Recurrent Neural Networks

مثل معظم تطبيقاتنا من البداية، تم تصميم القسم 9.5 لتوفير نظرة ثاقبة حول كيفية عمل كل مكون. ولكن عندما تستخدم RNNs كل يوم أو تكتب رمز الإنتاج، سترغبفي الاعتماد أكثر على المكتبات التي تقلل من وقت التنفيذ (من خلال توفير كود المكتبة للنماذج والدوال الشائعة) ووقت الحساب (عن طريق تحسين الخروج من تطبيقات المكتبة هذه). سيوضح لك هذا القسم كيفية تنفيذ نموذج اللغة نفسه بشكل أكثر كفاءة باستخدام واجهة برمجة التطبيقات عالية المستوى التي يوفرها إطار عمل التعلم العميق الخاص بك. نبدأ، كمافي السابق، بتحميل مجموعة بيانات The Time Machine.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

9.6.1. تعريف النموذج Defining the Model

نحدد الفئة التالية باستخدام RNN المنفذة بواسطة واجهات برمجة التطبيقات عالية المستوى API.

```
class RNN(d2l.Module):  #@save
    def __init__(self, num_inputs, num_hiddens):
        super().__init__()
        self.save_hyperparameters()
        self.rnn = nn.RNN(num_inputs, num_hiddens)

    def forward(self, inputs, H=None):
        return self.rnn(inputs, H)
```

موروثًا من فئة RNNLMScratch في القسم 9.5، تحدد فئة RNNLM التالية نموذجًا للغة قائمًا على RNN. لاحظ أننا بحاجة إلى إنشاء طبقة إخراج منفصلة متصلة بالكامل.

```
class RNNLM(d2l.RNNLMScratch):  #@save
    def init_params(self):
        self.linear =
tf.keras.layers.Dense(self.vocab_size)

    def output_layer(self, hiddens):
        return tf.transpose(self.linear(hiddens), (1, 0,
2))
```

9.6.2. التدريب والتنبؤ Training and Predicting

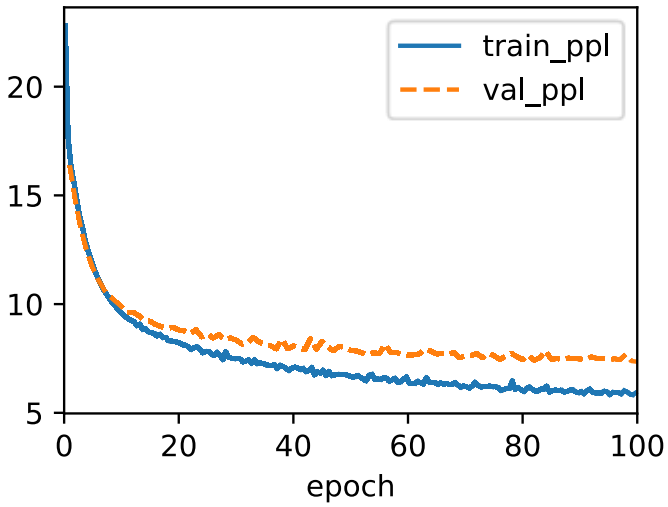
قبل تدريب النموذج، دعنا نتوقع باستخدام نموذج تمت تهيئته باستخدام أوزان عشوائية. نظرًا لأننا لم ندرب الشبكة، فسوف تولد تنبؤات لا معنى لها.

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
rnn = RNN(num_hiddens=32)
model = RNNLM(rnn, vocab_size=len(data.vocab), lr=1)
model.predict('it has', 20, data.vocab)
```

```
'it hasicaj<unk>zbjqyzttxte<unk>con'
```

بعد ذلك، نقوم بتدريب نموذجنا ، والاستفادة من واجهة برمجة التطبيقات عالية المستوى API.

```
with d2l.try_gpu():
    trainer = d2l.Trainer(max_epochs=100,
gradient_clip_val=1)
trainer.fit(model, data)
```



بالمقارنة مع القسم 9.5، يحقق هذا النموذج ارتباكًا مشابهًا comparable perplexity، ولكنه يعمل بشكل أسرع بسبب عمليات التنفيذ المحسّنة. كمافي السابق، يمكننا إنشاء الرموز المتوقعة predicted tokens بعد سلسلة البادئة المحددة.

```
model.predict('it has', 20, data.vocab)
```

```
'it has and the thice the t'
```

9.6.3. الملخص

توفر واجهات برمجة التطبيقات عالية المستوى API في أطر التعلم العميق تطبيقات RNNs القياسية. تساعدك هذه المكتبات على تجنب إضاعة الوقتفي إعادة تنفيذ النماذج القياسية. علاوة على ذلك، غالبًا ما يتم تحسين تطبيقات إطار العمل بشكل كبير، مما يؤدي إلى مكاسب كبيرةفي الأداء (الحسابي) مقارنة بالتطبيقات من البداية.

9.6.4. التمارين

1. هل يمكنك جعل نموذج RNN يعاني من الضبط الزائد overfitting باستخدام واجهات برمجة التطبيقات عالية المستوى API؟
2. نفذ نموذج الانحدار الذاتي autoregressive model للقسم 9.1 باستخدام RNN.

9.7 الانتشار الخلفي عبر الزمن Backpropagation Through Time

إذا أكملت التمارين الواردة في القسم 9.5 ، فستلاحظ أن قص التدرج gradient clipping أمر حيوي لمنع التدرجات الضخمة العرضية من التدريب المزعزع للاستقرار. لقد ألمحنا إلى أن

التدرجات المتفجرة تنبع من الانتشار الخلفي عبر تسلسلات طويلة. قبل تقديم عدد كبير من بنى RNN الحديثة ، دعنا نلقي نظرة فاحصة على كيفية عمل الانتشار الخلفي backpropagation في نماذج التسلسل بتفاصيل رياضية. نأمل أن تجلب هذه المناقشة بعض الدقة لمفهوم التلاشي وانفجار التدرجات. إذا كنت تتذكر مناقشتنا حول الانتشار الأمامي والخلفي من خلال الرسوم البيانية الحسابية عندما قدمنا MLPs في القسم 5.3 ، فيجب أن يكون الانتشار الأمامي في RNNs مباشرًا نسبيًا. يسمى تطبيق الانتشار الخلفي في RNNsالانتشار الخلفي عبر الزمن backpropagation through time (Werbos، 1990). يتطلب هذا الإجراء منا توسيع expand (أو unroll) الرسم البياني الحسابي لخطوة واحدة في وقت واحد في كل مرة. إن RNN الموسعة هو أساسًا شبكة عصبية تلقائية ذات خاصية خاصة تتكرر فيها نفس المعلمات في جميع أنحاء الشبكة غير المنضبطة ، وتظهر في كل خطوة زمنية. بعد ذلك ، تمامًا كما هو الحال في أي شبكة عصبية امامية التغذية ، يمكننا تطبيق قاعدة السلسلة ، الانتشار الخلفي للتدرجات عبر unrolled net. يجب جمع التدرج gradient فيما يتعلق بكل معلمة عبر جميع الأماكن التي تحدث فيها المعلمة في unrolled net. يجب أن يكون التعامل مع مثل هذا الربط بالوزن مألوفًا في فصولنا حول الشبكات العصبية التلافيفية.

تنشأ المضاعفات لأن التسلسلات يمكن أن تكون طويلة نوعًا ما. ليس من غير المعتاد العمل مع تسلسلات نصية تتكون من أكثر من ألف رمز. لاحظ أن هذا يطرح مشاكل من وجهة نظر حسابية (الكثير من الذاكرة) والتحسين (عدم الاستقرار العددي). المدخلات من الخطوة الأولى تمر عبر أكثر من 1000 ضرب مصفوفة قبل الوصول إلى المخرجات ، وهناك حاجة إلى 1000 ضرب مصفوفة أخرى لحساب التدرج. نقوم الآن بتحليل الخطأ الذي يمكن أن يحدث وكيفية معالجته في الممارسة العملية.

9.7.1. تحليل التدرجات في RNNs

نبدأ بنموذج مبسط لكيفية عمل ال RNN. يتجاهل هذا النموذج تفاصيل حول تفاصيل الحالة المخفية وكيفية تحديثها. لا يميز الترميز الرياضي هنا بشكل صريح بين القيم القياسية scalars والمتجهات vectors والمصفوفات matrices. نحن نحاول فقط تطوير بعض الحدس. في هذا النموذج المبسط ، نشير إلى الحالة المخفية h_t وx_tكمدخلات وo_tكمخرج في الخطوة الزمنية t. تذكر مناقشاتنا في القسم 9.4.2 أن الإدخال والحالة المخفية يمكن تسلسلهما قبل ضربهما بمتغير وزن واحد في الطبقة المخفية. وبالتالي ، فإننا نستخدم w_h وw_o لنؤشر أوزان الطبقة المخفية وطبقة المخرجات ، على التوالي. نتيجة لذلك ، فإن الحالات المخفية والمخرجات في كل مرة من الخطوات هي

$$\begin{aligned} h_t &= f(x_t, h_{t-1}, w_h), \\ o_t &= g(h_t, w_o), \end{aligned} \tag{9.7.1}$$

حيث f وg هي تحويلات الطبقة المخفية وطبقة الإخراج ، على التوالي. ومن ثم ، لدينا سلسلة من القيم $\{\dots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \dots\}$ التي تعتمد على بعضها البعض من خلال الحساب المتكرر recurrent computation. الانتشار الأمامي forward propagation واضح إلى حد ما. كل ما نحتاجه هو تكرار الخطوات الثلاثية (x_t, h_t, o_t) مرة واحدة في كل مرة. يتم بعد ذلك تقييم التناقض بين المخرجات o_t والهدف المطلوب y_t بواسطة دالة هدف T عبر جميع الخطوات الزمنية

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t).$$

بالنسبة إلى backpropagation ، تكون الأمور أكثر تعقيداً بعض الشيء ، خاصةً عندما نحسب التدرجات فيما يتعلق بمعلمات w_h الدالة الموضوعية L. على وجه التحديد ، من خلال قاعدة السلسلة،

$$\begin{aligned}\frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial g(h_t, w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h}.\end{aligned} \tag{9.7.3}$$

يسهل حساب العامل الأول والثاني للضرب في (9.7.3). العامل الثالث $\partial h_t/\partial w_h$ هو المكان الذي تصبح فيه الأشياء خادعة ، لأننا نحتاج إلى حساب تأثير المعلمة w_h على h_t بشكل متكرر. وفقًا للحساب المتكرر في (9.7.1)، h_t يعتمد على كلا h_{t-1} و w_h ، وحيث يعتمد حساب h_{t-1} على w_h أيضًا. وبالتالي ، فإن تقييم إجمالي المشتق لـ h_t فيما يتعلق بـ w_h باستخدام عوائد قاعدة السلسلة

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}. \tag{9.7.4}$$

لاشتقاق التدرج أعلاه ، افترض أن لدينا ثلاثة متواليات $\{a_t\}, \{b_t\}, \{c_t\}$ تحقق $a_0 = 0$ و$a_t = b_t + c_t a_{t-1}$ من أجل $t = 1, 2, \dots$. ثم لاجل $t \geq 1$ من السهل أن تظهر

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j\right) b_i. \tag{9.7.5}$$

بتعويض a_t، b_t وc_t وفقًا لـ

$$\begin{aligned} a_t &= \frac{\partial h_t}{\partial w_h}, \\ b_t &= \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \\ c_t &= \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}, \end{aligned} \quad (9.7.6)$$

حساب التدرج في (9.7.4) يحقق $a_t = b_t + c_t a_{t-1}$ وبالتالي ، في (9.7.5) ، يمكننا إزالة الحساب المتكرر في (9.7.4) مع

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}. \quad (9.7.7)$$

بينما يمكننا استخدام قاعدة السلسلة لحساب $\partial h_t / \partial w_h$ بشكل تكراري recursively، يمكن أن تصبح هذه السلسلة طويلة جدًا عندما t تكون كبيرة. دعونا نناقش عددًا من الاستراتيجيات للتعامل مع هذه المشكلة.

9.7.1.1. الحساب الكامل Full Computation

قد تكون إحدى الأفكار هي حساب المبلغ الكامل full sum في (9.7.7). ومع ذلك ، فإن هذا بطيء جدًا ويمكن أن تتفجر التدرجات ، نظرًا لأن التغييرات الطفيفة في الظروف الأولية يمكن أن تؤثر على النتيجة كثيرًا. بمعنى أنه يمكننا رؤية أشياء مشابهة لتأثير الفراشة butterfly effect، حيث تؤدي التغييرات الطفيفة في الظروف الأولية إلى تغييرات غير متناسبة في النتيجة. هذا بشكل عام غير مرغوب فيه. بعد كل شيء ، نحن نبحث عن مقدرات قوية تعمم جيدًا. ومن ثم فإن هذه الإستراتيجية لا تُستخدم أبدًا في الممارسة العملية.

9.7.1.2. اقتطاع خطوات الوقت Truncating Time Steps

بدلاً من ذلك ، يمكننا اقتطاع المجموع truncate the sum في (9.7.7) بعد الخطوات. هذا ما كنا نناقشه حتى الآن. هذا يؤدي إلى تقريب التدرج الحقيقي ، ببساطة عن طريق إنهاء المجموع عند $\partial h_{t-\tau} / \partial w_h$. في الممارسة العملية ، هذا يعمل بشكل جيد. هذا هو ما يشار إليه عادة باسم الانتشار الخلفي المقتطع عبر الزمن truncated backpropgation through time (Jaeger، 2002). ومن عواقب ذلك أن النموذج يركز في المقام الأول على التأثير قصير المدى بدلاً من العواقب طويلة المدى. هذا أمر مرغوب فيه بالفعل ، لأنه يوجه التقدير نحو نماذج أبسط وأكثر استقرارًا.

9.7.1.3. الاقتطاع العشوائي Randomized Truncation

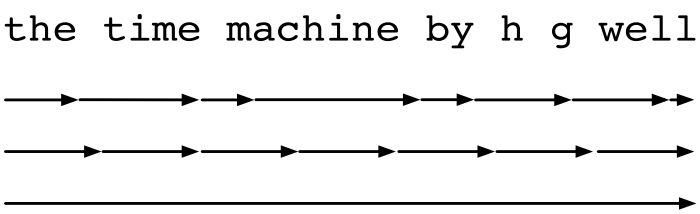
أخيرًا ، يمكننا الاستعاضة $\partial h_t / \partial w_h$ بمتغير عشوائي يكون صحيحًا في التوقع ولكنه يقطع التسلسل. يتم تحقيق ذلك باستخدام تسلسل ξ_t مع معرف مسبقًا $0 \leq \pi_t \leq 1$ ، حيث

$P(\xi_t = 0) = 1 - \pi_t$ و $P(\xi_t = \pi_t^{-1}) = \pi_t$ لذلك $E[\xi_t] = 1$. نستخدم هذا لاستبدال التدرج $\partial h_t/\partial w_h$ في (9.7.4) بـ

$$z_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \xi_t \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}.$$

يتبع من تعريف ξ_t ذلك $E[z_t] = \partial h_t/\partial w_h$. حيث ينتهي الحساب المتكرر $\xi_t = 0$ في تلك الخطوة الزمنية t. يؤدي هذا إلى مجموع اوزان للتسلسلات ذات أطوال متفاوتة ، حيث تكون التسلسلات الطويلة نادرة ولكنها ذات وزن زائد overweighted بشكل مناسب. تم اقتراح هذه الفكرة من قبل Tallec و Ollivier (2017).

9.7.1.4. مقارنة الاستراتيجيات Comparing Strategies



الشكل 9.7.1 مقارنة استراتيجيات حساب التدرجات في RNNs. من أعلى إلى أسفل: الاقتطاع العشوائي randomized truncation والاقتطاع المنتظم regular truncation والحساب الكامل full computation.

يوضح الشكل 9.7.1 الاستراتيجيات الثلاث عند تحليل الأحرف القليلة الأولى من The Time Machine باستخدام الانتشار الخلفي عبر الزمن لـ RNNs:

- الصف الأول هو الاقتطاع العشوائي randomized truncation الذي يقسم النص إلى مقاطع ذات أطوال مختلفة.
- الصف الثاني هو الاقتطاع المنتظم regular truncation الذي يقسم النص إلى تتابعات من نفس الطول. هذا ما كنا نفعله في تجارب RNN.
- الصف الثالث هو الانتشار الخلفي الكامل عبر الزمن الذي يؤدي إلى تعبير غير عملي حسابيًا.

لسوء الحظ، على الرغم من جاذبيته من الناحية النظرية، فإن الاقتطاع العشوائي لا يعمل بشكل أفضل بكثير من الاقتطاع المنتظم، ويرجع ذلك على الأرجح إلى عدد من العوامل. أولاً، تأثير المشاهدة بعد عدد من خطوات الانتشار الخلفيفي الماضي كافٍ تمامًا لالتقاط التبعياتفي الممارسة العملية. ثانيًا، يتعارض التباين المتزايد مع حقيقة أن التدرج يكون أكثر دقة بمزيد من

الخطوات. ثالثًا، نريدفي الواقع نماذج لها نطاق قصير من التفاعلات. وبالتالي، فإن الانتشار الخلفي المقتطع بانتظام عبر الزمن له تأثير تنظيم طفيف يمكن أن يكون مرغوبًا فيه.

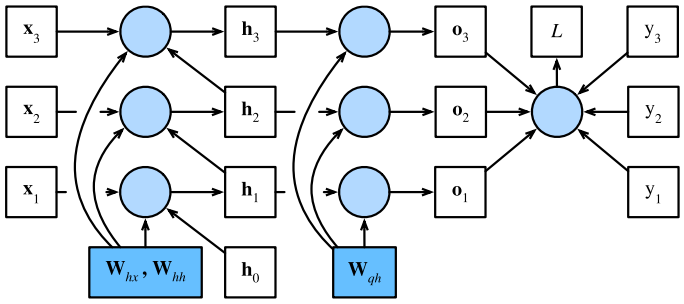
9.7.2. الانتشار الخلفي عبر الزمن بالتفصيل Backpropagation Through Time in Detail

بعد مناقشة المبدأ العام، دعونا نناقش الانتشار الخلفي عبر الزمن بالتفصيل. بخلاف التحليل الواردفي القسم 9.7.1، سنعرض فيما يلي كيفية حساب تدرجات الدالة الموضوعية فيما يتعلق بجميع معلمات النموذج المتحللة. لتبسيط الأمور، فإننا نعتبر RNN بدون معلمات التحيز، والتي تستخدم دالة التنشيط الخاصة بهافي الطبقة المخفية تعيين الهوية identity mapping $(\phi(x) = x)$. بالنسبة للخطوة الزمنية t، دع إدخال المثال الفردي والهدف يكونان $\mathbf{x}_t \in \mathbb{R}^d$، y_tعلى التوالي. يتم حساب الحالة المخفية $\mathbf{h}_t \in \mathbb{R}^h$ والإخراج $\mathbf{o}_t \in \mathbb{R}^q$ كـ

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1},\\ \mathbf{o}_t &= \mathbf{W}_{qh}\mathbf{h}_t,\end{aligned}$$

حيث $\mathbf{W}_{hx} \in \mathbb{R}^{h\times d}$، $\mathbf{W}_{hh} \in \mathbb{R}^{h\times h}$ و$\mathbf{W}_{qh} \in \mathbb{R}^{q\times h}$ هي معلمات الوزن. $l(\mathbf{o}_t, y_t)$ تدل على الخطأفي الخطوة الزمنية t. دالتنا الموضوعية هي الخطأ بمرور الوقت T من بداية التسلسل لذلك:

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{o}_t, y_t).$$



الشكل 9.7.2 رسم بياني حسابي يوضح التبعيات لنموذج RNN بثلاث خطوات زمنية. تمثل المربعات متغيرات (غير مظللة) أو معلمات (مظللة) وتمثل الدوائر عوامل التشغيل.

من أجل تصور التبعيات بين متغيرات النموذج والمعلمات أثناء حساب RNN، يمكننا رسم مخطط بياني حسابي للنموذج، كما هو موضحفي الشكل 9.7.2. على سبيل المثال، يعتمد حساب

الحالات المخفية للخطوة الزمنية 3، $\mathbf{h}_3$ على معلمات النموذج $\mathbf{W}_{hx}$ و$\mathbf{W}_{hh}$ الحالة المخفية لخطوة الوقت الأخيرة $\mathbf{h}_2$ وإدخال الخطوة الزمنية الحالية $\mathbf{x}_3$.

كما ذكر للتو ، فإن معلمات النموذج في الشكل 9.7.2 هي $\mathbf{W}_{hx}$ و$\mathbf{W}_{hh}$ و $\mathbf{W}_{qh}$. بشكل عام ، يتطلب تدريب هذا النموذج حساب التدرج فيما يتعلق بهذه المعلمات، $\partial L/\partial \mathbf{W}_{hx}$ و$\partial L/\partial \mathbf{W}_{hh}$ و $\partial L/\partial \mathbf{W}_{qh}$. وفقًا للتبعيات الواردة في الشكل 9.7.2 ، يمكننا اجتياز الاتجاه المعاكس للأسهم لحساب وتخزين التدرجات بدورها. للتعبير بمرونة عن ضرب المصفوفات والمتجهات والكميات ذات الأشكال المختلفة في قاعدة السلسلة ، نستمر في استخدام العامل prod كما هو موضح في القسم 5.3.

بادئ ذي بدء ، يعد التفريق بين الدالة الموضوعية فيما يتعلق بإخراج النموذج في أي خطوة زمنية t أمرًا بسيطًا إلى حد ما:

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial l(\mathbf{o}_t, y_t)}{T \cdot \partial \mathbf{o}_t} \in \mathbb{R}^q.$$

الآن، يمكننا حساب انحدار الهدف فيما يتعلق بالمعامل $\mathbf{W}_{qh}$ في طبقة المخرجات: $\partial L/\partial \mathbf{W}_{qh} \in \mathbb{R}^{q \times h}$. بناءً على الشكل 9.7.2 ، يعتمد الهدف L على $\mathbf{W}_{qh}$ بواسطة $\mathbf{o}_1, \ldots, \mathbf{o}_T$. باستخدام قاعدة السلسلة ينتج

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{qh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t} \mathbf{h}_t^\top,$$

حيث $\partial L/\partial \mathbf{o}_t$ تعطى بواسطة (9.7.11).

بعد ذلك ، كما هو موضح في الشكل 9.7.2 ، في الخطوة الزمنية النهائية T، تعتمد دالة الهدف على الحالة المخفية $\mathbf{h}_T$ فقط عبر $\mathbf{o}_T$. لذلك ، يمكننا بسهولة إيجاد التدرج $\partial L/\partial \mathbf{h}_T \in \mathbb{R}^h$ باستخدام قاعدة السلسلة:

$$\frac{\partial L}{\partial \mathbf{h}_T} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_T}, \frac{\partial \mathbf{o}_T}{\partial \mathbf{h}_T}\right) = \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_T}.$$

يصبح الأمر أكثر تعقيدًا في أي خطوة زمنية $t < T$، حيث تعتمد الدالة الموضوعية L على $\mathbf{h}_t$ عبر $\mathbf{h}_{t+1}$ و$\mathbf{o}_t$. وفقًا لقاعدة السلسلة ، يمكن حساب التدرج للحالة المخفية $\partial L/\partial \mathbf{h}_t \in \mathbb{R}^h$ في أي خطوة زمنية $t < T$ بشكل متكرر على النحو التالي:

$$\frac{\partial L}{\partial \mathbf{h}_t} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_{t+1}}, \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right) + \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right) = \mathbf{W}_{hh}^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_t}.$$

للتحليل ، يعطي توسيع الحساب المتكرر لأي خطوة زمنية $1 \leq t \leq T$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} (\mathbf{W}_{hh}^\top)^{T-i} \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}.$$

يمكننا أن نرى من (9.7.15) أن هذا المثال الخطي البسيط يعرض بالفعل بعض المشكلات الرئيسية لنماذج التسلسل الطويل: إنه ينطوي على إمكانات كبيرة جدًا من $\mathbf{W}_{hh}^\top$. في ذلك ، تختفي القيم الذاتية eigenvalues الأصغر من 1 وتتباعد القيم الذاتية الأكبر من 1. هذا غير مستقر عدديًا ، والذي يتجلى في شكل تدرجات متلاشية ومتفجرة. تتمثل إحدى طرق معالجة ذلك في اقتطاع الخطوات الزمنية بحجم مناسب حسابيًا كما تمت مناقشته في القسم 9.7.1. من الناحية العملية ، يمكن أيضًا إجراء هذا الاقتطاع عن طريق فصل التدرج بعد عدد معين من الخطوات الزمنية. لاحقًا ، سنرى كيف يمكن لنماذج التسلسل الأكثر تعقيدًا مثل الذاكرة طويلة قصيرة المدى long short-term memory أن تخفف من حدة هذا الأمر أكثر.

أخيرًا ، يوضح الشكل 9.7.2 أن دالة الهدف L تعتمد على معلمات النموذج $\mathbf{W}_{hx}$ و $\mathbf{W}_{hh}$ في الطبقة المخفية عبر الحالات المخفية $\mathbf{h}_1, \ldots, \mathbf{h}_T$. لحساب التدرجات فيما يتعلق بهذه المعلمات $\partial L/\partial \mathbf{W}_{hx} \in \mathbb{R}^{h \times h}$ و $\partial L/\partial \mathbf{W}_{hh} \in \mathbb{R}^{h \times d}$ نطبق قاعدة السلسلة التي تعطي

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}_{hx}} &= \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hx}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^\top, \\
\frac{\partial L}{\partial \mathbf{W}_{hh}} &= \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^\top,
\end{aligned}$$

حيث $\partial L/\partial \mathbf{h}_t$ يتم حسابها بشكل متكرر بواسطة (9.7.13) و (9.7.14) هي الكمية الرئيسية التي تؤثر على الاستقرار العددي.

نظرًا لأن الانتشار الخلفي عبر الزمن هو تطبيق الانتشار الخلفي في RNNs ، كما أوضحنا في القسم 5.3 ، فإن تدريب RNNs يبدل الانتشار الأمامي مع الانتشار الخلفي عبر الزمن. إلى جانب ذلك ، يحسب الانتشار الخلفي عبر الزمن ويخزن التدرجات المذكورة أعلاه بدوره. على وجه التحديد ، يتم إعادة استخدام القيم الوسيطة المخزنة لتجنب تكرار العمليات الحسابية ، مثل تخزين $\partial L/\partial \mathbf{h}_t$ لاستخدامها في حساب كل من $\partial L/\partial \mathbf{W}_{hx}$ و $\partial L/\partial \mathbf{W}_{hh}$.

9.7.3. الملخص

الانتشار الخلفي عبر الزمن هو مجرد تطبيق الانتشار الخلفي لتسلسل النماذج ذات الحالة المخفية. هناك حاجة إلى الاقتطاع Truncation من أجل الراحة الحسابية والاستقرار العددي،

مثل الاقتطاع المنتظم regular truncation والاقتطاع العشوائي randomized truncation. يمكن أن تؤدي القوى العالية للمصفوفات إلى قيم ذاتية متباعدة أو متلاشية. يتجلى هذا في شكل تدرجات متفجرة أو متلاشية. للحساب الفعال ، يتم تخزين القيم الوسيطة مؤقتًا أثناء الانتشار الخلفي عبر الوقت.

9.7.4. التمارين

1. افترض أن لدينا مصفوفة متماثلة $\mathbf{M} \in \mathbb{R}^{n \times n}$ مع قيم ذاتية (eigenvalues) λ_i المتجهات الذاتية المقابلة لها هي $\mathbf{v}_i (i = 1, \ldots, n)$. دون فقدان العمومية ، افترض أنها مرتبة بالترتيب $|\lambda_i| \geq |\lambda_{i+1}|$.
2. أظهر أن $\mathbf{M}^k$ لديه قيم ذاتية eigenvalues λ_i^k
3. إثبت أنه بالنسبة إلى المتجه العشوائي $\mathbf{x} \in \mathbb{R}^n$ ، مع وجود احتمال $\mathbf{M}^k\mathbf{x}$ كبير ، سيتم محاذاة إلى حد كبير مع المتجه الذاتي (eigenvector) $\mathbf{v}_1$ لـ $\mathbf{M}$. إضف طابع رسيماً على هذا البيان.
4. ماذا تعني النتيجة أعلاه للتدرجات في RNNs؟
5. إلى جانب قص التدرج gradient clipping ، هل يمكنك التفكير في أي طرق أخرى للتعامل مع انفجار التدرج في الشبكات العصبية المتكررة RNN؟

10

الشبكات العصبية المتكررة الحديثة

10. الشبكات العصبية المتكررة الحديثة Modern Recurrent Neural Networks

قدم الفصل السابق الأفكار الرئيسية وراء الشبكات العصبية المتكررة (RNNs). ومع ذلك، تمامًا كما هو الحال مع الشبكات العصبية التلافيفية CNN، كان هناك قدر هائل من الابتكارفي معماريات RNN، وبلغت ذروتهافي العديد من التصميمات المعقدة التي أثبتت نجاحهافي الممارسة. على وجه الخصوص، تتميز التصميمات الأكثر شيوعًا بآليات للتخفيف من عدم الاستقرار العددي numerical instability السيئ السمعة الذي تواجهه RNNs، كما يتجلى في اختفاء التدرجات وانفجارها. تذكر أننافي القسم 9 تعاملنا مع انفجار التدرج من خلال تطبيق استدلال قص متدرج حاد gradient clipping heuristic. على الرغم من فعالية هذا الاختراق، فإنه يترك مشكلة اختفاء التدرجات مفتوحة.

في هذا الفصل، نقدم الأفكار الرئيسية وراء أنجح بُنى RNN للتسلسل، والتي تنبع من مقالتين تم نشرهمافي عام 1997. المقالة الأولى، الذاكرة طويلة قصيرة المدى LSTM (Hochreiter and Schmidhuber، 1997) ، تقدم خلية الذاكرة ، وهي وحدة الحساب التي تحل محل العقد التقليديةفي الطبقة المخفية للشبكة. باستخدام خلايا الذاكرة هذه، تكون الشبكات قادرة على التغلب على صعوبات التدريب التي واجهتها الشبكات المتكررة السابقة. حدسيًا، تتجنب خلية الذاكرة مشكلة التدرج المتلاشي عن طريق الاحتفاظ بالقيمفي الحالة الداخلية لكل خلية ذاكرة متتالية على طول حافة متكررة بوزن 1 عبر العديد من الخطوات الزمنية المتتالية. تساعد مجموعة من البوابات المضاعفة الشبكة على تحديد كل من المدخلات التي يجب السماح بدخولهافي حالة الذاكرة، ومتى يجب أن يؤثر محتوى حالة الذاكرة على إخراج النموذج.

المقالة الثانية، الشبكات العصبية المتكررة ثنائية الاتجاه Bidirectional Recurrent Neural Networks (Schuster and Paliwal، 1997) ، تقدم بُنية يتم فيها استخدام المعلومات من كل من المستقبل (الخطوات الزمنية اللاحقة) والماضي (الخطوات الزمنية السابقة) لتحديد المخرجاتفي أي نقطةفي تسلسل. هذا على عكس الشبكات السابقة، حيث يمكن أن تؤثر المدخلات السابقة فقط على المخرجات. أصبحت RNNs ثنائية الاتجاه الدعامة الأساسية لمهام وضع العلامات التسلسليةفي معالجة اللغة الطبيعية، من بين مهام أخرى لا تعد ولا تحصى. لحسن الحظ، لا يتعارض الابتكاران مع بعضهما البعض، وقد تم دمجهما بنجاح لتصنيف الصوتيات phoneme classification (Graves and Schmidhuber، 2005) والتعرف على خط اليد handwriting recognition (Graves et al.، 2008).

ستشرح الأقسام الأولىفي هذا الفصل بُنية LSTM، وهي نسخة أخف وزنًا تسمى الوحدة المتكررة ذات البوابات gated recurrent unit (GRU)، والأفكار الرئيسية وراء شبكات RNN ثنائية الاتجاه وشرح موجز لكيفية تكديس طبقات RNN معًا لتكوين شبكات RNN عميقة. بعد ذلك، سوف نستكشف تطبيق RNNsفي مهام التسلسل إلى التسلسل، وإدخال الترجمة الآلية جنبًا إلى جنب مع الأفكار الرئيسية مثل معماريات المشفر ـ مفكك الشفرة encoder-decoder architectures والبحث الشعاعي beam search.

10.1. الذاكرة طويلة قصيرة المدى Long Short-Term (LSTM) Memory

بعد فترة وجيزة من تدريب أول RNNs على غرار Elman باستخدام الانتشار الخلفي، (1990,Elman)، أصبحت مشاكل تعلم التبعيات طويلة المدى (بسبب التلاشي والانفجار التدرجات) بارزة، حيث ناقش Bengio و Hochreiter المشكلة، (1994,Bengio et al.)، (2001,Hochreiter et al.). كان Hochreiter قد أوضح هذه المشكلةفي وقت مبكر من أطروحة الماجستير عام 1991، على الرغم من أن النتائج لم تكن معروفة على نطاق واسع لأن الأطروحة كتبت باللغة الألمانية. بينما يساعد قص التدرجفي انفجار التدرجات، يبدو أن التعامل مع التدرجات المتلاشية يتطلب حلاً أكثر تفصيلاً. جاءت واحدة من أولى التقنيات وأكثرها نجاحًا لمعالجة التدرجات المتلاشيةفي شكل نموذج الذاكرة طويلة قصيرة المدى (LSTM) بسبب Hochreiter and Schmidhuber (1997). تشبه LSTM الشبكات العصبية المتكررة القياسية ولكن هنا يتم استبدال كل عقدة متكررة عادية بخلية ذاكرة memory cell. تحتوي كل خلية ذاكرة على حالة داخلية internal state، أي عقدة ذات حافة متكررة متصلة ذاتيًا بوزن ثابت 1، مما يضمن أن التدرج يمكن أن يمر عبر العديد من الخطوات الزمنية دون أن يتلاشى أو ينفجر.

يأتي مصطلح "الذاكرة طويلة قصيرة المدى" من الحدس التالي. تمتلك الشبكات العصبية المتكررة البسيطة ذاكرة طويلة المدى على شكل أوزان. تتغير الأوزان ببطء أثناء التدريب، مما يؤدي إلى ترميز المعرفة العامة حول البيانات. لديهم أيضًا ذاكرة قصيرة المدىفي شكل عمليات تنشيط سريعة الزوال، والتي تنتقل من كل عقدة إلى عقد متتالية. يقدم نموذج LSTM نوعًا وسيطًا للتخزين عبر خلية الذاكرة. خلية الذاكرة هي وحدة مركبة، مبنية من عقد أبسطفي نمط اتصال محدد، مع تضمين جديد للعقد المضاعفة.

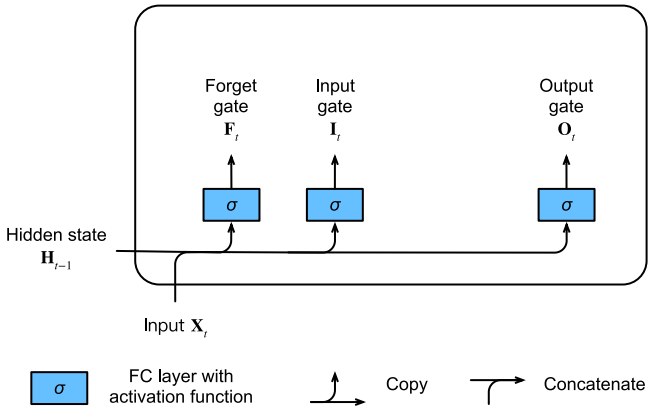
10.1.1. خلية ذاكرة ذات البوابات Gated Memory Cell

تم تجهيز كل خلية ذاكرة بحالة داخلية internal state وعدد من البوابات المضاعفة multiplicative gates التي تحدد ما إذا كان (1) إدخال معين يجب أن يؤثر على الحالة

الداخلية (بوابة الإدخال)، (2) يجب مسح الحالة الداخلية إلى (بوابة النسيان forget gate)، و (3) يجب السماح للحالة الداخلية للخلايا العصبية بالتأثير على ناتج الخلية (بوابة الإخراج output gate).

10.1.1.1. الحالة الخفية ذات البوابات Gated Hidden State

الفرق الرئيسي بين vanilla RNNs وLSTMs هو أن الأخير يدعم الحالة المخفية ذات البوابات Gated Hidden State. هذا يعني أن لدينا آليات مخصصة لتحديد متى يجب تحديث حالة مخفية وأيضًا متى يجب إعادة تعيينها reset. يتم تعلم هذه الآليات وهي تتناول المخاوف المذكورة أعلاه. على سبيل المثال، إذا كان الرمز token الأول ذا أهمية كبيرة، فسوف نتعلم عدم تحديث الحالة المخفية بعد الملاحظة الأولى. وبالمثل، سوف نتعلم تخطي الملاحظات المؤقتة غير ذات الصلة. أخيرًا، سوف نتعلم إعادة ضبط الحالة الكامنة عند الحاجة. نناقش هذا بالتفصيل أدناه.



الشكل 10.1.1 حساب بوابة الإدخال وبوابة النسيان وبوابة الإخراج في نموذج LSTM.

10.1.1.2. بوابة الإدخال، نسيت البوابة، وبوابة الإخراج Input Gate, Forget Gate, and Output Gate

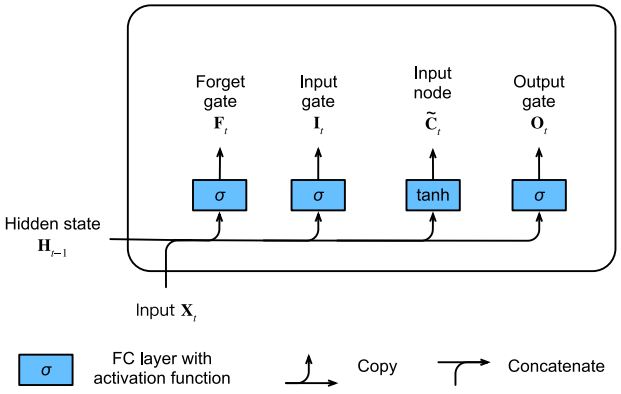
تغذية البيانات data feeding في بوابات LSTM هي المدخلات في الخطوة الزمنية الحالية والحالة المخفية للخطوة الزمنية السابقة، كما هو موضح في الشكل 10.1.1. ثلاث طبقات متصلة بالكامل مع دوال التنشيط sigmoid تحسب قيم بوابات الإدخال والنسيان والإخراج. نتيجة للتنشيط sigmoid، تقع جميع قيم البوابات الثلاثة في نطاق. بالإضافة إلى ذلك، نحتاج إلى عقدة إدخال input node، يتم حسابها عادةً بدالة تنشيط tanh. حدسيًا، تحدد بوابة الإدخال input gate مقدار قيمة عقدة الإدخال التي يجب إضافتها إلى الحالة الداخلية لخلية الذاكرة الحالية. تحدد بوابة النسيان forget gate ما إذا كنت تريد الاحتفاظ بالقيمة الحالية للذاكرة أو مسحها.

وتحدد بوابة الإخراج output gate ما إذا كان يجب أن تؤثر خلية الذاكرة على الإخراج في الخطوة الزمنية الحالية.

رياضيًا، افترض أن هناك وحدات مخفية h ، وحجم الدُفعة n ، وعدد المدخلات d. وبالتالي، يكون الإدخال $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ والحالة المخفية للخطوة الزمنية السابقة هي $\mathbf{H}_{t-1} \in \mathbb{R}^{n\times h}$. في المقابل، يتم تعريف البوابات في الخطوة الزمنية t على النحو التالي: بوابة الإدخال هي $\mathbf{I}_t \in \mathbb{R}^{n\times h}$ ، وبوابة النسيان $\mathbf{F}_t \in \mathbb{R}^{n\times h}$ ، وبوابة الإخراج هي $\mathbf{O}_t \in \mathbb{R}^{n\times h}$. يتم حسابها على النحو التالي:

$$\begin{aligned}\mathbf{I}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i),\\ \mathbf{F}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f),\\ \mathbf{O}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o),\end{aligned}$$

حيث $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{d\times h}$ و $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h\times h}$ هي معلمات الوزن و $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1\times h}$ هي معلمات التحيز. لاحظ أن البث broadcasting (انظر القسم 2.1.4) يتم تشغيله أثناء عملية الجمع. نحن نستخدم دوال sigmoid (كما هو مقدم في القسم 5.1) لتعيين قيم الإدخال إلى الفاصل الزمني $(0,1)$.



الشكل 10.1.2 حساب عقدة الإدخال في نموذج LSTM.

10.1.1.3. عقدة الإدخال Input Node

بعد ذلك نقوم بتصميم خلية الذاكرة memory cell. نظرًا لأننا لم نحدد عمل البوابات المختلفة بعد، فإننا نقدم أولاً عقدة الإدخال (Input Node) $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n\times h}$. يتشابه حسابها مع حساب البوابات الثلاثة الموضحة أعلاه، ولكن باستخدام دالة ذات قيم نطاق $(-1,1)$ لدالة التنشيط. هذا يؤدي إلى المعادلة التالية في الخطوة الزمنية t

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c),$$

حيث $\mathbf{W}_{xc} \in \mathbb{R}^{d\times h}$ و $\mathbf{W}_{hc} \in \mathbb{R}^{h\times h}$ هي معلمات الوزن و $\mathbf{b}_c \in \mathbb{R}^{1\times h}$ هي معلمة تحيز.

يظهر توضيح سريع لعقدة الإدخالفي الشكل 10.1.2.

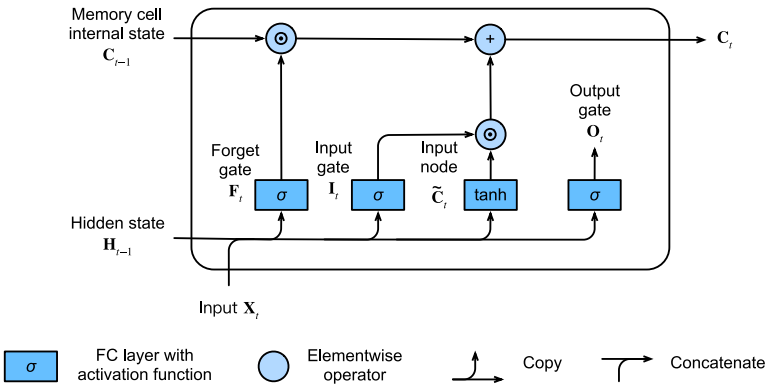
10.1.1.4. الحالة الداخلية لخلية الذاكرة Memory Cell Internal State

في LSTMs، تتحكم بوابة الإدخال $\mathbf{I}_t$في مقدار البيانات الجديدة التي نأخذهافي الاعتبار عن طريق $\tilde{\mathbf{C}}_t$وتعالج بوابة النسيان $\mathbf{F}_t$ مقدار الحالة الداخلية $\mathbf{C}_{t-1} \in \mathbb{R}^{n\times h}$ للخلية القديمة التي نحتفظ بها. باستخدام عامل ضرب هادامار ⊙ Hadamard (elementwise) ،نصل إلى معادلة التحديث التالية:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$

إذا كانت بوابة النسيان دائمًا 1 وكانت بوابة الإدخال دائمًا 0، فستظل الحالة الداخلية $\mathbf{C}_{t-1}$ لخلية الذاكرة ثابتة إلى الأبد، وستمر دون تغييرفي كل خطوة زمنية لاحقة. ومع ذلك، فإن بوابات الإدخال ونسيان البوابات تمنح النموذج المرونة لتعلم متى يجب الحفاظ على هذه القيمة دون تغيير ومتى تشوشها استجابةً للمدخلات اللاحقة.في الممارسة العملية، يخفف هذا التصميم من مشكلة التدرج المتلاشي، مما ينتج عنه نماذج يسهل تدريبها كثيرًا، خاصةً عند مواجهة مجموعات البيانات ذات أطوال التسلسل الطويل.

وهكذا نصل إلى مخطط التدفق flow diagramفي الشكل 10.1.3.



الشكل 10.1.3 حساب الحالة الداخلية لخلية الذاكرةفي نموذج LSTM.

10.1.1.5. الحالة المخفية Hidden State

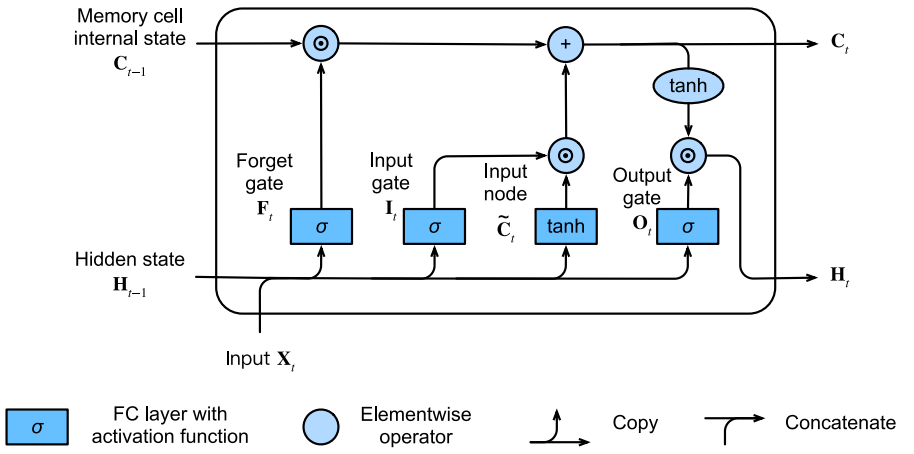
أخيرًا، نحتاج إلى تحديد كيفية حساب إخراج خلية الذاكرة، أي الحالة المخفية (hidden state) $\mathbf{H}_t \in \mathbb{R}^{n\times h}$، كما تراها الطبقات الأخرى. هذا هو المكان الذي يتم فيه تشغيل بوابة

الإخراج. في LSTMs، نطبق أولاً على الحالة الداخلية لخلية الذاكرة ثم نطبق ضرب نقطي أخر، هذه المرة باستخدام بوابة الإخراج. هذا يضمن أن تكون قيم $\mathbf{H}_t$ دائمًافي الفاصل الزمني $(-1,1)$:

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t).$$

عندما تكون بوابة الإخراج قريبة من 1، فإننا نسمح للحالة الداخلية لخلية الذاكرة بالتأثير على الطبقات اللاحقة غير المحظورة، بينما بالنسبة لقيم بوابة الإخراج القريبة من 0، فإننا نمنع الذاكرة الحالية من التأثير على الطبقات الأخرى للشبكةفي الخطوة الزمنية الحالية. لاحظ أن خلية الذاكرة يمكنها تجميع المعلومات عبر العديد من الخطوات الزمنية دون التأثير على بقية الشبكة (طالما أن بوابة الإخراج تأخذ قيمًا قريبة من 0)، ثم تؤثر فجأة على الشبكةفي خطوة زمنية لاحقة بمجرد بوابة الإخراج تقلب من قيم قريبة من 0 إلى قيم قريبة من 1.

يحتوي الشكل 10.1.4 على رسم توضيحي لتدفق البيانات.



الشكل 10.1.4 حساب الحالة المخفيةفي نموذج LSTM.

10.1.2. التنفيذ من البداية Implementation from Scratch

الآن دعونا ننفذ LSTM من البداية. مثل التجاربفي القسم 9.5، نقوم أولاً بتحميل مجموعة بيانات The Time Machine.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.1.2.1. تهيئة معلمات النموذج Initializing Model Parameters

بعد ذلك، نحتاج إلى تحديد وتهيئة معلمات النموذج. كمافي السابق، يحدد المعامل الفائق `num_hiddens` عدد الوحدات المخفية. نقوم بتهيئة الأوزان بعد توزيع Gaussian مع انحراف معياري 0.01، وقمنا بتعيين التحيزات على 0.

