یادگیری عمیق

جلسه ۲۳

بهترین روش‌های پیشرفته برای یادگیری عمیق

Advanced Deep-Learning Best Practices

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/deep

بهترین روش‌های پیشرفته برای یادگیری عمیق

ADVANCED DEEP-LEARNING BEST PRACTICES

هدف این فصل:
بررسی تعدادی ابزار قدرتمند
که ما را به توسعه‌ی مدل‌ها در مرزهای دانش برای مسائل دشوار نزدیک‌تر می‌کند.

استفاده از API تابعی در کراس
Using the Keras Functional API

برای ساخت مدل‌های گراف–گونه،
به‌اشتراک‌گذاری یک لایه میان ورودی‌های مختلف،
استفاده از مدل‌های کراس دقیقاً مانند توابع پایتون

استفاده از پس‌خوانی‌های کراس
Using the Keras Callbacks

برای بهره‌برداری از مدل‌های آموزش‌دیده

استفاده از ابزار مصورسازی TensorBoard
Using the TensorBoard Visualization Tool

برای نظارت بر مدل‌ها در حین آموزش

بهترین روش‌های تجربه شده در یادگیری عمیق
Deep-Learning Best Practices

مانند نرمال‌سازی دسته‌ای، اتصالات مانده‌ای،
بهینه‌سازی هایپرپارامترها و مدل‌های دسته‌جمعی

یادگیری عمیق

بهترین روش‌های پیشرفته
برای یادگیری عمیق

۱

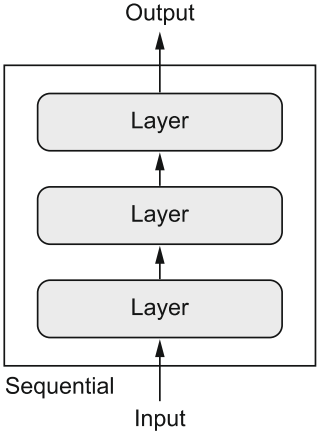
فراتر از مدل‌های ترتیبی: API تابعی در Keras

فراتر از مدل‌های ترتیبی: API تابعی در Keras

مدل ترتیبی: یک پشته‌ی خطی از لایه‌ها

GOING BEYOND THE SEQUENTIAL MODEL: THE KERAS FUNCTIONAL API

مدل ترتیبی این فرض را در نظر می‌گیرد:
شبکه دقیقاً دارای یک ورودی و یک خروجی است و
شبکه یک پشته‌ی خطی از لایه‌هاست.



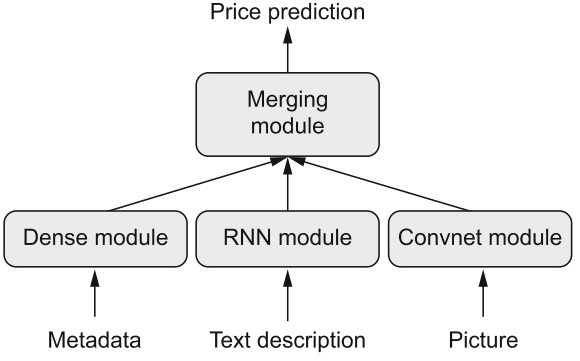
این پیکربندی بسیار معمول و متداول است.
اما این مجموعه از فرضیات در برخی موارد بیش از حد انعطاف‌ناپذیر است.
برخی شبکه‌ها نیازمند چندین ورودی مستقل یا چند خروجی و برخی نیازمند انشعابات درونی بین لایه‌ها هستند،
و ظاهر آنها شبیه گراف‌هایی از لایه‌ها (به‌جای پشته‌ای از لایه‌ها) می‌شود.

فراتر از مدل‌های ترتیبی: API تابعی در Keras

یک مدل با چند ورودی

A Multi-Input Model

برخی وظایف نیازمند ورودی‌های چندوجهی / multimodal هستند:
ادغام داده‌های دریافتی از منابع ورودی مختلف، پردازش هر نوع از داده‌ها با استفاده از انواع لایه‌های عصبی گوناگون