```
class LSTMScratch(d2l.Module):  #@save
    def __init__(self, num_inputs, num_hiddens,
sigma=0.01):
        super().__init__()
        self.save_hyperparameters()

        init_weight = lambda *shape:
tf.Variable(tf.random.normal(shape) * sigma)
        triple = lambda: (init_weight(num_inputs,
num_hiddens),
                          init_weight(num_hiddens,
num_hiddens),

tf.Variable(tf.zeros(num_hiddens)))

        self.W_xi, self.W_hi, self.b_i = triple()  #
Input gate
        self.W_xf, self.W_hf, self.b_f = triple()  #
Forget gate
        self.W_xo, self.W_ho, self.b_o = triple()  #
Output gate
        self.W_xc, self.W_hc, self.b_c = triple()  #
Input node
```

يتم تعريف النموذج الفعلي كما هو موضح أعلاه، ويتألف من ثلاث بوابات وعقدة إدخال. لاحظ أنه يتم تمرير الحالة المخفية فقط إلى طبقة الإخراج.

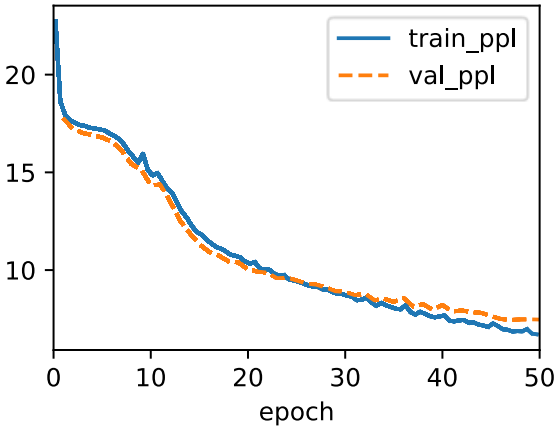
```
@d2l.add_to_class(LSTMScratch)
def forward(self, inputs, H_C=None):
    H, C = None, None if H_C is None else H_C
    outputs = []
    for X in inputs:
        I = tf.sigmoid(tf.matmul(X, self.W_xi) + (
            tf.matmul(H, self.W_hi) if H is not None
else 0) + self.b_i)
        if H is None:
            H, C = tf.zeros_like(I), tf.zeros_like(I)
        F = tf.sigmoid(tf.matmul(X, self.W_xf) +
                       tf.matmul(H, self.W_hf) +
self.b_f)
        O = tf.sigmoid(tf.matmul(X, self.W_xo) +
                       tf.matmul(H, self.W_ho) +
self.b_o)
```

```
        C_tilde = tf.tanh(tf.matmul(X, self.W_xc) +
                          tf.matmul(H, self.W_hc) +
self.b_c)
        C = F * C + I * C_tilde
        H = O * tf.tanh(C)
        outputs.append(H)
    return outputs, (H, C)
```

10.1.2.2. التدريب والتنبؤ Training and Prediction

دعنا ندرب نموذج LSTM عن طريق إنشاء مثيل لفئة `RNNLMScratch` كما هو موضح في القسم 9.5.

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
with d2l.try_gpu():
    lstm = LSTMScratch(num_inputs=len(data.vocab),
num_hiddens=32)
    model = d2l.RNNLMScratch(lstm,
vocab_size=len(data.vocab), lr=4)
trainer = d2l.Trainer(max_epochs=50,
gradient_clip_val=1)
trainer.fit(model, data)
```

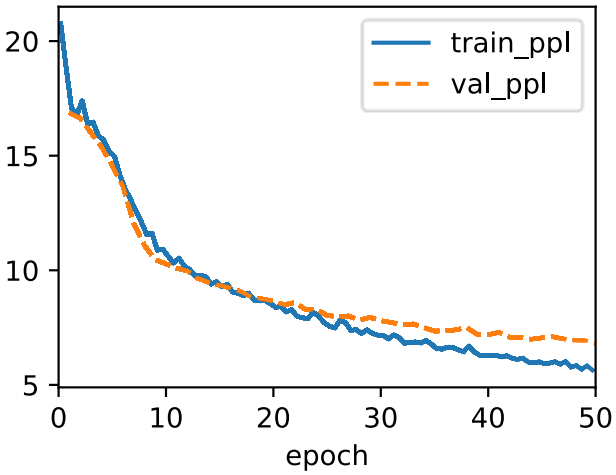


10.1.3. التنفيذ المختصر Concise Implementation

باستخدام واجهات برمجة التطبيقات عالية المستوى API، يمكننا إنشاء نموذج LSTM مباشرةً. هذا يلخص جميع تفاصيل التكوين التي أوضحناها أعلاه. الكود أسرع بشكل ملحوظ لأنه يستخدم عوامل مجمعة بدلاً من بايثون للعديد من التفاصيل التي أوضحناها من قبل.

```
class LSTM(d2l.RNN):
    def __init__(self, num_hiddens):
```

```
        d2l.Module.__init__(self)
        self.save_hyperparameters()
        self.rnn = tf.keras.layers.LSTM(
                num_hiddens, return_sequences=True,
                return_state=True, time_major=True)

    def forward(self, inputs, H_C=None):
        outputs, *H_C = self.rnn(inputs, H_C)
        return outputs, H_C

lstm = LSTM(num_hiddens=32)
with d2l.try_gpu():
    model = d2l.RNNLM(lstm, vocab_size=len(data.vocab),
lr=4)
trainer.fit(model, data)
```




```
model.predict('it has', 20, data.vocab)
```

```
'it has the the the the the'
```

LSTM هي النموذج الأولي الكامن المتغير الانحدار التلقائي مع التحكمفي الحالة غير البديهية. تم اقتراح العديد من المتغيرات على مر السنين، على سبيل المثال، طبقات متعددة multiple layers، وصلات متبقية residual connections، أنواع مختلفة من التنظيم. ومع ذلك، فإن تدريب LSTMs ونماذج التسلسل الأخرى (مثل GRUs) مكلف للغاية بسبب التبعية طويلة المدى long range dependency للتسلسل. لاحقًا سنواجه نماذج بديلة مثل المحولات transformers التي يمكن استخدامهافي بعض الحالات.

10.1.4. الملخص

بينما تم نشر LSTMsفي عام 1997، فقد برزت بشكل أكبر مع بعض الانتصاراتفي مسابقات التنبؤفي منتصف العقد الأول من القرن الحادي والعشرين، وأصبحت النماذج السائدة للتعلم المتسلسل من عام 2011 حتى وقت قريب مع ظهور نماذج المحولات transformer models، بدءًا من عام 2017. حتى مخترعي المحولات يدينون ببعض أفكارهم الرئيسية لابتكارات التصميم المعماري التي قدمتها LSTM. تحتوي LSTM على ثلاثة أنواع من البوابات: بوابات الإدخال input gates وبوابات النسيان forget gates وبوابات الإخراج output gates التي تتحكمفي تدفق المعلومات. يتضمن إخراج الطبقة المخفية لـ LSTM الحالة المخفية والحالة الداخلية لخلية الذاكرة. يتم تمرير الحالة المخفية فقط إلى طبقة الإخراج بينما تكون الحالة الداخلية لخلية الذاكرة داخلية بالكامل. يمكن أن تخفف LSTMs من التلاشي وانفجار التدرجات.

10.1.5. التمارين

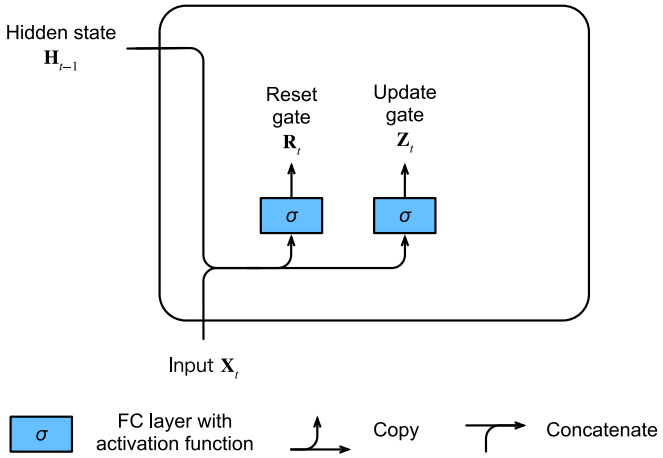
1. اضبط المعلمات الفائقة وحلل تأثيرها على وقت التشغيل والارتباك وتسلسل الإخراج.
2. كيف ستحتاج إلى تغيير النموذج لتوليد كلمات مناسبة بدلاً من تسلسل الأحرف؟
3. قارن التكلفة الحسابية لـ GRUs و LSTMs و RNNs العادية لبُعد مخفي معين. قم بإيلاء اهتمام خاص لتكلفة التدريب والاستدلال.
4. نظرًا لأن خلية الذاكرة المرشحة تضمن أن نطاق القيمة يقع بين 1− و 1 باستخدام الدالة tanh ، فلماذا تحتاج الحالة المخفية إلى استخدام الدالة مرة أخرى للتأكد من أن نطاق قيمة الإخراج يقع بين 1− و1 ؟
5. نفذ نموذج LSTM لتنبؤ السلاسل الزمنية بدلاً من توقع تسلسل الأحرف.

10.2. الوحدات المتكررة ذات البوابات (GRU) Gated Recurrent Units

نظرًا لأن RNNs وخاصة بنية LSTM (القسم 10.1) اكتسبت شعبية بسرعة خلال 2010، بدأ عدد من المقالاتفي تجربة البُنى المبسطة على أمل الاحتفاظ بالفكرة الرئيسية لدمج آليات البوابة الداخلية internal state والحالة المضاعفة multiplicative gating ولكن بهدف إسراع الحساب. قدمت الوحدة المتكررة ذات البوابات The gated recurrent unit (GRU) (Cho et al.، 2014) نسخة مبسطة من خلية ذاكرة LSTM التي تحقق غالبًا أداءً مشابهًا ولكن مع ميزة كونها أسرعفي الحساب (Chung et al.، 2014).

10.2.1. بوابة إعادة الضبط وبوابة التحديث Reset Gate and Update Gate

هنا، يتم استبدال بوابات LSTM الثلاثة بوابتين: بوابة إعادة الضبط reset gate وبوابة التحديث update gate. كما هو الحال مع LSTMs، تُعطى هذه البوابات تنشيطات sigmoid، مما يجبر قيمها على الوقوع في الفاصل الزمني(0,1). بشكل حدسي، تتحكم بوابة إعادة الضبط في مقدار الحالة السابقة التي قد لا نزال نريد تذكرها. وبالمثل، ستسمح لنا بوابة التحديث بالتحكم في مدى كون الحالة الجديدة مجرد نسخة من الحالة القديمة. يوضح الشكل 10.2.1 المدخلات لكل من بوابات إعادة الضبط والتحديث في GRU، بالنظر إلى مدخلات الخطوة الزمنية الحالية والحالة المخفية للخطوة الزمنية السابقة. يتم إعطاء مخرجات بوابتين بواسطة طبقتين متصلتين بالكامل بدالة التنشيط sigmoid.



الشكل 10.2.1 حساب بوابة إعادة الضبط وبوابة التحديث في نموذج GRU.

رياضيًا، لخطوة زمنية معينة t ، افترض أن الإدخال عبارة عن عدد من الدفعات الصغيرة $\mathbf{X}_t \in \mathbb{R}^{n\times d}$: عدد الأمثلة:$n$، عدد المدخلات:$d$) والحالة المخفية للخطوة الزمنية السابقة هي $\mathbf{H}_{t-1} \in \mathbb{R}^{n\times h}$ (عدد الوحدات المخفية:h). بعد ذلك، يتم حساب بوابة إعادة التعيين $\mathbf{R}_t \in \mathbb{R}^{n\times h}$ وبوابة التحديث $\mathbf{Z}_t \in \mathbb{R}^{n\times h}$ على النحو التالي:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xr} + \mathbf{H}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z),$$

حيث $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d\times h}$ و $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h\times h}$ هي معلمات الوزن و $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1\times h}$ هي معلمات التحيز.

10.2.2. الحالة المخفية المرشحة Candidate Hidden State

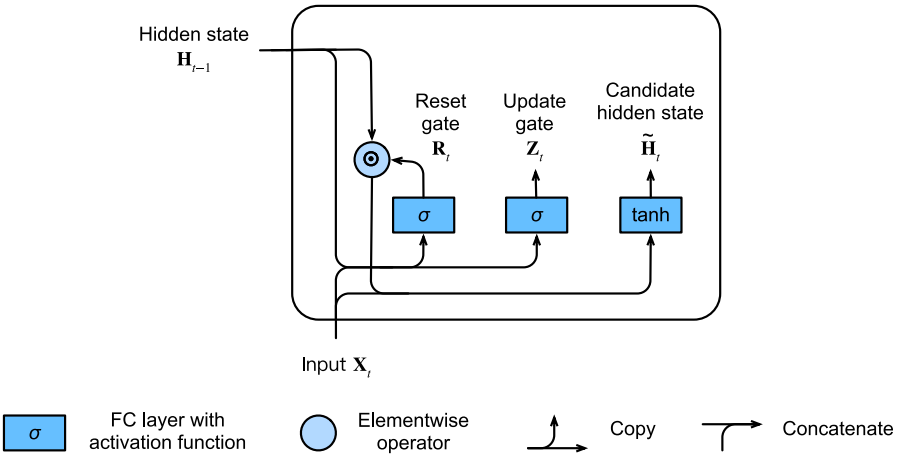
بعد ذلك، نقوم بدمج بوابة إعادة الضبط $\mathbf{R}_t$ بآلية التحديث المنتظمة regular updating mechanism في (9.4.5)، مما يؤدي إلى الحالة المخفية المرشحة candidate $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ التالي في الخطوة الزمنية t:

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h\right), \quad (10.2.2)$$

حيث $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ و $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ هي معلمات الوزن، $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ هو التحيز، والرمز $\odot$ هو عامل ضرب Hadamard (بشكل عنصري elementwise). هنا نستخدم دالة تنشيط tanh.

والنتيجة مرشح candidate، لأننا ما زلنا بحاجة إلى دمج عمل بوابة التحديث. بالمقارنة مع (9.4.5)، يمكن الآن تقليل تأثير الحالات السابقة مع الضرب الأولي لـ $\mathbf{R}_t$ و $\mathbf{H}_{t-1}$ في (10.2.2). عندما تكون الإدخالات في بوابة إعادة الضبط $\mathbf{R}_t$ قريبة من 1، فإننا نسترد vanilla RNN كما في (9.4.5). بالنسبة لجميع إدخالات بوابة إعادة الضبط $\mathbf{R}_t$ القريبة من 0، تكون حالة المرشح المخفية نتيجة MLP مع إدخال $\mathbf{X}_t$. وبالتالي يتم إعادة تعيين reset أي حالة مخفية موجودة مسبقًا إلى الإعدادات الافتراضية.

يوضح الشكل 10.2.2 التدفق الحسابي بعد تطبيق بوابة إعادة الضبط reset gate.



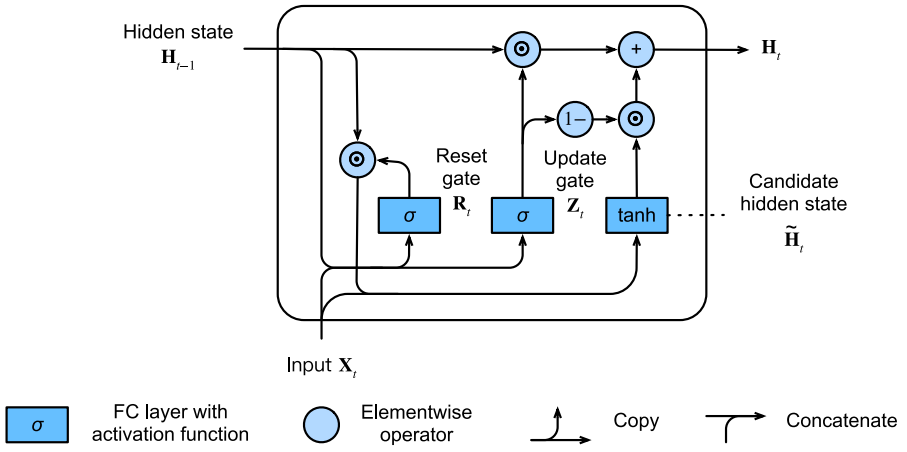
الشكل 10.2.2 حساب الحالة المخفية المرشحة في نموذج GRU.

10.2.3. الحالة المخفية Hidden State

أخيرًا، نحتاج إلى دمج تأثير بوابة التحديث $\mathbf{Z}_t$. يحدد هذا إلى أي مدى تتطابق الحالة المخفية الجديدة $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ مع الحالة القديمة $\mathbf{H}_{t-1}$ مقابل مدى تشابهها مع الحالة الجديدة المرشحة $\tilde{\mathbf{H}}_t$. يمكن استخدام بوابة التحديث $\mathbf{Z}_t$ لهذا الغرض، وذلك ببساطة عن طريق أخذ مجموعات محدبة عنصرية من $\mathbf{H}_{t-1}$ و$\tilde{\mathbf{H}}_t$. هذا يؤدي إلى معادلة التحديث النهائية لـ GRU:

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

عندما تكون بوابة التحديث $\mathbf{Z}_t$ قريبة من 1، فإننا ببساطة نحتفظ بالحالة القديمة.في هذه الحالة، يتم تجاهل المعلومات الواردة من $\mathbf{X}_t$ ، مما يؤدي بشكل فعال إلى تخطي الخطوة الزمنية tفي سلسلة التبعية.في المقابل، عندما تكون $\mathbf{Z}_t$ قريبة من 0، تقترب الحالة الكامنة الجديدة $\mathbf{H}_t$ من الحالة الكامنة المرشحة $\tilde{\mathbf{H}}_t$. يوضح الشكل 10.2.3 التدفق الحسابي بعد تشغيل بوابة التحديث.



الشكل 10.2.3 حساب الحالة المخفيةفي نموذج GRU.

باختصار، تمتلك GRU السمتين المميزتين التاليتين:

- تساعد بوابات إعادة الضبط Reset gates تعيين البوابات في التقاط التبعيات قصيرة المدى في التسلسل.
- تساعد بوابات التحديث Update gates على التقاط التبعيات طويلة المدى في التسلسل.

10.2.3.1. التنفيذ من البداية Implementation from Scratch

للحصول على فهم أفضل لنموذج GRU، دعنا نطبقه من البداية.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.2.4. تهيئة معلمات النموذج Initializing Model Parameters

الخطوة الأولى هي تهيئة معلمات النموذج. نرسم الأوزان من توزيع غاوسي مع الانحراف المعياري ليكون `sigma` ونضبط التحيز على 0. يحدد المعامل الفائق `num_hiddens` عدد الوحدات المخفية. نقوم بإنشاء جميع الأوزان والتحيزات المتعلقة ببوابة التحديث وبوابة إعادة الضبط والحالة المخفية المرشحة.

```
class GRUScratch(d2l.Module):
    def __init__(self, num_inputs, num_hiddens,
sigma=0.01):
        super().__init__()
        self.save_hyperparameters()

        init_weight = lambda *shape:
tf.Variable(tf.random.normal(shape) * sigma)
        triple = lambda: (init_weight(num_inputs,
num_hiddens),
                          init_weight(num_hiddens,
num_hiddens),

tf.Variable(tf.zeros(num_hiddens)))

        self.W_xz, self.W_hz, self.b_z = triple()  #
Update gate
        self.W_xr, self.W_hr, self.b_r = triple()  #
Reset gate
        self.W_xh, self.W_hh, self.b_h = triple()  #
Candidate hidden state
```

10.2.5. تعريف النموذج Defining the Model

الآن نحن جاهزون لتعريف حساب GRU الأمامي. هيكلها هو نفسه هيكل خلية RNN الأساسية، فيما عدا أن معادلات التحديث أكثر تعقيداً.

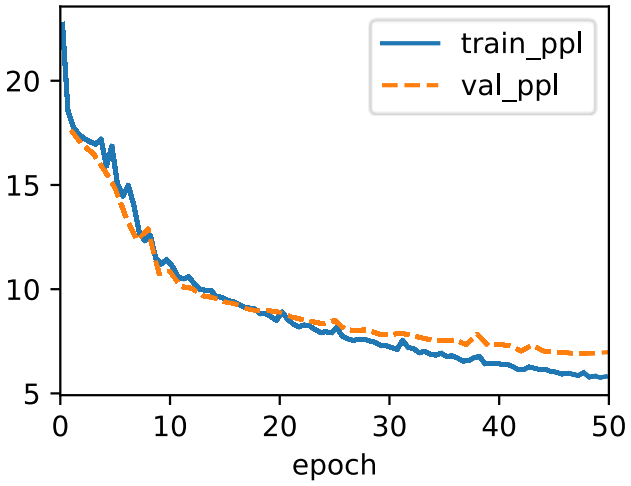
```
@d2l.add_to_class(GRUScratch)
def forward(self, inputs, H=None):
    matmul_H = lambda A, B: tf.matmul(A, B) if H is not
None else 0
    outputs = []
    for X in inputs:
        Z = tf.sigmoid(tf.matmul(X, self.W_xz) + (
```

```
            tf.matmul(H, self.W_hz) if H is not None
else 0) + self.b_z)
        if H is None: H = tf.zeros_like(Z)
        R = tf.sigmoid(tf.matmul(X, self.W_xr) +
                       tf.matmul(H, self.W_hr) +
self.b_r)
        H_tilde = tf.tanh(tf.matmul(X, self.W_xh) +
                          tf.matmul(R * H, self.W_hh) +
self.b_h)
        H = Z * H + (1 - Z) * H_tilde
        outputs.append(H)
    return outputs, (H, )
```

10.2.6. التدريب Training

يعمل تدريب نموذج لغوي على مجموعة بيانات The Time Machine بالطريقة نفسها تمامًا كما في القسم 9.5.

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
with d2l.try_gpu():
    gru = GRUScratch(num_inputs=len(data.vocab),
num_hiddens=32)
    model = d2l.RNNLMScratch(gru,
vocab_size=len(data.vocab), lr=4)
trainer = d2l.Trainer(max_epochs=50,
gradient_clip_val=1)
trainer.fit(model, data)
```



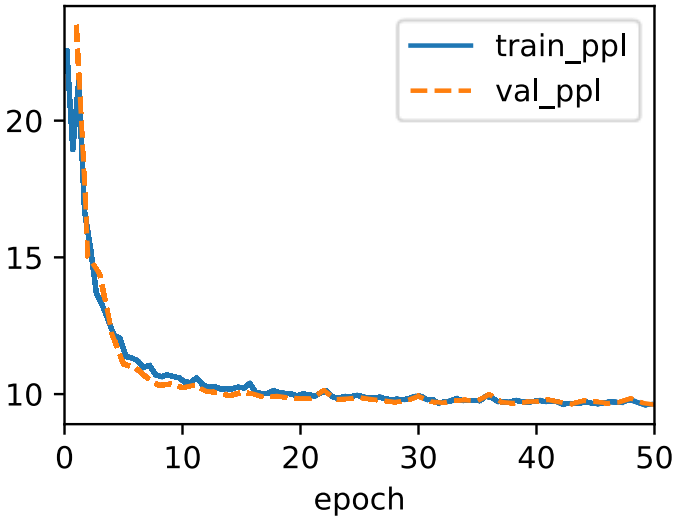
10.2.6.1. التنفيذ المختصر Concise Implementation

في واجهات برمجة التطبيقات عالية المستوى API، يمكننا إنشاء نموذج GPU مباشرةً. هذا يلخص كل تفاصيل التكوين التي أوضحناها أعلاه.

```
class GRU(d2l.RNN):
    def __init__(self, num_inputs, num_hiddens):
        d2l.Module.__init__(self)
        self.save_hyperparameters()
        self.rnn = tf.keras.layers.GRU(num_hiddens,
return_sequences=True,

return_state=True)
```

الكود أسرعفي التدريب لأنه يستخدم عوامل مترجمة compiled operators بدلاً من بايثون.

```
gru = GRU(num_inputs=len(data.vocab), num_hiddens=32)
with d2l.try_gpu():
    model = d2l.RNNLM(gru, vocab_size=len(data.vocab),
lr=4)
trainer.fit(model, data)
```



بعد التدريب، نقوم بطباعة الارتباك perplexityفي مجموعة التدريب والتسلسل المتوقع predicted sequence بعد البادئة المقدمة.

```
model.predict('it has', 20, data.vocab)
```

```
'it has the the the the the'
```

10.2.6.2. الملخص

مقارنةً بـ LSTMs، تحقق وحدات GRU أداءً مشابهًا ولكنها تميل إلى أن تكون أخف من الناحية الحسابية. بشكل عام، مقارنةً بـ RNNs البسيطة، يمكن لـ RNNs ذات البوابات مثل LSTMs وGRUs التقاط التبعيات بشكل أفضل للتسلسلات ذات مسافات الخطوة الزمنية الكبيرة. تحتوي وحدات GRU على RNNs الأساسية كحالة قصوى عند تشغيل بوابة إعادة التعيين reset gate. يمكنهم أيضًا تخطي التكرارات اللاحقة عن طريق تشغيل بوابة التحديث update gate.

10.2.6.3. التمارين

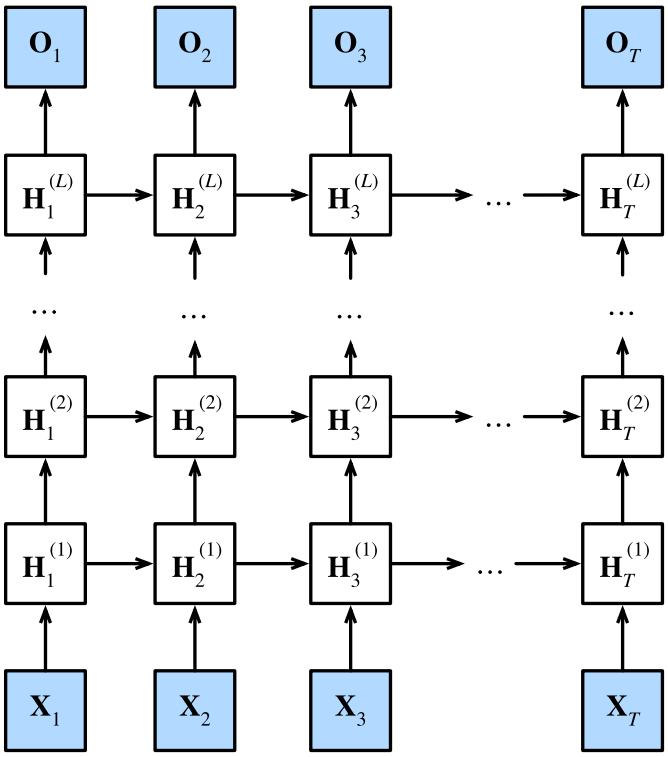
1. افترض أننا نريد فقط استخدام الإدخال في الخطوة الزمنية t' للتنبؤ بالإخراج في الخطوة الزمنية $t > t'$. ما هي أفضل القيم لبوابات إعادة التعيين والتحديث لكل خطوة زمنية؟
2. اضبط المعلمات الفائقة وحلل تأثيرها على وقت التشغيل والارتباك وتسلسل الإخراج.
3. قارن بين وقت التشغيل والارتباك وسلاسل الإخراج لتطبيقات `rnn.RNN` و `rnn.GRU` مع بعضها البعض.
4. ماذا يحدث إذا قمت بتنفيذ أجزاء فقط من GRU ، على سبيل المثال ، مع بوابة إعادة الضبط فقط أو بوابة تحديث فقط؟

10.3. الشبكات العصبية المتكررة العميقة Deep Recurrent Neural Networks

حتى الآن، ركزنا على تعريف الشبكات التي تتكون من إدخال تسلسلي وطبقة RNN مخفية واحدة وطبقة إخراج. على الرغم من وجود طبقة مخفية واحدة فقط بين الإدخالفي أي خطوة زمنية والمخرجات المقابلة، إلا أن هناك إحساسًا بأن هذه الشبكات عميقة. يمكن أن تؤثر المدخلات من الخطوة الأولى على المخرجاتفي الخطوة الزمنية النهائية T (غالبًا 100 أو 1000 خطوة لاحقًا). تمر هذه المدخلات عبر تطبيقات الطبقة المتكررة T قبل الوصول إلى الناتج النهائي. ومع ذلك، غالبًا ما نرغب أيضًافي الاحتفاظ بالقدرة على التعبير عن العلاقات المعقدة بين المدخلاتفي خطوة زمنية معينة والمخرجاتفي نفس الخطوة الزمنية. وبالتالي فإننا غالبًا ما نبني RNNs التي تكون عميقة ليس فقطفي اتجاه الوقت ولكن أيضًافي اتجاه الإدخال إلى الإخراج. هذا هو بالضبط مفهوم العمق الذي واجهناه بالفعلفي تطويرنا لـ MLPs وCNNs العميقة.

الطريقة القياسية لبناء هذا النوع من RNN العميقة بسيطة بشكل مذهل: نحن نكدس RNNs فوق بعضنا البعض. بالنظر إلى تسلسل الطول T، ينتج RNN الأول سلسلة من المخرجات،

وكذلك الطول T. هذه، بدورها، تشكل المدخلات إلى طبقة RNN التالية.في هذا القسم القصير، نوضح نمط التصميم هذا ونقدم مثالًا بسيطًا لكيفية ترميز مثل هذه RNNs المكدسة. أدناه،في الشكل 10.3.1، نوضح RNN عميقًا مع طبقات مخفية. تعمل كل حالة مخفية على إدخال متسلسل وتنتج مخرجات متسلسلة. علاوة على ذلك، تعتمد أي خلية RNN (المربع الأبيضفي الشكل 10.3.1)في كل خطوة زمنية على كل من قيمة الطبقة نفسهافي الخطوة الزمنية السابقة وقيمة الطبقة السابقةفي نفس الخطوة الزمنية.



الشكل 10.3.1 معمارية شبكة RNN العميقة.

بشكل رسمي، افترض أن لدينا الدفعات الصغيرة مع الادخال $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (عدد الأمثلة:n ، عدد المدخلاتفي كل مثال:d)في الخطوة الزمنية.في نفس الخطوة الزمنية t ، ليكن الحالة المخفية لـ l^{th} الطبقة المخفية ($l = 1, \ldots, L$) تكون $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n\times h}$ (عدد الوحدات المخفية: h) ومتغير طبقة الإخراج يكون $\mathbf{O}_t \in \mathbb{R}^{n\times q}$ (عدد النواتج:q). ضبط $\mathbf{H}_t^{(0)} = \mathbf{X}_t$، يتم حساب الحالة المخفية لـ l^{th} الطبقة المخفية التي تستخدم دالة التنشيط ϕ_l على النحو التالي:

$$\mathbf{H}_t^{(l)} = \phi_l(\mathbf{H}_t^{(l-1)}\mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)}),$$

حيث الأوزان $\mathbf{W}_{xh}^{(l)} \in \mathbb{R}^{h \times h}$ و $\mathbf{W}_{hh}^{(l)} \in \mathbb{R}^{h \times h}$، جنبًا إلى جنب مع التحيز $\mathbf{b}_h^{(l)} \in \mathbb{R}^{1 \times h}$، هي معلمات نموذج l^{th} الطبقة المخفية.

في النهاية، يعتمد حساب الطبقة المخرجة فقط على الحالة المخفية لـ L^{th} الطبقة المخفية النهائية:

$$\mathbf{O}_t = \mathbf{H}_t^{(L)} \mathbf{W}_{hq} + \mathbf{b}_q,$$

حيث الوزن $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ والتحيز $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ هما معلمات النموذج للطبقة الناتجة.

تمامًا كما هو الحال مع MLPs، فإن عدد الطبقات المخفية L وعدد الوحدات المخفية h هي معلمات فائقة يمكننا ضبطها. تقع عروض طبقة RNN الشائعة (h) في النطاق (64,2056) ، بينما تقع الأعماق الشائعة (L) في النطاق (1,8). بالإضافة إلى ذلك، يمكننا بسهولة الحصول على RNN ذي بوابات عميقة عن طريق استبدال حساب الحالة المخفيةفي (10.3.1) بحساب LSTM أو GRU.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.3.1. التنفيذ من البداية Implementation from Scratch

لتنفيذ RNN متعدد الطبقات من البداية، يمكننا التعامل مع كل طبقة على أنها مثيل `RNNScratch` مع معلماتها القابلة للتعلم.

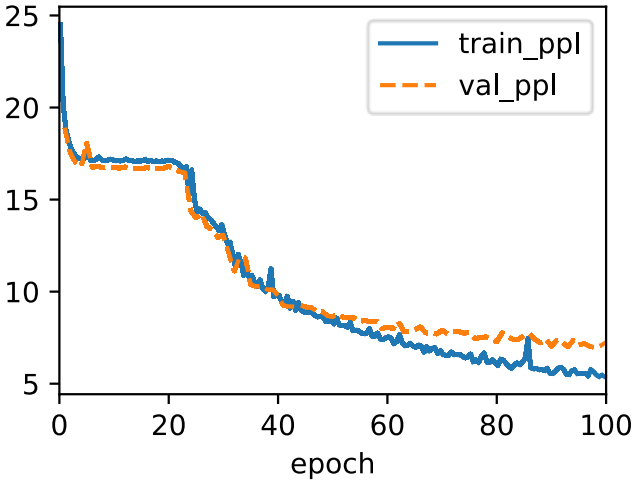
```
class StackedRNNScratch(d2l.Module):
    def __init__(self, num_inputs, num_hiddens,
num_layers, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.rnns = [d2l.RNNScratch(num_inputs if i==0
else num_hiddens,
                                    num_hiddens, sigma)
                     for i in range(num_layers)]
```

يقوم الحساب الأمامي متعدد الطبقات ببساطة بإجراء حساب إلى الأمام طبقة تلو الأخرى.

```
@d2l.add_to_class(StackedRNNScratch)
def forward(self, inputs, Hs=None):
    outputs = inputs
    if Hs is None: Hs = [None] * len(inputs)
    for i in range(self.num_layers):
        outputs, Hs[i] = self.rnns[i](outputs, Hs[i])
    return outputs, Hs
```

على سبيل المثال، نقوم بتدريب نموذج GRU عميق على مجموعة بيانات The Time Machine (كما في القسم 9.5). لتبسيط الأمور، قمنا بتعيين عدد الطبقات على 2.

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
with d2l.try_gpu():
    rnn_block =
StackedRNNScratch(num_inputs=len(data.vocab),
                                num_hiddens=32,
num_layers=2)
    model = d2l.RNNLMScratch(rnn_block,
vocab_size=len(data.vocab), lr=2)
trainer = d2l.Trainer(max_epochs=100,
gradient_clip_val=1)
trainer.fit(model, data)
```



10.3.2. التنفيذ المختصر Concise Implementation

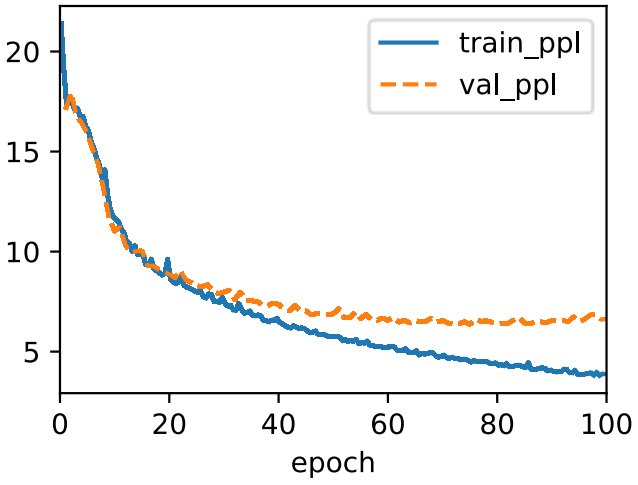
لحسن الحظ، فإن العديد من التفاصيل اللوجستية المطلوبة لتنفيذ طبقات متعددة من RNN متاحة بسهولةفي واجهات برمجة التطبيقات عالية المستوى API. سيستخدم تطبيقنا الموجز مثل هذه الدوال المدمجة. يعمم الكود الرمز الذي استخدمناه سابقًافي القسم 10.2، مما يسمح بتحديد عدد الطبقات بشكل صريح بدلاً من اختيار الافتراضي لطبقة واحدة.

```
class GRU(d2l.RNN):  #@save
    def __init__(self, num_hiddens, num_layers,
dropout=0):
        d2l.Module.__init__(self)
        self.save_hyperparameters()
```

```
        gru_cells =
[tf.keras.layers.GRUCell(num_hiddens, dropout=dropout)
                     for _ in range(num_layers)]
        self.rnn = tf.keras.layers.RNN(gru_cells,
return_sequences=True,

return_state=True, time_major=True)

    def forward(self, X, state=None):
        outputs, *state = self.rnn(X, state)
        return outputs, state
```

تشبه القرارات المعمارية مثل اختيار المعلمات الفائقة إلى حد كبير تلك الواردةفي القسم 10.2. نختار نفس عدد المدخلات والمخرجات لأن لدينا رموز مميزة distinct tokens، أي حجم_المفردات vocab_size. لا يزال عدد الوحدات المخفية 32. الاختلاف الوحيد هو أننا نختار الآن عددًا غير أساسي من الطبقات المخفية عن طريق تحديد قيمة عدد الطبقات.

```
gru = GRU(num_hiddens=32, num_layers=2)
with d2l.try_gpu():
    model = d2l.RNNLM(gru, vocab_size=len(data.vocab),
lr=2)
trainer.fit(model, data)
```




```
model.predict('it has', 20, data.vocab)
```

```
'it has i the time travelle'
```

10.3.3. الملخص

في RNNs العميقة، يتم تمرير معلومات الحالة المخفية إلى الخطوة الزمنية التالية للطبقة الحالية والخطوة الزمنية الحالية للطبقة التالية. توجد العديد من النكهات المختلفة لـ RNNs العميقة، مثل LSTMs أو GRUs أو vanilla RNNs. بشكل ملائم، تتوفر جميع هذه النماذج كأجزاء من واجهات برمجة التطبيقات عالية المستوى لأطر التعلم العميق. يتطلب تهيئة النماذج الاهتمام. بشكل عام، تتطلب شبكات RNN العميقة قدرًا كبيرًا من العمل (مثل معدل التعلم والقص clipping) لضمان التقارب المناسب.

10.3.4. التمارين

1. استبدل GRU بـ LSTM وقارن الدقة وسرعة التدريب.
2. قم بزيادة بيانات التدريب لتشمل كتبًا متعددة. إلى أي مدى يمكن أن تذهب على مقياس الارتباك perplexity scale ؟
3. هل تريد الجمع بين مصادر مؤلفين مختلفين عند نمذجة النص؟ لماذا هذه الفكرة جيدة؟ ما الخطأ الذي يمكن أن يحدث؟

10.4. الشبكات العصبية المتكررة ثنائية الاتجاه Bidirectional Recurrent Neural Networks

حتى الآن، كان مثالنا العملي لمهمة التعلم المتسلسل هو نمذجة اللغة، حيث نهدف إلى توقع الرمز التالي مع الأخذفي الاعتبار جميع الرموز السابقةفي تسلسل.في هذا السيناريو، نرغب فقطفي الشرط على السياق الأيسر، وبالتالي يبدو التسلسل أحادي الاتجاه unidirectional chaining لـ RNN القياسي مناسبًا. ومع ذلك، هناك العديد من سياقات مهام التعلم المتسلسلة الأخرى حيث يكون من الجيد تمامًا تكييف التنبؤفي كل خطوة زمنية على كل من السياق الأيمن والأيسر. ضعفي اعتبارك، على سبيل المثال، جزءًا من اكتشاف الكلام part of speech detection. لماذا لا يجب أن نأخذ السياقفي كلا الاتجاهينفي الاعتبار عند تقييم جزء الكلام المرتبط بكلمة معينة؟

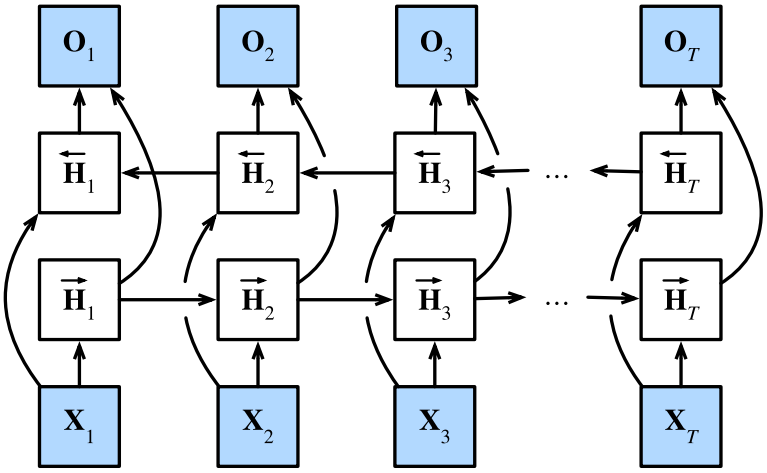
مهمة أخرى شائعة ــ غالبًا ما تكون مفيدة كتمرين مسبق قبل ضبط نموذج على مهمة فعلية مثيرة للاهتمام ــ وهي إخفاء الرموز العشوائيةفي مستند نصي ثم تدريب نموذج تسلسل للتنبؤ بقيم الرموز المفقودة. لاحظ أنه بناءً على ما يأتي بعد الفراغ، فإن القيمة المحتملة للرمز المفقود تتغير بشكل كبير:

- I am ___.
- I am ___ hungry.

- I am ___ hungry, and I can eat half a pig.

في الجملة الأولى، يبدو أن كلمة "happy " هي المرشح المحتمل. يبدو أن الكلمتين "not" و "very" معقولتانفي الجملة الثانية، لكن "not" تبدو غير متوافقة مع الجمل الثالثة.

لحسن الحظ، هناك تقنية بسيطة تحول أي RNN أحادي الاتجاه unidirectional إلى RNN ثنائي الاتجاه bidirectional (Schuster and Paliwal، 1997). نحن ببساطة ننفذ طبقتين من RNN أحادي الاتجاه مرتبطان ببعضهما البعضفي اتجاهات متعاكسة وتعمل على نفس المدخلات (الشكل 10.4.1). بالنسبة لطبقة RNN الأولى، يكون الإدخال الأول هو $\mathbf{x}_1$ والمدخل الأخير $\mathbf{x}_T$ ، ولكن بالنسبة لطبقة RNN الثانية، يكون الإدخال الأول $\mathbf{x}_T$ والمدخل الأخير هو $\mathbf{x}_1$. لإنتاج ناتج طبقة RNN ثنائية الاتجاه هذه، نقوم ببساطة بربط المخرجات المقابلة لطبقتين RNN الأساسيتين أحادي الاتجاه معًا.



الشكل 10.4.1 معمارية RNN ثنائية الاتجاه.

رسميًا لأي خطوة زمنية t ، نعتبر إدخال الدفعات الصغيرة هو $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (عدد الأمثلة:n ، عدد المدخلاتفي كل مثال:d) ودع دالة تنشيط الطبقة المخفية تكون ϕ.في البنية ثنائية الاتجاه، الحالات المخفية للأمام والخلف لهذه الخطوة الزمنية هي $\overrightarrow{\mathbf{H}}_t \in \mathbb{R}^{n\times h}$ و $\overleftarrow{\mathbf{H}}_t \in \mathbb{R}^{n\times h}$، على التوالي، حيث h هو عدد الوحدات المخفية. تحديثات الحالة المخفية للأمام والخلف هي كما يلي:

$$\begin{aligned}\overrightarrow{\mathbf{H}}_t &= \phi(\mathbf{X}_t\mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1}\mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}),\\ \overleftarrow{\mathbf{H}}_t &= \phi(\mathbf{X}_t\mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1}\mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}),\end{aligned}$$

حيث الأوزان $\mathbf{W}_{xh}^{(f)} \in \mathbb{R}^{d\times h}, \mathbf{W}_{hh}^{(f)} \in \mathbb{R}^{h\times h}, \mathbf{W}_{xh}^{(b)} \in \mathbb{R}^{d\times h}$, and $\mathbf{W}_{hh}^{(b)} \in \mathbb{R}^{h\times h}$ و $\mathbf{W}_{hh}^{(b)} \in \mathbb{R}^{h\times h}$ والتحيزات $\mathbf{b}_h^{(b)} \in \mathbb{R}^{1\times h}$ هي جميع معلمات النموذج.

بعد ذلك، نجمع بين الحالات المخفية الأمامية والخلفية $\overrightarrow{\mathbf{H}}_t$ و $\overleftarrow{\mathbf{H}}_t$ للحصول على الحالة المخفية $\mathbf{H}_t \in \mathbb{R}^{n\times 2h}$ التي سيتم تغذيتهافي طبقة الإخراج.في شبكات RNN ثنائية الاتجاه العميقة ذات الطبقات المخفية المتعددة، يتم تمرير هذه المعلومات كمدخلات إلى الطبقة ثنائية الاتجاه التالية. أخيرًا، طبقة الإخراج تحسب الإخراج $\mathbf{O}_t \in \mathbb{R}^{n\times q}$ (عدد المخرجات:q):

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$

هنا، مصفوفة الوزن $\mathbf{W}_{hq} \in \mathbb{R}^{2h\times q}$ والتحيز $\mathbf{b}_q \in \mathbb{R}^{1\times q}$ هما معلمات نموذج طبقة الإخراج. بينما من الناحية الفنية، يمكن أن يكون للاتجاهين أعداد مختلفة من الوحدات المخفية، إلا أن اختيار التصميم هذا نادرًا ما يتم إجراؤه عمليًا. نعرض الآن تنفيذاً بسيطًا لـ RNN ثنائي الاتجاه bidirectional RNN.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.4.1. التنفيذ من البداية Implementation from Scratch

لتنفيذ RNN ثنائي الاتجاه من البداية، يمكننا تضمين مثيلين `RNNScratch` أحادي الاتجاه مع معلمات منفصلة قابلة للتعلم.

```
class BiRNNScratch(d2l.Module):
    def __init__(self, num_inputs, num_hiddens,
sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.f_rnn = d2l.RNNScratch(num_inputs,
num_hiddens, sigma)
        self.b_rnn = d2l.RNNScratch(num_inputs,
num_hiddens, sigma)
        self.num_hiddens *= 2  # The output dimension
will be doubled
```

يتم تحديث حالات RNNs الأمامية والخلفية بشكل منفصل، بينما يتم توصيل مخرجات هاتين RNNs.

```
@d2l.add_to_class(BiRNNScratch)
def forward(self, inputs, Hs=None):
    f_H, b_H = Hs if Hs is not None else (None, None)
    f_outputs, f_H = self.f_rnn(inputs, f_H)
```

```
    b_outputs, b_H = self.b_rnn(reversed(inputs), b_H)
    outputs = [tf.concat((f, b), -1) for f, b in
zip(f_outputs, b_outputs)]
    return outputs, (f_H, b_H)
```

10.4.2. التنفيذ المختصر Concise Implementation

باستخدام واجهات برمجة التطبيقات عالية المستوى API، يمكننا تنفيذ RNN ثنائية الاتجاه بشكل أكثر إيجازًا. هنا نأخذ نموذج GRU كمثال.

10.4.2.1. الملخص

في RNNs ثنائية الاتجاه bidirectional RNNs، يتم تحديد الحالة المخفية لكل خطوة زمنيةفي وقت واحد بواسطة البيانات قبل وبعد الخطوة الزمنية الحالية. RNNs ثنائية الاتجاه مفيدةفي الغالب لتشفير التسلسل وتقدير الملاحظاتفي سياق ثنائي الاتجاه. RNNs ثنائية الاتجاه مكلفة للغاية للتدريب بسبب سلاسل التدرج الطويلة long gradient chains.

10.4.2.2. التمارين

1. إذا كانت الاتجاهات المختلفة تستخدم عددًا مختلفًا من الوحدات المخفية، فكيف سيتغير الشكل $\mathbf{H}_t$ ؟
2. صمم RNN ثنائي الاتجاه مع طبقات مخفية متعددة.
3. تعدد المعاني Polysemy شائعفي اللغات الطبيعية. على سبيل المثال، لكلمة "bank" معاني مختلفةفي السياقات " i went to the bank to deposit cash " و " i went to the bank to sit down ". كيف يمكننا تصميم نموذج شبكة عصبية بحيث يتم إرجاع تمثيل متجه للكلمةفي السياقفي ضوء تسلسل السياق والكلمة؟ ما نوع البنى العصبية المفضل للتعامل مع تعدد المعاني polysemy؟

10.5. الترجمة الآلية ومجموعة البيانات Machine Translation and the Dataset

من بين الإنجازات الرئيسية التي أدت إلى اهتمام واسع النطاق بـ RNNs الحديثة، كان هناك تقدم كبيرفي المجال التطبيقي للترجمة الآلية machine translation الإحصائية. هنا، يتم تقديم النموذج بجملة بلغة واحدة ويجب أن يتنبأ بالجمل المقابلفي لغة أخرى. لاحظ أن الجمل هنا قد تكون ذات أطوال مختلفة، وأن الكلمات المقابلةفي الجملتين قد لا تظهر بنفس الترتيب، بسبب الاختلافاتفي التركيب النحوي للغتين.

العديد من المشاكل لها طابع التعيين mapping بين اثنين من هذه التتابعات "غير المحاذاة unaligned ". تتضمن الأمثلة التعيين من مطالبات الحوار إلى الردود أو من الأسئلة إلى

الإجابات. بشكل عام، تسمى هذه المشكلات مشاكل التسلسل إلى التسلسل sequence-to-sequence (seq2seq) وهي محور تركيزنا لكل من الجزء المتبقي من هذا الفصل والكثير من القسم 11.

في هذا القسم، نقدم مشكلة الترجمة الآلية machine translation ومثالًا لمجموعة البيانات dataset التي سنستخدمهافي الأمثلة اللاحقة. لعقود من الزمان، كانت الصيغ الإحصائية للترجمة بين اللغات شائعة (Brown et al.، 1990، Brown et al.، 1988)، حتى قبل أن يعمل الباحثون على اتباع نهج الشبكة العصبية (غالبًا ما يتم تجميع الطرق معًا تحت مصطلح الترجمة الآلية العصبية neural machine translation).

أولاً، سنحتاج إلى رمز جديد لمعالجة بياناتنا. على عكس نمذجة اللغة التي رأيناهافي القسم 9.3، يتكون كل مثال هنا من تسلسلين نصيين منفصلين، أحدهما بلغة المصدر والآخر (الترجمة)في اللغة الهدف. ستوضح مقتطفات التعليمات البرمجية التالية كيفية تحميل البيانات المعالجة مسبقًا إلى الدفعات الصغيرة للتدريب.

```
import os
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.5.1. تنزيل مجموعة البيانات ومعالجتها مسبقًا Downloading and Preprocessing the Dataset

للبدء، نقوم بتنزيل مجموعة بيانات إنجليزية–فرنسية تتكون من أزواج جمل ثنائية اللغة bilingual sentence pairs من مشروع Tatoeba. كل سطرفي مجموعة البيانات عبارة عن زوج محدد بعلامات جدولة يتكون من تسلسل نص إنجليزي وتسلسل نص فرنسي مترجم. لاحظ أن كل تسلسل نصي يمكن أن يكون مجرد جملة واحدة، أو فقرة من جمل متعددة.في مشكلة الترجمة الآلية هذه حيث تتم ترجمة اللغة الإنجليزية إلى الفرنسية، تسمى اللغة الإنجليزية لغة المصدر source language وتسمى الفرنسية اللغة الهدف target language.

```
class MTFraEng(d2l.DataModule):  #@save
    def _download(self):
        d2l.extract(d2l.download(
            d2l.DATA_URL+'fra-eng.zip', self.root,
            '94646ad1522d915e7b0f9296181140edcf86a4f5'))
        with open(self.root + '/fra-eng/fra.txt',
encoding='utf-8') as f:
            return f.read()

data = MTFraEng()
```

```
raw_text = data._download()
print(raw_text[:75])
```

```
Go. Va !
Hi. Salut !
Run!        Cours !
Run!        Courez !
Who?        Qui ?
Wow!        Ça alors !
```

بعد تنزيل مجموعة البيانات dataset، ننتقل إلى العديد من خطوات المعالجة المسبقة لبيانات النص الخام. على سبيل المثال، نستبدل المسافات غير المنقسمة بمسافة، ونحول الأحرف الكبيرة إلى أحرف صغيرة، ونضع مسافة بين الكلمات وعلامات الترقيم.

```
@d2l.add_to_class(MTFraEng)  #@save
def _preprocess(self, text):
    # Replace non-breaking space with space
    text = text.replace('\u202f', ' ').replace('\xa0', '
')
    # Insert space between words and punctuation marks
    no_space = lambda char, prev_char: char in ',.!?'
and prev_char != ' '
    out = [' ' + char if i > 0 and no_space(char, text[i
- 1]) else char
           for i, char in enumerate(text.lower())]
    return ''.join(out)

text = data._preprocess(raw_text)
print(text[:80])
```

```
go .        va !
hi .        salut !
run !       cours !
run !       courez !
who ?       qui ?
wow !       ça alors !
```

10.5.2. الترميز Tokenization

على عكس الترميز على مستوى الحرف في القسم 9.3، بالنسبة للترجمة الآلية، فإننا نفضل الترميز على مستوى الكلمات هنا (تستخدم النماذج الحديثة اليوم تقنيات ترميز أكثر تعقيدًا). تقوم طريقة `_tokenize` التالية بترميز أول أزواج تسلسل نصي `max_examples`، حيث يكون كل رمز إما كلمة أو علامة ترقيم. نلحق الرمز المميز "<eos>" بنهاية كل تسلسل للإشارة إلى نهاية التسلسل. عندما يتنبأ نموذج عن طريق إنشاء رمز تسلسلي بعد الرمز، يمكن أن يشير إنشاء

الرمز المميز "<eos>" إلى اكتمال تسلسل الإخراج.في النهاية، تُرجع الطريقة أدناه قائمتين من قوائم الرموز المميزة: `src` و `tgt`. على وجه التحديد ، `src[i]` هي قائمة من الرموز من تسلسل النصفي اللغة المصدر (الإنجليزية هنا) و `tgt[i]` هو ذلكفي اللغة الهدف (الفرنسية هنا).

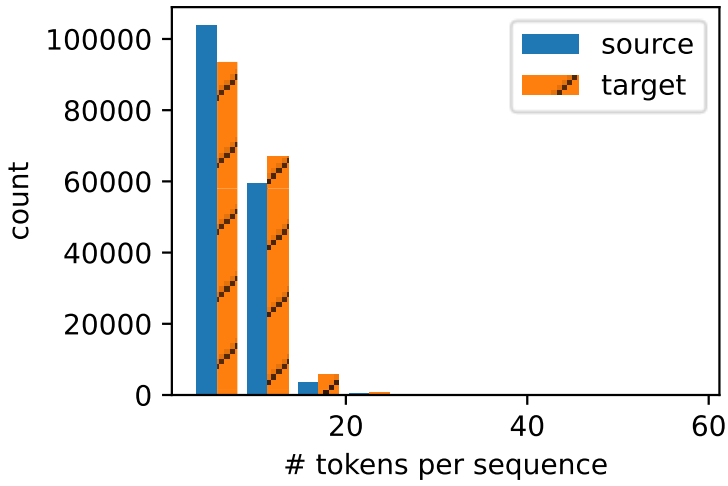
```
@d2l.add_to_class(MTFraEng)  #@save
def _tokenize(self, text, max_examples=None):
    src, tgt = [], []
    for i, line in enumerate(text.split('\n')):
        if max_examples and i > max_examples: break
        parts = line.split('\t')
        if len(parts) == 2:
            # Skip empty tokens
            src.append([t for t in f'{parts[0]} <eos>'.split(' ') if t])
            tgt.append([t for t in f'{parts[1]} <eos>'.split(' ') if t])
    return src, tgt

src, tgt = data._tokenize(text)
src[:6], tgt[:6]
```

```
([['go', '.', '<eos>'],
  ['hi', '.', '<eos>'],
  ['run', '!', '<eos>'],
  ['run', '!', '<eos>'],
  ['who', '?', '<eos>'],
  ['wow', '!', '<eos>']],
 [['va', '!', '<eos>'],
  ['salut', '!', '<eos>'],
  ['cours', '!', '<eos>'],
  ['courez', '!', '<eos>'],
  ['qui', '?', '<eos>'],
  ['ça', 'alors', '!', '<eos>']])
```

دعنا نرسم المدرج التكراري لعدد الرموز لكل تسلسل نصي.في مجموعة البيانات الإنجليزية الفرنسية البسيطة هذه، تحتوي معظم التسلسلات النصية على أقل من 20 رمزًا.

```
#@save
def show_list_len_pair_hist(legend, xlabel, ylabel,
xlist, ylist):
    """Plot the histogram for list length pairs."""
    d2l.set_figsize()
```

```
    _, _, patches = d2l.plt.hist(
        [[len(l) for l in xlist], [len(l) for l in
ylist]])
    d2l.plt.xlabel(xlabel)
    d2l.plt.ylabel(ylabel)
    for patch in patches[1].patches:
        patch.set_hatch('/')
    d2l.plt.legend(legend)

show_list_len_pair_hist(['source', 'target'], '# tokens
per sequence',
                        'count', src, tgt);
```



10.5.3. تحميل التسلسلات ذات الطول الثابت Loading Sequences of Fixed Length

تذكر أنهفي نمذجة اللغة لكل مثال تسلسل، سواء كان مقطعًا من جملة واحدة أو امتدادًا عبر جمل متعددة، يكون له طول ثابت. تم تحديد ذلك بواسطة الوسيطة `num_steps` (عدد الخطوات الزمنية أو الرموز tokens)في القسم 9.3.في الترجمة الآلية، يكون كل مثال زوجًا من تسلسل النص المصدر والهدف، حيث قد يكون لتسلسل النصين أطوال مختلفة.

لتحقيق الكفاءة الحسابية، لا يزال بإمكاننا معالجة مجموعة صغيرة من تسلسلات النصفي وقت واحد عن طريق الاقتطاع truncation والحشو padding. افترض أن كل تسلسلفي نفس الدفعات الصغيرة minibatch يجب أن يكون له نفس الطول `num_steps`. إذا كان التسلسل النصي يحتوي على أقل من `num_steps` للرموز، فسنستمرفي إلحاق الرمز المميز "<pad>" بنهايته حتى يصل طوله إلى عدد الخطوات `num_steps`. خلافًا لذلك، سنقوم باقتطاع

تسلسل النص من خلال أخذ الرموز لعدد الخطوات الأولى num_steps فقط والتخلص من الباقي. بهذه الطريقة، سيكون لكل تسلسل نصي نفس الطول ليتم تحميلهفي دفعات صغيرة من نفس الشكل. إلى جانب ذلك، نسجل أيضًا طول تسلسل المصدر باستثناء الرموز للحشو padding. ستحتاج بعض النماذج إلى هذه المعلومات التي سنغطيها لاحقًا.

نظرًا لأن مجموعة بيانات الترجمة الآلية تتكون من أزواج من اللغات، يمكننا بناء مفردتين لكل من اللغة المصدر واللغة الهدف بشكل منفصل. باستخدام الترميز على مستوى الكلمة، سيكون حجم المفردات أكبر بكثير من ذلك باستخدام الترميز على مستوى الحرف. للتخفيف من هذا، نتعامل هنا مع الرموز غير المتكررة التي تظهر أقل من مرتين كرمز غير معروف ("<unk>"). كما سنشرح لاحقًا (الشكل 10.7.1)، عند التدريب باستخدام التسلسلات المستهدفة، يمكن أن يكون إخراج وحدة مفكك الشفرة decoder output (رموز للتسمية label tokens) هو نفس مدخلات مفكك الشفرة (رموز المستهدفة target tokens) ، والتي يتم إزاحتها بواسطة رمز واحد وسيتم استخدام الرمز لبداية التسلسل "<bos>" كأول رمز للدخل للتنبؤ بالتسلسل المستهدف (الشكل 10.7.3).

```
@d2l.add_to_class(MTFraEng)  #@save
def __init__(self, batch_size, num_steps=9,
num_train=512, num_val=128):
    super(MTFraEng, self).__init__()
    self.save_hyperparameters()
    self.arrays, self.src_vocab, self.tgt_vocab =
self._build_arrays(
        self._download())


@d2l.add_to_class(MTFraEng)  #@save
def _build_arrays(self, raw_text, src_vocab=None,
tgt_vocab=None):
    def _build_array(sentences, vocab, is_tgt=False):
        pad_or_trim = lambda seq, t: (
            seq[:t] if len(seq) > t else seq + ['<pad>']
* (t - len(seq)))
        sentences = [pad_or_trim(s, self.num_steps) for
s in sentences]
        if is_tgt:
            sentences = [['<bos>'] + s for s in
sentences]
        if vocab is None:
            vocab = d2l.Vocab(sentences, min_freq=2)
```

```
        array = tf.constant([vocab[s] for s in
sentences])
        valid_len = tf.reduce_sum(
            tf.cast(array != vocab['<pad>'], tf.int32),
1)
        return array, vocab, valid_len
    src, tgt =
self._tokenize(self._preprocess(raw_text),
                                 self.num_train +
self.num_val)
    src_array, src_vocab, src_valid_len =
_build_array(src, src_vocab)
    tgt_array, tgt_vocab, _ = _build_array(tgt,
tgt_vocab, True)
    return ((src_array, tgt_array[:,:-1], src_valid_len,
tgt_array[:,1:]),
            src_vocab, tgt_vocab)
```

10.5.4. قراءة مجموعة البيانات Reading the Dataset

أخيرًا، نحدد طريقة `get_dataloader` لإرجاع مكرر البيانات data iterator.

```
@d2l.add_to_class(MTFraEng)  #@save
def get_dataloader(self, train):
    idx = slice(0, self.num_train) if train else
slice(self.num_train, None)
    return self.get_tensorloader(self.arrays, train,
idx)
```

دعنا نقرأ الدفعة الأولى من مجموعة البيانات الإنجليزية–الفرنسية.

```
data = MTFraEng(batch_size=3)
src, tgt, src_valid_len, label =
next(iter(data.train_dataloader()))
print('source:', tf.cast(src, tf.int32))
print('decoder input:', tf.cast(tgt, tf.int32))
print('source len excluding pad:',
tf.cast(src_valid_len, tf.int32))
print('label:', tf.cast(label, tf.int32))
```

```
source: tf.Tensor(
[[ 79   5   0   3   4   4   4   4   4]
 [ 28 150   2   3   4   4   4   4   4]
 [ 69   0   3   4   4   4   4   4   4]], shape=(3, 9),
dtype=int32)
decoder input: tf.Tensor(
```

```
[[  3  49  37   6   0   4   5   5   5]
 [  3 206  31   0   4   5   5   5   5]
 [  3 210   6   0   4   5   5   5   5]], shape=(3, 9),
dtype=int32)
source len excluding pad: tf.Tensor([4 4 3], shape=(3,),
dtype=int32)
label: tf.Tensor(
[[ 49  37   6   0   4   5   5   5   5]
 [206  31   0   4   5   5   5   5   5]
 [210   6   0   4   5   5   5   5   5]], shape=(3, 9),
dtype=int32)
```

نعرض أدناه زوجًا من التسلسلات المصدر والهدف التي تتم معالجتها بواسطة طريقة `_build_arrays` أعلاه (في تنسيق السلسلة string format).

```
@d2l.add_to_class(MTFraEng)  #@save
def build(self, src_sentences, tgt_sentences):
    raw_text = '\n'.join([src + '\t' + tgt for src, tgt
in zip(
        src_sentences, tgt_sentences)])
    arrays, _, _ = self._build_arrays(
        raw_text, self.src_vocab, self.tgt_vocab)
    return arrays

src, tgt, _,  _ = data.build(['hi .'], ['salut .'])
print('source:',
data.src_vocab.to_tokens(tf.cast(src[0], tf.int32)))
print('target:',
data.tgt_vocab.to_tokens(tf.cast(tgt[0], tf.int32)))
```

```
source: ['hi', '.', '<eos>', '<pad>', '<pad>', '<pad>',
'<pad>', '<pad>', '<pad>']
target: ['<bos>', 'salut', '.', '<eos>', '<pad>',
'<pad>', '<pad>', '<pad>', '<pad>']
```

10.5.5. الملخص

في معالجة اللغة الطبيعية، تشير الترجمة الآلية إلى مهمة التعيين التلقائي من تسلسل يمثل سلسلة نصيةفي لغة المصدر إلى سلسلة تمثل ترجمة معقولةفي لغة الهدف. باستخدام الترميز على مستوى الكلمة word-level tokenization، سيكون حجم المفردات أكبر بكثير من ذلك باستخدام الترميز على مستوى الحرف character-level tokenization، لكن أطوال التسلسل ستكون أقصر بكثير. لتقليل حجم المفردات الكبير، يمكننا التعامل مع الرموز غير المتكررة على أنها رمز مميز "غير معروف unknown". يمكننا اقتطاع truncate وحشو pad تسلسلات النص بحيث

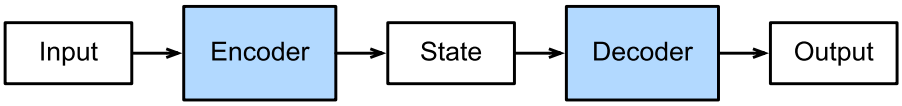
يكون لكل منهم نفس الطول ليتم تحميله ف الدفعات الصغيرة minibatches. غالبًا ما تستخدم التطبيقات الحديثة تسلسلات بأطوال مماثلة لتجنب إهدار الحساب المفرط على الحشو.

10.5.6. التمارين

1. جرب قيمًا مختلفة للوسيطة `max_examples` في طريقة `_tokenize`. كيف يؤثر ذلك على أحجام مفردات اللغة المصدر واللغة الهدف؟
2. لا يحتوي النص في بعض اللغات مثل الصينية واليابانية على مؤشرات حدود الكلمات (على سبيل المثال، المسافة). هل لا يزال الترميز على مستوى الكلمة فكرة جيدة لمثل هذه الحالات؟ لما ولما لا؟

10.6. معمارية المشفر ومفكك الشفرة Encoder-Decoder Architecture

في مسائل seq2seq العامة مثل الترجمة الآلية (القسم 10.5)، تكون المدخلات والمخرجات ذات أطوال متفاوتة وغير محاذية. يتمثل النهج القياسي لمعالجة هذا النوع من البيانات في تصميم معمارية المشفر–مفكك الشفرة encoder-decoder architecture (الشكل 10.6.1) تتكون من مكونين رئيسيين: المشفر encoder يأخذ تسلسلًا متغير الطول كمدخل، ومفكك الشفرة decoder يعمل كنموذج لغة شرطي، مع الأخذ في الاعتبار المدخلات المشفرة والسياق الأيسر للتسلسل المستهدف والتنبؤ بالرمز التالي في التسلسل المستهدف.



الشكل 10.6.1 معمارية المشفر ومفكك الشفرة.

لنأخذ الترجمة الآلية من الإنجليزية إلى الفرنسية كمثال. بالنظر إلى تسلسل الإدخال باللغة الإنجليزية: "They" ، "are" ، "watching" ، "." ، تقوم معمارية المشفر–مفكك الشفرة هذه أولاً بتشفير المدخلات ذات الطول المتغير إلى حالة، ثم تقوم بفك تشفير الحالة لإنشاء التسلسل المترجم، رمزاً رمزاً، كمخرج: "Ils" ، "regardent" ، "." . نظرًا لأن معمارية المشفر–مفكك الشفرة تشكل أساسًا لنماذج seq2seq المختلفة في الأقسام اللاحقة، فإن هذا القسم سيحول هذه المعمارية إلى واجهة سيتم تنفيذها لاحقًا.

10.6.1. المشفر Encoder

في واجهة المشفر، نحدد فقط أن المشفر يأخذ تسلسلات متغيرة الطول كمدخل `X`. سيتم توفير التنفيذ بواسطة أي نموذج يرث كلاس `Encoder` الأساسية هذه.

```
import tensorflow as tf
```

```
from d2l import tensorflow as d2l


#@save
class Encoder(tf.keras.layers.Layer):
    """The base encoder interface for the encoder-
decoder architecture."""
    def __init__(self):
        super().__init__()

    # Later there can be additional arguments (e.g.,
length excluding padding)
    def call(self, X, *args):
        raise NotImplementedError
```

10.6.2. مفكك الشفرة Decoder

في واجهة مفكك الشفرة التالية، نضيف دالة `init_state` إضافية لتحويل إخراج المشفر (`enc_outputs`) إلى الحالة المشفرة encoded state. لاحظ أن هذه الخطوة قد تتطلب مدخلات إضافية، مثل الطول الصالح valid length للإدخال، والذي تم شرحهفي القسم 10.5. لإنشاء رمز تسلسل متغير الطول بواسطة الرمز token،في كل مرة قد تقوم مفكك الشفرة بتعيين إدخال (على سبيل المثال، الرمز الذي تم إنشاؤهفي الخطوة الزمنية السابقة) والحالة المشفرةفي رمز إخراجفي الخطوة الزمنية الحالية.

```
#@save
class Decoder(tf.keras.layers.Layer):
    """The base decoder interface for the encoder-
decoder architecture."""
    def __init__(self):
        super().__init__()

    # Later there can be additional arguments (e.g.,
length excluding padding)
    def init_state(self, enc_outputs, *args):
        raise NotImplementedError

    def call(self, X, state):
        raise NotImplementedError
```

10.6.3. وضع المشفر ومفكك الشفرة معًا Putting the Encoder and Decoder Together

في الانتشار الأمامي forward propagation، يتم استخدام خرج المشفر لإنتاج الحالة المشفرة، وسيتم استخدام هذه الحالة أيضًا بواسطة مفكك الشفرة كأحد مدخلاته.

```
#@save
class EncoderDecoder(d2l.Classifier):
    """The base class for the encoder-decoder
architecture."""
    def __init__(self, encoder, decoder):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder

    def call(self, enc_X, dec_X, *args):
        enc_outputs = self.encoder(enc_X, *args,
training=True)
        dec_state = self.decoder.init_state(enc_outputs,
*args)
        # Return decoder output only
        return self.decoder(dec_X, dec_state,
training=True)[0]
```

في القسم التالي، سنرى كيفية تطبيق RNNs لتصميم نماذج seq2seq بناءً على معمارية المشفر–مفكك الشفرة هذه.

10.6.4. الملخص

يمكن أن تتعامل معماريات المشفر–مفكك الشفرة مع المدخلات والمخرجات التي تتكون من متواليات متغيرة الطول وبالتالي فهي مناسبة لمشاكل seq2seq مثل الترجمة الآلية. يأخذ المشفر encoder تسلسلًا متغير الطول كمدخل ويحوله إلى حالة ذات شكل ثابت. يقوم مفكك الشفرة decoder بتعيين الحالة المشفرة لشكل ثابت إلى تسلسل متغير الطول.

10.6.5. التمارين

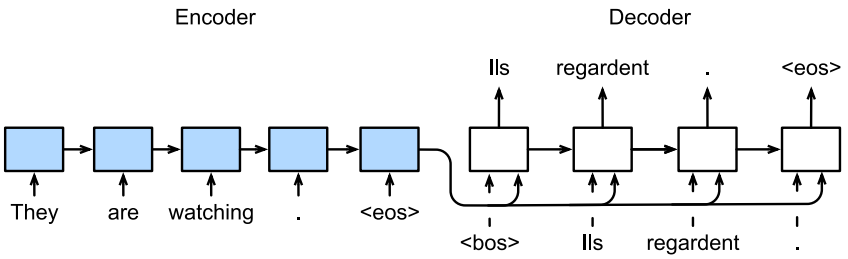
1. لنفترض أننا نستخدم الشبكات العصبية لتنفيذ معمارية المشفر–مفكك الشفرة. هل يجب أن يكون المشفر ومفكك الشفرة من نفس نوع الشبكة العصبية؟
2. إلى جانب الترجمة الآلية، هل يمكنك التفكيرفي تطبيق آخر حيث يمكن تطبيق معمارية الشفرة–مفكك الشفرة؟

10.7. المشفر-مفكك الشفرة Seq2Seq للترجمة الآلية Encoder-Decoder Seq2Seq for Machine Translation

فيما يسمى بالمشكلات seq2seq مثل الترجمة الآلية machine translation (كما تمت مناقشتهفي القسم 10.5)، حيث تتكون كل من المدخلات والمخرجات من متواليات (تسلسلات) متغيرة الطول غير محاذاة variable-length unaligned sequences، فإننا نعتمد عمومًا على معماريات المشفر_مفكك الشفرة encoder-decoder architecture (القسم 10.6).في هذا القسم، سوف نوضح تطبيق معمارية المشفر_مفكك الشفرة، حيث يتم تنفيذ كل من المشفر ومفكك الشفرة على أنهما RNNs، لمهمة الترجمة الآلية (Cho et al.، 2014، Sutskever et al.، 2014).

هنا، سيأخذ المشفر RNN تسلسلًا متغير الطول كمدخل ويحوله إلى حالة مخفية ذات شكل ثابت. لاحقًا،في القسم 11، سنقدم آليات الانتباه attention mechanisms، والتي تسمح لنا بالوصول إلى المدخلات المشفرة دون الحاجة إلى ضغط المدخلات بالكاملفي تمثيل واحد ذي طول ثابت.

بعد ذلك، لإنشاء تسلسل الإخراج، رمز واحدفي كل مرة، سيتنبأ نموذج وحدة مفكك الشفرة decoder model، الذي يتكون من RNN منفصل، بكل رمز مستهدف متتالي بالنظر إلى كل من تسلسل الإدخال والرموز السابقةفي المخرجات. أثناء التدريب، عادةً ما يكون مفكك الشفرة مشروطًا بالرموز السابقةفي التسمية الحقيقية "Ground-Truth" الرسمي. ومع ذلك،في وقت الاختبار، سنرغبفي تكييف كل إخراج من مفكك الشفرة على الرموز التي تم توقعها بالفعل. لاحظ أنه إذا تجاهلنا المشفر، فإن مفكك الشفرةفي بنية seq2seq تتصرف تمامًا مثل نموذج اللغة العادي. يوضح الشكل 10.7.1 كيفية استخدام RNNs تعلم التسلسل للتسلسلفي الترجمة الآلية.



الشكل 10.7.1 تعلم التسلسل للتسلسل باستخدام مشفر RNN ومفكك شفرة RNN.

في الشكل 10.7.1، يشير الرمز "<eos>" إلى نهاية التسلسل. يمكن أن يتوقف نموذجنا عن إجراء تنبؤات بمجرد إنشاء هذا الرمز.في الخطوة الزمنية الأولية لمفكك شفرة RNN، هناك قراران تصميم خاصان يجب أن تكون على دراية بهما: أولاً، نبدأ كل إدخال برمز لبداية التسلسل "<bos>". ثانيًا، قد نقوم بتغذية الحالة المخفية النهائية للمشفرفي مفكك الشفرةفي كل خطوة زمنية لمفكك الشفرة (Cho et al.، 2014).في بعض التصميمات الأخرى، مثل Sutskever et al. (2014)، تُستخدم الحالة المخفية النهائية لمشفر RNN لبدء الحالة المخفية لمفكك الشفرة فقطفي خطوة مفكك الشفرة الأولى.

10.7.1. إجبار المعلم Teacher Forcing

أثناء تشغيل المشفر على تسلسل الإدخال أمر بسيط نسبيًا، فإن كيفية التعامل مع مدخلات ومخرجات مفكك الشفرة تتطلب مزيدًا من العناية. يُطلق على النهج الأكثر شيوعًا أحيانًا اسم إجبار المعلم teacher forcing. هنا، يتم تغذية التسلسل الهدف الأصلي (تسميات الرمز token labels)في مفكك الشفرة كمدخل. بشكل أكثر تحديدًا، يتم تجميع رمز بداية التسلسل الخاص والتسلسل المستهدف الأصلي، باستثناء الرمز النهائي، كمدخل إلى مفكك الشفرة،في حين أن إخراج مفكك الشفرة (تسميات التدريب) هو التسلسل الهدف الأصلي، ويتم إزاحته بواسطة رمز واحد: "<bos>"، "Ils"، "regardent"، "."، "<eos>" (الشكل 10.7.1).

تطبيقنافي القسم 10.5.3 بيانات تدريب مُعدّة لإجبار المعلم، حيث يشبه تحويل الرموز للتعلم تحت الإشراف الذاتي تدريب النماذج اللغويةفي القسم 9.3. تتمثل الطريقة البديلةفي تغذية الرمز المتوقع من الخطوة الزمنية السابقة كمدخل حالي إلى مفكك الشفرة.

فيما يلي، نشرح التصميم الموضحفي الشكل 10.7.1 بمزيد من التفصيل. سنقوم بتدريب هذا النموذج للترجمة الآلية على مجموعة البيانات الإنجليزية–الفرنسية كما هو مقدمفي القسم 10.5.

```
import collections
import math
import tensorflow as tf
from d2l import tensorflow as d2l
```

10.7.2. المشفر Encoder

تذكر أن المشفر يحول تسلسل إدخال متغير الطول إلى متغير سياق ثابت الشكل (انظر الشكل 10.7.1).

ضعفي اعتبارك مثال تسلسل واحد (حجم الدُفعة 1). افترض أن تسلسل الإدخال هو $x_1, \ldots, x_T$ حيث x_t هو t^{th} الرمز.في الخطوة الزمنية t ، يقوم RNN بتحويل متجه ميزة الإدخال $\mathbf{x}_t$ لـ x_t

والحالة المخفية $\mathbf{h}_{t-1}$ من الخطوة الزمنية السابقة إلى الحالة المخفية الحالية $\mathbf{h}_t$. يمكننا استخدام دالة f للتعبير عن تحول الطبقة المتكررة لـ RNN:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}).$$

بشكل عام، يحول المشفر الحالات المخفيةفي جميع الخطوات الزمنية إلى متغير سياق من خلال دالة مخصصة q:

$$\mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T).$$

على سبيل المثال،في الشكل 10.7.1، متغير السياق هو فقط الحالة المخفية $\mathbf{h}_T$ المقابلة لتمثيل المشفر RNN بعد معالجة الرمز النهائي لتسلسل الإدخال.

في هذا المثال، استخدمنا RNN أحادي الاتجاه unidirectional لتصميم المشفر، حيث تعتمد الحالة المخفية فقط على المدخلات اللاحقةفي وقبل الخطوة الزمنية للحالة المخفية. يمكننا أيضًا إنشاء مشفرات باستخدام RNNs ثنائية الاتجاه bidirectional.في هذه الحالة، تعتمد الحالة المخفية على الخطوة اللاحقة قبل الخطوة الزمنية وبعدها (بمافي ذلك الإدخالفي الخطوة الزمنية الحالية)، والتي تشفر معلومات التسلسل بأكمله.

فلنبدأ الآنفي تنفيذ مشفر RNN. لاحظ أننا نستخدم طبقة التضمين embedding layer للحصول على متجه المعالم لكل رمزفي تسلسل الإدخال. وزن طبقة التضمين هو مصفوفة، حيث يتوافق عدد الصفوف مع حجم مفردات الإدخال (`vocab_size`) ويتوافق عدد الأعمدة مع أبعاد متجه المعالم (`embed_size`). بالنسبة لأي فهرس رمز إدخال i ، تجلب طبقة التضمين لـ i^{th} الصف (بدءًا من 0) من مصفوفة الوزن لإرجاع متجه الميزة الخاص بها. هنا نقوم بتنفيذ المشفر باستخدام GRU متعدد الطبقات.

```
class Seq2SeqEncoder(d2l.Encoder):  #@save
    """The RNN encoder for sequence to sequence
learning."""
    def __init__(self, vocab_size, embed_size,
num_hiddens, num_layers,
                 dropout=0):
        super().__init__()
        self.embedding =
tf.keras.layers.Embedding(vocab_size, embed_size)
        self.rnn = d2l.GRU(num_hiddens, num_layers,
dropout)

    def call(self, X, *args):
```

```
        # X shape: (batch_size, num_steps)
        embs = self.embedding(tf.transpose(X))
        # embs shape: (num_steps, batch_size,
embed_size)
        output, state = self.rnn(embs)
        # output shape: (num_steps, batch_size,
num_hiddens)
        # state shape: (num_layers, batch_size,
num_hiddens)
        return output, state
```

دعنا نستخدم مثالًا ملموسًا لتوضيح تنفيذ المشفر أعلاه. أدناه، نقوم بإنشاء مثيل لـ GRU مشفر من طبقتين عدد وحداته المخفية هو 16. بالنظر إلى الدفعات الصغيرة minibatch لمدخلات التسلسل X (حجم الدُفعة: 4، عدد خطوات الوقت: 9)، الحالات المخفية للطبقة الأخيرة طوال الوقت الخطوات (المخرجات التي يتم إرجاعها بواسطة الطبقات المتكررة للمشفر) هي موتر للشكل (عدد خطوات الوقت، حجم الدفعة، عدد الوحدات المخفية).

```
vocab_size, embed_size, num_hiddens, num_layers = 10, 8,
16, 2
batch_size, num_steps = 4, 9

encoder = Seq2SeqEncoder(vocab_size, embed_size,
num_hiddens, num_layers)
X = tf.zeros((batch_size, num_steps))
outputs, state = encoder(X)

d2l.check_shape(outputs, (num_steps, batch_size,
num_hiddens))
```

نظرًا لأننا نستخدم GRU هنا، فإن شكل الحالات المخفية متعددة الطبقاتفي الخطوة الزمنية النهائية هو (عدد الطبقات المخفية، حجم الدُفعة، عدد الوحدات المخفية).

10.7.3. مفكك الشفرة Decoder

بالنظر إلى تسلسل الإخراج المستهدف $y_1, y_2, \ldots, y_{T'}$ لكل خطوة زمنية t' (نستخدم t' للتمييز عن خطوات وقت تسلسل الإدخال)، يقوم مفكك الشفرة بتعيين احتمالية متوقعة لكل رمز ممكن يحدثفي الخطوة $y_{t'+1}$ المشروطة بالرموز السابقةفي الهدف $y_1, \ldots, y_{t'}$ ومتغير السياق $\mathbf{c}$ ، أي $P(y_{t'+1} \mid y_1, \ldots, y_{t'}, \mathbf{c})$.

للتنبؤ بالرمز اللاحق $t' + 1$في التسلسل المستهدف، يأخذ مفكك شفرة RNN الرمز الهدف للخطوة السابقة $y_{t'}$ ، وحالة RNN المخفية من الخطوة الزمنية السابقة $\mathbf{s}_{t'-1}$ ، ومتغير السياق $\mathbf{c}$ كمدخلات، ويحولهم إلى الحالة المخفية $\mathbf{s}_{t'}$في الخطوة الزمنية الحالية. يمكننا استخدام دالة g للتعبير عن تحول الطبقة المخفية لمفكك الشفرة:

$$\mathbf{s}_{t'} = g(y_{t'-1}, \mathbf{c}, \mathbf{s}_{t'-1}).$$

بعد الحصول على الحالة المخفية لمفكك الشفرة، يمكننا استخدام طبقة الإخراج وعملية softmaxلحساب التوزيع التنبئي $p(y_{t'+1} \mid y_1, \ldots, y_{t'}, \mathbf{c})$ على رمز الإخراج التالي $t' + 1$.

باتباع الشكل 10.7.1، عند تنفيذ مفكك الشفرة على النحو التالي، فإننا نستخدم الحالة المخفية مباشرةًفي الخطوة الزمنية النهائية للمشفر لتهيئة الحالة المخفية لمفكك الشفرة. يتطلب ذلك أن يكون لمشفر RNN ومفكك شفرة RNN نفس عدد الطبقات والوحدات المخفية. لمزيد من دمج معلومات تسلسل الإدخال المشفر، يكون متغير السياق متسلسلًا مع إدخال مفكك الشفرة في جميع خطوات الوقت. للتنبؤ بالتوزيع الاحتمالي للرمز الناتج، نستخدم طبقة متصلة بالكامل لتحويل الحالة المخفيةفي الطبقة الأخيرة من مفكك شفرة RNN.

```
class Seq2SeqDecoder(d2l.Decoder):
    """The RNN decoder for sequence to sequence
learning."""
    def __init__(self, vocab_size, embed_size,
num_hiddens, num_layers,
                 dropout=0):
        super().__init__()
        self.embedding =
tf.keras.layers.Embedding(vocab_size, embed_size)
        self.rnn = d2l.GRU(num_hiddens, num_layers,
dropout)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def init_state(self, enc_outputs, *args):
        return enc_outputs[1]

    def call(self, X, enc_state):
        # X shape: (batch_size, num_steps)
        # embs shape: (num_steps, batch_size,
embed_size)
        embs = self.embedding(tf.transpose(X))
```

```
        # context shape: (batch_size, num_hiddens)
        context = enc_state[-1]
        # Broadcast context to (num_steps, batch_size,
num_hiddens)
        context = tf.tile(tf.expand_dims(context, 0),
(embs.shape[0], 1, 1))
        # Concat at the feature dimension
        embs_and_context = tf.concat((embs, context), -
1)
        outputs, state = self.rnn(embs_and_context,
enc_state)
        outputs = tf.transpose(self.dense(outputs), (1,
0, 2))
        # outputs shape: (batch_size, num_steps,
vocab_size)
        # state shape: (num_layers, batch_size,
num_hiddens)
        return outputs, state
```

لتوضيح مفكك الشفرة المنفذ، أدناه نقوم بإنشاء مثيل لها باستخدام نفس المعلمات الفائقة من المشفر المذكورة أعلاه. كما نرى، يصبح شكل إخراج مفكك الشفرة (حجم الدُفعة، عدد الخطوات الزمنية، حجم المفردات)، حيث يخزن البعد الأخير للموتر توزيع الرمز المتوقع.