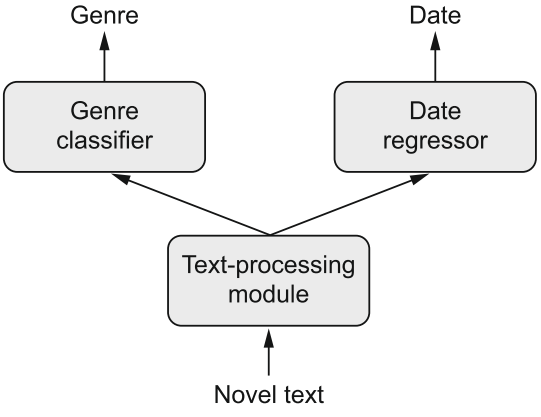
مثال: پیش‌بینی محتمل‌ترین قیمت بازار برای یک لباس دسته‌دوم
با استفاده از ۱) فراداده‌های فراهم‌شده توسط کاربر (برند، سن و ...)، ۲) توصیف متنی فراهم شده توسط کاربر، ۳) تصویر کالا
برای استفاده از هر سه ورودی به‌صورت همزمان
الف (روش خام): آموزش سه مدل مجزا و متوسط گیری وزن‌دار پیش‌بینی آنها
ب (روش بهتر): یادگیری **توأم** یک مدل دقیق‌تر از داده‌ها با استفاده از یک مدل که بتواند همه‌ی مدالیته‌های ورودی را به‌صورت همزمان ببیند (مدلی با سه شاخه‌ی ورودی).

فراتر از مدل‌های ترتیبی: API تابعی در Keras

یک مدل با چند خروجی (چند سر)

A Multi-Output (Or Multihead) Model

برخی وظایف نیازمند پیش‌بینی چند خصیصه‌ی تارگت برای داده‌های ورودی هستند.



مثال: طبقه‌بندی خودکار ژانر یک داستان و پیش‌بینی تاریخ تقریبی نوشته شدن آن
با استفاده از متن آن داستان

راه حل:

الف (روش خام): آموزش دو مدل مجزا: یکی برای ژانر و دیگری برای تاریخ

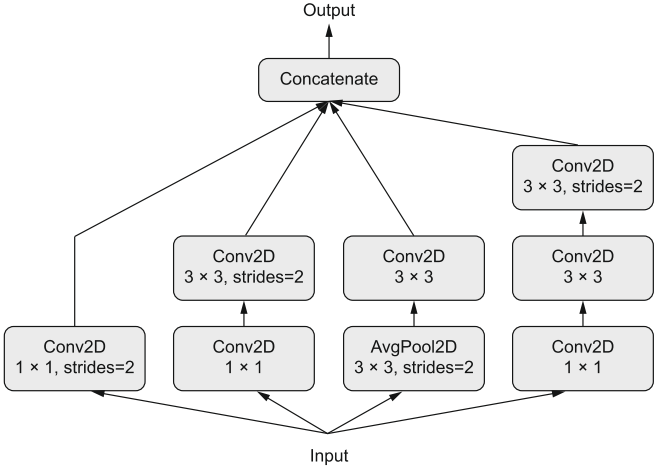
ب (روش بهتر): یادگیری **توأم** یک مدل دقیق‌تر از داده‌ها برای پیش‌بینی ژانر و تاریخ به‌صورت همزمان
(به‌دلیل وجود وابستگی آماری بین این خصیصه‌ها)
مدل دو خروجی (دو سر)

فراتر از مدل‌های ترتیبی: API تابعی در Keras

توپولوژی‌های غیرخطی شبکه

NONLINEAR NETWORK TOPOLOGY

بسیاری از معماری‌های عصبی اخیراً توسعه داده شده، نیازمند توپولوژی غیرخطی برای شبکه هستند:
شبکه‌های ساختاریافته به‌صورت گراف‌های جهت‌دار بدون دور (DAG: directed acyclic graphs)



مثال: خانواده‌ی شبکه‌های Inception
متکی به ماژول‌های Inception است
که در آن ورودی توسط چند شاخه‌ی کانوولوشنال موازی پردازش می‌شود
و سپس خروجی‌های آنها در قالب یک تانسور واحد ادغام می‌شود.

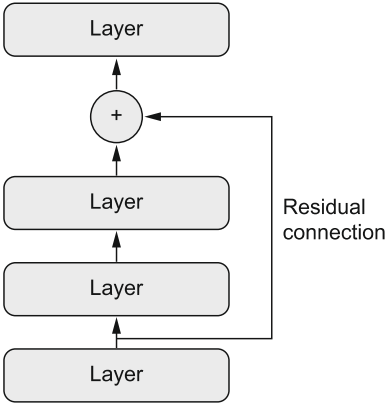
Christian Szegedy et al., "Going Deeper with Convolutions," Conference on Computer Vision and Pattern Recognition (2014), https://arxiv.org/abs/1409.4842.

فراتر از مدل‌های ترتیبی: API تابعی در Keras

اتصالات مانده‌ای

RESIDUAL CONNECTIONS

اخیراً اضافه کردن اتصالات مانده‌ای (residual connections) به مدل به یک ترند تبدیل شده است.
اولین نمونه در خانواده‌ی شبکه‌های ResNet ارائه شد.