```
decoder = Seq2SeqDecoder(vocab_size, embed_size,
num_hiddens, num_layers)
state = decoder.init_state(encoder(X))
outputs, state = decoder(X, state)

d2l.check_shape(outputs, (batch_size, num_steps,
vocab_size))
d2l.check_len(state, num_layers)
d2l.check_shape(state[0], (batch_size, num_hiddens))
```

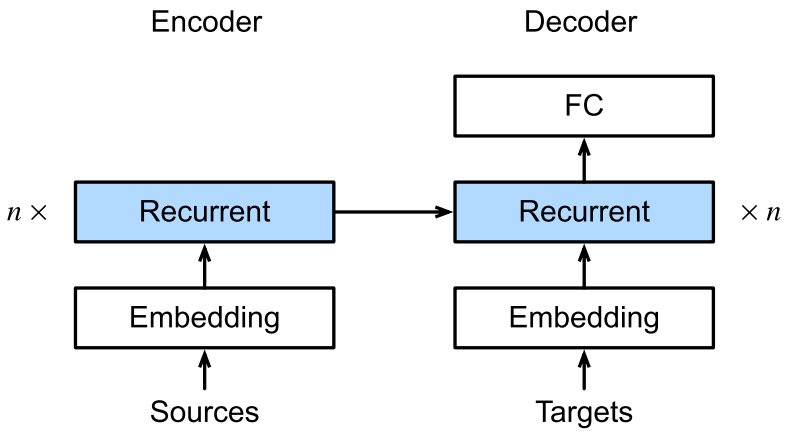
للتلخيص، يوضح الشكل 10.7.2 الطبقاتفي نموذج مشفر_مفكك شفرة RNN أعلاه.

10.7.4. المشفر-مفكك الشفرة لتعلم التسلسل للتسلسل Encoder-Decoder for Sequence to Sequence Learning

ينتج عن وضع كل ذلك معًافي التعليمات البرمجية ما يلي:

```
class Seq2Seq(d2l.EncoderDecoder):  #@save
    def __init__(self, encoder, decoder, tgt_pad, lr):
        super().__init__(encoder, decoder)
        self.save_hyperparameters()
```

```
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]),
train=False)

    def configure_optimizers(self):
        # Adam optimizer is used here
        return
tf.keras.optimizers.Adam(learning_rate=self.lr)
```



الشكل 10.7.2 طبقات في نموذج مشفر ـ مفكك شفرة RNN .

10.7.5. دالة الخطأ مع الاخفاء Loss Function with Masking

في كل خطوة زمنية، يتنبأ مفكك الشفرة بتوزيع احتمالي لرموز الإخراج. كما هو الحال مع نمذجة اللغة، يمكننا تطبيق softmax للحصول على التوزيع وحساب خطأ الانتروبيا المتقاطعة من أجل التحسين. تذكر القسم 10.5 أن رموز الحشو الخاصة يتم إلحاقها بنهاية التسلسلات بحيث يمكن تحميل التسلسلات ذات الأطوال المتفاوتة بكفاءةفي دفعات صغيرة من نفس الشكل. ومع ذلك، ينبغي استبعاد التنبؤ بالرموز للحشو من حسابات الخطأ. تحقيقا لهذه الغاية، يمكننا إخفاء mask المدخلات غير ذات الصلة بقيم صفرية بحيث يكون ضرب أي تنبؤ غير ذي صلة بصفر يساوي صفرًا.

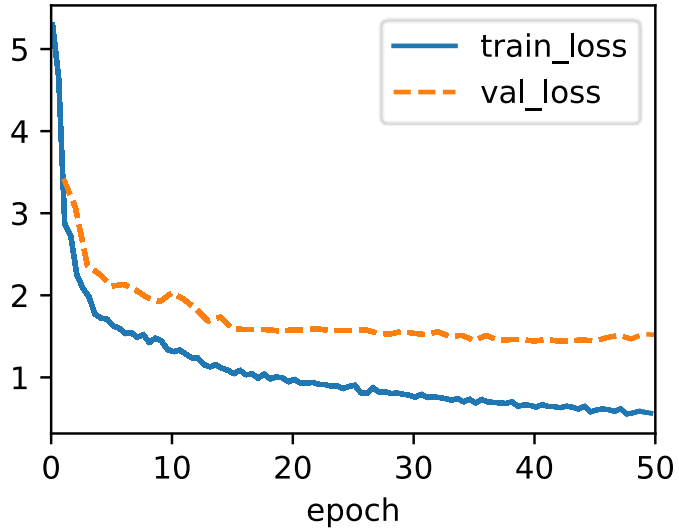
```
@d2l.add_to_class(Seq2Seq)
def loss(self, Y_hat, Y):
```

```
    l = super(Seq2Seq, self).loss(Y_hat, Y,
averaged=False)
    mask = tf.cast(tf.reshape(Y, -1) != self.tgt_pad,
tf.float32)
    return tf.reduce_sum(l * mask) / tf.reduce_sum(mask)
```

10.7.6. التدريب Training

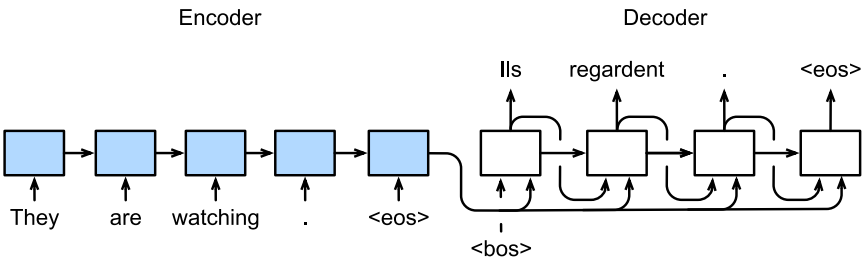
الآن يمكننا إنشاء وتدريب نموذج مشفر–مفكك شفرة RNN للتعلم التسلسل–التسلسل على مجموعة بيانات الترجمة الآلية.

```
data = d2l.MTFraEng(batch_size=128)
embed_size, num_hiddens, num_layers, dropout = 256, 256,
2, 0.2
with d2l.try_gpu():
    encoder = Seq2SeqEncoder(
        len(data.src_vocab), embed_size, num_hiddens,
num_layers, dropout)
    decoder = Seq2SeqDecoder(
        len(data.tgt_vocab), embed_size, num_hiddens,
num_layers, dropout)
    model = Seq2Seq(encoder, decoder,
tgt_pad=data.tgt_vocab['<pad>'],
                    lr=0.001)
trainer = d2l.Trainer(max_epochs=50,
gradient_clip_val=1)
trainer.fit(model, data)
```



10.7.7. التنبؤ Prediction

للتنبؤ بتسلسل الإخراج في كل خطوة، يتم إدخال الرمز المتوقع من الخطوة الزمنية السابقة في مفكك الشفرة كمدخل. تتمثل إحدى الإستراتيجيات البسيطة في أخذ عينة من أي رمز حدده مفكك الشفرة لأعلى احتمال عند التنبؤ في كل خطوة. كما هو الحال في التدريب، في الخطوة الزمنية الأولية، يتم تغذية رمز بداية التسلسل ("<bos>") في مفكك الشفرة. عملية التنبؤ هذه موضحة في الشكل 10.7.3. عندما يتم توقع رمز نهاية التسلسل("<eos>") ، يكون تنبؤ تسلسل الإخراج قد اكتمل.



الشكل 10.7.3 توقع رمز تسلسل الخرج برمز باستخدام مشفر ـ مفكك شفرة RNN.

في القسم التالي، سوف نقدم استراتيجيات أكثر تعقيداً بناءً على البحث الشعاعي beam search (القسم 10.8).

```
@d2l.add_to_class(d2l.EncoderDecoder)  #@save
def predict_step(self, batch, device, num_steps,
                 save_attention_weights=False):
    src, tgt, src_valid_len, _ = batch
    enc_outputs = self.encoder(src, src_valid_len,
training=False)
    dec_state = self.decoder.init_state(enc_outputs,
src_valid_len)
    outputs, attention_weights =
[tf.expand_dims(tgt[:,0], 1), ], []
    for _ in range(num_steps):
        Y, dec_state = self.decoder(outputs[-1],
dec_state, training=False)
        outputs.append(tf.argmax(Y, 2))
        # Save attention weights (to be covered later)
        if save_attention_weights:

attention_weights.append(self.decoder.attention_weights)
    return tf.concat(outputs[1:], 1), attention_weights
```

10.7.8. تقييم المتسلسلات المتوقعة Evaluation of Predicted Sequences

يمكننا تقييم تسلسل متوقع من خلال مقارنته بالتسلسل المستهدف (الحقيقة). ولكن ما هو بالضبط المقياس المناسب لمقارنة التشابه بين تسلسلين؟

BLEU (دراسة التقييم ثنائي اللغة Bilingual Evaluation Understudy) ، على الرغم من اقتراحهافي الأصل لتقييم نتائج الترجمة الآلية (Papineni et al.، 2002) ، فقد تم استخدامها على نطاق واسعفي قياس جودة تسلسل المخرجات للتطبيقات المختلفة. من حيث المبدأ، بالنسبة لأي n ـ Gramsفي التسلسل المتوقع، تقوم BLEU بتقييم ما إذاكانت هذه n ـ Grams تظهر في التسلسل المستهدف.

p_n تدل على دقة n ـ grams، وهي نسبة عدد n ـ Grams المتطابقةفي التسلسل المتوقع والهدف إلى عدد n ـ gramsفي التسلسل المتوقع. لشرح، بالنظر إلى التسلسل المستهدف A ، B، C ، D ، E ، F والتسلسل المتوقع،A ، B ، D ، لدينا، $p_1 = 4/5$ ، $p_2 = 3/4$ ، $p_3 = 1/3$ و $p_4 = 0$. إلى جانب ذلك، ليكن $\text{len}_{\text{label}}$ وlen_{pred} أن يكون عدد الرموزفي التسلسل المستهدف والتسلسل المتوقع، على التوالي. بعد ذلك، يتم تعريف BLEU على أنه

$$\exp\left(min\left(0,1-\frac{\text{len}_{\text{label}}}{\text{len}_{\text{pred}}}\right)\right)\prod_{n=1}^{k} p_n^{1/2^n},$$

حيث k هي أطول n ـ غرام للمطابقة.

استنادًا إلى تعريف BLEUفي (10.7.4)، كلما كان التسلسل المتوقع هو نفسه التسلسل المستهدف، يكون BLEU هو 1. علاوة على ذلك، نظرًا لأن مطابقة الغرام الأطول أكثر صعوبة، فإن BLEU تعين وزناً أكبر لدقة الغرام الأطول. على وجه التحديد، عندما يتم إصلاح p_n ، $p_n^{1/2^n}$ يزداد مع نمو n (يستخدم الورق الأصلي $p_n^{1/n}$). علاوة على ذلك، نظرًا لأن التنبؤ بالتسلسلات الأقصر يميل إلى الحصول على قيمة p_n أعلى، فإن المعامل قبل حد الضربفي (10.7.4) يعاقب التسلسلات الأقصر المتوقعة. على سبيل المثال، عندما $k = 2$ ، بالنظر إلى التسلسل المستهدف A ، B، C ، D ، E ، F والتسلسل المتوقع A ، B على الرغم من أن عامل الجزاء $\exp(1 - 6/2) \approx 0.14$ اقل من BLEU.

نقوم بتنفيذ مقياس BLEU على النحو التالي.

```
def bleu(pred_seq, label_seq, k):  #@save
    """Compute the BLEU."""
```

```
    pred_tokens, label_tokens = pred_seq.split(' '),
label_seq.split(' ')
    len_pred, len_label = len(pred_tokens),
len(label_tokens)
    score = math.exp(min(0, 1 - len_label / len_pred))
    for n in range(1, min(k, len_pred) + 1):
        num_matches, label_subs = 0,
collections.defaultdict(int)
        for i in range(len_label - n + 1):
            label_subs[' '.join(label_tokens[i: i + n])]
+= 1
        for i in range(len_pred - n + 1):
            if label_subs[' '.join(pred_tokens[i: i +
n])] > 0:
                num_matches += 1
                label_subs[' '.join(pred_tokens[i: i +
n])] -= 1
        score *= math.pow(num_matches / (len_pred - n +
1), math.pow(0.5, n))
    return score
```

في النهاية، نستخدم مشفر–مفكك شفرة RNN المدربة لترجمة بعض الجمل الإنجليزية إلى الفرنسية وحساب BLEU للنتائج.

```
engs = ['go .', 'i lost .', 'he\'s calm .', 'i\'m home
.']
fras = ['va !', 'j\'ai perdu .', 'il est calme .', 'je
suis chez moi .']
preds, _ = model.predict_step(
    data.build(engs, fras), d2l.try_gpu(),
data.num_steps)
for en, fr, p in zip(engs, fras, preds):
    translation = []
    for token in data.tgt_vocab.to_tokens(p):
        if token == '<eos>':
            break
        translation.append(token)
    print(f'{en} => {translation}, bleu,'
          f'{bleu(" ".join(translation), fr, k=2):.3f}')
```

```
go . => ['<unk>', '!'], bleu,0.000
i lost . => ["j'ai", '<unk>', '.'], bleu,0.000
he's calm . => ['<unk>', 'tom', '.'], bleu,0.000
```

```
i'm home . => ['je', 'suis', 'suis', '<unk>', '<unk>',
'gentil', 'gentil', 'gentil', 'aille'], bleu,0.280
```

10.7.9. الملخص

بعد تصميم معمارية المشفر–مفكك الشفرة encoder-decoder architecture، يمكننا استخدام اثنين من RNN لتصميم نموذج تعلم التسلسل للمتسلسل sequence to sequence learning.في تدريب المشفر–مفكك الشفرة، يقوم أسلوب إجبار المعلم teacher forcing بتغذية تسلسلات الإخراج الأصلية (على عكس التنبؤات)في مفكك الشفرة. عند تنفيذ المشفر ومفكك الشفرة، يمكننا استخدام RNNs متعددة الطبقات. يمكننا استخدام الأقنعة masks لتصفية الحسابات غير ذات الصلة، كما هو الحال عند حساب الخطأ. بالنسبة لتقييم تسلسل المخرجات، فإن BLEU هو مقياس شائع عن طريق مطابقة n ـ جرام بين التسلسل المتوقع والتسلسل المستهدف.

10.7.10. التمارين

1. هل يمكنك ضبط المعلمات الفائقة لتحسين نتائج الترجمة؟
2. أعد تشغيل التجربة بدون استخدام الأقنعةفي حساب الخطأ. ما هي النتائج التي تلاحظها؟ لماذا؟
3. إذا اختلف المشفر encoder ومفكك الشفرة decoderفي عدد الطبقات أو عدد الوحدات المخفية، فكيف يمكننا تهيئة الحالة المخفية لمفكك الشفرة؟
4. في التدريب، استبدل إجبار المعلم بتغذية التنبؤفي الخطوة الزمنية السابقةفي مفكك الشفرة. كيف يؤثر هذا على الأداء؟
5. أعد تشغيل التجربة عن طريق استبدال GRU بـ LSTM.
6. هل هناك أي طرق أخرى لتصميم طبقة الإخراج لمفكك الشفرة؟

10.8. البحث الشعاعي Beam Search

في القسم 10.7، قدمنا معمارية المشفر–مفكك الشفرة، والتقنيات القياسية لتدريبهم من طرف إلى طرف. ومع ذلك، عندما يتعلق الأمر بالتنبؤ بوقت الاختبار، فقد ذكرنا فقط استراتيجية الجشع greedy strategy، حيث نختارفي كل خطوة الرمز نظرًا لأعلى احتمالية متوقعة للظهور التالي، حتى،في وقت ما، نجد أننا توقعنا رمز مميز لنهاية التسلسل "<eos>".في هذا القسم، سنبدأ بإضفاء الطابع الرسمي على استراتيجية البحث الجشع Greedy Search هذه وتحديد بعض المشكلات التي يميل الممارسون إلى مواجهتها. بعد ذلك، قمنا بمقارنة هذه الإستراتيجية مع بديلين: بحث شامل exhaustive search (توضيحي ولكن ليس عمليًا) وبحث شعاعي beam search (الطريقة القياسيةفي الممارسة).

لنبدأ بإعداد التدوين الرياضي mathematical notation، واستعارة الاصطلاحات من القسم 10.7.في أي خطوة زمنية t' ، يُخرج مفكك الشفرة تنبؤات تمثل احتمالية كل رمزفي المفردات القادمةفي التسلسل (القيمة المحتملة لـ $y_{t'+1}$ ، المشروطة بالرموز السابقة $y_1, \ldots, y_{t'}$ ومتغير السياق $\mathbf{c}$ ، التي ينتجها المشفر لتمثيل تسلسل الإدخال. تحديد التكلفة الحسابية، يتم الإشارة إليها بواسطة $\mathcal{Y}$ مفردات الإخراج (بمافي ذلك الرمز الخاص بنهاية التسلسل "<eos>"). دعنا أيضًا نحدد الحد الأقصى لعدد الرموز لتسلسل المخرجات كما هو T'. هدفنا هو البحث عن مخرجات مثالية من جميع $\mathcal{O}(|\mathcal{Y}|^{T'})$ تسلسلات الإخراج الممكنة. لاحظ أن هذا يبالغ قليلاًفي تقدير عدد المخرجات المميزة لأنه لا توجد رموز لاحقة بعد حدوث المميز "<eos>". ومع ذلك، لأغراضنا، يلتقط هذا الرقم حجم مساحة البحث تقريبًا.

10.8.1. البحث الجشع Greedy Search

ضعفي اعتبارك استراتيجية البحث الجشع Greedy Search البسيطة من القسم 10.7. هنا،في أي خطوة زمنية t' ، نختار ببساطة الرمز بأعلى احتمالية شرطية من $\mathcal{Y}$ ، على سبيل المثال،

$$y_{t'} = \operatorname*{argmax}_{y \in \mathcal{Y}} P(y \mid y_1, \ldots, y_{t'-1}, \mathbf{c}).$$

بمجرد أن يخرج نموذجنا "<eos>" (أو نصل إلى الحد الأقصى للطول T')، يكتمل تسلسل الإخراج.

قد تبدو هذه الإستراتيجية معقولة، وهيفي الحقيقة ليست سيئة للغاية! بالنظر إلى مدى تساهله من الناحية الحسابية، ستتعرض لضغوط شديدة للحصول على المزيد من الفوائد مقابل أموالك. ومع ذلك، إذا وضعنا الكفاءة جانباً لمدة دقيقة، فقد يبدو من المعقول أكثر البحث عن التسلسل الأكثر احتمالاً، وليس تسلسل الرموز (المختارة بطمع) على الأرجح. اتضح أن هذين الكائنين يمكن أن يكونا مختلفين تمامًا. التسلسل الأكثر احتمالا هو الذي يقوم بتكبير التعبير $\prod_{t'=1}^{T'} P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c})$.في مثال الترجمة الآلية لدينا، إذا استعاد مفكك الشفرة حقًا احتمالات العملية التوليدية الأساسية، فإن هذا سيعطينا الترجمة الأكثر احتمالية. لسوء الحظ، ليس هناك ما يضمن أن البحث الجشع سوف يعطينا هذا التسلسل.

دعونا نوضح ذلك بمثال. افترض أن هناك أربعة رموز مميزة "A" و "B" و "C" و "<eos>"في قاموس الإخراج.في الشكل 10.8.1، تمثل الأرقام الأربعة تحت كل خطوة زمنية الاحتمالات الشرطية لتوليد "A" و "B" و "C" و "<eos>"في تلك الخطوة الزمنية، على التوالي.

Time step	1	2	3	4
A	0.5	0.1	0.2	0.0
B	0.2	0.4	0.2	0.2
C	0.2	0.3	0.4	0.2
<eos>	0.1	0.2	0.2	0.6

الشكل 10.8.1في كل خطوة زمنية، يقوم البحث الجشع بتحديد الرمز أعلى احتمالية شرطية.

في كل خطوة زمنية، يقوم البحث الجشع بتحديد الرمز بأعلى احتمالية شرطية. لذلك، سيتم توقع تسلسل الخرج "A" و "B" و "C" و "<eos>" (الشكل 10.8.1). الاحتمال الشرطي لتسلسل الإخراج هذا هو $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$.

بعد ذلك، دعونا نلقي نظرة على مثال آخرفي الشكل 10.8.2. على عكس الشكل 10.8.1،في الخطوة الزمنية 2 نختار الرمز المميز "C"في الشكل 10.8.2، والذي له ثاني أعلى احتمال شرطي.

Time step	1	2	3	4
A	0.5	0.1	0.1	0.1
B	0.2	0.4	0.6	0.2
C	0.2	0.3	0.2	0.1
<eos>	0.1	0.2	0.1	0.6

الشكل 10.8.2 تمثل الأرقام الأربعة الموجودة أسفل كل خطوة زمنية الاحتمالات الشرطية لتوليد "A" و "B" و "C" و "<eos>"في تلك الخطوة الزمنية.في الخطوة الزمنية 2، يتم تحديد الرمز المميز "C"، الذي يحتوي على ثاني أعلى احتمالية شرطية.

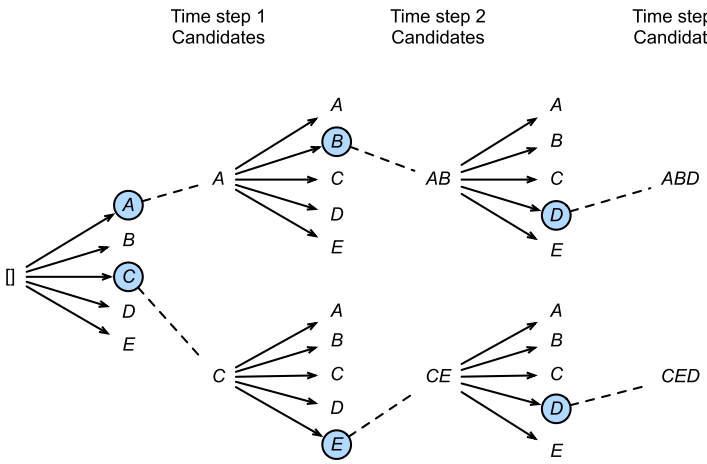
نظرًا لأن المخرجات اللاحقةفي الخطوتين 1 و2، التي تستند إليها الخطوة الزمنية 3، قد تغيرت من "A" و "B"في الشكل 10.8.1. إلى "A" و "C"في الشكل 10.8.2، كما تغير الاحتمال الشرطي لكل رمزفي الخطوة الزمنية 3في الشكل 10.8.2. لنفترض أننا اخترنا الرمز "B"في الوقت الخطوة 3. الآن الخطوة الزمنية 4 مشروطة بالإخراج اللاحقفي الخطوات الثلاث الأولى "A" و "C" و "B"، والتي تختلف عن "A" و "B "و "C"في الشكل 10.8.1. لذلك، فإن الاحتمال الشرطي لتوليد كل رمزفي الخطوة الزمنية 4في الشكل 10.8.2 يختلف أيضًا عن ذلكفي الشكل 10.8.1. ونتيجة لذلك، فإن الاحتمال الشرطي لتسلسل المخرجات "A" و "C" و "B" و "<eos>"في الشكل 10.8.2 $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$, أكبر من احتمال البحث الجشعفي الشكل I

10.8. 1.في هذا المثال، تسلسل المخرجات "A" و "B" و "C" و "<eos>" الذي تم الحصول عليه عن طريق البحث الجشع ليس هو التسلسل الأمثل.

10.8.2. البحث الشامل Exhaustive Search

إذا كان الهدف هو الحصول على التسلسل الأكثر احتمالًا، فقد نفكرفي استخدام البحث الشامل exhaustive search: قم بتعداد شامل لجميع تسلسلات المخرجات المحتملة مع احتمالاتها الشرطية، ثم إخراج التسلسل الذي يسجل أعلى احتمالية متوقعة.

في حين أن هذا سيعطينا بالتأكيد ما نرغب فيه، إلا أنه سيأتي بتكلفة حسابية باهظة $\mathcal{O}(|\mathcal{Y}|^{T'})$ ، وأسيةفي طول التسلسل وقاعدة هائلة من حجم المفردات. على سبيل المثال، عندما $|\mathcal{Y}| = 10000$ و $T' = 10$ ، سنحتاج إلى تقييم المتتاليات $10000^{10} = 10^{40}$. هذه أرقام صغيرة مقارنة بالتطبيقات الحقيقية ولكنها بالفعل تتجاوز قدرات أي أجهزة كمبيوتر متوقعة. من ناحية أخرى، فإن التكلفة الحسابية للبحث الجشع هي $\mathcal{O}(|\mathcal{Y}|T')$: رخيصة بأعجوبة ولكنها بعيدة عن المثالية. على سبيل المثال، عندما $|\mathcal{Y}| = 10000$ و$T' = 10$ ، نحتاج فقط إلى تقييم المتتاليات $10000 \times 10 = 10^5$.



الشكل 10.8.3 عملية البحث الشعاعي (حجم الحزمة: 2، الحد الأقصى لطول تسلسل الخرج: 3). تسلسلات الإخراج المرشح هي A وC و AB و CE وABD و CED.

10.8.3. البحث الشعاعي Beam Search

يمكنك عرض إستراتيجيات فك تشفير التسلسل على أنها تقع على طيف spectrum، حيث يقدم البحث الشعاعي Beam Search حلاً وسطًا بين كفاءة البحث الجشع ومثالية البحث الشامل. تتميز النسخة الأكثر وضوحًا من البحث الشعاعي بمعامل فائق واحد، حجم الشعاع

beam size، k. في الخطوة الزمنية 1، نختار الرموز بأعلى الاحتمالات المتوقعة. سيكون كل واحد منهم الرمز الأول لتسلسل الإخراج المرشح، على التوالي.في كل خطوة زمنية لاحقة، بناءً على تسلسل الإخراج المرشح kفي الخطوة الزمنية السابقة، نواصل تحديد تسلسلات إخراج المرشح k ذات أعلى الاحتمالات المتوقعة من الخيارات الممكنة.

يوضح الشكل 10.8.3 عملية البحث الشعاعي مع مثال. لنفترض أن مفردات الإخراج تحتوي على خمسة عناصر فقط: $\mathcal{Y} = \{A, B, C, D, E\}$ أحدها "<eos>". اجعل حجم الشعاع 2 ويكون الحد الأقصى لطول تسلسل الإخراج 3.في الخطوة الزمنية 1، افترض أن الرموز ذات الاحتمالات الشرطية الأعلى $P(y_1 \mid \mathbf{c})$ هي A و C.في الخطوة الزمنية 2، لكل $y_2 \in \mathcal{Y}$، نحسب:

$$P(A, y_2 \mid \mathbf{c}) = P(A \mid \mathbf{c})P(y_2 \mid A, \mathbf{c}),$$
$$P(C, y_2 \mid \mathbf{c}) = P(C \mid \mathbf{c})P(y_2 \mid C, \mathbf{c}),$$

واختر أكبر قيمتين من بين هذه القيم العشر، على سبيل المثال $P(A, B \mid \mathbf{c})$ و$P(C, E \mid \mathbf{c})$. ثمفي الخطوة الزمنية 3، لكل $y_3 \in \mathcal{Y}$ نقوم بحساب:

$$P(A, B, y_3 \mid \mathbf{c}) = P(A, B \mid \mathbf{c})P(y_3 \mid A, B, \mathbf{c}),$$
$$P(C, E, y_3 \mid \mathbf{c}) = P(C, E \mid \mathbf{c})P(y_3 \mid C, E, \mathbf{c}),$$

واختر أكبر قيمتين من بين هذه القيم العشر، على سبيل المثال $P(A, B, D \mid \mathbf{c})$ و$P(C, E, D \mid \mathbf{c})$. نتيجة لذلك، نحصل على ست متواليات إخراج مرشحة:

(i) A; (ii) C; (iii) A, B; (iv) C, E; (v) A, B, D; and (vi) C, E, D.

في النهاية، نحصل على مجموعة تسلسلات الإخراج المرشح النهائية بناءً على هذه التسلسلات الستة (على سبيل المثال، تجاهل الأجزاء بمافي ذلك وبعد "<eos>"). ثم نختار التسلسل ذي الدرجة الأعلى من الدرجة التالية كتسلسل الإخراج:

$$\frac{1}{L^\alpha} \log P(y_1, \ldots, y_L \mid \mathbf{c}) = \frac{1}{L^\alpha} \sum_{t'=1}^{L} \log P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c}),$$

حيث L هو طول تسلسل المرشح النهائي وعادة ما يتم تعيينه على 0.75. نظرًا لأن التسلسل الأطول يحتوي على مصطلحات لوغاريتمية أكثرفي جمع (10.8.4)، فإن المصطلح L^αفي المقام يعاقب التسلسلات الطويلة.

التكلفة الحسابية للبحث الشعاعي هي $\mathcal{O}(k|\mathcal{Y}|T')$. هذه النتيجة تقع بين نتيجة البحث الجشع ونتائج البحث الشامل. يمكن التعامل مع البحث الجشع كحالة خاصة للبحث الشعاعي التي تنشأ عند ضبط حجم الشعاع beam size على 1.

10.8.4. الملخص

تتضمن استراتيجيات البحث المتسلسل البحث الجشع والبحث الشامل والبحث الشعاعي. يوفر البحث الشعاعي مفاضلة بين الدقة مقابل التكلفة الحسابية من خلال اختياره المرن لحجم الشعاع.

10.8.5. التمارين

1. هل يمكننا التعامل مع البحث الشامل كنوع خاص من البحث الشعاعي؟ لما ولما لا؟
2. قم بتطبيق البحث الشعاعيفي مشكلة الترجمة الآليةفي القسم 10.7. كيف يؤثر حجم الشعاع على نتائج الترجمة وسرعة التنبؤ؟
3. استخدمنا نمذجة اللغة لإنشاء نص يتبع البادئات التي قدمها المستخدم user-provided prefixesفي القسم 9.5. ما نوع استراتيجية البحث التي تستخدمها؟ هل يمكنك تحسينه؟

11

آليات الانتباه والمحولات

11. آليات الانتباه والمحولات Attention Mechanisms and Transformers

يتلقى العصب البصري للنظام البصري الرئيسي مدخلات حسية هائلة، تتجاوز بكثير ما يمكن للدماغ معالجته بشكل كامل. لحسن الحظ، ليست كل المحفزات متساوية. مكّن التركيز البؤري للوعي وتركيزه الرئيسي من توجيه الانتباه إلى الأشياء محل الاهتمام، مثل الفرائس والحيوانات المفترسة،في البيئة المرئية المعقدة. إن القدرة على الانتباه إلى جزء صغير فقط من المعلومات لها أهمية تطورية، مما يسمح للبشر بالعيش والنجاح.

كان العلماء يدرسون الانتباه attentionفي مجال علم الأعصاب الإدراكي cognitive neuroscience منذ القرن التاسع عشر.في هذا الفصل، سنبدأ بمراجعة إطار عمل شائع لشرح كيفية نشر الانتباهفي المشهد المرئي. مستوحاة من إشارات الانتباه attention cuesفي هذا الإطار، سنصمم نماذج تستفيد من إشارات الانتباه هذه. والجدير بالذكر أن انحدار نواة Nadaraya-Watsonفي عام 1964 هو عرض بسيط للتعلم الآلي مع آليات الانتباه attention mechanisms. بعد ذلك، سوف نقدم دوال الانتباه التي تم استخدامها على نطاق واسعفي تصميم نماذج الانتباهفي التعلم العميق. على وجه التحديد، سوف نوضح كيفية استخدام هذه الدوال لتصميم انتباه باهدانو Bahdanau attention، وهو نموذج اهتمام رائدفي التعلم العميق يمكن أن يتماشى بشكل ثنائي الاتجاه وقابل للتفاضل.

مجهزة بأحدث تصميمات الانتباه متعدد الرؤوس multi-head attention والانتباه الذاتي self-attention، تعتمد بُنية المحولات فقط على آليات الانتباه attention mechanisms. سننتقل إلى وصف التصميم الأصلي المشفر-مفكك الشفرة للترجمة الآلية. ثم سنبين كيف يمكن المشفر الخاص به تمثيل الصور، مما يؤدي إلى تطوير محولات الرؤية vision transformers. عند تدريب نماذج كبيرة جداً على مجموعات بيانات كبيرة جداً (على سبيل المثال، 300 مليون صورة)، تتفوق محولات الرؤية على ResNets بشكل كبيرفي تصنيف الصور، مما يدل على قابلية التوسع الفائقة للمحولات. وبالتالي، تم استخدام المحولات على نطاق واسعفي التدريب المسبق على نطاق واسع، والتي يمكن تكييفها لأداء مهام مختلفة مع تحديث النموذج (على سبيل المثال، الضبط الدقيق fine tuning) أو لا (على سبيل المثال، عدد قليل من اللقطات few shot).في النهاية، سنراجع كيفية التصفية المسبقة للمحولات كمشفرات فقط encoder-only (على سبيل المثال، BERT)، ومشفر-مفكك الشفرة encoder-decoder (على سبيل المثال، T5)، ومفكك الشفرة فقط decoder-only (على سبيل المثال، سلسلة GPT). يشير النجاح المقنع للتدريب المسبق على نطاق واسع باستخدام

المحولاتفي مجالات متنوعة مثل اللغة والرؤية والكلام والتعلم المعزز إلى أن الأداء الأفضل يستفيد من النماذج الأكبر، والمزيد من بيانات التدريب، والمزيد من حوسبة التدريب.

11.1. إشارات الانتباه Attention Cues

شكرا لك على اهتمامك (انتباهك) بهذا الكتاب. الاهتمام مورد نادر:في الوقت الحالي تقرأ هذا الكتاب وتتجاهل الباقي. وبالتالي، على غرار المال، يتم دفع انتباهك بتكلفة الفرصة البديلة. للتأكد من أن استثمار اهتمامك الآن يستحق العناء، فقد تحمسنا بشدة لإيلاء اهتمامنا بعناية لإنتاج كتاب لطيف. الانتباه (الاهتمام) هو حجر الأساسفي قوس الحياة ويمسك بمفتاح التميز لأي عمل.

بما أن الاقتصاد يدرس تخصيص الموارد النادرة، فنحنفي عصر اقتصاد الانتباه attention economy، حيث يتم التعامل مع الاهتمام البشري على أنه سلعة محدودة وقيمة ونادرة يمكن تبادلها. تم تطوير العديد من نماذج الأعمال للاستفادة منها.في الموسيقى أو خدمات بث الفيديو، إما أن نولي اهتمامًا لإعلاناتهم أو ندفع المال لإخفائها. للنموفي عالم الألعاب عبر الإنترنت، إما أن نولي اهتمامًا للمشاركةفي المعارك، التي تجذب لاعبين جددًا، أو ندفع المال لنصبح أقوياء على الفور. لا شيء يأتي بالمجان.

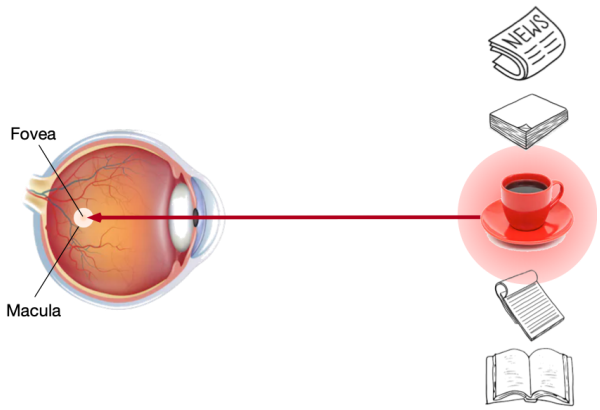
بشكل عام، المعلوماتفي بيئتنا ليست نادرة، الانتباه هو. عند فحص مشهد مرئي، يتلقى العصب البصري معلومات بترتيب 10^8 بتاتفي الثانية، وهو ما يتجاوز بكثير ما يمكن لدماغنا معالجته بشكل كامل. لحسن الحظ، تعلم أسلافنا من التجربة (المعروفة أيضًا باسم البيانات) أنه ليست كل المدخلات الحسية متساوية. على مدار تاريخ البشرية، مكنت القدرة على توجيه الانتباه إلى جزء بسيط من المعلومات ذات الأهمية عقولنا من تخصيص الموارد بشكل أكثر ذكاءً للبقاء والنمو والتواصل الاجتماعي، مثل اكتشاف الحيوانات المفترسة والفرائس والرفاق.

11.1.1. إشارات الانتباه في علم الأحياء Attention Cues in Biology

لشرح كيفية نشر انتباهنافي العالم المرئي، ظهر إطار عمل مكون من عنصرين وانتشر. تعود هذه الفكرة إلى ويليام جيمسفي تسعينيات القرن التاسع عشر، والذي يُعتبر "أب علم النفس الأمريكي" (جيمس، 2007).في هذا الإطار، يقوم الأشخاص بشكل انتقائي بتوجيه بؤرة الانتباه باستخدام كل من الإشارات اللاإرادية nonvolitional cue والإشارة الإرادية volitional cue.

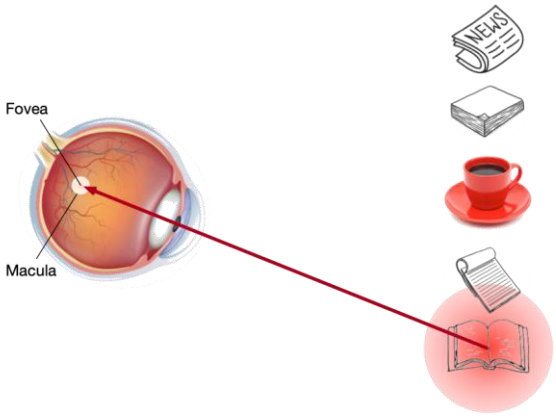
تعتمد الاشارة اللاإرادية nonvolitional cue على بروز العناصرفي البيئة ووضوحها. تخيل أن هناك خمسة أشياء أمامك: جريدة وورقة بحث وفنجان قهوة ودفتر ملاحظات وكتاب كمافي الشكل 11.1.1. بينما تتم طباعة جميع المنتجات الورقية باللونين الأسود والأبيض، فإن فنجان القهوة باللون الأحمر. بعبارة أخرى، هذه القهوة بارزة بشكل جوهري وواضحةفي هذه البيئة

المرئية، وتلفت الانتباه تلقائيًا ولا إراديًا. لذلك تقوم بإحضار النقرة المركزية fovea (مركز البقعة حيث تكون حدة البصر أعلى) على القهوة كما هو موضح في الشكل 11.1.1.



الشكل 11.1.1 باستخدام الإشارات اللاإرادية القائمة على النقطة المحفزة saliency (كوب أحمر، غير ورقي)، يتم توجيه الانتباه بشكل لا إرادي إلى القهوة.

بعد شرب القهوة، تصبح محتويًا على الكافيين وترغب في قراءة كتاب. لذلك تدير رأسك، وتعيد تركيز عينيك، وتنظر إلى الكتاب كما هو موضح في الشكل 11.1.2. يختلف عن الحالة في الشكل 11.1.1 حيث تحيزك القهوة نحو الاختيار بناءً على النقطة المحفزة saliency، في هذه الحالة التي تعتمد على المهمة، يمكنك تحديد الكتاب تحت التحكم المعرفي والإرادي. باستخدام الاشارة الإرادية volitional cue بناءً على معايير الاختيار المتغيرة، يكون هذا النوع من الانتباه أكثر تعمقًا. كما أنها أكثر قوة مع الجهد التطوعي للموضوع.



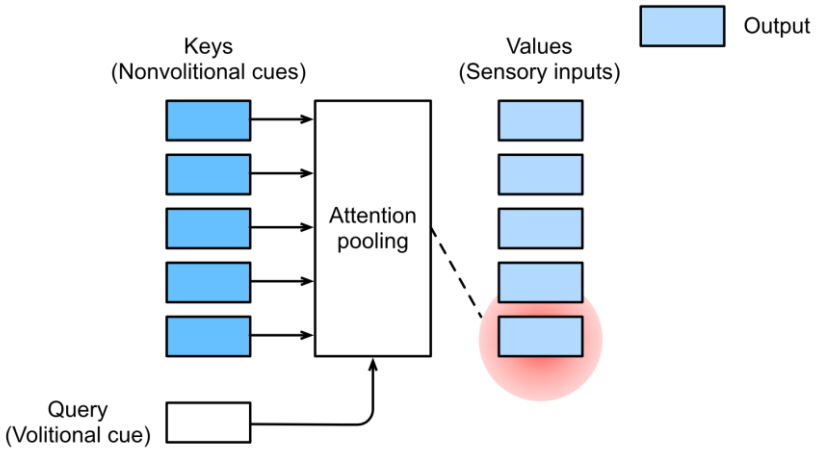
الشكل 11.1.2 باستخدام الاشارة الإرادية (الرغبة في قراءة كتاب) الذي يعتمد على المهمة، يتم توجيه الانتباه إلى الكتاب تحت السيطرة الإرادية.

11.1.2. الاستعلامات والمفاتيح والقيم Queries, Keys, and Values

مستوحاة من إشارات الانتباه اللاإرادي والإرادي التي تشرح النشر الانتباهي attentional deployment، فيما يلي سنصف إطار عمل لتصميم آليات الانتباه من خلال دمج إشارات الانتباه هاتين.

بادئ ذي بدء، ضعفي اعتبارك الحالة الأبسط حيث لا تتوفر سوى الإشارات غير المنطقية. لتحيز التحديد على المدخلات الحسية، يمكننا ببساطة استخدام طبقة متصلة بالكامل ذات معلمات أو حتى تجميع أقصى أو متوسط بدون معلمات.

لذلك، فإن ما يميز آليات الانتباه عن تلك الطبقات أو طبقات التجميع المتصلة بالكامل هو تضمين الإشارات الإرادية.في سياق آليات الانتباه، نشير إلى الإشارات الارادية على أنها استعلامات queries. بالنظر إلى أي استفسار، فإن آليات الانتباه تحيز الاختيار bias selection على المدخلات الحسية sensory inputs (على سبيل المثال، تمثيلات الميزات الوسيطة intermediate feature representations) عبر تجميع الانتباه attention pooling. تسمى هذه المدخلات الحسية القيم valuesفي سياق آليات الانتباه. بشكل عام، يتم إقران كل قيمة بمفتاح key، والذي يمكن التفكير فيه من خلال الإشارة غير المنطقية لتلك المدخلات الحسية. كما هو مبينفي الشكل 11.1.3، يمكننا تصميم تجميع الانتباه بحيث يمكن أن يتفاعل الاستعلام المحدد (الإشارات الإرادية) مع المفاتيح (الإشارات غير الارادية)، والتي توجه اختيار التحيز على القيم (المدخلات الحسية).



الشكل 11.1.3 تحيز الاختيار لآليات الانتباه على القيم (المدخلات الحسية) من خلال تجميع الانتباه، والذي يتضمن الاستعلامات (الإشارات الإرادية) والمفاتيح (الإشارات غير المنطقية).

لاحظ أن هناك العديد من البدائل لتصميم آليات الانتباه. على سبيل المثال، يمكننا تصميم نموذج انتباه غير قابل للتفاضل يمكن تدريبه باستخدام أساليب التعلم المعزز reinforcement learning (Mnih et al.، 2014). بالنظر إلى هيمنة إطار العمل في الشكل 11.1.3، ستكون النماذج في هذا الإطار محور اهتمامنا في هذا الفصل.

11.1.3. رسم الانتباه Visualization of Attention

يمكن التعامل مع متوسط التجميع Average pooling كمتوسط وزني للمدخلات، حيث تكون الأوزان موحدة. في الممارسة العملية، يجمع الانتباه القيم الإجمالية باستخدام متوسط الاوزان، حيث يتم حساب الأوزان بين الاستعلام المحدد والمفاتيح المختلفة.

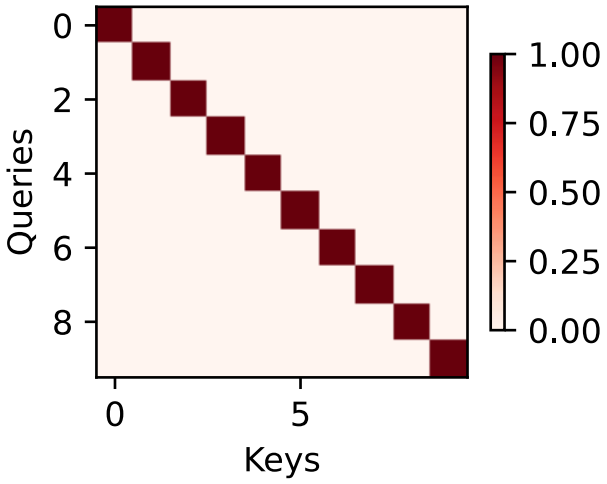
```
import tensorflow as tf
from d2l import tensorflow as d2l
```

لرسم أوزان الانتباه attention weights، نحدد دالة `show_heatmaps`. مصفوفات الإدخال `matrices` لها الشكل (عدد الصفوف للعرض، عدد الأعمدة للعرض، عدد الاستعلامات، عدد المفاتيح).

```
#@save
def show_heatmaps(matrices, xlabel, ylabel, titles=None,
figsize=(2.5, 2.5),
                  cmap='Reds'):
    """Show heatmaps of matrices."""
    d2l.use_svg_display()
    num_rows, num_cols = len(matrices), len(matrices[0])
    fig, axes = d2l.plt.subplots(num_rows, num_cols,
figsize=figsize,
                                 sharex=True,
sharey=True, squeeze=False)
    for i, (row_axes, row_matrices) in
enumerate(zip(axes, matrices)):
        for j, (ax, matrix) in enumerate(zip(row_axes,
row_matrices)):
            pcm = ax.imshow(matrix.numpy(), cmap=cmap)
            if i == num_rows - 1:
                ax.set_xlabel(xlabel)
            if j == 0:
                ax.set_ylabel(ylabel)
            if titles:
                ax.set_title(titles[j])
    fig.colorbar(pcm, ax=axes, shrink=0.6);
```

للتوضيح، نعتبر حالة بسيطة حيث يكون وزن الانتباه واحداً فقط عندما يكون الاستعلام والمفتاح متماثلين؛ وإلا فهو صفر.

```
attention_weights = tf.reshape(tf.eye(10), (1, 1, 10,
10))
show_heatmaps(attention_weights, xlabel='Keys',
ylabel='Queries')
```



في الأقسام التالية، غالبًا ما نستدعي هذه الدالة لرسم أوزان الانتباه.

11.1.4. الملخص

- الاهتمام (الانتباه) البشري Human attention هو مورد محدود وقيِّم ونادر.
- الموضوعات توجه الانتباه بشكل انتقائي باستخدام كل من الإشارات غير الارادية والإرادية. الأول يعتمد على النقطة المحفزة saliency والأخير يعتمد على المهمة task.
- تختلف آليات الانتباه عن الطبقات المتصلة بالكامل أو طبقات التجميع بسبب تضمين الإشارات الإرادية.
- تعمل آليات الانتباه على تحيز الاختيار bias selection على القيم (المدخلات الحسية) من خلال تجميع الانتباه، والذي يتضمن الاستعلامات (الإشارات الإرادية) والمفاتيح (الإشارات غير المنطقية). المفاتيح والقيم مقترنة.
- يمكننا رسم أوزان الانتباه بين الاستعلامات والمفاتيح.

11.1.5. التمارين

1. ماذا يمكن أن تكون الاشارة الإرادية عند فك تشفير رمز تسلسلي برمزفي الترجمة الآلية؟ ما هي الإشارات غير الارادية والمدخلات الحسية؟
2. أنشئ مصفوفة بشكل عشوائي واستخدم عملية softmax للتأكد من أن كل صف هو توزيع احتمالي صالح. ارسم أوزان الانتباه الناتج.

11.2. تجميع الانتباه Attention Pooling

أنت الآن تعرف المكونات الرئيسية لآليات الانتباه attention mechanisms ضمن الإطار الموضحفي الشكل 11.1.3. للتلخيص، تؤدي التفاعلات بين الاستعلامات queries (الإشارات الإرادية volitional cues) والمفاتيح keys (الإشارات غير الارادية nonvolitional cues) إلى تجميع الانتباه attention pooling. يجمع الاهتمام بشكل انتقائي القيم (المدخلات الحسية sensory inputs) لإنتاج المخرجات.في هذا القسم، سنصف تجميع الانتباه بمزيد من التفصيل لمنحك رؤية عالية المستوى لكيفية عمل آليات الانتباهفي الممارسة العملية. على وجه التحديد، نموذج انحدار نواة Nadaraya-Watson المقترحفي عام 1964 هو مثال بسيط ولكنه كامل لإظهار التعلم الآلي بآليات الانتباه.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.2.1. إنشاء مجموعة البيانات Generating the Dataset

لتبسيط الأمور، دعنا نفكرفي مشكلة الانحدار التالية: بالنظر إلى مجموعة بيانات من أزواج المدخلات والمخرجات $\{(x_1, y_1), \ldots, (x_n, y_n)\}$، كيف تعلم f كيفية توقع المخرجات $\hat{y} = f(x)$ عن أي مدخلات جديدة؟

نقوم هنا بإنشاء مجموعة بيانات اصطناعية artificial dataset وفقًا للدالة غير الخطية التالية بمصطلح الضوضاء ϵ:

$$y_i = 2\sin(x_i) + x_i^{0.8} + \epsilon,$$

حيث ϵ تخضع لتوزيع طبيعي بمتوسط صفري وانحراف معياري 0.5. يتم إنشاء كل من أمثلة التدريب 50 و50 أمثلة التحقق. لتصور نمط الانتباه بشكل أفضل لاحقًا، يتم فرز مدخلات التدريب.

```
class NonlinearData(d2l.DataModule):
    def __init__(self, n, batch_size):
        self.save_hyperparameters()
        f = lambda x: 2 * tf.sin(x) + x**0.8
```

```
        self.x_train = tf.sort(tf.random.uniform((n,1))
* 5, 0)
        self.y_train = f(self.x_train) +
tf.random.normal((n,1))
        self.x_val = tf.range(0, 5, 5.0/n)
        self.y_val = f(self.x_val)

    def get_dataloader(self, train):
        arrays = (self.x_train, self.y_train) if train
else (self.x_val, self.y_val)
        return self.get_tensorloader(arrays, train)

n = 50
data = NonlinearData(n, batch_size=10)
```

توضح الدالة التالية جميع أمثلة التدريب (التي تمثلها الدوائر)، ودالة توليد بيانات الحقيقة f بدون مصطلح الضوضاء (المُسمى "Truth")، ودالة التنبؤ المكتسبة (المُسمى بـ "Pred").

```
def plot_kernel_reg(y_hat):
    d2l.plot(data.x_val, [data.y_val, y_hat.numpy()],
'x', 'y', legend=['Truth', 'Pred'],
             xlim=[0, 5], ylim=[-1, 5])
    d2l.plt.plot(data.x_train, data.y_train, 'o',
alpha=0.5);
```

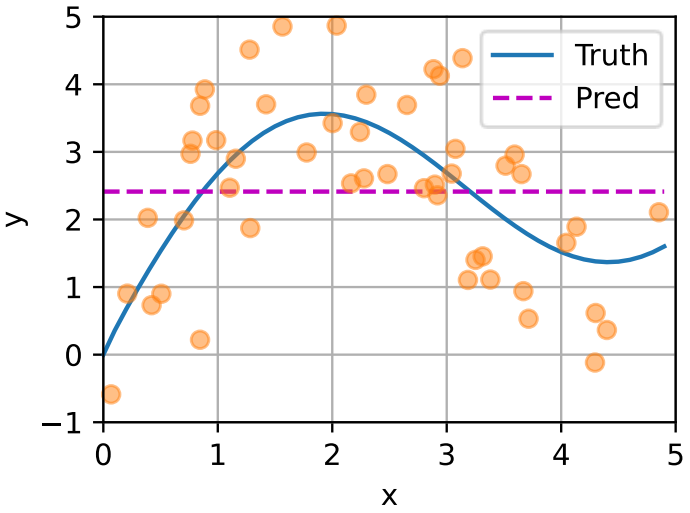
11.2.2. متوسط التجميع Average Pooling

نبدأ مع ربما "أغبى dumbest " مقدر estimatorفي العالم لمشكلة الانحدار هذه: استخدام متوسط التجميع إلى المتوسط على جميع مخرجات التدريب:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \quad y_i, \qquad (11.2.2)$$

الذي تم رسمه أدناه. كما نرى، هذا المقدّر ليس ذكيًا حقًا.

```
y_hat = tf.repeat(tf.reduce_mean(data.y_train), n)
plot_kernel_reg(y_hat)
```



11.2.3. تجميع الانتباه اللامعلمي Nonparametric Attention Pooling

من الواضح أن متوسط التجميع يتجاهل المدخلات x_i. تم اقتراح فكرة أفضل من قبل Nadaraya، (1964,Nadaraya) و Watson، (1964,Watson) لوزن المخرجات y_i وفقًا لمواقع إدخالها:

$$f(x) = \sum_{i=1}^{n} \frac{K(x-x_i)}{\sum_{j=1}^{n} K(x-x_j)} y_i, \qquad (11.2.3)$$

حيث K النواة kernel. المقدرفي (11.2.3) يسمى انحدار نواة Nadaraya-Watson. لن نتعمق هنافي تفاصيل النوى. استدعي إطار عمل آليات الانتباهفي الشكل 11.1.3. من منظور الانتباه، يمكننا إعادة كتابة (11.2.3)في شكل أكثر عمومية لتجميع الانتباه attention pooling:

$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i, (11.2.4)$$

حيث x هو الاستعلام وهو زوج المفتاح والقيمة (x_i, y_i). بمقارنة (11.2.4) و (11.2.2)، فإن تجميع الانتباه هنا هو متوسط الاوزان للقيم y_i. يتم تعيين وزن الانتباه $\alpha(x, x_i)$في (11.2.4) إلى القيمة المقابلة y_i بناءً على التفاعل بين الاستعلام x والمفتاح x_i المصمم بواسطة α. بالنسبة لأي استعلام، تعتبر أوزان الانتباه الخاصة به على جميع أزواج المفتاح–القيمة توزيعًا احتماليًا صالحًا: فهي غير سالبة ويصل مجموعهم إلى واحد.

لاكتساب حدس لتجميع الانتباه، ما عليك سوى التفكيرفي نواة غاوسية Gaussian kernel مُعرَّفة على أنها

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right).$$

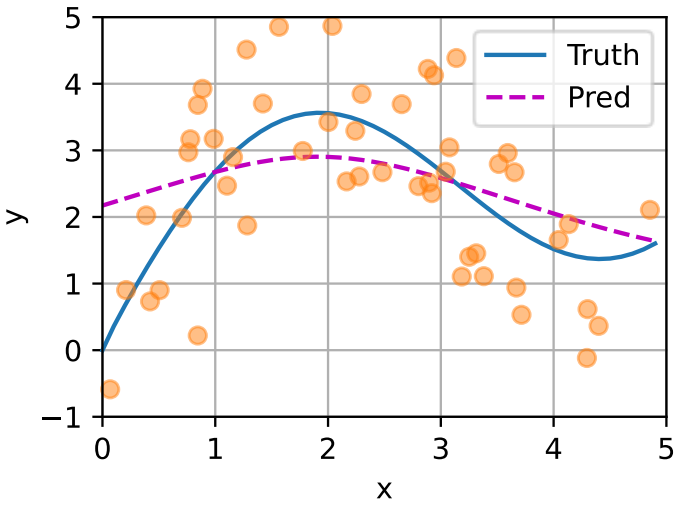
إدخال النواة الغاوسيةفي (11.2.4) و (11.2.3) يعطي:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\ &= \sum_{i=1}^{n} \text{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i. \end{aligned}$$

في (11.2.6)، سيحظى المفتاح x_i الأقرب إلى الاستعلام المعطى x بمزيد من الانتباه من خلال وزن انتباه أكبر larger attention weight يتم تعيينه لقيمة المفتاح المقابلة y_i.

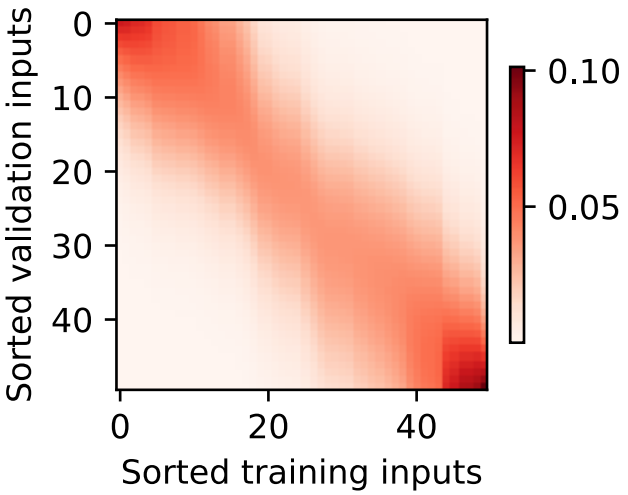
والجدير بالذكر أن انحدار نواة Nadaraya-Watson هو نموذج غير معلمي nonparametric model. وبالتالي (11.2.6) هو مثال لتجميع الانتباه اللامعلمي nonparametric attention pooling. فيما يلي، نرسم التنبؤ بناءً على نموذج الانتباه اللامعلمي هذا. الخط المتوقع سلس وأقرب إلى الحقيقة الأساسية من ذلك الناتج عن متوسط التجميع average pooling.

```
def diff(queries, keys):
    return tf.reshape(queries, (-1, 1)) -
tf.reshape(keys, (1, -1))

def attention_pool(query_key_diffs, values):
    attention_weights = tf.nn.softmax(-
query_key_diffs**2/2, axis=1)
    return tf.matmul(attention_weights, values),
attention_weights

y_hat, attention_weights = attention_pool(
    diff(data.x_val, data.x_train), data.y_train)
plot_kernel_reg(y_hat)
```



الآن دعونا نلقي نظرة على أوزان الانتباه. هنا مدخلات التحقق من الصحة validation inputs هي استعلامات بينما مدخلات التدريب training inputs هي مفاتيح. نظرًا لأنه يتم فرز كلا المدخلات، يمكننا أن نرى أنه كلما اقترب زوج المفتاح–الاستعلام، زاد وزن الانتباهفي تجميع الانتباه.

```
d2l.show_heatmaps([[attention_weights]],
                  xlabel='Sorted training inputs',
                  ylabel='Sorted validation inputs')
```



11.2.4. تجميع الانتباه المعلمي Parametric Attention Pooling

يتمتع انحدار نواة Nadaraya-Watson اللامعلمي بفائدة الاتساق consistency: نظرًا لبيانات كافية، يتقارب هذا النموذج مع الحل الأمثل. ومع ذلك، يمكننا بسهولة دمج المعلمات القابلة للتعلمفي تجميع الانتباه.

على سبيل المثال، يختلف قليلاً عن (11.2.6)،في المسافة التالية بين الاستعلام x والمفتاح x_i يتم ضربه بمعامل قابل للتعلم w:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp(-\frac{1}{2}((x-x_i)w)^2)}{\sum_{j=1}^{n} \exp(-\frac{1}{2}((x-x_j)w)^2)} y_i \\ &= \sum_{i=1}^{n} \text{softmax}(-\frac{1}{2}((x-x_i)w)^2) y_i. \end{aligned} \tag{11.2.7}$$

في باقي القسم، سنقوم بتدريب هذا النموذج من خلال تعلم معامل تجميع الانتباهفي (11.2.7).

11.2.4.1. ضرب مصفوفة الدُفعات Batch Matrix Multiplication

لحساب الانتباه بشكل أكثر كفاءة للدفعات الصغيرة minibatches، يمكننا الاستفادة من أدوات ضرب مصفوفة الدُفعات batch matrix multiplication التي توفرها أطر عمل التعلم العميق.

افترض أن أول دفعة صغيرة تحتوي على n مصفوفات $\mathbf{X}_1, \ldots, \mathbf{X}_n$ بالشكل $a \times b$، بينما تحتوي الدفعة الثانية على n مصفوفات $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$ بالشكل $b \times c$. ينتج عن ضرب المصفوفة الدفعية n مصفوفات $\mathbf{X}_1\mathbf{Y}_1, \ldots, \mathbf{X}_n\mathbf{Y}_n$ بالشكل $a \times c$. لذلك، بالنظر إلى موترين من الشكل $(n،a،b)$ و $(n،b،c)$، يكون شكل ناتج ضرب المصفوفة الدفعية هو $(n،a،c)$.

```
X = tf.ones((2, 1, 4))
Y = tf.ones((2, 4, 6))
d2l.check_shape(tf.matmul(X, Y), (2, 1, 6))
```

في سياق آليات الانتباه، يمكننا استخدام ضرب مصفوفة الدفعات الصغيرة لحساب المتوسطات الموزونة للقيمفي الدفعات الصغيرة.

```
weights = tf.ones((2, 10)) * 0.1
values = tf.reshape(tf.range(20.0), shape = (2, 10))
tf.matmul(tf.expand_dims(weights, axis=1),
tf.expand_dims(values, axis=-1)).numpy()
```

```
array([[[ 4.5]],

       [[14.5]]], dtype=float32)
```

11.2.4.2. تعريف النموذج Defining the Model

باستخدام ضرب مصفوفة الدفعات الصغيرة، نحدد أدناه النسخة المعلمية من انحدار نواة Nadaraya-Watson استنادًا إلى تجميع الانتباه المعلميفي (11.2.7).

```
class NWKernelRegression(d2l.Module):
    def __init__(self, keys, values, lr):
        super().__init__()
        self.save_hyperparameters()
        self.w = tf.Variable(tf.ones(1), trainable=True)

    def forward(self, queries):
        y_hat, self.attention_weights = attention_pool(
            diff(queries, self.keys) * self.w,
self.values)
        return y_hat

    def loss(self, y_hat, y):
        l = (tf.reshape(y_hat, -1) - tf.reshape(y, -1))
** 2 / 2
        return tf.reduce_mean(l)

    def configure_optimizers(self):
        return d2l.SGD(self.lr)
```

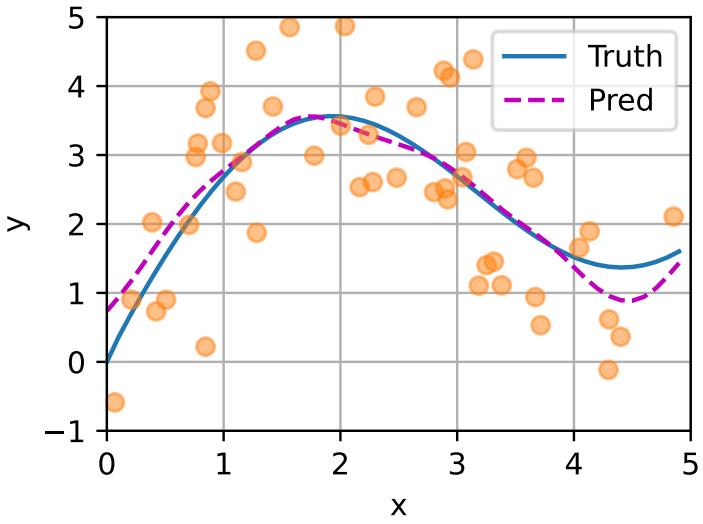
11.2.4.3. التدريب Training

فيما يلي، نقوم بتحويل مجموعة بيانات التدريب إلى مفاتيح وقيم لتدريب نموذج الانتباه.في تجميع الانتباه المعلمي، من أجل التبسيط، يأخذ أي إدخال تدريب فقط أزواج المفتاح–القيمة من جميع أمثلة التدريب للتنبؤ بمخرجاته.

```
model = NWKernelRegression(data.x_train, data.y_train,
lr=1)
model.board.display = False
trainer = d2l.Trainer(max_epochs=5)
trainer.fit(model, data)
```

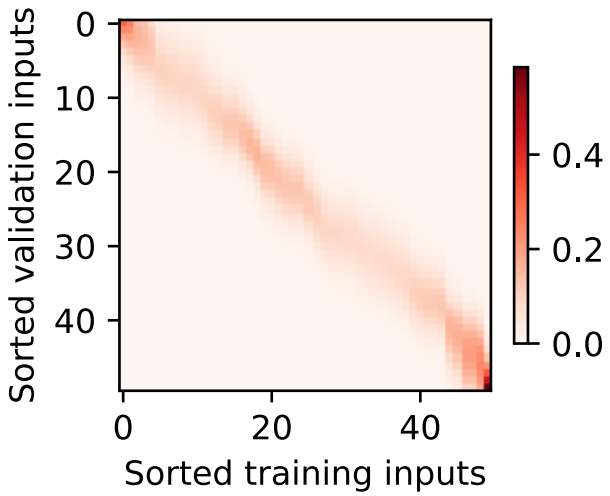
في محاولة لتلائم fit مجموعة بيانات التدريب بالضوضاء، يكون الخط المتوقع أقل سلاسة من نظيره اللامعلمي الذي تم رسمه مسبقًا.

```
plot_kernel_reg(model.forward(data.x_val))
```



بالمقارنة مع تجميع الانتباه اللامعلمي، تصبح المنطقة ذات أوزان الانتباه الكبيرة أكثر حدةفي الضبط المعلمي.

```
d2l.show_heatmaps([[model.attention_weights]],
                  xlabel='Sorted training inputs',
                  ylabel='Sorted validation inputs')
```



11.2.5. الملخص

- يعد انحدار نواة Nadaraya-Watson مثالاً على التعلم الآلي بآليات الانتباه.

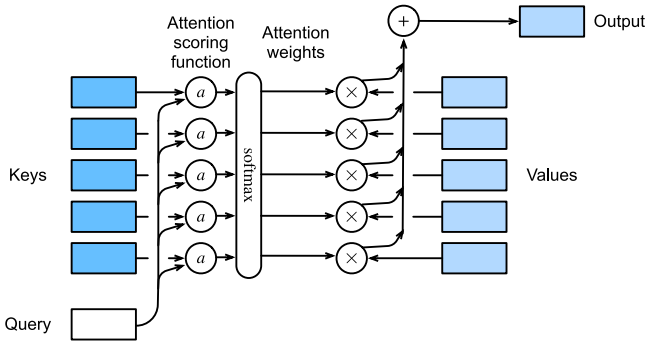
- تجميع الانتباه لانحدار نواة Nadaraya-Watson هو متوسط اوزان لمخرجات التدريب. من منظور الانتباه، يتم تعيين وزن الانتباه إلى قيمة بناءً على دالة استعلام والمفتاح المقترن بالقيمة.
- يمكن أن يكون تجميع الانتباه إما غير معلمي nonparametric أو معلمي parametric.

11.2.6. التمارين

1. قم بزيادة عدد الأمثلة التدريبية. هل يمكنك تعلم انحدار نواة Nadaraya-Watson اللامعلمية بشكل أفضل؟
2. ما هي قيمة ما تعلمناه wفي تجربة تجميع الانتباه المعلمية؟ لماذا تجعل المنطقة الموزونة أكثر حدة عند رسم أوزان الانتباه؟
3. كيف يمكننا إضافة المعلمات الفائقة إلى انحدار نواة Nadaraya-Watson اللامعلمي للتنبؤ بشكل أفضل؟
4. صمم انتباهاً معلمياً آخر لتجميع الانتباه لانحدار النواةفي هذا القسم. درب هذا النموذج الجديد وارسم أوزان انتباهه.

11.3. دوال تسجيل الانتباه Attention Scoring Functions

في القسم 11.2، استخدمنا نواة غاوسية لنمذجة التفاعلات بين الاستعلامات والمفاتيح. معالجة أس النواة الغاوسيةفي (11.2.6) كدالة لتسجيل الانتباه attention scoring function (أو دالة التسجيل scoring function للاختصار)، تم تغذية نتائج هذه الدالة بشكل أساسيفي عملية softmax. نتيجة لذلك، حصلنا على توزيع احتمالي (أوزان الانتباه) على القيم المقترنة بالمفاتيح.في النهاية، ناتج تجميع الانتباه هو ببساطة مجموع الاوزان للقيم بناءً على أوزان الانتباه هذه.



الشكل 11.3.1 حساب ناتج تجميع الانتباه كمتوسط الوزن للقيم.

على مستوى عالٍ، يمكننا استخدام الخوارزمية أعلاه لإنشاء مثيل لإطار آليات الانتباهفي الشكل 11.1.3. يوضح الشكل 11.3.1، الذي يشير إلى دالة تسجيل الانتباه من خلال a، كيف يمكن حساب ناتج تجميع الانتباه كمجموع اوزان من القيم. نظرًا لأن أوزان الانتباه هي توزيع احتمالي، فإن مجموع الاوزان هوفي الأساس متوسط موزون.

رياضيًا، افترض أن لدينا استعلامًا $\mathbf{q} \in \mathbb{R}^q$ وm أزواج المفتاح–القيمة $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)$، حيث كل $\mathbf{k}_i \in \mathbb{R}^k$ وكل $\mathbf{v}_i \in \mathbb{R}^v$. يتم إنشاء مثيل تجميع الانتباه f كمجموع الوزن للقيم:

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^v,$$

حيث يتم حساب وزن الانتباه (scalar) للاستعلام $\mathbf{q}$ والمفتاح $\mathbf{k}_i$ من خلال عملية softmax لدالة تسجيل الانتباه a التي تعين متجهين إلى القيمة القياسية scalar:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}.$$

كما نرى، تؤدي الخيارات المختلفة لدالة تسجيل الانتباه a إلى سلوكيات مختلفة لتجميع الانتباه attention pooling.في هذا القسم، نقدم دالتين شائعتين للتسجيل سنستخدمهما لتطوير آليات انتباه أكثر تعقيدًا لاحقًا.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.3.1. عملية Softmax المقنعة Masked Softmax Operation

كما ذكرنا للتو، يتم استخدام عملية softmax لإخراج توزيع احتمالي كأوزان للانتباه.في بعض الحالات، لا يجب تغذية جميع القيمفي عملية تجميع الانتباه. على سبيل المثال، من أجل معالجة الدفعات الصغيرة الفعالةفي القسم 10.5، تكون بعض التسلسلات النصية محشوة padded برموز خاصة special tokens لا تحمل أي معنى. لجذب الانتباه إلى الرموز ذات المعنى فقط كقيم، يمكننا تحديد طول تسلسل صالح (في عدد الرموز) لتصفية تلك التي تتجاوز هذا النطاق المحدد عند حساب softmax. بهذه الطريقة، يمكننا تنفيذ عملية softmax المقنعة هذهفي دالة `masked_softmax` التالية، حيث يتم إخفاء أي قيمة تتجاوز الطول الصالح على أنها صفر.

```
#@save
def masked_softmax(X, valid_lens):
    """Perform softmax operation by masking elements on
the last axis."""
```

```
    # X: 3D tensor, valid_lens: 1D or 2D tensor
    def _sequence_mask(X, valid_len, value=0):
        maxlen = X.shape[1]
        mask = tf.range(start=0, limit=maxlen,
dtype=tf.float32)[
            None, :] < tf.cast(valid_len[:, None],
dtype=tf.float32)

        if len(X.shape) == 3:
            return tf.where(tf.expand_dims(mask, axis=-
1), X, value)
        else:
            return tf.where(mask, X, value)

    if valid_lens is None:
        return tf.nn.softmax(X, axis=-1)
    else:
        shape = X.shape
        if len(valid_lens.shape) == 1:
            valid_lens = tf.repeat(valid_lens,
repeats=shape[1])

        else:
            valid_lens = tf.reshape(valid_lens, shape=-
1)
        # On the last axis, replace masked elements with
a very large negative
        # value, whose exponentiation outputs 0
        X = _sequence_mask(tf.reshape(X, shape=(-1,
shape[-1])), valid_lens,
                           value=-1e6)
        return tf.nn.softmax(tf.reshape(X, shape=shape),
axis=-1)
```

لتوضيح كيفية عمل هذه الدالة، ضعفي اعتبارك مجموعة صغيرة من مثالين لمصفوفة 4 × 2 ، حيث يكون الأطوال الصالحة لهذين المثالين اثنين وثلاثة على التوالي. نتيجة لعملية softmax المقنعة، يتم إخفاء القيم التي تتجاوز الأطوال الصالحة على أنها صفر.

```
masked_softmax(tf.random.uniform(shape=(2, 2, 4)),
tf.constant([2, 3]))
```

```
<tf.Tensor: shape=(2, 2, 4), dtype=float32, numpy=
```

```
array([[[0.4517647 , 0.54823536, 0.        , 0.
],
        [0.4112412 , 0.5887589 , 0.        , 0.
]],

       [[0.24448606, 0.34571627, 0.40979767, 0.
],
        [0.3740311 , 0.35151485, 0.27445397, 0.
]]], dtype=float32)>
```

وبالمثل، يمكننا أيضًا استخدام موتر ثنائي الأبعاد لتحديد أطوال صالحة لكل صف في كل مثال مصفوفة.

```
masked_softmax(tf.random.uniform((2, 2, 4)),
tf.constant([[1, 3], [2, 4]]))
```

```
<tf.Tensor: shape=(2, 2, 4), dtype=float32, numpy=
array([[[1.        , 0.        , 0.        , 0.
],
        [0.26554373, 0.36378106, 0.37067524, 0.
]],

       [[0.46347728, 0.5365227 , 0.        , 0.
],
        [0.16122091, 0.15872595, 0.29501283,
0.38504028]]], dtype=float32)>
```

11.3.2. الانتباه الإضافي Additive Attention

بشكل عام، عندما تكون الاستعلامات والمفاتيح متجهات ذات أطوال مختلفة، يمكننا استخدام الانتباه الإضافي كدالة لتسجيل النقاط. بالنظر إلى الاستعلام $\mathbf{q} \in \mathbb{R}^q$ والمفتاح $\mathbf{k} \in \mathbb{R}^k$ ، فإن دالة تسجيل الانتباه الإضافي additive attention scoring function

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh(\mathbf{W}_q\mathbf{q} + \mathbf{W}_k\mathbf{k}) \in \mathbb{R},$$

حيث المعلمات القابلة للتعلم $\mathbf{W}_q \in \mathbb{R}^{h\times q}$، $\mathbf{W}_k \in \mathbb{R}^{h\times k}$ و $\mathbf{w}_v \in \mathbb{R}^h$. أي ما يعادل (11.3.3)، يتم تسلسل الاستعلام والمفتاح وإدخالهمافي MLP بطبقة واحدة مخفية يكون عدد الوحدات المخفية فيها h ، وهي معلمة فائقة. باستخدام دالة التنشيط tanh وتعطيل شروط التحيز، فإننا ننفذ اهتمامًا إضافيًا فيما يلي.

```
#@save
class AdditiveAttention(tf.keras.layers.Layer):
```

```
    """Additive attention."""
    def __init__(self, key_size, query_size,
num_hiddens, dropout, **kwargs):
        super().__init__(**kwargs)
        self.W_k = tf.keras.layers.Dense(num_hiddens,
use_bias=False)
        self.W_q = tf.keras.layers.Dense(num_hiddens,
use_bias=False)
        self.w_v = tf.keras.layers.Dense(1,
use_bias=False)
        self.dropout = tf.keras.layers.Dropout(dropout)

    def call(self, queries, keys, values, valid_lens,
**kwargs):
        queries, keys = self.W_q(queries),
self.W_k(keys)
        # After dimension expansion, shape of queries:
(batch_size, no. of
        # queries, 1, num_hiddens) and shape of keys:
(batch_size, 1, no. of
        # key-value pairs, num_hiddens). Sum them up
with broadcasting
        features = tf.expand_dims(queries, axis=2) +
tf.expand_dims(
            keys, axis=1)
        features = tf.nn.tanh(features)
        # There is only one output of self.w_v, so we
remove the last
        # one-dimensional entry from the shape. Shape of
scores: (batch_size,
        # no. of queries, no. of key-value pairs)
        scores = tf.squeeze(self.w_v(features), axis=-1)
        self.attention_weights = masked_softmax(scores,
valid_lens)
        # Shape of values: (batch_size, no. of key-value
pairs, value
        # dimension)
        return tf.matmul(self.dropout(
            self.attention_weights, **kwargs), values)
```

دعنا نوضح فئة `AdditiveAttention` أعلاه مع مثال لعبة ، حيث الأشكال (حجم الدُفعة، عدد الخطوات أو طول التسلسلفي الرموز، حجم الميزة) للاستعلامات والمفاتيح والقيم هي

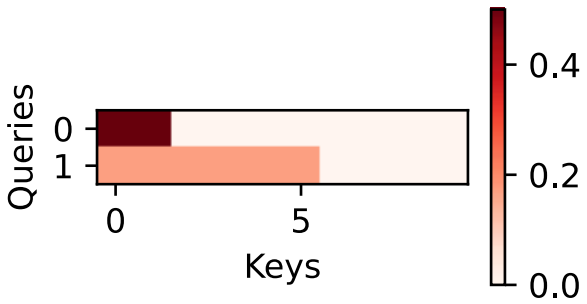
(2،1،20) ، (2،10،2) ، و (2،10،4)، على التوالي. ناتج تجميع الانتباه له شكل (حجم الدُفعة، عدد خطوات الاستعلامات، حجم الميزة للقيم).

```
queries, keys = tf.random.normal(shape=(2, 1, 20)),
tf.ones((2, 10, 2))
# The two value matrices in the values minibatch are
identical
values = tf.repeat(tf.reshape(
    tf.range(40, dtype=tf.float32), shape=(1, 10, 4)),
repeats=2, axis=0)
valid_lens = tf.constant([2, 6])

attention = AdditiveAttention(key_size=2, query_size=20,
num_hiddens=8,
                              dropout=0.1)
attention(queries, keys, values, valid_lens,
training=False)
```

```
<tf.Tensor: shape=(2, 1, 4), dtype=float32, numpy=
array([[[ 2.      ,  3.      ,  4.      ,  5.      ]],

       [[10.      , 11.      , 12.000001, 13.      ]]],
dtype=float32)>
```

على الرغم من أن الانتباه الإضافي additive attention يحتوي على معلمات قابلة للتعلم learnable parameters، نظرًا لأن كل مفتاح هو نفسه في هذا المثال، فإن أوزان الانتباه تكون منتظم uniform، ويتم تحديدها من خلال الأطوال الصالحة المحددة.

```
d2l.show_heatmaps(tf.reshape(attention.attention_weights
, (1, 1, 2, 10)),
                  xlabel='Keys', ylabel='Queries')
```



11.3.3. انتباه الضرب النقطي المقاس Scaled Dot-Product Attention

يمكن أن يكون التصميم الأكثر كفاءة من الناحية الحسابية لدالة التسجيل مجرد ضرب نقطي dot product. ومع ذلك، تتطلب عملية الضرب النقطي أن يكون لكل من الاستعلام والمفتاح نفس طول المتجه، على سبيل المثال d. افترض أن جميع عناصر الاستعلام والمفتاح عبارة عن متغيرات عشوائية مستقلة بمتوسط صفري وتباين الوحدة. حاصل الضرب النقطي لكلا المتجهين له متوسط صفري وتباين قدره d. للتأكد من أن تباين الضرب النقطي لا يزال واحداً بغض النظر عن طول المتجه، فإن دالة تسجيل انتباه المنتج النقطي المقاس scaled dot-product attention scoring function

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}/\sqrt{d}$$

قسمة الضرب النقطي على $\sqrt{d}$. من الناحية العملية، غالبًا ما نفكرفي الدفعات الصغيرة من أجل الكفاءة، مثل الاهتمام الحاسوبي للاستعلامات n وأزواج القيمة والمفاتيح m، حيث تكون الاستعلامات والمفاتيح ذات طول d والقيم ذات طول v. يتم قياس انتباه الضرب النقطي للاستعلامات $\mathbf{Q} \in \mathbb{R}^{n\times d}$ والمفاتيح $\mathbf{K} \in \mathbb{R}^{m\times d}$ والقيم $\mathbf{V} \in \mathbb{R}^{m\times v}$

$$\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n\times v}.$$

في التنفيذ التالي لانتباه الضرب النقطي المقاس scaled dot product attention، نستخدم التسرب dropout من أجل تسوية النموذج.

```
#@save
class DotProductAttention(tf.keras.layers.Layer):
    """Scaled dot product attention."""
    def __init__(self, dropout, num_heads=None):
        super().__init__()
        self.dropout = tf.keras.layers.Dropout(dropout)
        self.num_heads = num_heads  # To be covered
later

    # Shape of queries: (batch_size, no. of queries, d)
    # Shape of keys: (batch_size, no. of key-value
pairs, d)
    # Shape of values: (batch_size, no. of key-value
pairs, value dimension)
    # Shape of valid_lens: (batch_size,) or (batch_size,
no. of queries)
```

```
    def call(self, queries, keys, values,
valid_lens=None, window_mask=None,
             **kwargs):
        d = queries.shape[-1]
        scores = tf.matmul(queries, keys,
transpose_b=True)/tf.math.sqrt(
            tf.cast(d, dtype=tf.float32))
        if window_mask is not None:  # To be covered
later
            num_windows = window_mask.shape[0]
            n, num_queries, num_kv_pairs = scores.shape
            # Shape of window_mask: (num_windows, no. of
queries,
            # no. of key-value pairs)
            scores = tf.reshape(
                scores,
(n//(num_windows*self.num_heads), num_windows,
                            self.num_heads, num_queries,
num_kv_pairs
                           )) + tf.expand_dims(
                tf.expand_dims(window_mask, 1), 0)
            scores = tf.reshape(scores, (n, num_queries,
num_kv_pairs))
        self.attention_weights = masked_softmax(scores,
valid_lens)
        return
tf.matmul(self.dropout(self.attention_weights,
**kwargs), values)
```

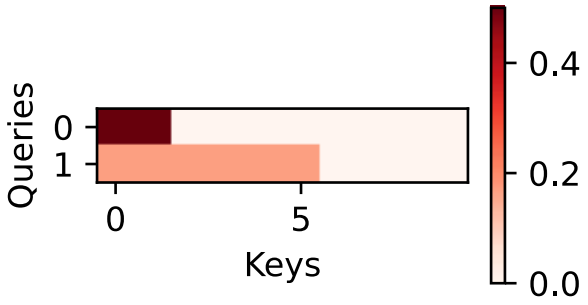
لإثبات فئة `DotProductAttention` أعلاه، نستخدم نفس المفاتيح والقيم والأطوال الصالحة من مثال اللعبة السابق للاهتمام الإضافي. بالنسبة لعملية الضرب النقطي، نجعل حجم ميزة الاستعلامات هو نفسه حجم المفاتيح.

```
queries = tf.random.normal(shape=(2, 1, 2))
attention = DotProductAttention(dropout=0.5)
attention(queries, keys, values, valid_lens,
training=False)
```

```
<tf.Tensor: shape=(2, 1, 4), dtype=float32, numpy=
array([[[ 2.       ,  3.       ,  4.       ,  5.       ]],

       [[10.       , 11.       , 12.000001, 13.       ]]],
dtype=float32)>
```

كما هو الحال في عرض الانتباه الإضافي additive attention، نظرًا لأن المفاتيح تحتوي على نفس العنصر الذي لا يمكن تمييزه بأي استعلام، يتم الحصول على أوزان انتباه موحدة uniform attention weights.

```
d2l.show_heatmaps(tf.reshape(attention.attention_weights
, (1, 1, 2, 10)),
                  xlabel='Keys', ylabel='Queries')
```



11.3.4. الملخص

- يمكننا حساب ناتج تجميع الانتباه كمتوسط الوزن للقيم، حيث تؤدي الاختيارات المختلفة لدالة تسجيل الانتباه إلى سلوكيات مختلفة لتجميع الانتباه.
- عندما تكون الاستعلامات والمفاتيح متجهات ذات أطوال مختلفة، يمكننا استخدام دالة تسجيل الانتباه الإضافي. عندما تكون متطابقة، تكون دالة تسجيل انتباه المنتج النقطي المقاسة أكثر كفاءة من الناحية الحسابية.

11.3.5. التمارين

1. قم بتعديل المفاتيح في مثال اللعبة وارسم أوزان الانتباه. هل الانتباه الإضافي وانتباه المنتج النقطي المقاس لا يزالان ينتجان نفس أوزان الانتباه؟ لما ولما لا؟
2. باستخدام ضرب المصفوفة فقط، هل يمكنك تصميم دالة تسجيل جديدة للاستعلامات والمفاتيح ذات أطوال متجهات مختلفة؟
3. عندما يكون للاستعلامات والمفاتيح نفس طول المتجه، فهل يعد جمع المتجهات تصميمًا أفضل من حاصل الضرب النقطي لدالة التسجيل؟ لما ولما لا؟

11.4. انتباه Bahdanau

لقد درسنا مشكلة الترجمة الآلية في القسم 10.7، حيث صممنا معمارية المشفر ـ مفكك الشفرة على أساس اثنين من RNNs لتعلم التسلسل ـ التسلسل sequence to sequence

learning. على وجه التحديد، يقوم مشفر RNN بتحويل تسلسل متغير الطول إلى متغير سياق ذو شكل ثابت، ثم يقوم مفكك شفرة RNN بإنشاء رمز تسلسل الإخراج (الهدف) بواسطة الرمز token بناءً على الرموز المتولدة ومتغير السياق. ومع ذلك، على الرغم من أن جميع رموز الإدخال (المصدر) ليست مفيدة لفك تشفير رمز معين، إلا أن متغير السياق نفسه الذي يشفر تسلسل الإدخال بالكامل لا يزال مستخدمًافي كل خطوة فك تشفير.

في تحد منفصل ولكنه مرتبط بتوليد الكتابة اليدوية لتسلسل نصي معين، صمم Graves نموذجًا مختلفًا للانتباه لمحاذاة أحرف النص مع تتبع القلم الأطول بكثير، حيث تتحرك المحاذاةفي اتجاه واحد فقط (Graves، 2013). مستوحاة من فكرة تعلم المحاذاة، Bahdanau et al. اقترح نموذج اهتمام متباين دون قيود المحاذاة أحادية الاتجاه الشديدة (Bahdanau et al.، 2014). عند توقع رمز، إذا لم تكن جميع الرموز للإدخال ذات صلة، فإن النموذج يحاذي (أو يحضر attends) فقط لأجزاء من تسلسل الإدخال ذات الصلة بالتنبؤ الحالي. يتم تحقيق ذلك من خلال معاملة متغير السياق كناتج لتجميع الانتباه.

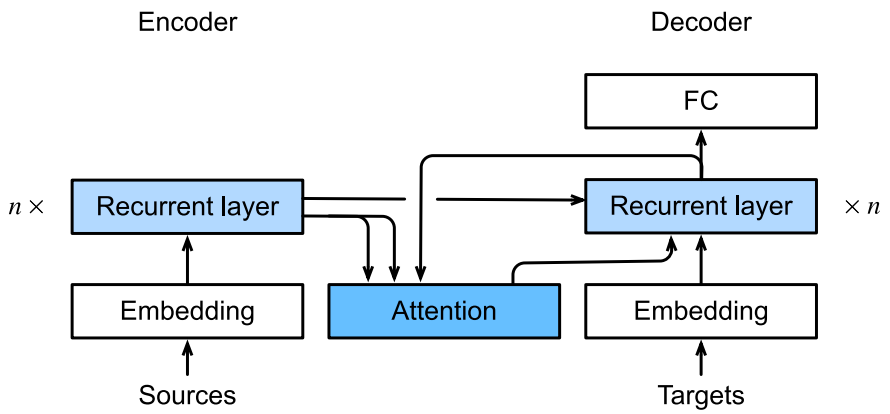
11.4.1. النموذج Model

عند وصف انتباه Bahdanau لمشفر ـ مفكك شفرة RNN أدناه، سوف نتبع نفس الترميزفي القسم 10.7. النموذج الجديد القائم على الانتباه هو نفسه الموجودفي القسم 10.7 فيما عدا أنه يتم استبدال متغير السياق $\mathbf{c}$في (10.7.3) بـ $\mathbf{c}_{t'}$في أي خطوة زمنية $\mathbf{c}_{t'}$ لفك التشفير. لنفترض أن هناك رموزًا Tفي تسلسل الإدخال، متغير السياقفي الخطوة الزمنية t' لفك التشفير هو ناتج تجميع الانتباه:

$$\mathbf{c}_{t'} = \sum_{t=1}^{T} \alpha(\mathbf{s}_{t'-1}, \mathbf{h}_t)\mathbf{h}_t,$$

حيث تكون الحالة المخفية $\mathbf{s}_{t'-1}$ لوحدة لمفكك الشفرةفي الخطوة الزمنية $t'-1$ هي الاستعلام، وتكون الحالات المخفية $\mathbf{h}_t$ للمشفر هي المفاتيح والقيم، ويتم حساب وزن الانتباه α كمافي (11.3.2) باستخدام دالة تسجيل الانتباه الإضافي المقاسة بواسطة (11.3.3).

تختلف قليلاً عن معمارية مشفر ـ مفكك شفرة vanilla RNNفي الشكل. 10.7.2، تم توضيح نفس المعمارية مع انتباه Bahdanauفي الشكل 11.4.1.



الشكل 11.4.1 طبقاتفي نموذج مشفر_مفكك شفرة RNN مع اهتمام Bahdanau.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.4.2. تعريف مفكك الشفرة مع الانتباه Defining the Decoder with Attention

لتنفيذ مشفر_مفكك شفرة RNN مع انتباه Bahdanau، نحتاج فقط إلى إعادة تعريف مفكك الشفرة decoder. لرسم أوزان الانتباه المكتسبة بشكل أكثر ملاءمة، تحدد فئة `AttentionDecoder` التالية الواجهة الأساسية مفكك الشفرة بآليات الانتباه.

```
#@save
class AttentionDecoder(d2l.Decoder):
    """The base attention-based decoder interface."""
    def __init__(self):
        super().__init__()

    @property
    def attention_weights(self):
        raise NotImplementedError
```

الآن دعنا نطبق مفكك شفرة RNN مع انتباه Bahdanauفي فئة `Seq2SeqAttentionDecoder` التالية. تتم تهيئة حالة مفكك الشفرة بـ (1) حالات الطبقة النهائية المشفرة المخفيةفي جميع خطوات الوقت (كمفاتيح وقيم الانتباه) ؛ (2) الحالة المخفية لجميع طبقات المشفرفي الخطوة الزمنية النهائية (لتهيئة الحالة المخفية لمفكك الشفرة)؛ و (3) الطول الصالح للمشفر (لاستبعاد رموز الحشوفي تجميع الانتباه).في كل خطوة زمنية لمفكك الشفرة، تُستخدم الحالة المخفية للطبقة النهائية لمفكك الشفرةفي الخطوة الزمنية

السابقة كاستعلام للانتباه. نتيجة لذلك، يتم تسلسل كل من إخراج الانتباه وتضمين المدخلات كمدخلات لمفكك شفرة RNN.