یک **اتصال مانده‌ای**، متشکل است از:
تزریق مجدد بازنمایی قبلی به جریان downstream داده‌ها
با افزودن یک تانسور خروجی قبلی به یک تانسور خروجی بعدی
مزیت: جلوگیری از از دست رفتن اطلاعات در طول جریان پردازش داده‌ها

Kaiming He et al., “Deep Residual Learning for Image Recognition,” Conference on Computer Vision and Pattern Recognition (2015), https://arxiv.org/abs/1512.03385.

آشنایی با API تابعی

INTRODUCTION TO THE FUNCTIONAL API

با استفاده از API تابعی، می‌توانیم تانسورها را مستقیماً دستکاری کنیم.
لایه‌ها به‌عنوان تابع‌هایی در نظر گرفته می‌شوند
که تانسورهایی را به‌عنوان ورودی دریافت و تانسورهایی را به‌عنوان خروجی تحویل می‌دهند.

```
from keras import Input, layers

input_tensor = Input(shape=(32,))
```

A tensor

```
dense = layers.Dense(32, activation='relu')
```

A layer is a function.

```
output_tensor = dense(input_tensor)
```

A layer may be called on a tensor, and it returns a tensor.

آشنایی با API تابعی

مثال حداقلی

INTRODUCTION TO THE FUNCTIONAL API

یک مثال حداقلی برای نمایش یک مدل Sequential ساده و معادل آن در functional API

```
from keras.models import Sequential, Model
from keras import layers
from keras import Input

seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = Model(input_tensor, output_tensor)

model.summary()
```

Sequential model, which you already know about

Its functional equivalent

The Model class turns an input tensor and output tensor into a model.

Let's look at it!

آشنایی با API تابعی

مثال حداقلی: نمایش

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

آشنایی با API تابعی

وقوع خطا در صورت وجود اشکال

از شیئ Model تنها با استفاده از یک تانسور ورودی و یک تانسور خروجی نمونه‌سازی می‌شود.
در پشت صحنه، کراس هر لایه‌ای که در مسیر بین تانسور ورودی و تانسور خروجی وجود دارد را بازیابی می‌کند و آنها را در یک ساختمان‌داده‌ی گراف–گونه قرار می‌دهد (یک مدل)

دلیل کار کردن این فرآیند این است که تانسور خروجی با تبدیل تکرارشونده‌ی تانسور ورودی حاصل می‌شود.
اگر تلاش کنیم مدلی از ورودی‌ها و خروجی‌هایی بسازیم که با هم مرتبط نیستند،
با خطای زمان اجرا مواجه می‌شویم.

```
>>> unrelated_input = Input(shape=(32,))
>>> bad_model = model = Model(unrelated_input, output_tensor)
```

```
RuntimeError: Graph disconnected: cannot
obtain value for tensor
  Tensor("input_1:0", shape=(?, 64), dtype=float32) at layer "input_1".
```

این خطا در اصل به ما می‌گوید که:
کراس نمی‌تواند از تانسور خروجی مشخص‌شده به input_1 برسد.

آشنایی با API تابعی

کامپایل کردن، آموزش و ارزیابی مدل‌های ساخته شده با functional API

کامپایل کردن، آموزش و ارزیابی مدل‌های ساخته شده با functional API مشابه مدل Sequential است:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

import numpy as np
x_train = np.random.random((1000, 64))
y_train = np.random.random((1000, 10))

model.fit(x_train, y_train, epochs=10, batch_size=128)

score = model.evaluate(x_train, y_train)
```

Compiles the model

Generates dummy Numpy data to train on

Trains the model for 10 epochs

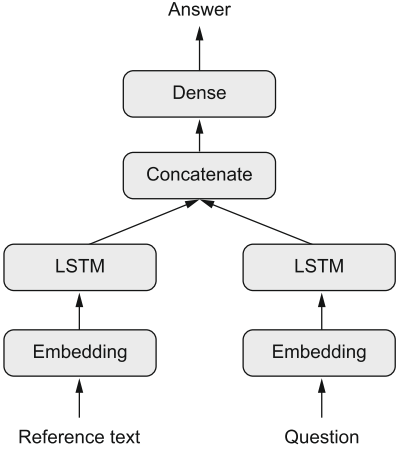
Evaluates the model

مدل‌های چند-ورودی

MULTI-INPUT MODELS

از functional API می‌توان برای ساخت مدل‌های دارای چند ورودی استفاده کرد.
معمولاً این مدل‌ها در برخی نقاط شاخه‌های ورودی خود را از طریق یک لایه‌ی ترکیب‌گر، ترکیب می‌کنند.

مثال: یک مدل پاسخ به پرسش:
با دو ورودی: یک پرسش به زبان طبیعی + متن مرجع (مانند یک مقاله) حاوی اطلاعاتی برای پاسخ به آن پرسش
خروجی: پاسخ یک‌کلمه‌ای از یک فهرست واژگان از پیش‌تعریف شده



باید دو شاخه‌ی مستقل ایجاد کنیم که
ورودی متن و ورودی پرسش
را به‌صورت بردارهای بازنمایی کدگذاری می‌کنند.
سپس، این بردارها را به‌یکدیگر الحاق می‌کنیم.
آن‌گاه یک طبقه‌بندی‌کننده‌ی softmax
بر بالای بازنمایی‌های الحاق‌شده قرار می‌دهیم.

مدل‌های چند-ورودی

مثال: مدل پاسخ به پرسش

Functional API implementation of a two-input question-answering model