```
class Seq2SeqAttentionDecoder(AttentionDecoder):
    def __init__(self, vocab_size, embed_size,
num_hiddens, num_layers,
                 dropout=0):
        super().__init__()
        self.attention =
d2l.AdditiveAttention(num_hiddens, num_hiddens,

num_hiddens, dropout)
        self.embedding =
tf.keras.layers.Embedding(vocab_size, embed_size)
        self.rnn =
tf.keras.layers.RNN(tf.keras.layers.StackedRNNCells(
            [tf.keras.layers.GRUCell(num_hiddens,
dropout=dropout)
             for _ in range(num_layers)]),
return_sequences=True,

return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def init_state(self, enc_outputs, enc_valid_lens):
        # Shape of outputs: (batch_size, num_steps,
num_hiddens).
        # Length of list hidden_state is num_layers,
where the shape of its
        # element is (batch_size, num_hiddens)
        outputs, hidden_state = enc_outputs
        return (tf.transpose(outputs, (1, 0, 2)),
hidden_state,
                enc_valid_lens)

    def call(self, X, state, **kwargs):
        # Shape of output enc_outputs: # (batch_size,
num_steps, num_hiddens)
        # Length of list hidden_state is num_layers,
where the shape of its
        # element is (batch_size, num_hiddens)
```

```
        enc_outputs, hidden_state, enc_valid_lens =
state
        # Shape of the output X: (num_steps, batch_size,
embed_size)
        X = self.embedding(X)  # Input X has shape:
(batch_size, num_steps)
        X = tf.transpose(X, perm=(1, 0, 2))
        outputs, self._attention_weights = [], []
        for x in X:
            # Shape of query: (batch_size, 1,
num_hiddens)
            query = tf.expand_dims(hidden_state[-1],
axis=1)
            # Shape of context: (batch_size, 1,
num_hiddens)
            context = self.attention(query, enc_outputs,
enc_outputs,
                                     enc_valid_lens,
**kwargs)
            # Concatenate on the feature dimension
            x = tf.concat((context, tf.expand_dims(x,
axis=1)), axis=-1)
            out = self.rnn(x, hidden_state, **kwargs)
            hidden_state = out[1:]
            outputs.append(out[0])

self._attention_weights.append(self.attention.attention_
weights)
        # After fully connected layer transformation,
shape of outputs:
        # (batch_size, num_steps, vocab_size)
        outputs = self.dense(tf.concat(outputs, axis=1))
        return outputs, [enc_outputs, hidden_state,
enc_valid_lens]

    @property
    def attention_weights(self):
        return self._attention_weights
```

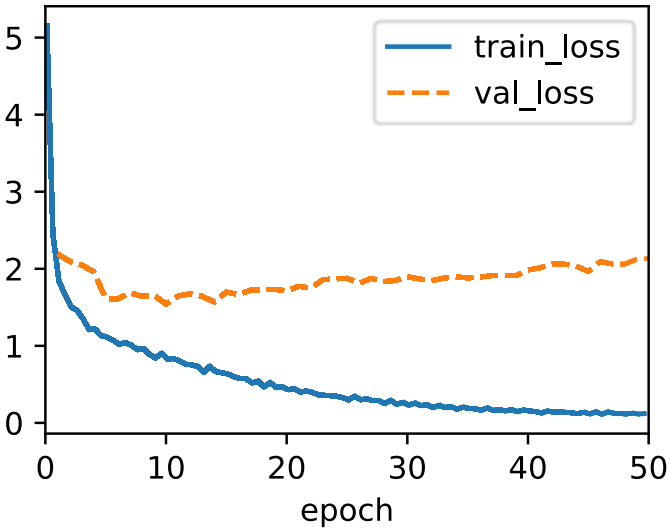
فيما يلي، نختبر مفكك الشفرة المنفذ مع انتباه Bahdanau باستخدام الدفعات الصغيرة من 4 مدخلات تسلسلية من 7 خطوات زمنية.

```
vocab_size, embed_size, num_hiddens, num_layers = 10, 8,
16, 2
batch_size, num_steps = 4, 7
encoder = d2l.Seq2SeqEncoder(vocab_size, embed_size,
num_hiddens, num_layers)
decoder = Seq2SeqAttentionDecoder(vocab_size,
embed_size, num_hiddens,
                                  num_layers)
X = tf.zeros((batch_size, num_steps))
state = decoder.init_state(encoder(X, training=False),
None)
output, state = decoder(X, state, training=False)
d2l.check_shape(output, (batch_size, num_steps,
vocab_size))
d2l.check_shape(state[0], (batch_size, num_steps,
num_hiddens))
d2l.check_shape(state[1][0], (batch_size, num_hiddens))
```

11.4.3. التدريب Training

على غرار القسم 10.7.6، نحدد هنا المعلمات الفائقة، وننشئ مثيلًا للمشفر ومفكك الشفرة بانتباه Bahdanau، ونقوم بتدريب هذا النموذج على الترجمة الآلية.

```
data = d2l.MTFraEng(batch_size=128)
embed_size, num_hiddens, num_layers, dropout = 256, 256,
2, 0.2
with d2l.try_gpu():
    encoder = d2l.Seq2SeqEncoder(
        len(data.src_vocab), embed_size, num_hiddens,
num_layers, dropout)
    decoder = Seq2SeqAttentionDecoder(
        len(data.tgt_vocab), embed_size, num_hiddens,
num_layers, dropout)
    model = d2l.Seq2Seq(encoder, decoder,
tgt_pad=data.tgt_vocab['<pad>'],
                        lr=0.005)
trainer = d2l.Trainer(max_epochs=50,
gradient_clip_val=1)
trainer.fit(model, data)
```



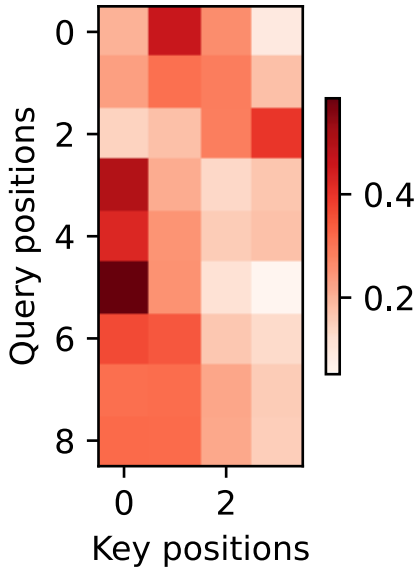
بعد تدريب النموذج، نستخدمه لترجمة بعض الجمل الإنجليزية إلى الفرنسية وحساب نقاطهم في BLEU

```
engs = ['go .', 'i lost .', 'he\'s calm .', 'i\'m home
.']
fras = ['va !', 'j\'ai perdu .', 'il est calme .', 'je
suis chez moi .']
preds, _ = model.predict_step(
    data.build(engs, fras), d2l.try_gpu(),
data.num_steps)
for en, fr, p in zip(engs, fras, preds):
    translation = []
    for token in data.tgt_vocab.to_tokens(p):
        if token == '<eos>':
            break
        translation.append(token)
    print(f'{en} => {translation}, bleu,'
          f'{d2l.bleu(" ".join(translation), fr,
k=2):.3f}')
```

```
go . => ['va', '!'], bleu,1.000
i lost . => ["j'ai", 'perdu', '.'], bleu,1.000
he's calm . => ['il', 'est', 'mouillé', '.'], bleu,0.658
i'm home . => ['je', 'suis', 'chez', 'moi', '.'],
bleu,1.000
```

من خلال رسم أوزان الانتباه عند ترجمة الجملة الإنجليزية الأخيرة، يمكننا أن نرى أن كل استعلام يعين أوزانًا غير موحدة على أزواج المفتاح–القيمة. يوضح أنهفي كل خطوة من خطوات فك التشفير، يتم تجميع أجزاء مختلفة من تسلسل الإدخال بشكل انتقائيفي تجميع الانتباه.

```
_, dec_attention_weights = model.predict_step(
    data.build([engs[-1]], [fras[-1]]), d2l.try_gpu(),
data.num_steps, True)
attention_weights = tf.reshape(
    tf.concat([step[0][0][0] for step in
dec_attention_weights], 0),
    (1, 1, -1, data.num_steps))

# Plus one to include the end-of-sequence token
d2l.show_heatmaps(attention_weights[:, :, :, :len(engs[-
1].split()) + 1],
                  xlabel='Key positions', ylabel='Query
positions')
```



11.4.4. الملخص

- عند توقع رمز token، إذا لم تكن جميع الرموز للإدخال ذات صلة، فإن مشفر–مفكك شفرة RNN مع انتباه Bahdanau تجمع بشكل انتقائي أجزاء مختلفة من تسلسل الإدخال. يتم تحقيق ذلك من خلال معاملة متغير السياق كناتج لتجميع الانتباه الإضافي.

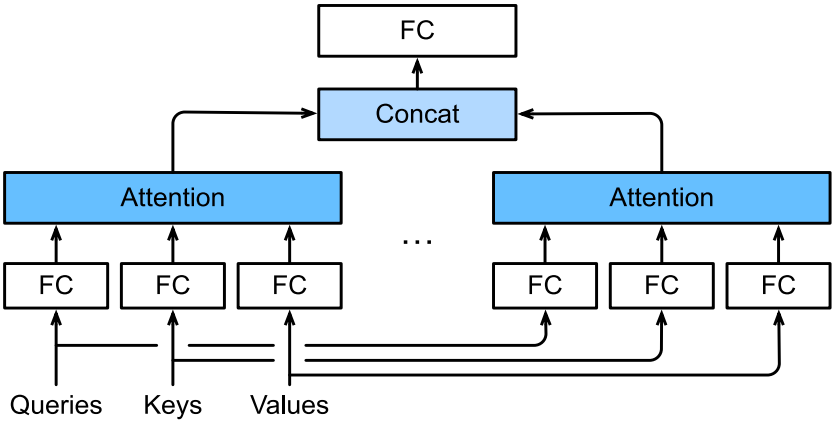
- في مشفر–مفكك شفرةRNN ، يتعامل انتباه Bahdanau مع الحالة المخفية لمفكك الشفرةفي الخطوة الزمنية السابقة على أنها الاستعلام، والحالات المخفيةفي جميع الخطوات الزمنية كمفاتيح وقيم.

11.4.5. التمارين

1. استبدل GRU بـ LSTMفي التجربة.
2. قم بتعديل التجربة لاستبدال دالة تسجيل الانتباه الإضافي additive attention scoring function بالمنتج النقطي المقاس scaled dot-product. كيف تؤثر على كفاءة التدريب؟

11.5. الانتباه متعدد الرؤوس Multi-Head Attention

من الناحية العملية، بالنظر إلى نفس مجموعة الاستعلامات والمفاتيح والقيم، قد نرغبفي أن يجمع نموذجنا المعرفة من السلوكيات المختلفة لنفس آلية الانتباه، مثل التقاط التبعيات من نطاقات مختلفة (على سبيل المثال، المدى الأقصر مقابل المدى الأطول) ضمن تسلسل. وبالتالي، قد يكون من المفيد السماح لآلية انتباهنا بالاستخدام المشترك لمساحات تمثيل فرعية مختلفة من الاستعلامات والمفاتيح والقيم.



الشكل 11.5.1 الانتباه متعدد الرؤوس، حيث يتم ربط عدة رؤوس متسلسلة ثم تحويلها خطيًا.

تحقيقًا لهذه الغاية، بدلاً من إجراء تجميع واحد للانتباه، يمكن تحويل الاستعلامات والمفاتيح والقيم باستخدام الإسقاطات الخطية المكتسبة بشكل مستقل h. ثم يتم إدخال هذه الاستعلامات والمفاتيح والقيم المتوقعة على شكل hفي تجميع الانتباه بشكل متوازٍ.في النهاية، يتم ربط نواتج تجميع الانتباه h وتحويلها بإسقاط خطي مكتسب آخر لإنتاج الناتج النهائي. يُطلق على هذا التصميم الاهتمام متعدد الرؤوس multi-head attention، حيث يكون كل من مخرجات

تجميع الانتباه h رأسًا (Vaswani et al.، 2017). باستخدام طبقات متصلة بالكامل لإجراء تحويلات خطية قابلة للتعلم، يصف الشكل 11.5.1 الانتباه متعدد الرؤوس.

11.5.1. النموذج Model

قبل تقديم الانتباه متعدد الرؤوس، دعنا نضفي الطابع الرسمي على هذا النموذج رياضيًا. بالنظر إلى الاستعلام $\mathbf{q} \in \mathbb{R}^{d_q}$ والمفتاح $\mathbf{k} \in \mathbb{R}^{d_k}$ والقيمة $\mathbf{v} \in \mathbb{R}^{d_v}$، يتم حساب كل رأس انتباه $\mathbf{h}_i (i = 1, \ldots, h)$ على أنه

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v},$$

حيث المعلمات القابلة للتعلم $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$ و$\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$ و$\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$، وهو عبارة عن تجميع للانتباه، مثل الاهتمام الإضافي واهتمام المنتج النقطي المقاس في القسم 11.3. ناتج الانتباه متعدد الرؤوس هو تحول خطي آخر عبر معلمات قابلة للتعلم $\mathbf{W}_o \in \mathbb{R}^{p_o \times hp_v}$ لتسلسل الرؤوس h:

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}.$$

بناءً على هذا التصميم، قد يحضر كل رأس أجزاء مختلفة من الإدخال. يمكن التعبير عن دوال أكثر تعقيدًا من متوسط الوزن البسيط.

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.5.2. التنفيذ Implementation

في تنفيذنا، نختار انتباه المنتج النقطي المقاس لكل رأس من الانتباه متعدد الرؤوس. لتجنب النمو الكبيرفي التكلفة الحسابية وتكلفة المعلمات، قمنا بتعيين $p_q = p_k = p_v = p_o/h$. لاحظ أنه يمكن حساب الرؤوس h بالتوازي إذا قمنا بتعيين عدد مخرجات التحويلات الخطية للاستعلام والمفتاح والقيمة الى $p_q h = p_k h = p_v h = p_o$.في التنفيذ التالي، p_o يتم تحديده عبر الوسيطة `num_hiddens`.

```
#@save
class MultiHeadAttention(d2l.Module):
    """Multi-head attention."""
    def __init__(self, key_size, query_size, value_size,
num_hiddens,
                 num_heads, dropout, bias=False,
**kwargs):
```

```
        super().__init__()
        self.num_heads = num_heads
        self.attention =
d2l.DotProductAttention(dropout, num_heads)
        self.W_q = tf.keras.layers.Dense(num_hiddens,
use_bias=bias)
        self.W_k = tf.keras.layers.Dense(num_hiddens,
use_bias=bias)
        self.W_v = tf.keras.layers.Dense(num_hiddens,
use_bias=bias)
        self.W_o = tf.keras.layers.Dense(num_hiddens,
use_bias=bias)

    def call(self, queries, keys, values, valid_lens,
window_mask=None,
             **kwargs):
        # Shape of queries, keys, or values:
        # (batch_size, no. of queries or key-value
pairs, num_hiddens)
        # Shape of valid_lens: (batch_size,) or
(batch_size, no. of queries)
        # After transposing, shape of output queries,
keys, or values:
        # (batch_size * num_heads, no. of queries or
key-value pairs,
        # num_hiddens / num_heads)
        queries = self.transpose_qkv(self.W_q(queries))
        keys = self.transpose_qkv(self.W_k(keys))
        values = self.transpose_qkv(self.W_v(values))

        if valid_lens is not None:
            # On axis 0, copy the first item (scalar or
vector) for num_heads
            # times, then copy the next item, and so on
            valid_lens = tf.repeat(valid_lens,
repeats=self.num_heads, axis=0)

        # Shape of output: (batch_size * num_heads, no.
of queries,
        # num_hiddens / num_heads)
        output = self.attention(queries, keys, values,
valid_lens,
```

```
                                window_mask, **kwargs)

        # Shape of output_concat: (batch_size, no. of
queries, num_hiddens)
        output_concat = self.transpose_output(output)
        return self.W_o(output_concat)
```

للسماح بالحساب المتوازي لرؤوس متعددة، تستخدم فئة `MultiHeadAttention` المذكورة أعلاه طريقتين للتبديل على النحو المحدد أدناه. على وجه التحديد، تعكس طريقة `transpose_output` عملية طريقة `transpose_qkv`.

```
@d2l.add_to_class(MultiHeadAttention)  #@save
def transpose_qkv(self, X):
    """Transposition for parallel computation of
multiple attention heads."""
    # Shape of input X: (batch_size, no. of queries or
key-value pairs,
    # num_hiddens). Shape of output X: (batch_size, no.
of queries or
    # key-value pairs, num_heads, num_hiddens /
num_heads)
    X = tf.reshape(X, shape=(X.shape[0], X.shape[1],
self.num_heads, -1))
    # Shape of output X: (batch_size, num_heads, no. of
queries or key-value
    # pairs, num_hiddens / num_heads)
    X = tf.transpose(X, perm=(0, 2, 1, 3))
    # Shape of output: (batch_size * num_heads, no. of
queries or key-value
    # pairs, num_hiddens / num_heads)
    return tf.reshape(X, shape=(-1, X.shape[2],
X.shape[3]))

@d2l.add_to_class(MultiHeadAttention)  #@save
def transpose_output(self, X):
    """Reverse the operation of transpose_qkv."""
    X = tf.reshape(X, shape=(-1, self.num_heads,
X.shape[1], X.shape[2]))
    X = tf.transpose(X, perm=(0, 2, 1, 3))
    return tf.reshape(X, shape=(X.shape[0], X.shape[1],
-1))
```

دعونا نختبر فئة `MultiHeadAttention` المنفذة لدينا باستخدام مثال لعبة حيث المفاتيح والقيم هي نفسها. نتيجة لذلك، يكون شكل ناتج الانتباه متعدد الرؤوس هو (حجم_الدفعة `batch_size`، عدد_الاستعلامات `num_queries`، عدد الحالات المخفية `num_hiddens`).

```
num_hiddens, num_heads = 100, 5
attention = MultiHeadAttention(num_hiddens, num_hiddens,
num_hiddens,
                               num_hiddens, num_heads,
0.5)

batch_size, num_queries, num_kvpairs, valid_lens = 2, 4,
6, tf.constant([3, 2])
X = tf.ones((batch_size, num_queries, num_hiddens))
Y = tf.ones((batch_size, num_kvpairs, num_hiddens))
d2l.check_shape(attention(X, Y, Y, valid_lens,
training=False),
                (batch_size, num_queries, num_hiddens))
```

11.5.3. الملخص

- يجمع الانتباه متعدد الرؤوس بين معرفة نفس تجميع الانتباه عبر مساحات تمثيل فرعية مختلفة من الاستعلامات والمفاتيح والقيم.
- لحساب رؤوس متعددة للانتباه متعدد الرؤوس بشكل متوازٍ، هناك حاجة إلى معالجة موتر مناسبة.

11.5.4. التمارين

1. ارسم أوزان الانتباه لرؤوس متعددةفي هذه التجربة.
2. افترض أن لدينا نموذجًا مدربًا يعتمد على الانتباه متعدد الرؤوس ونريد تقليم رؤوس الانتباه الأقل أهمية لزيادة سرعة التنبؤ. كيف يمكننا تصميم تجارب لقياس أهمية رأس الانتباه؟

11.6. الانتباه الذاتي والتشفير الموضعي Self-Attention and Positional Encoding

في التعلم العميق، غالبًا ما نستخدم شبكات CNN أو RNN لتشفير التسلسل sequence. الآن مع آليات الانتباه attention mechanisms، تخيل أننا نقوم بتغذية سلسلة من الرموزفي تجميع الانتباه بحيث تعمل نفس المجموعة من الرموز كاستعلامات ومفاتيح وقيم. على وجه التحديد، يحضر كل استعلام جميع أزواج المفتاح–القيمة ويولد ناتج انتباه واحد. نظرًا لأن

الاستعلامات والمفاتيح والقيم تأتي من نفس المكان، فإن هذا يؤدي إلى الانتباه الذاتي self-attention (Lin et al., 2017، Vaswani et al., 2017)، والذي يُسمى أيضًا الانتباه الداخلي intra-attention (Cheng et al., 2016، Parikh et al., 2016، Paulus et al., 2017).في هذا القسم، سنناقش ترميز التسلسل باستخدام الانتباه الذاتي، بمافي ذلك استخدام معلومات إضافية لترتيب التسلسل.

```
import numpy as np
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.6.1. الانتباه الذاتي Self-Attention

بالنظر إلى تسلسل من رموز الإدخال $\mathbf{x}_1, \ldots, \mathbf{x}_n$ حيث أي $\mathbf{x}_i \in \mathbb{R}^d$ $(1 \leq i \leq n)$ ، فإن الانتباه الذاتي الخاص به ينتج تسلسلًا بنفس الطول $\mathbf{y}_1, \ldots, \mathbf{y}_n$ ، حيث

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d$$

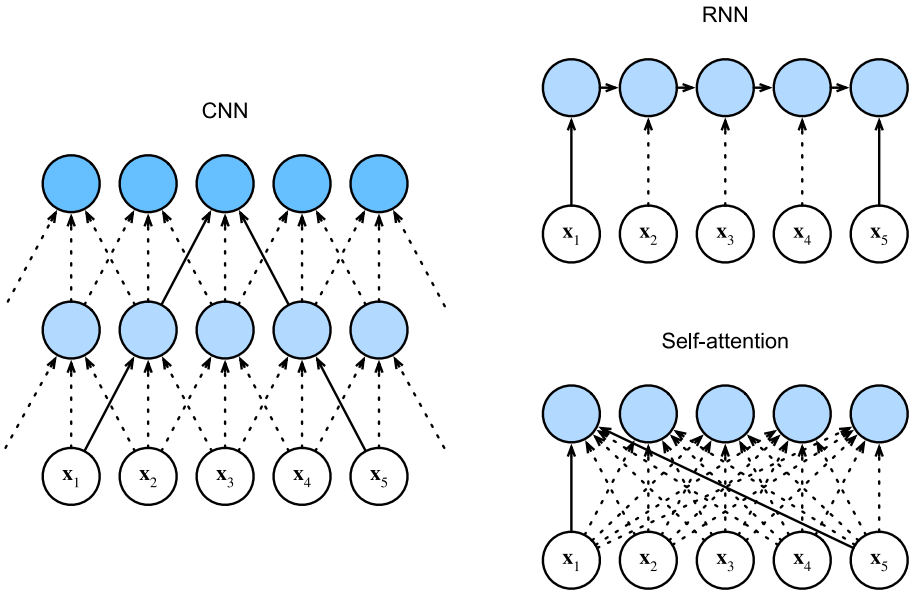
حسب تعريف تجميع الانتباه fفي (11.3.1). باستخدام الانتباه متعدد الرؤوس multi-head attention، يحسب مقتطف الكود التالي الانتباه الذاتي لموتّر ذي شكل (حجم الدُفعة، عدد الخطوات الزمنية أو طول التسلسل بالرموز،d). موتر الإخراج له نفس الشكل.

```
num_hiddens, num_heads = 100, 5
attention = d2l.MultiHeadAttention(num_hiddens,
num_hiddens, num_hiddens,
                                   num_hiddens,
num_heads, 0.5)

batch_size, num_queries, valid_lens = 2, 4,
tf.constant([3, 2])
X = tf.ones((batch_size, num_queries, num_hiddens))
d2l.check_shape(attention(X, X, X, valid_lens,
training=False),
                (batch_size, num_queries, num_hiddens))
```

11.6.2. مقارنة CNNs وRNNs والانتباه الذاتي Comparing CNNs, RNNs, and Self-Attention

دعونا نقارن معماريات لتعيين تسلسل من الرموز إلى تسلسل آخر متساوي الطول، حيث يتم تمثيل كل رمز إدخال أو إخراج بواسطة متجه ذي أبعاد. على وجه التحديد، سننظرفي شبكات CNN وRNNs والانتباه الذاتي. سنقارن التعقيد الحسابي والعمليات المتسلسلة وأطوال المسار القصوى. لاحظ أن العمليات المتسلسلة تمنع الحساب المتوازي،في حين أن المسار

الأقصر بين أي مجموعة من مواضع التسلسل يجعل من السهل تعلم التبعيات بعيدة المدى ضمن التسلسل (Hochreiter et al.، 2001).



الشكل 11.6.1 مقارنة CNN (تم حذف الرموز للحشو) وRNN ومعماريات الانتباه الذاتي.

ضعفي اعتبارك طبقة تلافيفية حجم نواتها k. سنقدم مزيدًا من التفاصيل حول معالجة التسلسل باستخدام شبكات CNNفي فصول لاحقة.في الوقت الحالي، نحتاج فقط إلى معرفة أنه نظرًا لأن طول التسلسل هو n ، فإن عدد قنوات الإدخال والإخراج كلاهما d ، فإن التعقيد الحسابي للطبقة التلافيفية هو d. وكما يوضح الشكل 11.6.1، فإن شبكات CNN مرتبة بشكل هرمي، لذا توجد هناك $\mathcal{O}(1)$ عمليات تسلسلية ويكون أقصى طول للمسير هو $\mathcal{O}(n/k)$. على سبيل المثال، وضمن المجال المؤثر لشبكة CNN ذات طبقتين بحجم نواة 3في الشكل 11.6.1.

عند تحديث الحالة المخفية لـ RNNs، فإن ضرب مصفوفة الوزن $d \times d$ والحالة المخفية ذات الأبعاد d لها تعقيد حسابي قدره $\mathcal{O}(d^2)$. نظرًا لأن طول التسلسل هو n ، فإن التعقيد الحسابي للطبقة المتكررة هو $\mathcal{O}(nd^2)$. وفقاً للشكل 11.6.1، توجد $\mathcal{O}(n)$ عمليات متسلسلة لا يمكن موازنتها ويكون طول المسير الأقصى كذلك $\mathcal{O}(n)$.

في الانتباه الذاتي، تكون الاستعلامات والمفاتيح والقيم كلها مصفوفات $n \times d$. ضعفي اعتبارك انتباه الضرب النقطي المقاس(11.3.5)، حيث يتم ضرب المصفوفة $n \times d$ بمصفوفة $d \times n$ ، ثم يتم ضرب المصفوفة الناتجة $n \times n$في مصفوفة $n \times d$. نتيجة لذلك، الانتباه الذاتي

له تعقيد حسابي $\mathcal{O}(n^2d)$. كما نرى في الشكل 11.6.1، يرتبط كل رمز بشكل مباشر بأي رمز آخر من خلال الانتباه الذاتي. لذلك، يمكن أن يكون الحساب موازيًا للعمليات المتسلسلة $\mathcal{O}(1)$ ويكون الحد الأقصى لطول المسار وكذلك $\mathcal{O}(1)$.

الكل في الكل، تتمتع كل من شبكات CNN والانتباه الذاتي بحساب موازٍ والانتباه الذاتي بأقصر طول للمسار. ومع ذلك، فإن التعقيد الحسابي التربيعي فيما يتعلق بطول التسلسل يجعل الانتباه الذاتي بطيئًا للغاية للتسلسلات الطويلة جدًا.

11.6.3. الترميز الموضعي Positional Encoding

على عكس RNNs التي تعالج بشكل متكرر الرموز للتسلسل واحدًا تلو الآخر، يتخلى الانتباه الذاتي عن العمليات المتسلسلة لصالح الحساب المتوازي. لاستخدام معلومات ترتيب التسلسل، يمكننا حقن معلومات موضعية مطلقة أو نسبية عن طريق إضافة ترميز موضعي positional encoding إلى تمثيلات الإدخال. يمكن تعلم الترميزات الموضعية أو إصلاحها. فيما يلي، نصف ترميزًا موضعيًا ثابتًا استنادًا إلى دوال الجيب وجيب التمام (Vaswani et al.، 2017).

افترض أن تمثيل الإدخال $\mathbf{X} \in \mathbb{R}^{n\times d}$ يحتوي على ذات الأبعاد المضمنة d لرموز n التسلسل. إخراج الترميز الموضعي $\mathbf{X} + \mathbf{P}$ باستخدام مصفوفة تضمين موضعية $\mathbf{P} \in \mathbb{R}^{n\times d}$ من نفس الشكل، يكون العنصر الموجود في الصف i^{th} والعمود $(2j)^{\text{th}}$ أو العمود $(2j+1)^{\text{th}}$ هو

$$\begin{aligned} p_{i,2j} &= \sin\left(\frac{i}{10000^{2j/d}}\right), \\ p_{i,2j+1} &= \cos\left(\frac{i}{10000^{2j/d}}\right). \end{aligned} \tag{11.6.2}$$

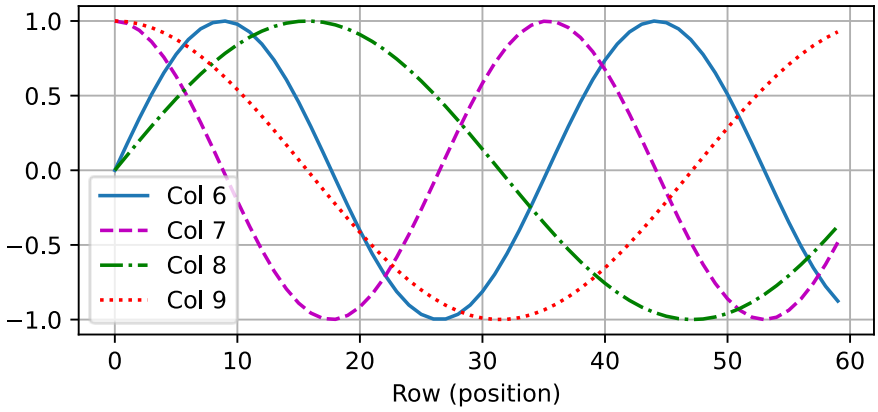
للوهلة الأولى، يبدو تصميم الدالة المثلثية هذا غريبًا. قبل شرح هذا التصميم، دعنا نطبقه أولاً في فئة `PositionalEncoding` التالية.

```
#@save
class PositionalEncoding(tf.keras.layers.Layer):
    """Positional encoding."""
    def __init__(self, num_hiddens, dropout,
max_len=1000):
        super().__init__()
        self.dropout = tf.keras.layers.Dropout(dropout)
        # Create a long enough P
        self.P = np.zeros((1, max_len, num_hiddens))
        X = np.arange(max_len,
dtype=np.float32).reshape(
            -1,1)/np.power(10000, np.arange(
```

```
            0, num_hiddens, 2, dtype=np.float32) /
num_hiddens)
        self.P[:, :, 0::2] = np.sin(X)
        self.P[:, :, 1::2] = np.cos(X)

    def call(self, X, **kwargs):
        X = X + self.P[:, :X.shape[1], :]
        return self.dropout(X, **kwargs)
```

في مصفوفة التضمين الموضعي **P** ، تتوافق الصفوف مع المواضع داخل تسلسل وتمثل الأعمدة أبعاد ترميز موضعية مختلفة.في المثال أدناه، يمكننا أن نرى أن الاعمدة 6^{th} و 7^{th} لمصفوفة التضمين الموضعية وأعمدة لها تردد أعلى من الأعمدة 8^{th} و 9^{th}. يرجع سبب الإزاحة بين 6^{th} و 7^{th} (نفس الأعمدة 8^{th} و9^{th}) إلى تبديل دالات الجيب وجيب التمام.

```
encoding_dim, num_steps = 32, 60
pos_encoding = PositionalEncoding(encoding_dim, 0)
X = pos_encoding(tf.zeros((1, num_steps, encoding_dim)),
training=False)
P = pos_encoding.P[:, :X.shape[1], :]
d2l.plot(np.arange(num_steps), P[0, :, 6:10].T,
xlabel='Row (position)',
         figsize=(6, 2.5), legend=["Col %d" % d for d in
np.arange(6, 10)])
```



11.6.3.1. معلومات الموضع المطلقة Absolute Positional Information

لمعرفة كيف يرتبط التردد المنخفض بشكل رتيب على طول بُعد التشفير بالمعلومات الموضعية المطلقة absolute positional information، دعنا نطبع التمثيلات الثنائية لـ 0,1, ...,7.

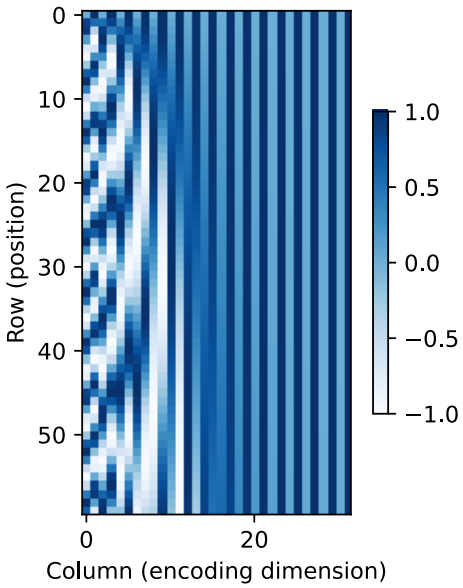
كما نرى، فإن أقل بت، وثاني أدنى بت، وثالث أدنى بت، تتناوب على كل رقم، وكل رقمين، وكل أربعة أرقام، على التوالي.

```
for i in range(8):
    print(f'{i} in binary is {i:>03b}')
```

```
0 in binary is 000
1 in binary is 001
2 in binary is 010
3 in binary is 011
4 in binary is 100
5 in binary is 101
6 in binary is 110
7 in binary is 111
```

في التمثيلات الثنائية، يكون للبت الأعلى تردد أقل من البت الأقل. وبالمثل، كما هو موضحفي خريطة الحرارة أدناه، يقلل الترميز الموضعي الترددات على طول بُعد الترميز باستخدام الدوال المثلثية. نظرًا لأن المخرجات عبارة عن أرقام عائمة، فإن مثل هذه التمثيلات المستمرة تكون أكثر كفاءةفي استخدام المساحة من التمثيلات الثنائية.

```
P = tf.expand_dims(tf.expand_dims(P[0, :, :], axis=0),
axis=0)
d2l.show_heatmaps(P, xlabel='Column (encoding
dimension)',
                  ylabel='Row (position)', figsize=(3.5,
4), cmap='Blues')
```



11.6.3.2. المعلومات الموضعية النسبية Relative Positional Information

إلى جانب التقاط المعلومات الموضعية المطلقة، يتيح الترميز الموضعي أعلاه أيضًا للنموذج أن يتعلم بسهولة الحضور من خلال المواضع النسبية relative positions. هذا لأنه بالنسبة لأي إزاحة موضع ثابت، يمكن تمثيل الترميز الموضعيفي الموضع بإسقاط خطي linear projection لذلك الموضع.

يمكن تفسير هذا الإسقاط رياضيًا. للدلالة $\omega_j = 1/10000^{2j/d}$ ،أي زوج من $(p_{i,2j}, p_{i,2j+1})$في(11.6.2) يمكن إسقاطه خطيًا الى يمكن إسقاط لأي إزاحة ثابتة δ:

$$\begin{aligned}
&\begin{bmatrix} \cos(\delta\omega_j) & \sin(\delta\omega_j) \\ -\sin(\delta\omega_j) & \cos(\delta\omega_j) \end{bmatrix} \begin{bmatrix} p_{i,2j} \\ p_{i,2j+1} \end{bmatrix} \\
&= \begin{bmatrix} \cos(\delta\omega_j)\sin(i\omega_j) + \sin(\delta\omega_j)\cos(i\omega_j) \\ -\sin(\delta\omega_j)\sin(i\omega_j) + \cos(\delta\omega_j)\cos(i\omega_j) \end{bmatrix} \\
&= \begin{bmatrix} \sin((i+\delta)\omega_j) \\ \cos((i+\delta)\omega_j) \end{bmatrix} \\
&= \begin{bmatrix} p_{i+\delta,2j} \\ p_{i+\delta,2j+1} \end{bmatrix},
\end{aligned} \tag{11.6.3}$$

حيث لا تعتمد مصفوفة الإسقاط 2×2 على أي مؤشر موضع i.

11.6.4. الملخص

- في الانتباه الذاتي self-attention، تأتي الاستعلامات والمفاتيح والقيم من نفس المكان.
- تتمتع كل من شبكات CNN والانتباه الذاتي بحسابات موازية ويكون الانتباه الذاتي بأقصر طول للمسار. ومع ذلك، فإن التعقيد الحسابي التربيعي فيما يتعلق بطول التسلسل يجعل الانتباه الذاتي بطيئًا للغاية للتسلسلات الطويلة جدًا.
- لاستخدام معلومات ترتيب التسلسل، يمكننا حقن معلومات موضعية مطلقة أو نسبية عن طريق إضافة ترميز موضعي positional encoding إلى تمثيلات الإدخال.

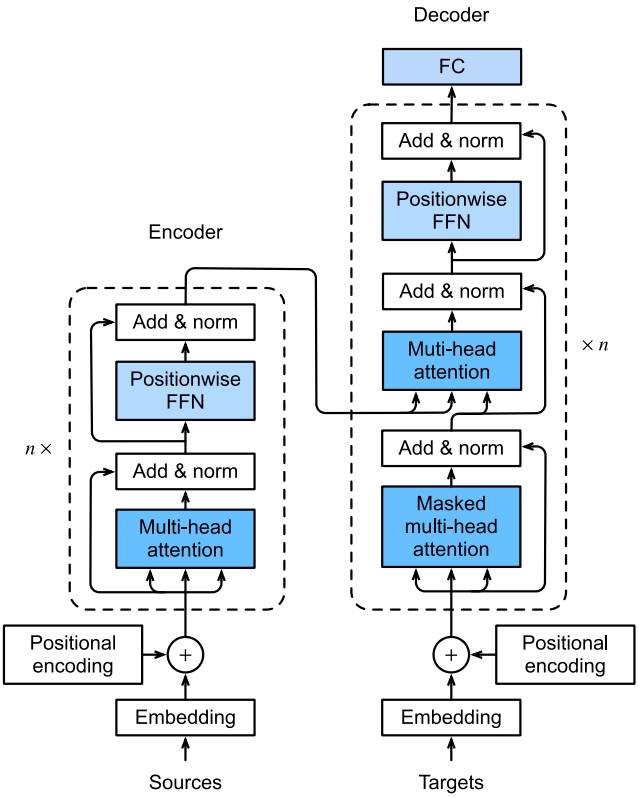
11.6.5. التمارين

1. افترض أننا نصمم بُنية عميقة لتمثيل تسلسل من خلال تكديس طبقات الانتباه الذاتي مع الترميز الموضعي. ماذا يمكن أن تكون المشاكل؟
2. هل يمكنك تصميم طريقة ترميز موضعي قابلة للتعلم؟
3. هل يمكننا تخصيص تضمينات متعلمة learned embeddings مختلفة وفقًا لازاحات مختلفة بين الاستعلامات والمفاتيح التي تتم مقارنتهافي الانتباه الذاتي؟

تلميح: يمكنك الرجوع إلى التضمينات ذات الموضع النسبي (Huang et al.، 2018، Shaw et al.، 2018).

11.7. معمارية المحولات The Transformer Architecture

لقد قارنا شبكات CNN وRNN والانتباه الذاتي في القسم 11.6.2. والجدير بالذكر أن الانتباه الذاتي يتمتع بكل من الحساب المتوازي وأقصر طول أقصى للمسار. لذلك، بطبيعة الحال، من الجذاب تصميم معماريات عميقة باستخدام الانتباه الذاتي. على عكس نماذج الانتباه الذاتي السابقة التي لا تزال تعتمد على RNNs لتمثيل المدخلات (Cheng et al.، 2016، Lin et al.، 2017، Paulus et al.، 2017)، يعتمد نموذج المحولات transformer model فقط على آليات الانتباه دون أي تلافيف أو طبقة متكررة (Vaswani et al، 2017). على الرغم من اقتراحها في الأصل لتعلم التسلسل لتسلسل sequence to sequence learning على البيانات النصية، إلا أن المحولات كانت منتشرة في مجموعة واسعة من تطبيقات التعلم العميق الحديثة، مثل مجالات اللغة والرؤية والكلام والتعلم المعزز.



الشكل 11.7.1 معمارية المحولات.

11.7.1. النموذج Model

كمثال على معمارية المشفر–مفكك الشفرة encoder–decoder architecture، يعرض الشكل 11.7.1 المعمارية الكلية للمحول. كما نرى، يتكون المحول من مشفر ومفكك شفرة. يختلف عن انتباه Bahdanau لتعلم التسلسل إلى التسلسل في الشكل 11.4.1، تتم إضافة إدخال (المصدر) والإخراج (الهدف) المتسلسل مع الترميز الموضعي قبل إدخاله في المشفر ومفكك الشفرة الذي يكدس الوحدات بناءً على الانتباه الذاتي.

نقدم الآن نظرة عامة على بُنية المحولات في الشكل 11.7.1. على مستوى عالٍ، يكون مشفر المحول transformer encoder عبارة عن كومة من طبقات متعددة متطابقة، حيث تحتوي كل طبقة على طبقتين فرعيتين (يُشار إليهما على أنهما طبقة فرعية sublayer). الأول هو تجميع الانتباه الذاتي متعدد الرؤوس والثاني عبارة عن شبكة تغذية للأمام في موضعها. على وجه التحديد، في الانتباه الذاتي للمشفر، تكون الاستعلامات والمفاتيح والقيم كلها من مخرجات طبقة التشفير السابقة. مستوحى من تصميم ResNet في القسم 8.6، يتم استخدام اتصال متبقي حول كلتا الطبقتين الفرعيتين. في المحول، لأي إدخال $\mathbf{x} \in \mathbb{R}^d$ في أي موضع في التسلسل، نطلب $\text{sublayer}(\mathbf{x}) \in \mathbb{R}^d$ حتى يكون الاتصال المتبقي $\mathbf{x} + \text{sublayer}(\mathbf{x}) \in \mathbb{R}^d$ ممكنًا. هذه الإضافة من الاتصال المتبقي يتبعها مباشرة تسوية الطبقة (Ba et al., 2016). نتيجة لذلك، يقوم مشفر المحول بإخراج تمثيل متجه ذات الأبعاد d لكل موضع في تسلسل الإدخال.

مفكك شفرة المحولات transformer decoder هي أيضًا كومة من طبقات متطابقة متعددة مع توصيلات متبقية وتسوية طبقة. إلى جانب الطبقتين الفرعيتين الموصوفتين في المشفر، تُدرج وحدة مفكك الشفرة طبقة فرعية ثالثة، تُعرف باسم انتباه مشفر–مفكك الشفرة encoder–decoder attention، بين هاتين الطبقتين. في انتباه مشفر–مفكك الشفرة، تكون الاستعلامات من مخرجات طبقة وحدة مفكك الشفرة السابقة، والمفاتيح والقيم من مخرجات مشفر المحول. في وحدة مفكك الشفرة، يكون الانتباه الذاتي والاستعلامات والمفاتيح والقيم كلها من مخرجات طبقة مفكك الشفرة السابقة. ومع ذلك، يُسمح لكل موضع في مفكك الشفرة بالحضور فقط إلى جميع المواضع في مفكك الشفرة حتى ذلك الموضع. يحافظ هذا الانتباه المقنع masked attention على خاصية الانحدار التلقائي، مما يضمن أن التنبؤ يعتمد فقط على رموز الإخراج التي تم إنشاؤها.

لقد وصفنا بالفعل ونفذنا الانتباه متعدد الرؤوس استنادًا إلى عمليات ضرب نقطية مقاسة scaled dot-products في القسم 11.5 والترميز الموضعي في القسم 11.6.3. فيما يلي سنقوم بتنفيذ باقي نموذج المحولات.

```
import numpy as np
```

```
import pandas as pd
import tensorflow as tf
from d2l import tensorflow as d2l
```

11.7.2. شبكات التغذية الأمامية الموضعية Positionwise Feed-Forward Networks

تعمل شبكة التغذية الأمامية الموضعية positionwise feed-forward network على تحويل التمثيل في جميع مواضع التسلسل باستخدام نفس MLP. هذا هو السبب في أننا نطلق عليه اسم الموضع positionwise. في التنفيذ أدناه، سيتم تحويل الإدخال X مع الشكل (حجم الدُفعة، عدد الخطوات الزمنية أو طول التسلسل بالرموز، عدد الوحدات المخفية أو بُعد الميزة) بواسطة MLP ثنائي الطبقات إلى شكل موتر إخراج (حجم الدُفعة، عدد الخطوات الزمنية، `ffn_num_outputs`).

```
#@save
class PositionWiseFFN(tf.keras.layers.Layer):
    """Positionwise feed-forward network."""
    def __init__(self, ffn_num_hiddens,
ffn_num_outputs):
        super().__init__()
        self.dense1 =
tf.keras.layers.Dense(ffn_num_hiddens)
        self.relu = tf.keras.layers.ReLU()
        self.dense2 =
tf.keras.layers.Dense(ffn_num_outputs)

    def call(self, X):
        return self.dense2(self.relu(self.dense1(X)))
```

يوضح المثال التالي أن البعد الأعمق للموتر يتغير إلى عدد المخرجات في شبكة التغذية الأمامية الموضعية. نظرًا لأن نفس MLP يتحول في جميع المواضع، عندما تكون المدخلات في جميع هذه المواضع هي نفسها، فإن مخرجاتها متطابقة أيضًا.

```
ffn = PositionWiseFFN(4, 8)
ffn(tf.ones((2, 3, 4)))[0]
```

```
<tf.Tensor: shape=(3, 8), dtype=float32, numpy=
array([[ 0.31384668,  0.01483253, -0.38529113,
0.22501771, -0.0185459 ,
        -0.15193778, -0.65797377, -0.7980111 ],
       [ 0.31384668,  0.01483253, -0.38529113,
0.22501771, -0.0185459 ,
        -0.15193778, -0.65797377, -0.7980111 ],
```

```
       [ 0.31384668,  0.01483253, -0.38529113,
0.22501771, -0.0185459 ,
        -0.15193778, -0.65797377, -0.7980111 ]],
dtype=float32)>
```

11.7.3. الاتصال المتبقي وتسوية الطبقة Residual Connection and Layer Normalization

دعنا الآن نركز على مكون "الإضافة والمعيار add & norm "في الشكل 11.7.1. كما وصفنافي بداية هذا القسم، هذا اتصال متبقي residual connection يتبعه مباشرة تسوية طبقة layer normalization. كلاهما مفتاح للبنى العميقة الفعالة.

في القسم 8.5، أوضحنا كيف أحدث تسوية الدُفعات وإعادة المقاييس عبر الأمثلة داخل minibatch. كما تمت مناقشتهفي القسم 8.5.2.3، فإن تسوية الطبقة هي نفسها تسوية الدُفعات باستثناء أن الأول يتم تسويته عبر بُعد الميزة، وبالتالي الاستمتاع بفوائد استقلالية المقياس واستقلالية حجم الدُفعة. على الرغم من تطبيقاته المنتشرةفي الرؤية الحاسوبية، فإن تسوية الدفعات عادة ما يكون أقل فعالية من الناحية التجريبية من تسوية الطبقةفي مهام معالجة اللغة الطبيعية، والتي غالبًا ما تكون مدخلاتها متواليات متغيرة الطول.

يقارن مقتطف الشفرة التالي التسوية عبر أبعاد مختلفة حسب تسوية الطبقة وتسوية الدُفعات.

```
ln = tf.keras.layers.LayerNormalization()
bn = tf.keras.layers.BatchNormalization()
X = tf.constant([[1, 2], [2, 3]], dtype=tf.float32)
print('layer norm:', ln(X), '\nbatch norm:', bn(X))
```

```
layer norm: tf.Tensor(
[[-0.998006   0.9980061]
 [-0.9980061  0.998006 ]], shape=(2, 2), dtype=float32)
batch norm: tf.Tensor(
[[0.99950033 1.9990007 ]
 [1.9990007  2.998501  ]], shape=(2, 2), dtype=float32)
```

الآن يمكننا تنفيذ فئة `AddNorm` باستخدام اتصال متبقي متبوعًا بتسوية الطبقة. يتم تطبيق التسرب Dropout أيضًا للتنظيم regularization.

```
#@save
class AddNorm(tf.keras.layers.Layer):
    """Residual connection followed by layer
normalization."""
    def __init__(self, norm_shape, dropout):
        super().__init__()
        self.dropout = tf.keras.layers.Dropout(dropout)
```

```
        self.ln =
tf.keras.layers.LayerNormalization(norm_shape)

    def call(self, X, Y, **kwargs):
        return self.ln(self.dropout(Y, **kwargs) + X)
```

يتطلب الاتصال المتبقي أن يكون المدخلان من نفس الشكل بحيث يكون لموتّر الإخراج أيضًا نفس الشكل بعد عملية الإضافة.

```
# Normalized_shape is: [i for i in
range(len(input.shape))][1:]
add_norm = AddNorm([1, 2], 0.5)
d2l.check_shape(add_norm(tf.ones((2, 3, 4)), tf.ones((2,
3, 4)),
                         training=False), (2, 3, 4))
```

11.7.4. المشفر Encoder

مع جميع المكونات الأساسية لتجميع مشفر المحولات، فلنبدأ بتنفيذ طبقة واحدة داخل المشفر. تحتوي فئة `TransformerEncoderBlock` التالية على طبقتين فرعيتين: الانتباه الذاتي متعدد الرؤوس وشبكات التغذية الأمامية الموضعية ، حيث يتم استخدام اتصال متبقي متبوعًا بتسوية الطبقة حول كلتا الطبقتين الفرعيتين.

```
#@save
class TransformerEncoderBlock(tf.keras.layers.Layer):
    """Transformer encoder block."""
    def __init__(self, key_size, query_size, value_size,
num_hiddens,
                 norm_shape, ffn_num_hiddens, num_heads,
dropout, bias=False):
        super().__init__()
        self.attention = d2l.MultiHeadAttention(
            key_size, query_size, value_size,
num_hiddens, num_heads, dropout,
            bias)
        self.addnorm1 = AddNorm(norm_shape, dropout)
        self.ffn = PositionWiseFFN(ffn_num_hiddens,
num_hiddens)
        self.addnorm2 = AddNorm(norm_shape, dropout)

    def call(self, X, valid_lens, **kwargs):
        Y = self.addnorm1(X, self.attention(X, X, X,
valid_lens, **kwargs),
```

```
                                **kwargs)
        return self.addnorm2(Y, self.ffn(Y), **kwargs)
```

كما نرى، أي طبقةفي مشفر المحول لا تغير شكل المدخلات الخاصة بها.

```
X = tf.ones((2, 100, 24))
valid_lens = tf.constant([3, 2])
norm_shape = [i for i in range(len(X.shape))][1:]
encoder_blk = TransformerEncoderBlock(24, 24, 24, 24,
norm_shape, 48, 8, 0.5)
d2l.check_shape(encoder_blk(X, valid_lens,
training=False), X.shape)
```

في تنفيذ مشفر المحولات التالي، نقوم بتكديس عدد من مثيلات فئات `TransformerEncoderBlock` المذكورة أعلاه. نظرًا لأننا نستخدم الترميز الموضعي الثابت الذي تكون قيمه دائمًا بين `-1` و `1`، فإننا نضرب قيم تضمين المدخلات القابلة للتعلم بواسطة الجذر التربيعي لبعد التضمين لإعادة القياس قبل تلخيص تضمين الإدخال والترميز الموضعي.

```
#@save
class TransformerEncoder(d2l.Encoder):
    """Transformer encoder."""
    def __init__(self, vocab_size, key_size, query_size,
value_size,
                 num_hiddens, norm_shape,
ffn_num_hiddens, num_heads,
                 num_blks, dropout, bias=False):
        super().__init__()
        self.num_hiddens = num_hiddens
        self.embedding =
tf.keras.layers.Embedding(vocab_size, num_hiddens)
        self.pos_encoding =
d2l.PositionalEncoding(num_hiddens, dropout)
        self.blks = [TransformerEncoderBlock(
            key_size, query_size, value_size,
num_hiddens, norm_shape,
            ffn_num_hiddens, num_heads, dropout, bias)
for _ in range(
            num_blks)]

    def call(self, X, valid_lens, **kwargs):
        # Since positional encoding values are between -
1 and 1, the embedding
```

```
        # values are multiplied by the square root of
the embedding dimension
        # to rescale before they are summed up
        X = self.pos_encoding(self.embedding(X) *
tf.math.sqrt(
            tf.cast(self.num_hiddens,
dtype=tf.float32)), **kwargs)
        self.attention_weights = [None] * len(self.blks)
        for i, blk in enumerate(self.blks):
            X = blk(X, valid_lens, **kwargs)
            self.attention_weights[
                i] =
blk.attention.attention.attention_weights
        return X
```

أدناه نحدد معلمات فائقة لإنشاء مشفر المحول من طبقتين. شكل إخراج مشفر المحول هو (حجم الدُفعة، عدد الخطوات الزمنية، num_hiddens).

11.7.5. مفكك الشفرة Decoder

كما هو مبين في الشكل 11.7.1، يتكون مفكك شفرة المحول من عدة طبقات متطابقة. يتم تنفيذ كل طبقة في فئة TransformerDecoderBlock التالية ، والتي تحتوي على ثلاث طبقات فرعية: الانتباه الذاتي لمفكك الشفرة ، وانتباه المشفر ـ مفكك الشفرة ، وشبكات التغذية الأمامية الموضعية. تستخدم هذه الطبقات الفرعية اتصالًا متبقيًا حولها متبوعًا بتطبيع الطبقة.

كما وصفنا سابقًا في هذا القسم، في مفكك الشفرة الانتباه الذاتي متعددة الرؤوس المقنعة (الطبقة الفرعية الأولى)، تأتي الاستعلامات والمفاتيح والقيم من مخرجات طبقة مفكك الشفرة السابقة. عند تدريب نماذج التسلسل إلى التسلسل، تعرف الرموز في جميع المواضع (الخطوات الزمنية) لتسلسل الإخراج. ومع ذلك، أثناء التنبؤ، يتم إنشاء تسلسل الإخراج رمزًا؛ وبالتالي، في أي خطوة زمنية لمفكك الشفرة، يمكن استخدام الرموز التي تم إنشاؤها فقط في الانتباه الذاتي لمفكك الشفرة. للحفاظ على الانحدار التلقائي في مفكك الشفرة، يحدد الانتباه الذاتي المقنع dec_valid_lens بحيث لا يحضر أي استعلام سوى جميع المواضع في مفكك الشفرة حتى موضع الاستعلام.

```
class TransformerDecoderBlock(tf.keras.layers.Layer):
    # The i-th block in the transformer decoder
    def __init__(self, key_size, query_size, value_size,
num_hiddens,
                 norm_shape, ffn_num_hiddens, num_heads,
dropout, i):
```

```
        super().__init__()
        self.i = i
        self.attention1 = d2l.MultiHeadAttention(
            key_size, query_size, value_size,
num_hiddens, num_heads, dropout)
        self.addnorm1 = AddNorm(norm_shape, dropout)
        self.attention2 = d2l.MultiHeadAttention(
            key_size, query_size, value_size,
num_hiddens, num_heads, dropout)
        self.addnorm2 = AddNorm(norm_shape, dropout)
        self.ffn = PositionWiseFFN(ffn_num_hiddens,
num_hiddens)
        self.addnorm3 = AddNorm(norm_shape, dropout)

    def call(self, X, state, **kwargs):
        enc_outputs, enc_valid_lens = state[0], state[1]
        # During training, all the tokens of any output
sequence are processed
        # at the same time, so state[2][self.i] is None
as initialized. When
        # decoding any output sequence token by token
during prediction,
        # state[2][self.i] contains representations of
the decoded output at
        # the i-th block up to the current time step
        if state[2][self.i] is None:
            key_values = X
        else:
            key_values = tf.concat((state[2][self.i],
X), axis=1)
        state[2][self.i] = key_values
        if kwargs["training"]:
            batch_size, num_steps, _ = X.shape
            # Shape of dec_valid_lens: (batch_size,
num_steps), where every
            # row is [1, 2, ..., num_steps]
            dec_valid_lens = tf.repeat(
                tf.reshape(tf.range(1, num_steps + 1),
                           shape=(-1, num_steps)),
repeats=batch_size, axis=0)
        else:
            dec_valid_lens = None
```

```
        # Self-attention
        X2 = self.attention1(X, key_values, key_values,
dec_valid_lens,
                             **kwargs)
        Y = self.addnorm1(X, X2, **kwargs)
        # Encoder-decoder attention. Shape of
enc_outputs:
        # (batch_size, num_steps, num_hiddens)
        Y2 = self.attention2(Y, enc_outputs,
enc_outputs, enc_valid_lens,
                             **kwargs)
        Z = self.addnorm2(Y, Y2, **kwargs)
        return self.addnorm3(Z, self.ffn(Z), **kwargs),
state
```

لتسهيل عمليات الضرب النقطي المقاسةفي عمليات المشفر_مفكك الشفرة للانتباه والإضافةفي التوصيلات المتبقية، يكون بُعد السمة (num_hiddens) لمفكك الشفرة هو نفس بُعد المشفر.

```
decoder_blk = TransformerDecoderBlock(24, 24, 24, 24,
[1, 2], 48, 8, 0.5, 0)
X = tf.ones((2, 100, 24))
state = [encoder_blk(X, valid_lens), valid_lens, [None]]
d2l.check_shape(decoder_blk(X, state,
training=False)[0], X.shape)
```

الآن نقوم ببناء مفكك شفرة المحولات بالكامل المكونة من num_blks مثيلات TransformerDecoderBlock.في النهاية، طبقة متصلة بالكامل تحسب التنبؤ لجميع رموز الإخراج الممكنة بحجم vocab_size. يتم تخزين كل من أوزان الانتباه الذاتي لمفكك الشفرة وأوزان الانتباه الخاصة بالمشفر_مفكك الشفرة من أجل الرسم اللاحق.

```
class TransformerDecoder(d2l.AttentionDecoder):
    def __init__(self, vocab_size, key_size, query_size,
value_size,
                 num_hiddens, norm_shape,
ffn_num_hiddens, num_heads,
                 num_blks, dropout):
        super().__init__()
        self.num_hiddens = num_hiddens
        self.num_blks = num_blks
        self.embedding =
tf.keras.layers.Embedding(vocab_size, num_hiddens)
```

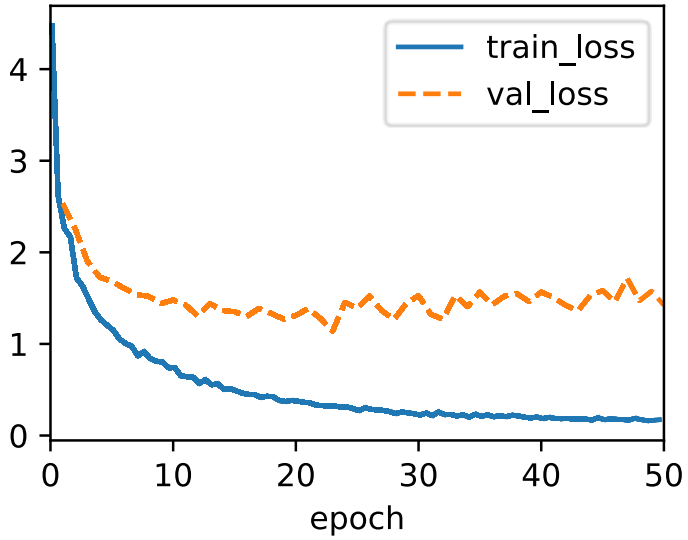
```
        self.pos_encoding =
d2l.PositionalEncoding(num_hiddens, dropout)
        self.blks = [TransformerDecoderBlock(
            key_size, query_size, value_size,
num_hiddens, norm_shape,
            ffn_num_hiddens, num_heads, dropout, i)
                    for i in range(num_blks)]
        self.dense = tf.keras.layers.Dense(vocab_size)

    def init_state(self, enc_outputs, enc_valid_lens):
        return [enc_outputs, enc_valid_lens, [None] *
self.num_blks]

    def call(self, X, state, **kwargs):
        X = self.pos_encoding(self.embedding(X) *
tf.math.sqrt(
            tf.cast(self.num_hiddens,
dtype=tf.float32)), **kwargs)
        # 2 attention layers in decoder
        self._attention_weights = [[None] *
len(self.blks) for _ in range(2)]
        for i, blk in enumerate(self.blks):
            X, state = blk(X, state, **kwargs)
            # Decoder self-attention weights
            self._attention_weights[0][i] = (

blk.attention1.attention.attention_weights)
            # Encoder-decoder attention weights
            self._attention_weights[1][i] = (

blk.attention2.attention.attention_weights)
        return self.dense(X), state

    @property
    def attention_weights(self):
        return self._attention_weights
```

11.7.6. التدريب Training

لنبدأ نموذج المشفر–مفكك الشفرة باتباع بُنية المحول. نحدد هنا أن كلاً من مشفر المحول ومفكك شفرة المحول لهما طبقتان باستخدام الانتباه رباعي الرؤوس. على غرار القسم 10.7.6،

نقوم بتدريب نموذج المحول على تعلم التسلسل لتسلسل على مجموعة بيانات الترجمة الآلية الإنجليزية–الفرنسية.

```
data = d2l.MTFraEng(batch_size=128)
num_hiddens, num_blks, dropout = 256, 2, 0.2
ffn_num_hiddens, num_heads = 64, 4
key_size, query_size, value_size = 256, 256, 256
norm_shape = [2]
with d2l.try_gpu():
    encoder = TransformerEncoder(
        len(data.src_vocab), key_size, query_size,
value_size, num_hiddens,
        norm_shape, ffn_num_hiddens, num_heads,
num_blks, dropout)
    decoder = TransformerDecoder(
        len(data.tgt_vocab), key_size, query_size,
value_size, num_hiddens,
        norm_shape, ffn_num_hiddens, num_heads,
num_blks, dropout)
    model = d2l.Seq2Seq(encoder, decoder,
tgt_pad=data.tgt_vocab['<pad>'],
                        lr=0.001)
trainer = d2l.Trainer(max_epochs=50,
gradient_clip_val=1)
trainer.fit(model, data)
```



بعد التدريب، نستخدم نموذج المحول لترجمة بعض الجمل الإنجليزية إلى الفرنسية وحساب نقاطهم في BLEU.

```
engs = ['go .', 'i lost .', 'he\'s calm .', 'i\'m home
.']
fras = ['va !', 'j\'ai perdu .', 'il est calme .', 'je
suis chez moi .']
preds, _ = model.predict_step(
    data.build(engs, fras), d2l.try_gpu(),
data.num_steps)
for en, fr, p in zip(engs, fras, preds):
    translation = []
    for token in data.tgt_vocab.to_tokens(p):
        if token == '<eos>':
            break
        translation.append(token)
    print(f'{en} => {translation}, bleu,'
          f'{d2l.bleu(" ".join(translation), fr,
k=2):.3f}')
```

```
go . => ['va', '!'], bleu,1.000
i lost . => ["j'ai", 'perdu', '.'], bleu,1.000
he's calm . => ['il', 'est', 'mouillé', 'est',
'mouillé', 'est', 'mouillé', 'mouillé', '.'], bleu,0.343
i'm home . => ['je', 'suis', 'chez', 'moi', 'suis',
'chez', 'moi', 'suis', 'chez'], bleu,0.522
```

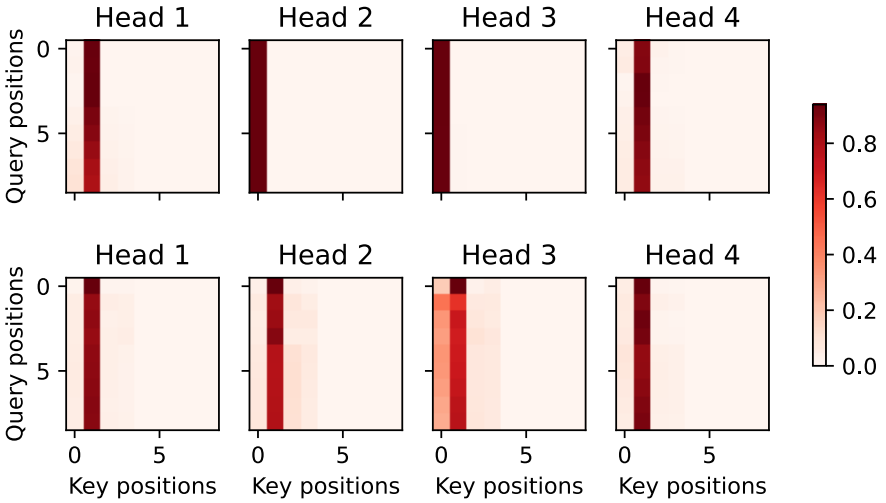
دعونا نتخيل أوزان انتباه المحولات عند ترجمة الجملة الإنجليزية الأخيرة إلى الفرنسية. شكل أوزان الانتباه الذاتي للمشفر هو (عدد طبقات المشفر، وعدد رؤوس الانتباه، وعدد الخطوات أو عدد الاستعلامات، وعدد الخطوات أو عدد أزواج المفتاح_القيمة).

```
_, dec_attention_weights = model.predict_step(
    data.build([engs[-1]], [fras[-1]]), d2l.try_gpu(),
data.num_steps, True)
enc_attention_weights = tf.reshape(
    tf.concat(model.encoder.attention_weights, 0),
    (num_blks, num_heads, -1, data.num_steps))
d2l.check_shape(enc_attention_weights,
                (num_blks, num_heads, data.num_steps,
data.num_steps))
```

في الانتباه الذاتي للمشفر، تأتي كل من الاستعلامات والمفاتيح من تسلسل الإدخال نفسه. نظرًا لأن الرموز للحشو لا تحمل معنى، مع تحديد طول صالح لتسلسل الإدخال، فلا يوجد استعلام

يحضر لمواضع الرموز للحشو. فيما يلي، يتم تقديم طبقتين من أوزان الانتباه متعددة الرؤوس صفًا تلو الآخر. يحضر كل رئيس بشكل مستقل بناءً على فضاءات تمثيل منفصلة من الاستعلامات والمفاتيح والقيم.

```
d2l.show_heatmaps(
    enc_attention_weights, xlabel='Key positions',
ylabel='Query positions',
    titles=['Head %d' % i for i in range(1, 5)],
figsize=(7, 3.5))
```



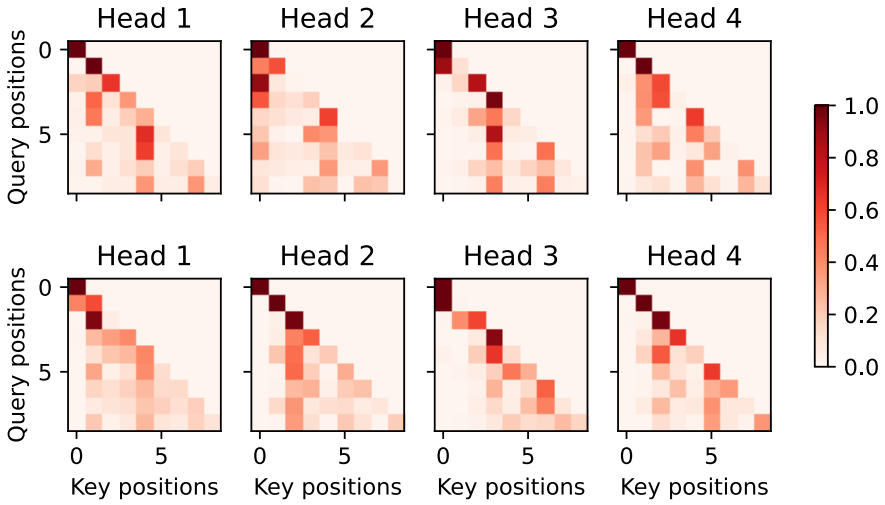
لرسم كل من أوزان الانتباه الذاتي مفكك الشفرة وأوزان الانتباه الخاصة بالمشفر_مفكك الشفرة، نحتاج إلى مزيد من التلاعب بالبيانات. على سبيل المثال، نملأ أوزان الانتباه المقنعة بصفر. لاحظ أن أوزان الانتباه الذاتي لمفكك الشفرة وأوزان الانتباه بالمشفر_مفكك الشفرة لها نفس الاستعلامات: رمز بداية التسلسل متبوعًا برموز الإخراج وربما الرموز لنهاية التسلسل.

```
dec_attention_weights_2d = [head[0] for step in
dec_attention_weights
                            for attn in step
                            for blk in attn for head in
blk]
dec_attention_weights_filled = tf.convert_to_tensor(

np.asarray(pd.DataFrame(dec_attention_weights_2d).fillna
(
        0.0).values).astype(np.float32))
```

```
dec_attention_weights =
tf.reshape(dec_attention_weights_filled, shape=(
    -1, 2, num_blks, num_heads, data.num_steps))
dec_self_attention_weights, dec_inter_attention_weights
= tf.transpose(
    dec_attention_weights, perm=(1, 2, 3, 0, 4))

d2l.check_shape(dec_self_attention_weights,
                (num_blks, num_heads, data.num_steps,
data.num_steps))
d2l.check_shape(dec_inter_attention_weights,
                (num_blks, num_heads, data.num_steps,
data.num_steps))
```

نظرًا لخاصية الانحدار التلقائي للانتباه الذاتي لمفكك الشفرة، لا يوجد استعلام يحضر أزواج القيمة والمفتاح بعد موضع الاستعلام.

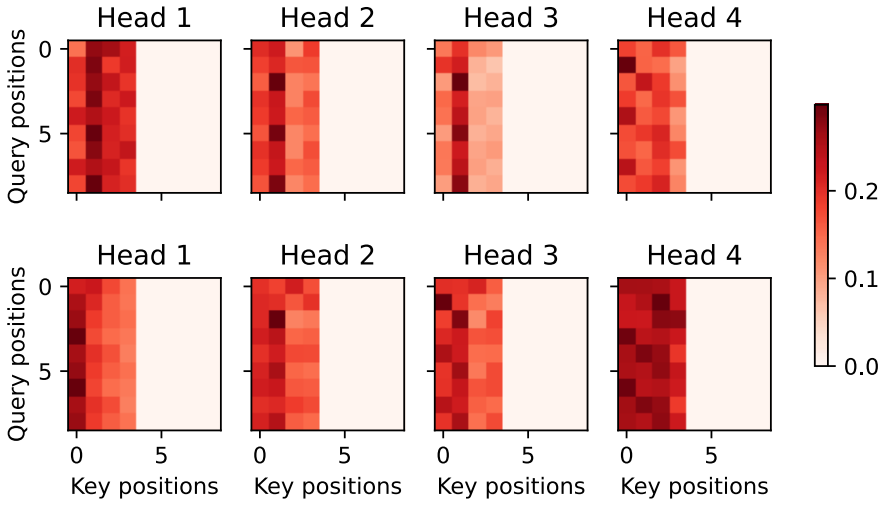
```
d2l.show_heatmaps(
    dec_self_attention_weights[:, :, :, :],
    xlabel='Key positions', ylabel='Query positions',
    titles=['Head %d' % i for i in range(1, 5)],
figsize=(7, 3.5))
```



على غرار الحالةفي الانتباه الذاتي للمشفر، عبر الطول الصحيح المحدد لتسلسل الإدخال، لا يوجد استعلام من تسلسل الإخراج يحضر تلك الرموز للحشو من تسلسل الإدخال.

```
d2l.show_heatmaps(
    dec_inter_attention_weights, xlabel='Key positions',
```

```
    ylabel='Query positions', titles=['Head %d' % i for
i in range(1, 5)],
    figsize=(7, 3.5))
```



على الرغم من اقتراح بُنية المحولاتفي الأصل للتعلم من التسلسل إلى التسلسل، كما سنكتشف لاحقًافي الكتاب، غالبًا ما يتم استخدام إما مشفر المحولات أو مفكك شفرة المحولات بشكل فرديفي مهام التعلم العميق المختلفة.

11.7.7. الملخص

- المحول transformer هو مثال على بُنية المشفر–مفكك الشفرة encoder–decoder architecture، على الرغم من أنه يمكن استخدام المشفر أو مفكك الشفرة بشكل فرديفي الممارسة العملية.
- في المحول، يتم استخدام الانتباه الذاتي متعدد الرؤوس لتمثيل تسلسل الإدخال وتسلسل الإخراج، على الرغم من أن مفكك الشفرة يجب أن يحافظ على خاصية الانحدار التلقائي عبر إصدار مقنع.
- تعتبر كل من الاتصالات المتبقية وتسوية الطبقةفي المحول مهمة لتدريب نموذج عميق للغاية.
- تعمل شبكة التغذية الأمامية الموضعيةفي نموذج المحولات على تحويل التمثيلفي جميع مواضع التسلسل باستخدام نفس MLP.

11.7.8. التمارين

1. قم بتدريب محول أعمق في التجارب. كيف تؤثر على سرعة التدريب وأداء الترجمة؟
2. هل من الجيد استبدال الانتباه المحسّن للضرب النقطي بانتباه إضافي في المحول؟ لماذا؟
3. لنمذجة اللغة، هل يجب أن نستخدم مشفر المحول أو مفكك شفرة المحول أو كليهما؟ كيف تصمم هذه الطريقة؟
4. ما الذي يمكن أن يمثل تحديات للمحولات إذا كانت تسلسلات الإدخال طويلة جدًا؟ لماذا؟
5. كيفية تحسين كفاءة الحوسبة والذاكرة للمحولات؟ تلميح: يمكنك الرجوع إلى مقالة الاستطلاع التي أعدها Tay et al. (Tay et al، 2020).

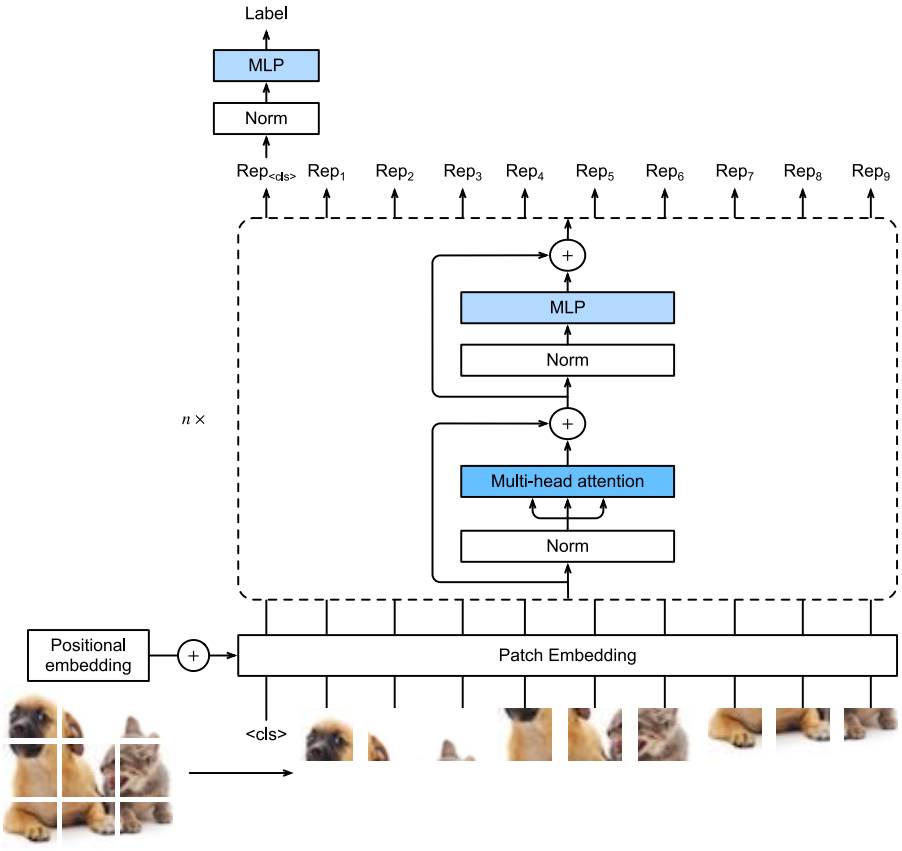
11.8 محولات الرؤية Transformers for Vision

تم اقتراح بنية المحولات في البداية لتعلم التسلسل الى تسلسل، مثل الترجمة الآلية. مع الفعالية العالية، أصبحت المحولات لاحقًا النموذج المفضل في العديد من مهام معالجة اللغة الطبيعية (Brown et al.، 2020، Devlin et al.، 2018، Radford et al.، 2018، Radford et al.، 2019، Raffel et al.، 2020). ومع ذلك، في مجال الرؤية الحاسوبية، استندت العمارة المهيمنة إلى شبكات CNN (القسم 8). هل يمكننا تكييف المحولات لبيانات الصورة النموذجية؟ أثار هذا السؤال اهتمامًا كبيرًا في مجتمع الرؤية الحاسوبية. (Ramachandran et al.، 2019) لاستبدال الالتفاف بالانتباه الذاتي. ومع ذلك، فإن استخدامه للأنماط المتخصصة في الانتباه يجعل من الصعب توسيع نطاق النماذج على مسرعات الأجهزة. (Cordonnier et al.، 2020) أثبت نظريًا أن الانتباه الذاتي يمكن أن يتعلم التصرف بشكل مشابه للالتفاف. بشكل تجريبي، تم أخذ الرقع (patches) 2×2 من الصور كمدخلات، لكن حجم الرقعة الصغير يجعل النموذج قابل للتطبيق فقط على بيانات الصورة ذات الدقة المنخفضة.

بدون قيود محددة على حجم الرقعة، تستخرج محولات الرؤية vision transformers (ViTs) رقعاً من الصور وتغذيها في مشفر المحول transformer encoder للحصول على تمثيل عالمي، والذي سيتم تحويله أخيرًا من أجل التصنيف (Dosovitskiy et al.، 2021). والجدير بالذكر أن المحولات تُظهر قابلية تطوير أفضل من شبكات CNN: عند تدريب نماذج أكبر على مجموعات بيانات أكبر، تتفوق محولات الرؤية على شبكات ResNets بهامش كبير. على غرار الارضية لتصميم معمارية الشبكة في معالجة اللغة الطبيعية، أصبحت المحولات أيضًا مغيرًا للعبة في الرؤية الحاسوبية.

11.8.1. النموذج Model

الشكل 11.8.1 يصور العمارة النموذجية لمحولات الرؤية vision transformers. تتكون هذه البنية من جذع stem يقوم بتحويل الصور الى رقع patches، وجسم body يعتمد على مشفر المحولات متعدد الطبقات multi-layer transformer encoder، ورأس head يحول التمثيل العام إلى تسمية الإخراج output label.



شكل 11.8.1 معمارية محولات الرؤية.في هذا المثال، يتم تقسيم الصورة إلى 9 رقع patches. يتم تحويل الرمز المميز "<cls>" و9 رقع صورة مسطحة عبر تضمين التصحيح وn كتل مشفر المحولات إلى 10 تمثيلات، على التوالي. يتم أيضًا تحويل تمثيل "<cls>" إلى تسمية الإخراج.

ضعفي اعتبارك صورة إدخال بالارتفاع h والعرض w والقنوات c. تحديد ارتفاع وعرض الرقعة على حد سواء، يتم تقسيم الصورة إلى سلسلة من الرقع $m = hw/p^2$ ، حيث يتم تسوية كل رقعة إلى متجه طول cp^2. بهذه الطريقة، يمكن معالجة رقع الصور بشكل مشابه للرموز tokens في تسلسل النص بواسطة مشفر المحولات transformer encoders. يتم عرض علامة مميزة

خاصة "<cls>" (فئة) ورقع الصورة المسطحة m خطيًافي سلسلة من المتجهات $m+1$ ، مُلخَّصة بتضمينات موضعية قابلة للتعلم. يحول مشفر المحول متعدد الطبقات متجهات الإدخال إلى نفس المقدار من تمثيلات متجه الإخراج بنفس الطول. إنه يعمل تمامًا بنفس طريقة مشفر المحول الأصليفي الشكل 11.7.1، ويختلف فقطفي موضع التسوية. نظرًا لأن الرمز "<cls>" يحضر جميع رقع الصور عبر الانتباه الذاتي (انظر الشكل 11.6.1)، فإن تمثيله من خرج مشفر المحول سيتحول إلى تسمية الإخراج.

```
import torch
from torch import nn
from d2l import torch as d2l
```

11.8.2. تضمين الرقعة Patch Embedding

لتنفيذ محول الرؤية، لنبدأ بتضمين الرقعةفي الشكل 11.8.1. يمكن تبسيط تقسيم الصورة إلى رقع وإسقاط هذه الرقع المسطحة خطيًا كعملية التفاف واحدة، حيث يتم تعيين كل من حجم النواة وحجم الخطوة على حجم الرقعة patch size.

```
class PatchEmbedding(nn.Module):
    def __init__(self, img_size=96, patch_size=16,
num_hiddens=512):
        super().__init__()
        def _make_tuple(x):
            if not isinstance(x, (list, tuple)):
                return (x, x)
            return x
        img_size, patch_size = _make_tuple(img_size),
_make_tuple(patch_size)
        self.num_patches = (img_size[0] //
patch_size[0]) * (
            img_size[1] // patch_size[1])
        self.conv = nn.LazyConv2d(num_hiddens,
kernel_size=patch_size,
                                  stride=patch_size)

    def forward(self, X):
        # Output shape: (batch size, no. of patches, no.
of channels)
        return self.conv(X).flatten(2).transpose(1, 2)
```

في المثال التالي، أخذ صور بارتفاع وعرض img_size كمدخلات ، مخرجات تضمين الرقعة (img_size//patch_size)**2 رقع يتم عرضها خطيًا على متجهات بطول num_hiddens.

```
img_size, patch_size, num_hiddens, batch_size = 96, 16,
512, 4
patch_emb = PatchEmbedding(img_size, patch_size,
num_hiddens)
X = torch.randn(batch_size, 3, img_size, img_size)
d2l.check_shape(patch_emb(X),
                (batch_size, (img_size//patch_size)**2,
num_hiddens))
```

```
/home/d2l-worker/miniconda3/envs/d2l-en-release-
0/lib/python3.9/site-
packages/torch/nn/modules/lazy.py:178: UserWarning: Lazy
modules are a new feature under heavy development so
changes to the API or functionality can happen at any
moment.
  warnings.warn('Lazy modules are a new feature under
heavy development '
```

11.8.3. مشفر محول الرؤية Vision Transformer Encoder

يختلف MLP الخاص بمشفر محول الرؤية اختلافًا طفيفًا عن FFN من حيث الموضع لمشفّر المحول الأصلي (انظر القسم 11.7.2). أولاً، هنا تستخدم دالة التنشيط الوحدة الخطية للخطأ الغاوسي Gaussian error linear unit (GELU)، والتي يمكن اعتبارها نسخة أكثر سلاسة من ReLU(2016,Hendrycks and Gimpel). ثانيًا، يتم تطبيق التسرب dropout على إخراج كل طبقة متصلة بالكامل في MLP من أجل التنظيم.

```
class ViTMLP(nn.Module):
    def __init__(self, mlp_num_hiddens, mlp_num_outputs,
dropout=0.5):
        super().__init__()
        self.dense1 = nn.LazyLinear(mlp_num_hiddens)
        self.gelu = nn.GELU()
        self.dropout1 = nn.Dropout(dropout)
        self.dense2 = nn.LazyLinear(mlp_num_outputs)
        self.dropout2 = nn.Dropout(dropout)

    def forward(self, x):
        return
self.dropout2(self.dense2(self.dropout1(self.gelu(
```

```
            self.dense1(x)))))
```

يتبع تنفيذ كتلة مشفر محول الرؤية تصميم ما قبل التسوية الواردفي الشكل 11.8.1، حيث يتم تطبيق التسوية مباشرة قبل الانتباه متعدد الرؤوس أو MLP. على عكس ما بعد التسوية (" add & norm "في الشكل 11.7.1)، حيث يتم وضع التسوية مباشرة بعد التوصيلات المتبقية، يؤدي التسوية المسبقة إلى تدريب أكثر فعالية أو كفاءة للمحولات (Baevski and Auli، 2018، Wang et al.، 2019، Xiong وآخرون، 2020).

```
class ViTBlock(nn.Module):
    def __init__(self, num_hiddens, norm_shape,
mlp_num_hiddens,
                 num_heads, dropout, use_bias=False):
        super().__init__()
        self.ln1 = nn.LayerNorm(norm_shape)
        self.attention =
d2l.MultiHeadAttention(num_hiddens, num_heads,
                                                dropout,
use_bias)
        self.ln2 = nn.LayerNorm(norm_shape)
        self.mlp = ViTMLP(mlp_num_hiddens, num_hiddens,
dropout)

    def forward(self, X, valid_lens=None):
        X = self.ln1(X)
        return X + self.mlp(self.ln2(
            X + self.attention(X, X, X, valid_lens)))
```

كما هو الحالفي القسم 11.7.4، فإن أي كتلة مشفر لمحول الرؤية لا تغير شكل إدخالها.

```
X = torch.ones((2, 100, 24))
encoder_blk = ViTBlock(24, 24, 48, 8, 0.5)
encoder_blk.eval()
d2l.check_shape(encoder_blk(X), X.shape)
```

11.8.4. وضع كل شيء معا Putting It All Together

إن المرور الأمامي لمحولات الرؤية أدناه واضح ومباشر. أولاً، يتم إدخال صور الإدخالفي مثيل `PatchEmbedding` ، والذي يتم ربط مخرجاته مع تضمين الرمز "<cls>". يتم تلخيصهافي التضمينات الموضعية القابلة للتعلم قبل التسرب dropout. ثم يتم إدخال الإخراجفي مشفر المحولات التي تكدس `num_blks` من مثيلات فئة `ViTBlock`. أخيرًا، يتم عرض تمثيل الرمز المميز "<cls>" بواسطة رأس الشبكة.

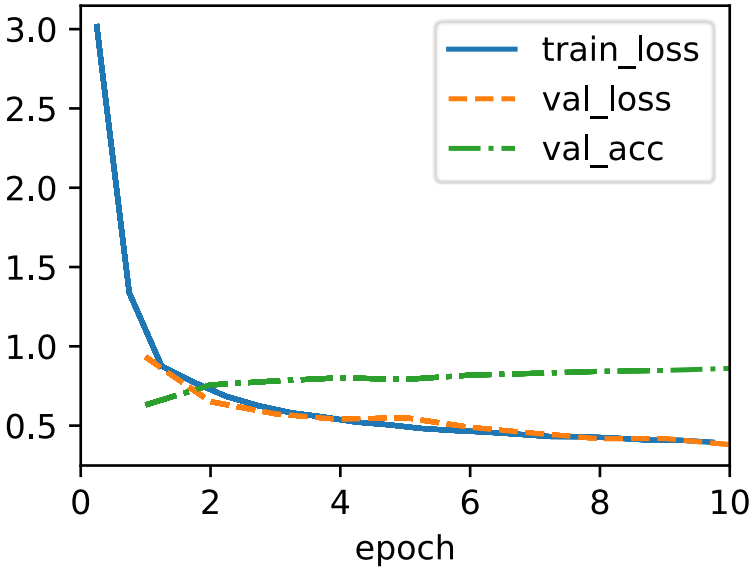
```
class ViT(d2l.Classifier):
```

```
    """Vision transformer."""
    def __init__(self, img_size, patch_size,
num_hiddens, mlp_num_hiddens,
                 num_heads, num_blks, emb_dropout,
blk_dropout, lr=0.1,
                 use_bias=False, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.patch_embedding = PatchEmbedding(
            img_size, patch_size, num_hiddens)
        self.cls_token = nn.Parameter(torch.zeros(1, 1,
num_hiddens))
        num_steps = self.patch_embedding.num_patches + 1
# Add the cls token
        # Positional embeddings are learnable
        self.pos_embedding = nn.Parameter(
            torch.randn(1, num_steps, num_hiddens))
        self.dropout = nn.Dropout(emb_dropout)
        self.blks = nn.Sequential()
        for i in range(num_blks):
            self.blks.add_module(f"{i}", ViTBlock(
                num_hiddens, num_hiddens,
mlp_num_hiddens,
                num_heads, blk_dropout, use_bias))
        self.head =
nn.Sequential(nn.LayerNorm(num_hiddens),
                                    nn.Linear(num_hiddens,
num_classes))

    def forward(self, X):
        X = self.patch_embedding(X)
        X = torch.cat((self.cls_token.expand(X.shape[0],
-1, -1), X), 1)
        X = self.dropout(X + self.pos_embedding)
        for blk in self.blks:
            X = blk(X)
        return self.head(X[:, 0])
```

11.8.5. التدريب Training

إن تدريب محول الرؤية على مجموعة بيانات Fashion-MNIST يشبه تمامًا كيفية تدريب شبكات CNN في القسم 8.

```
img_size, patch_size = 96, 16
num_hiddens, mlp_num_hiddens, num_heads, num_blks = 512,
2048, 8, 2
emb_dropout, blk_dropout, lr = 0.1, 0.1, 0.1
model = ViT(img_size, patch_size, num_hiddens,
mlp_num_hiddens, num_heads,
            num_blks, emb_dropout, blk_dropout, lr)
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128,
resize=(img_size, img_size))
trainer.fit(model, data)
```



11.8.6. الملخص والمناقشة

قد تلاحظ أنه بالنسبة لمجموعات البيانات الصغيرة مثل Fashion-MNIST، فإن محول الرؤية المطبق لدينا لا يتفوق على ResNetفي القسم 8.6. يمكن إجراء ملاحظات مماثلة حتى على مجموعة بيانات ImageNet (1.2 مليون صورة). هذا لأن المحولات تفتقر إلى تلك المبادئ المفيدةفي الالتفاف، مثل ثبات الترجمة translation invariance والمحلية locality (القسم 7.1). ومع ذلك، تتغير الصورة عند تدريب نماذج أكبر على مجموعات بيانات أكبر (على سبيل المثال، 300 مليون صورة)، حيث تتفوق محولات الرؤية على شبكات ResNets بهامش كبيرفي تصنيف الصور، مما يدل على التفوق الجوهري للمحولاتفي قابلية التوسع scalability (Dosovitskiy et al.، 2021). أدى إدخال محولات الرؤية إلى تغيير مشهد تصميم الشبكة لنمذجة بيانات الصورة. وسرعان ما تم عرضها على مجموعة بيانات ImageNet

باستخدام استراتيجيات تدريب فعالة للبيانات من DeiT، (Touvron et al., 2021). ومع ذلك، فإن التعقيد التربيعي للانتباه الذاتي (القسم 11.6) يجعل بُنية المحولات أقل ملاءمة للصور عالية الدقة. نحو شبكة العمود الفقري للأغراض العامةفي الرؤية الحاسوبية، عالجت محولات Swin التعقيد الحسابي التربيعي فيما يتعلق بحجم الصورة (القسم 11.6.2) وأضافت مقدمات تشبه الالتفاف الخلفي back convolution-like priors، مما أدى إلى توسيع قابلية تطبيق المحولات إلى مجموعة من مهام الرؤية الحاسوبية التي تتجاوز تصنيف الصور مع أحدث النتائج (Liu et al.، 2021).

11.8.7. التمارين

1. كيف تؤثر قيمة `img_size` على وقت التدريب؟
2. بدلاً من إسقاط تمثيل الرمز "<cls>" على الإخراج، كيف يتم عرض تمثيلات الرقعة المتوسطة؟ قم بتنفيذ هذا التغيير وانظر كيف يؤثر على الدقة.
3. هل يمكنك تعديل المعلمات الفائقة لتحسين دقة محول الرؤية؟

11.9 التدريب المسبق على نطاق واسع باستخدام المحولات Large-Scale Pretraining with Transformers

حتى الآنفي تصنيف الصور وتجارب الترجمة الآلية، تم تدريب النماذج على مجموعات البيانات مع أمثلة المدخلات والمخرجات من البداية لأداء مهام محددة. على سبيل المثال، تم تدريب أحد المحولات باستخدام أزواج من الإنجليزية والفرنسية (القسم 11.7) بحيث يمكن لهذا النموذج ترجمة إدخال النص الإنجليزي إلى الفرنسية. نتيجة لذلك، يصبح كل نموذج خبيرًا محددًا حساسًا حتى للتحول الطفيففي توزيع البيانات (القسم 4.7). بالنسبة للنماذج المعممة بشكل أفضل، أو حتى المتخصصين الأكثر كفاءة الذين يمكنهم أداء مهام متعددة مع أو بدون تكيف، فإن نماذج التدريب المسبق pretraining models على البيانات الكبيرة كانت شائعة بشكل متزايد.

بالنظر إلى بيانات أكبر للتدريب المسبق، تعمل بنية المحولات بشكل أفضل مع زيادة حجم النموذج وحساب التدريب، مما يدل على سلوك القياس المتفوق. على وجه التحديد، يتم قياس أداء نماذج اللغة القائمة على المحولات كقانون قوة مع مقدار معلمات النموذج ورموز التدريب وحساب التدريب (Kaplan et al.، 2020). تتضح قابلية تطوير المحولات أيضًا من خلال الأداء المعزز بشكل كبير من محولات الرؤية الأكبر المدربة على بيانات أكبر (تمت مناقشتهافي القسم 11.8). تشمل قصص النجاح الأحدث غاتو Gato، وهو نموذج عام يمكنه لعب أتاري، والتعليق على الصور، والدردشة، والعمل كإنسان آلي (Reed et al.، 2022). Gato هو محول واحد يتسع بشكل جيد عند اختباره مسبقًا على طرائق متنوعة بمافي ذلك النص والصور وعزم

الدوران المشترك وضغط الأزرار. والجدير بالذكر أن كل هذه البيانات متعددة الوسائط يتم تسلسلها إلى تسلسل مسطح من الرموز tokens، والتي يمكن معالجتها على غرار الرموز النصية text tokens (القسم 11.7) أو رقع الصور image patches (القسم 11.8) بواسطة المحولات.

قبل النجاح المقنع لمحولات التدريب المسبق للبيانات متعددة الوسائط، كانت المحولات مقيدة على نطاق واسع بكمية كبيرة من النصوص. تم اقتراح معمارية المحولاتفي الشكل 11.7.1في الأصل للترجمة الآلية، وتتألف من مشفر لتمثيل تسلسلات الإدخال ومفكك شفرة لتوليد متواليات الهدف.في المقام الأول، يمكن استخدام المحولاتفي ثلاثة أوضاع مختلفة: المشفر فقط encoder-only، والمشفر-مفكك الشفرة encoder-decoder ومفكك الشفرة فقط decoder-only.في ختام هذا الفصل، سنراجع هذه الأوضاع الثلاثة ونوضح قابلية التوسعفي المحولات قبل التدريب.

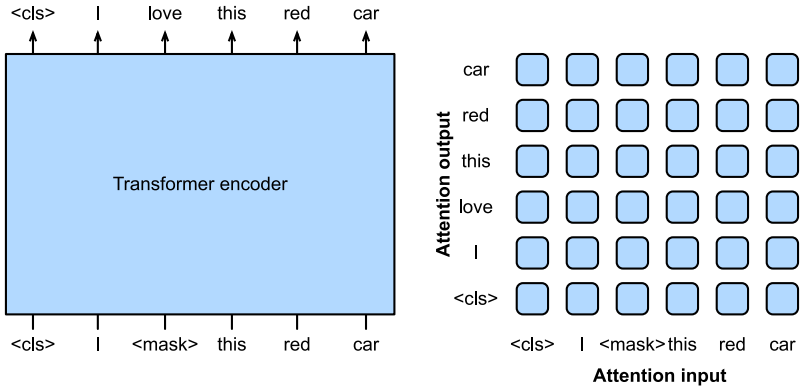
11.9.1. المشفر فقط Encoder-Only

عند استخدام مشفر المحول فقط، يتم تحويل سلسلة من الرموز للإدخال إلى نفس عدد التمثيلات التي يمكن إسقاطها بشكل أكبرفي الإخراج (على سبيل المثال، التصنيف). يتكون مشفر المحولات من طبقات الانتباه الذاتي، حيث تحضر جميع الرموز للإدخال مع بعضها البعض. على سبيل المثال، محولات الرؤية الموضحةفي الشكل 11.8.1 هي عبارة عن مشفر فقط encoder-only، حيث تقوم بتحويل سلسلة من رقع صورة الإدخال إلى تمثيل رمز "<cls>". نظرًا لأن هذا التمثيل يعتمد على جميع الرموز للإدخال، فإنه يتم عرضه بشكل أكبرفي تسميات التصنيف. تم استلهام هذا التصميم من محول سابق يعمل بالمشفر فقط تم اختباره مسبقًا على النص: BERT (تمثيلات التشفير ثنائية الاتجاه من المحولات Bidirectional Encoder Representations from Transformers) (Devlin et al.، 2018).

تم تدريب BERT مسبقًا على تسلسلات النص باستخدام نمذجة اللغة المُقنعة masked language modeling: يتم إدخال نص الإدخال مع الرموز المُقنعة masked tokens عشوائيًا في مشفر المحولات للتنبؤ بالرموز المقنعة. كما هو موضحفي الشكل 11.9.1، تسلسل النص الأصلي "I" و "love" و "this" و "red" و "car" مُجهز مسبقًا بالرمز "<cls>" و "<mask>" يستبدل الرمز بشكل عشوائي "love "؛ ثم يتم التقليل من خطأ الانتروبيا بين الرمز المقنع "love" وتنبؤاته أثناء التدريب المسبق. لاحظ أنه لا يوجد قيدفي نمط الانتباه الخاص بمشفرات المحولات (يمين الشكل 11.9.1) لذلك يمكن لجميع الرموز أن تتعامل مع بعضها البعض. وبالتالي، فإن التنبؤ بـ "love " يعتمد على رموز الإدخال قبل وبعدهفي التسلسل. هذا هو السبب في أن BERT هو "مشفر ثنائي الاتجاه bidirectional encoder ". بدون الحاجة إلى وضع

التسميات اليدوية manual labeling، يمكن استخدام بيانات نصية كبيرة الحجم من الكتب وويكيبيديا للتدريب المسبق على BERT.

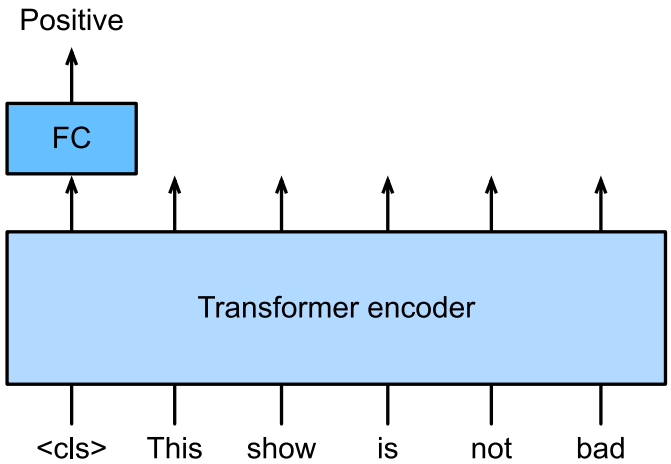
11.9.1.1. التدريب المسبق لبيرت Pretraining BERT



الشكل 11.9.1 على اليسار: التدريب المسبق لـ BERT بنمذجة اللغة المقنعة. يعتمد توقع رمز "love" المقنع على جميع رموز الإدخال قبل وبعد "love". على اليمين: نمط الانتباهفي مشفر المحولات. كل رمز مميز على طول المحور الرأسي يحضر جميع رموز الإدخال على طول المحور الأفقي.

11.9.1.2. الضبط الدقيق لبيرت Fine-Tuning BERT

يمكن ضبط BERT المدروس مسبقًا لمهام الترميز النهائية التي تتضمن نصًا واحدًا أو أزواجًا نصية. أثناء الضبط الدقيق Fine-Tuning، يمكن إضافة طبقات إضافية إلى BERT باستخدام معلمات عشوائية: سيتم تحديث هذه المعلمات ومعلمات BERT سابقة التدريب لتلائم fit بيانات التدريب الخاصة بمهام downstream.



يوضح الشكل 11.9.2 ضبط BERT لتحليل المشاعر sentiment analysis. مشفر المحول عبارة عن BERT تم اختباره مسبقًا، والذي يأخذ تسلسل نصي كمدخل ويغذي تمثيل "<cls>" (تمثيل عالمي للمدخلات)في طبقة إضافية متصلة بالكامل للتنبؤ بالمشاعر. أثناء الضبط الدقيق، يتم تقليل فقدان الانتروبيا بين التنبؤ والتسمية على بيانات تحليل المشاعر عبر خوارزميات قائمة على التدرج، حيث يتم تدريب الطبقة الإضافية من نقطة الصفر بينما يتم تحديث معلمات BERT المحددة مسبقًا. يقوم BERT بأكثر من مجرد تحليل المشاعر. طورت التمثيلات اللغوية العامة التي تعلمها BERT المكونة من 350 مليون متغير من 250 مليار رمز تدريب من أحدث ما توصلت إليه مهام اللغة الطبيعية مثل تصنيف النص الفردي single text classification، وتصنيف أزواج النص text pair classification أو الانحدار regression، ووضع علامات على النص text tagging، والإجابة على الأسئلة question answering.

قد تلاحظ أن هذه المهام النهائية تتضمن فهم أزواج النص. يتسبب تدريب BERT المسبقفي خسارة أخرى للتنبؤ بما إذا كانت إحدى الجمل تتبع الأخرى مباشرة. ومع ذلك، تم العثور لاحقًا على هذه الخسارة غير مفيدة عند إجراء اختبار مسبق لـ RoBERTa، وهو متغير BERT من نفس الحجم، على 2000 مليار رمز (Liu et al.، 2019). مشتقات أخرى من BERT حسنت معماريات النماذج أو أهداف ما قبل التدريب، مثل ALBERT (فرض مشاركة المعلمات)، (Lan et al.,2019)، SpanBERT (تمثل وتوقع مساحات النص representing and predicting spans of text)، (Joshi et al.,2020) DistilBERT (خفيف الوزن عبر تقطير المعرفة lightweight via knowledge distillation) (Sanh et al.، 2019)، وELECTRA (اكتشاف الرموز المستبدلة replaced token detection) (Clark et al.، 2020). علاوة على ذلك، استوحى BERT المحولات من التدريب المسبقفي الرؤية الحاسوبية، مثل محولات الرؤية (Dosovitskiy et al.، 2021)، ومحولات Swin (Liu et al.,2021)، وMAE (المشفرات التلقائية المقنعة masked autoencoders) (He et al.، 2022).

11.9.2. المشفر-مفكك الشفرة Encoder-Decoder

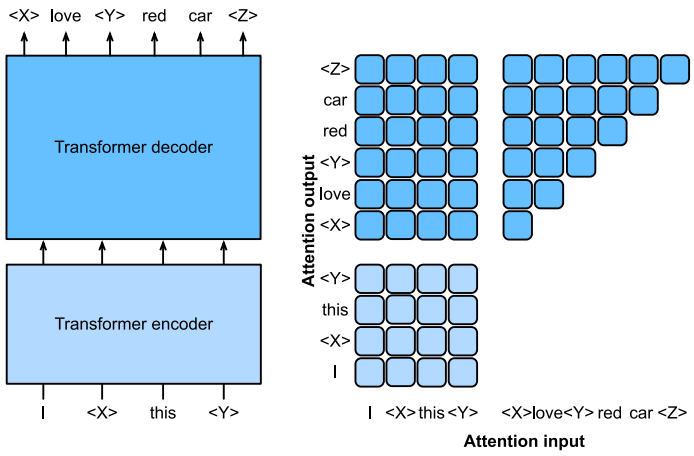
نظرًا لأن مشفر المحول يحول سلسلة من رموز الإدخال إلى نفس عدد تمثيلات الإخراج، لا يمكن لوضع المشفر فقط إنشاء سلسلة من الطول التعسفي كما هو الحالفي الترجمة الآلية. تم اقتراح بُنية المحولاتفي الأصل للترجمة الآلية، وتحتوي أيضًا على مفكك الشفرة تتنبأ بشكل تلقائي بالتسلسل المستهدف للطول التعسفي ، رمزًا رمزًا ، مشروطًا بكل من إخراج المشفر وإخراج مفكك الشفرة: (1) للتكييف على إخراج المشفر ، يسمح الانتباه المتبادل للمشفر- مفكك الشفرة (الانتباه متعدد الرؤوس لمفكك الشفرةفي الشكل 11.7.1) للرموز المستهدفة بالحضور إلى جميع الرموز للإدخال ؛ (2) يتحقق التكييف على خرج مفكك الشفرة من خلال

نمط انتباه سببي causal attention pattern (الانتباه المقنع متعدد الرؤوس لمفكك الشفرة في الشكل 11.7.1) ، حيث لا يمكن لأي رمز مستهدف أن يحضر إلا إلى الرموز السابقة والحالية في التسلسل المستهدف.

للتدريب المسبق لمشفر_مفكك شفرة المحولات بما يتجاوز بيانات الترجمة الآلية التي تحمل علامات بشرية، فإن BART، (2019,Lewis et al.) و T5، (2020,Raffel et al.) هما محولات مقترحةفي نفس الوقت لمشفر_مفكك شفرة تم تدريبهما مسبقًا على نصوص كبيرة الحجم. يحاول كلاهما إعادة بناء النص الأصلي في أهدافهم المدربة سابقاً، بينما يؤكد الأول على المدخلات المزعجة noising input (على سبيل المثال، الإخفاء والحذف والتبديل والتناوب) والأخير يسلط الضوء على توحيد المهام المتعددة مع دراسات الاجتثاث الشاملة.

11.9.2.1. التدريب المسبق لـ T5

كمثال على مشفر_مفكك شفرة المحولات سابقة التدريب ، يوحِّد T5 (محول نقل النص إلى نص Text-to-Text Transfer Transformer) العديد من المهام مثل مشكلة النص إلى النص نفسه: بالنسبة لأي مهمة ، يكون إدخال المشفر هو وصف المهمة (على سبيل المثال ، "تلخيص Summarize " ، ":") متبوعًا بإدخال مهمة (على سبيل المثال ، سلسلة من الرموز المميزة من مقالة a sequence of tokens from an article) ، ويتنبأ مفكك الشفرة بإخراج بالمهمة (على سبيل المثال ، سلسلة من الرموز المميزة تلخص مقالة الإدخال sequence of tokens summarizing the input article). لأداء النص إلى نص، يتم تدريب T5 على إنشاء بعض نص الهدف المشروط على إدخال النص.



الشكل 11.9.3 على اليسار: التدريب المسبق على T5 من خلال توقع فترات متتالية. الجملة الأصلية هي "I"، "love"، "this"، "red"، "car"، حيث يتم استبدال "love" برمز "<X>"،

ومتتالية "red"، "car" استبدالها برمز مميز "<Y>". التسلسل المستهدف ينتهي برمز خاص "<Z>". على اليمين: نمط الانتباهفي مشفر_مفكك شفرة المحولات.في الانتباه الذاتي للمشفر (المربع السفلي)، تحضر جميع الرموز للإدخال بعضها البعض؛في الانتباه المتبادل للمشفر ومفكك الشفرة (المستطيل العلوي)، يحضر كل رمز مستهدف جميع الرموز للإدخال؛في الانتباه الذاتي لمفكك الشفرة (المثلث العلوي)، يحضر كل رمز مميز الهدف الرموز الحالية والماضية فقط (السببية causal).

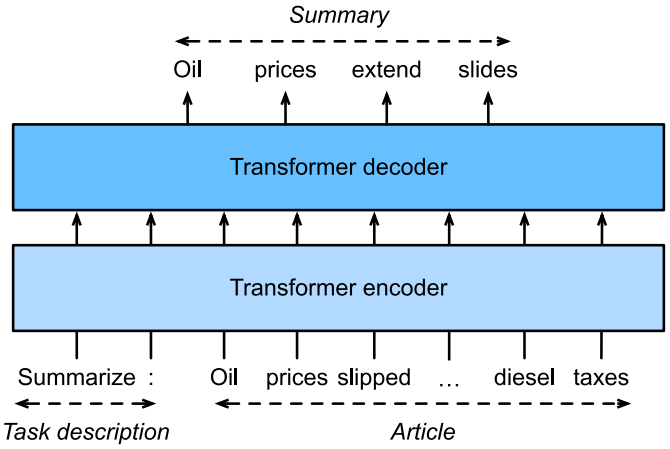
للحصول على مدخلات ومخرجات من أي نص أصلي، يتم تدريب T5 مسبقًا على التنبؤ بفترات متتالية. على وجه التحديد، يتم استبدال الرموز من النص بشكل عشوائي برموز خاصة حيث يتم استبدال كل فترة متتالية بالرمز نفسه. تأمل المثالفي الشكل 11.9.3، حيث النص الأصلي هو "I"، "love"، "this"، "red"، "car". يتم استبدال الرموز "love"، "red"، "car" بشكل عشوائي برموز خاصة. نظرًا لأن "red" و "car" يمثلان امتدادًا متتاليًا، يتم استبدالهما بنفس الرمز الخاص. نتيجة لذلك، تسلسل الإدخال هو "I"، X>"<X"، "هذا"، "<Y>"، والتسلسل الهدف هو "<X>"، "love"، "<Y>"، " red "، " car "، " <Z> "، حيث "<Z> "هو رمز آخر يشير إلى النهاية. كما هو مبينفي الشكل 11.9.3، فإن مفكك الشفرة لديه نمط انتباه سببي لمنع نفسه من الانتباه بالرموز المستقبلية أثناء التنبؤ بالتسلسل.

في T5، يُشار أيضًا إلى التنبؤ بامتداد متتالي consecutive span على أنه إعادة بناء نص تالف reconstructing corrupted text. مع هذا الهدف، يتم تدريب T5 مسبقًا على 1000 مليار رمز من بيانات (Colossal Clean Crawled Corpus) C4، والتي تتكون من نص إنجليزي نظيف من الويب (Raffel et al.، 2020).

11.9.2.2. الضبط الدقيق لأ T5 Fine-Tuning T5

على غرار BERT، يحتاج T5 إلى ضبط دقيق (تحديث معلمات T5) على بيانات التدريب الخاصة بالمهمة لأداء هذه المهمة. تشمل الاختلافات الرئيسية عن الضبط الدقيق لـ BERT ما يلي: (1) يتضمن إدخال T5 أوصاف المهام؛ (2) يمكن أن يولد T5 تسلسلات ذات طول تعسفي باستخدام مفكك شفرة المحول؛ (3) لا توجد طبقات إضافية مطلوبة.

يوضح الشكل 11.9.4 ضبط T5 بدقة باستخدام تلخيص النص كمثال.في هذه المهمة المتلقية للمعلومات، تعتبر الرموز لوصف المهمة ":"Summarize" متبوعة برموز المقالة المميزة هي مدخلات إلى للمشفر.



الشكل 11.9.4 الضبط الدقيق لـ T5 لتلخيص النص. يتم تغذية كل من وصف المهمة ورموز المقالةفي مشفر المحول للتنبؤ بالملخص.

بعد الضبط الدقيق، حققت T5 (T5-11B) التي تبلغ 11 مليار معلمة نتائج متطورة على معايير الترميز المتعددة (مثل التصنيف) والتوليد (على سبيل المثال، التلخيص). منذ إطلاقه، تم استخدام T5 على نطاق واسعفي الأبحاث اللاحقة. على سبيل المثال، تم تصميم محولات التبديل على أساس T5 لتنشيط مجموعة فرعية من المعلمات لتحسين الكفاءة الحسابية (Fedus et al.، 2022).في نموذج تحويل النص إلى صورة يسمى Imagen، يتم إدخال النص إلى مشفر T5 مجمد frozen T5 encoder (T5-XXL) مع 4.6 مليار معلمة (Saharia et al.، 2022). تشير الأمثلة الواقعية لتحويل النص إلى صورةفي الشكل 11.9.5 إلى أن مشفر T5 وحده قد يمثل النص بشكل فعال حتى بدون ضبطه بدقة.

Teddy bears swimming at the Olympics 400m Butterfly event.

A cute corgi lives in a house made out of sushi.

A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

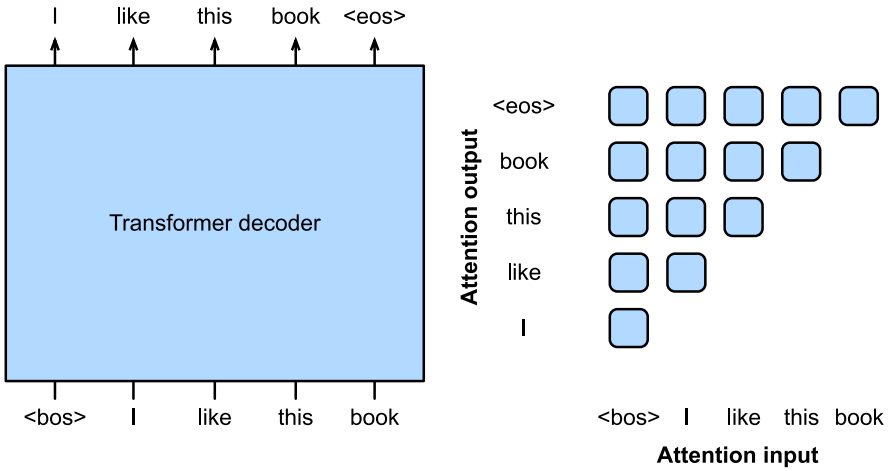
الشكل 11.9.5 أمثلة على تحويل النص إلى صورة من خلال نموذج Imagen، الذي يكون مشفر النص الخاص به من T5 (الأرقام مأخوذة من Saharia et al. (2022)).

11.9.3. مفكك الشفرة فقط Decoder-Only

لقد راجعنا مشفر المحولات فقط ومشفر–مفكك الشفرة المحولات. بدلاً من ذلك، تزيل مفكك شفرة المحولات فقط كامل المشفر والطبقة الفرعية لمفكك الشفرة مع الانتباه المتبادل للمشفر–مفكك الشفرة من بُنية المشفر–مفكك الشفرة الأصلية الموضحةفي الشكل 11.7.1. في الوقت الحاضر، كانت مفكك شفرة المحولات فقط هي العمارة الفعليةفي نمذجة اللغة على نطاق واسع (القسم 9.3)، والتي تستفيد من مجموعات النصوص غير المصنفة الوفيرةفي العالم عبر التعلم تحت الإشراف الذاتي.

11.9.3.1. GPT و GPT-2

باستخدام النمذجة اللغوية كهدف للتدريب، يختار نموذج GPT (التدريب المسبق التوليدي generative pre-training) مفكك شفرة المحولات كعمودها الفقري (Radford et al.، 2018).



الشكل 11.9.6 اليسار: التدريب المسبق على GPT بنمذجة اللغة. التسلسل المستهدف هو تسلسل الإدخال الذي تم إزاحته بواسطة رمز واحد. كل من "<bos>" و "<eos>" هما رمزان مميزان يرمزان إلى بداية ونهاية التسلسلات، على التوالي. اليمين: نمط الانتباهفي مفكك شفرة المحولات. كل رمز مميز على طول المحور الرأسي يحضر فقط الرموز السابقة على طول المحور الأفقي (سببي).

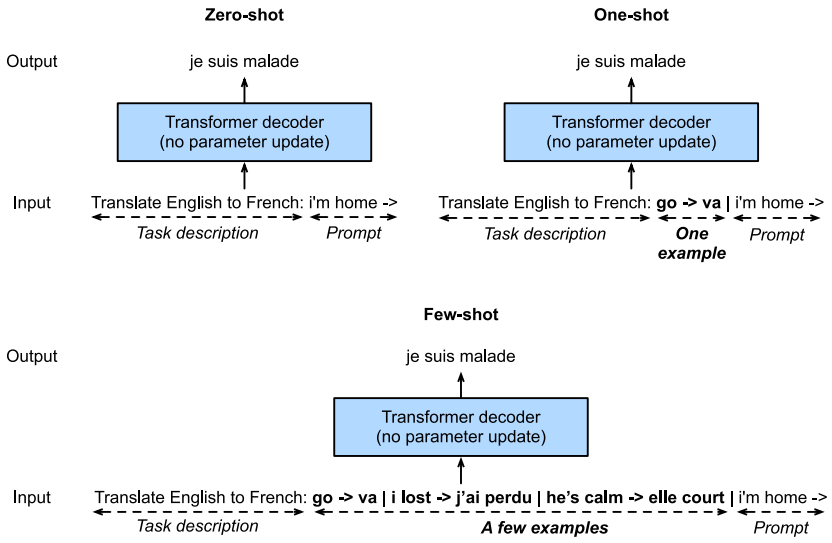
بعد تدريب نموذج اللغة الانحدار الذاتي كما هو موضحفي القسم 9.3.3، يوضح الشكل 11.9.6 التدريب المسبق لـ GPT باستخدام مشفر المحولات، حيث يكون التسلسل المستهدف هو تسلسل الإدخال الذي تم إزاحته برمز واحد. لاحظ أن نمط الانتباه السببيفي وحدة مفكك شفرة

المحولات يفرض أن كل رمز لا يمكن أن يحضر إلا إلى الرموز السابقة (لا يمكن للتنبؤ بالرمز المميز أن يحضر إلى الرموز المستقبلية).

تحتوي GPT على 100 مليون معلمة وتحتاج إلى ضبطها لمهام downstream الفردية. تم تقديم نموذج لغة مفكك شفرة محول أكبر بكثير، GPT-2 ، بعد عام واحد (Radford et al.، 2019). بالمقارنة مع مفكك شفرة المحولات الأصليةفي GPT، تم اعتماد التسوية المسبقة pre-normalization (الذي تمت مناقشتهفي القسم 11.8.3) والتهيئة المحسنة وقياس الوزن في GPT-2. حصل GPT-2 الذي تم تدريبه مسبقًا على 40 غيغابايت من النص، والذي يبلغ حجمه 1.5 مليار معلمة، على أحدث النتائج المتعلقة بمعايير نمذجة اللغة والنتائج الواعدةفي مهام أخرى متعددة دون تحديث المعلمات أو البُنية.

GPT-3 .11.9.3.2

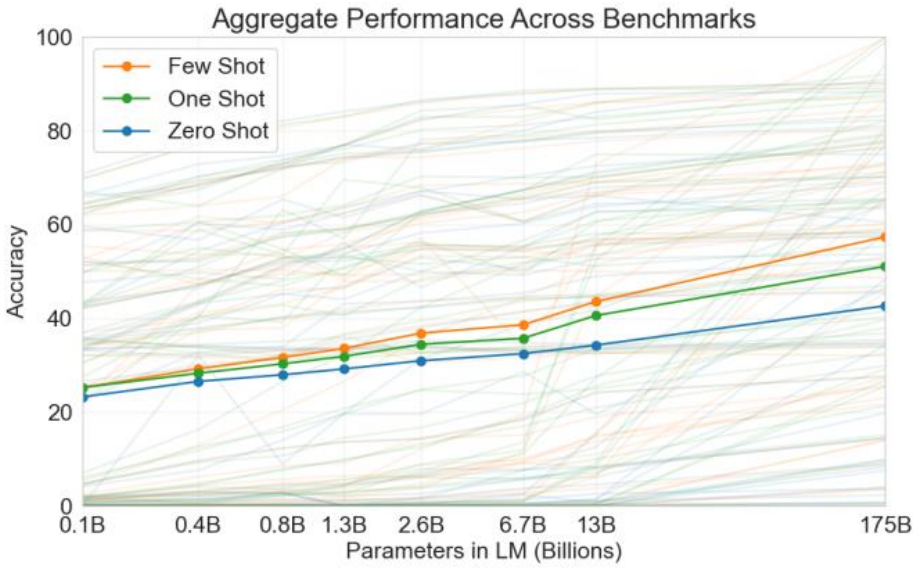
أظهر GPT-2 إمكانية استخدام نفس نموذج اللغة لمهام متعددة دون تحديث النموذج. يعد هذا أكثر كفاءة من الناحية الحسابية من الضبط الدقيق، والذي يتطلب تحديثات النموذج عبر حساب التدرج.



الشكل 11.9.7 التعلم بدون لقطة Zero-shot ولقطة واحدة one-shot ولقطات قليلة few-shot باستخدام نماذج اللغة (مفكك شفرة المحولات). لا حاجة لتحديث المعلمة.

قبل شرح الاستخدام الأكثر كفاءة من الناحية الحسابية لنماذج اللغة دون تحديث المعلمة، تذكر القسم 9.5 أنه يمكن تدريب نموذج اللغة لإنشاء تسلسل نصي مشروط ببعض تسلسل نص

البادئة. وبالتالي، قد ينتج عن نموذج اللغة الذي تم اختباره مسبقًا إخراج المهمة كتسلسل بدون تحديث المعلمة، بشرط تسلسل الإدخال مع وصف المهمة، وأمثلة الإدخال والمخرجات الخاصة بالمهمة، والموجه prompt (إدخال المهمة). يمكن تصنيف نموذج التعلم هذا بشكل أكبر إلى zero-shot، one-shot، وfew-shot، عندما لا يكون هناك أمثلة على المدخلات والمخرجات الخاصة بالمهمة، أو أمثلة قليلة، على التوالي (الشكل 11.9.7).

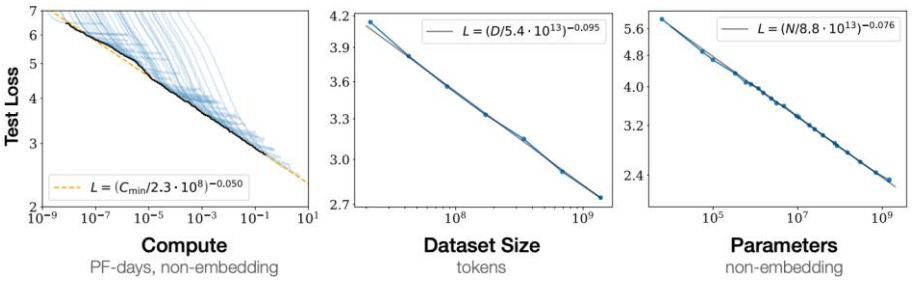


الشكل 11.9.8 الأداء الإجمالي لـ GPT-3 لجميع المعايير المعيارية المقومة بالدقة البالغ عددها 42 (التسمية التوضيحية مقتبسة والشكل مأخوذ من 2020) .Brown et al)).

تم اختبار هذه الإعدادات الثلاثةفي GPT-3، (2020,Brown et al.)، الذي يستخدم أكبر إصدار له البيانات وحجم النموذج بحوالي أمرين من حيث الحجم أكبر من تلك الموجودةفي GPT-2. يستخدم GPT-3 نفس بُنية مفكك شفرة المحولاتفي سلفها المباشر GPT-2 فيما عدا أن أنماط الانتباه (يمين الشكل 11.9.6) تكون متناثرةفي الطبقات المتناوبة. نظرًا لأن GPT-3 مُدرب مسبقًا بـ 300 مليار رمز، فإنه يعمل بشكل أفضل مع حجم نموذج أكبر، حيث يزيد أداء عدد قليل من اللقطات بسرعة أكبر (الشكل 11.9.8). على الرغم من التمتع بكفاءة حسابية، إلا أن التعلم قليل اللقطات لـ GPT-3 كان أقل من أداء النماذج الحديثة التي تم ضبطها والتي تتطلب تحديثات النموذج. ومع ذلك، فقد قام GPT-3 بتشغيل مجموعة واسعة من التطبيقات النهائية عبر الويب: فقد كان ينتج 4.5 مليار كلمة كل يوم حوالي تسعة أشهر من إصدار API الخاص به.

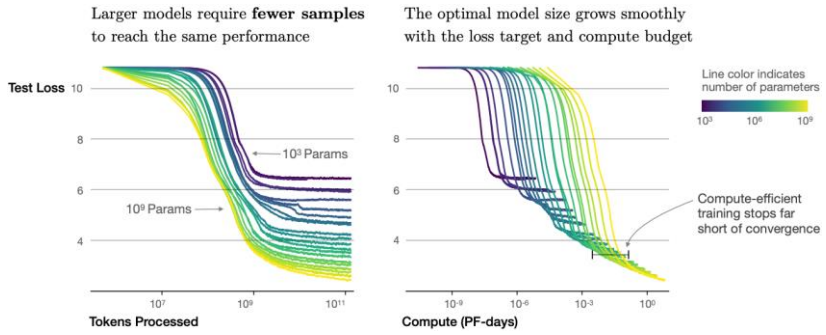
11.9.4. قابلية التوسع Scalability

يوضح الشكل 11.9.8 بشكل تجريبي قابلية التوسع للمحولاتفي نموذج اللغة GPT-3. بالنسبة لنمذجة اللغة، اقترحت دراسات تجريبية أكثر شمولاً حول قابلية توسيع المحولات تدريب محولات أكبر بمزيد من البيانات والحسابات (Kaplan et al.، 2020).



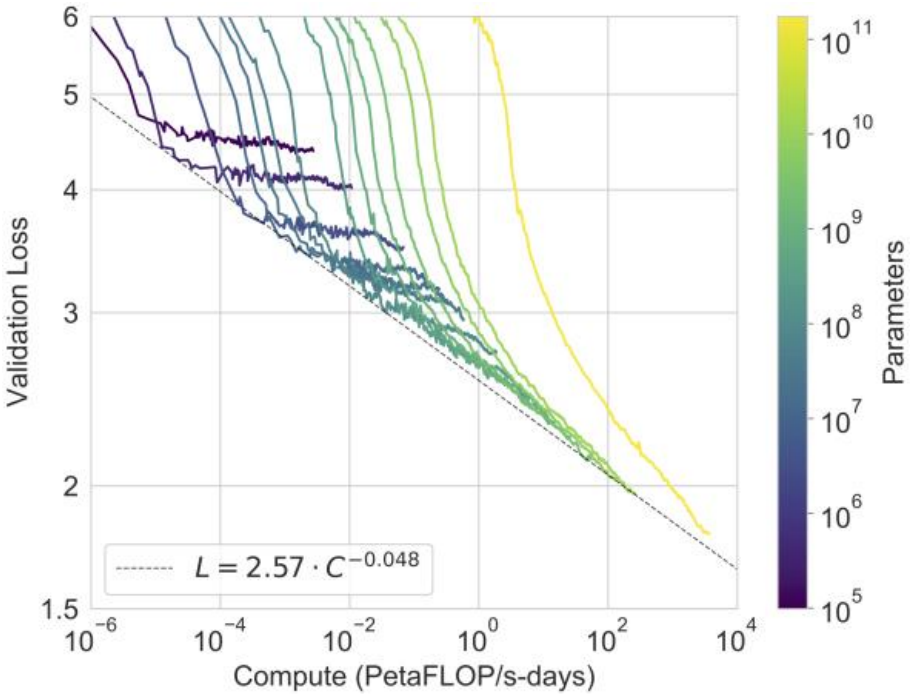
الشكل 11.9.9 يتحسن أداء نموذج لغة المحولات بسلاسة لأننا نزيد من حجم النموذج وحجم مجموعة البيانات وكمية الحوسبة المستخدمة للتدريب. لتحقيق الأداء الأمثل، يجب رفع مستوى جميع العوامل الثلاثة جنبًا إلى جنب. للأداء التجريبي علاقة قانون القوة مع كل عامل على حدة عندما لا يتم اختناقهما من قبل العاملين الآخرين (التسمية التوضيحية مقتبسة والشكل مأخوذ من Kaplan et al. (2020)).

كما هو مبينفي الشكل 11.9.9، يمكن ملاحظة دقة قانون القدرة المقاس power-law scalingفي الأداء فيما يتعلق بحجم النموذج (عدد المعلمات، باستثناء طبقات التضمين)، وحجم مجموعة البيانات (عدد الرموز للتدريب)، وكمية حساب التدريب (PetaFLOP / s-days، باستثناء طبقات التضمين). بشكل عام، تؤدي زيادة كل هذه العوامل الثلاثة جنبًا إلى جنب إلى أداء أفضل. ومع ذلك، لا تزال كيفية زيادتها جنبًا إلى جنب مسألة نقاش (Hoffmann et al.، 2022).



الشكل 11.9.10 دورات تدريب نموذج لغة المحولات (الشكل مأخوذ من Kaplan et al. (2020)).

إلى جانب الأداء المتزايد، تتمتع الطرز الكبيرة أيضًا بكفاءة أفضل للعينة من النماذج الصغيرة. يوضح الشكل 11.9.10 أن النماذج الكبيرة تحتاج إلى عدد أقل من عينات التدريب (الرموز التي تمت معالجتها) لأداء نفس المستوى الذي تحققه النماذج الصغيرة، ويتم قياس الأداء بسلاسة باستخدام الحساب.



الشكل 11.9.11 أداء GPT-3 (خطأ التحقق من الانتروبيا المتقاطعة) يتبع اتجاه قانون القوة مع مقدار الحساب المستخدم للتدريب. سلوك قانون القوة الذي لوحظ في Kaplan et al. (2020) يستمر لخطوتين إضافيتين من حيث الحجم مع انحرافات صغيرة فقط عن المنحنى المتوقع. تُستثنى معلمات التضمين من عدد الحسابات والمعلمات (التسمية التوضيحية مقتبسة والرقم مأخوذ من (Brown et al.,2020)).

سلوكيات القياس التجريبية في (Kaplan et al,2020) تم اختبارها في نماذج المحولات الكبيرة اللاحقة. على سبيل المثال، دعمت GPT-3 هذه الفرضية بأمرين آخرين من حيث الحجم في الشكل 11.9.11.

لقد ألهمت قابلية تطوير المحولات في سلسلة GPT نماذج لغة المحولات اللاحقة. بينما تم اتباع مفكك شفرة المحولات في GPT-3 إلى حد كبير في OPT (Open Pretrained Transformers)، (Zhang et al.,2022) باستخدام 7/1 فقط من البصمة الكربونية للسابق،

تم استخدام مفكك شفرة محول GPT-2في تدريب 530 ـ ميجاترون ـ تورينج NLG ((Smith et al.، 2022) بمليار متغير مع 270 مليار من رموز التدريب. بعد تصميم GPT-2، حقق Gopher، (2021,Rae et al.) الذي يبلغ حجمه 280 مليار معلمة والذي تم تدريبه مسبقًا بـ 300 مليار رمز أداءً متطورًا عبر الغالبيةفي حوالي 150 مهمة متنوعة. إن وراثة نفس البُنية واستخدام نفس الميزانية الحسابية لـ Gopher، Chinchilla، (2022,Hoffmann et al.) هو نموذج أصغر بكثير (70 مليار معلمة) يتدرب لفترة أطول (1.4 تريليون رمز تدريب)، متفوقًا على Gopherفي العديد من المهام. لمواصلة خط القياس لنمذجة اللغة، PaLM (نموذج لغة المسار Pathway Language Model) (Chowdhery et al.، 2022)، مفكك شفرة محول 540 مليار متغير مع تصميمات معدلة تم اختبارها مسبقًا على 780 مليار رمز، تفوقت على متوسط الأداء البشري على BIG-Bench المعيار (Srivastava et al.، 2022). مزيد من التدريب لـ PaLM على 38.5 مليار رمز يحتوي على نتائج محتوى علمي ورياضيفي Minerva، (2022,Lewkowycz et al.)، وهو نموذج لغوي كبير يمكنه الإجابة على ما يقرب من ثلث مشاكل المستوى الجامعي التي تتطلب التفكير الكمي، مثل الفيزياء والكيمياء وعلم الأحياء والاقتصاد.

11.9.5. الملخص والمناقشة

تم اختبار المحولات مسبقًا على أنها مشفر فقط encoder-only (على سبيل المثال، BERT)، مشفرـمفكك شفرة encoder-decoder (على سبيل المثال، T5)، ومفكك شفرة فقط decoder-only (على سبيل المثال، سلسلة GPT). قد يتم تكييف النماذج سابقة التدريب لأداء مهام مختلفة مع تحديث النموذج (على سبيل المثال، الضبط الدقيق fine tuning) أو لا (على سبيل المثال، عدد قليل من اللقطات few shot). تشير قابلية توسيع المحولات إلى أن الأداء الأفضل يستفيد من النماذج الأكبر، والمزيد من بيانات التدريب، والمزيد من حوسبة التدريب. نظرًا لأن المحولات تم تصميمها لأول مرة وفحصها مسبقًا للبيانات النصية، فإن هذا القسم يميل قليلاً نحو معالجة اللغة الطبيعية. ومع ذلك، يمكن العثور على تلك النماذج التي تمت مناقشتها أعلاهفي نماذج أحدث عبر طرائق متعددة. على سبيل المثال، تم تمديد (1)، (2022,Hoffmann et al.) إلى Flamingo، (2022,Alayrac et al.)، وهو نموذج لغة بصرية للتعلم قليل اللقطات؛ (2) GPT-2، (2019,Radford et al.) ومحول الرؤية يشفر النص والصورفي CLIP (التدريب المسبق على اللغة المتباينة Contrastive Language-Image Pre-training) (Radford et al.، 2021)، التي تم اعتماد صورتها ونصها لاحقًا.في نظام تحويل النص إلى صورة DALL-E 2 (Ramesh et al.، 2022). على الرغم من عدم وجود دراسات منهجية حول قابلية التوسعفي المحولاتفي التدريب المسبق متعدد الوسائط حتى الآن، فإن نموذج تحويل النص إلى صورة حديث بالكامل، Parti، (Yu et

.2022,al)، يُظهر إمكانية التوسع عبر الطرائق: يُظهر Parti أكبر أكثر قدرة على إنشاء صور عالية الدقة وفهم نص غني بالمحتوى (الشكل 11.9.12).

A portrait photo of a kangaroo wearing an orange hoodie and blue sunglasses standing on the grass in front of the Sydney Opera House holding a sign on the chest that says Welcome Friends!

الشكل 11.9.12 أمثلة للصور تم إنشاؤها من نفس النص بواسطة نموذج Parti بأحجام متزايدة (350M، 750M، 3B، 20B) (أمثلة مأخوذة من Yu et al (2022)).

11.9.6. التمارين

1. هل من الممكن ضبط T5 باستخدام الدفعات الصغيرة يتكون من مهام مختلفة؟ لما ولما لا؟ ماذا عن GPT-2؟
2. بالنظر إلى نموذج لغوي قوي، ما هي التطبيقات التي يمكنك التفكير فيها؟
3. لنفترض أنه تمت مطالبتك بضبط نموذج لغة لإجراء تصنيف للنص عن طريق إضافة طبقات إضافية. أين ستضيفهم؟ لماذا ا؟
4. ضع في اعتبارك مشاكل التسلسل الى التسلسل (على سبيل المثال، الترجمة الآلية) حيث يكون تسلسل الإدخال متاحًا دائمًا خلال تنبؤ تسلسل الهدف. ماذا يمكن أن تكون قيود النمذجة باستخدام مفكك شفرة المحولات فقط؟ لماذا؟

Dive into Deep Learning

Part 2

Modern Deep Learning Techniques

ASTON ZHANG, ZACHARY C. LIPTON, MU LI,
AND ALEXANDER J.SMOLA

Translated Into Arabic by
Dr. Alaa Taima