```
from keras.models import Model
from keras import layers
from keras import Input

text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

text_input = Input(shape=(None,), dtype='int32', name='text')

embedded_text = layers.Embedding(
    64, text_vocabulary_size)(text_input)

encoded_text = layers.LSTM(32)(embedded_text)
```

The text input is a variable-length sequence of integers. Note that you can optionally name the inputs.

Embeds the inputs into a sequence of vectors of size 64

مدل‌های چند-ورودی

مثال: مدل پاسخ به پرسش

Functional API implementation of a two-input question-answering model (cntd.)

```
encoded_text = layers.LSTM(32)(embedded_text)

question_input = Input(shape=(None,),
                       dtype='int32',
                       name='question')

embedded_question = layers.Embedding(
    32, question_vocabulary_size)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

concatenated = layers.concatenate([encoded_text, encoded_question],
                                  axis=-1)

answer = layers.Dense(answer_vocabulary_size,
                      activation='softmax')(concatenated)

model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
```

Encodes the vectors in a single vector via an LSTM

Same process (with different layer instances) for the question

Concatenates the encoded question and encoded text

Adds a softmax classifier on top

At model instantiation, you specify the two inputs and the output.

مدل‌های چند-ورودی

مثال: مدل پاسخ به پرسش: خوراندن داده‌ها به مدل دارای چند ورودی

Feeding data to a multi-input model

```
import numpy as np

num_samples = 1000
max_length = 100

text = np.random.randint(1, text_vocabulary_size,
                         size=(num_samples, max_length))

question = np.random.randint(1, question_vocabulary_size,
                             size=(num_samples, max_length))
answers = np.random.randint(0, 1,
                            size=(num_samples, answer_vocabulary_size))

model.fit([text, question], answers, epochs=10, batch_size=128)

model.fit({'text': text, 'question': question}, answers,
          epochs=10, batch_size=128)
```

Generates dummy Numpy data

Answers are one-hot encoded, not integers

Fitting using a list of inputs

Fitting using a dictionary of inputs (only if inputs are named)

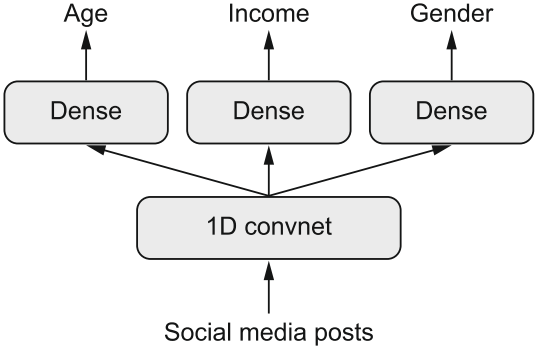
دو گزینه برای آموزش مدل دارای دو ورودی: ۱) می‌توانیم لیستی از آرایه‌های numPy را به‌عنوان ورودی به مدل بخورانیم.
۲) می‌توانیم یک دیکشنری در اختیار آن قرار بدهیم که نام‌های ورودی را به آرایه‌های numPy نگاشت دهد.

مدل‌های چند-خروجی

MULTI-OUTPUT MODELS

از functional API می‌توان برای ساخت مدل‌های دارای چند خروجی (چند سر) استفاده کرد.
مانند شبکه‌ای که تلاش می‌کند ویژگی‌های مختلف داده‌ها را به‌صورت همزمان پیش‌بینی کند.

مثال: شبکه‌ای برای پیش‌بینی ویژگی‌های یک شخص از روی پست‌های او از رسانه‌های اجتماعی:
این شبکه یک سری از پست‌های مربوط به رسانه‌های اجتماعی را از یک شخص ناشناس دریافت می‌کند،
و تلاش می‌کند ویژگی‌های آن شخص (سن، جنسیت و سطح درآمد) را پیش‌بینی کند.



مدل‌های چند–خروجی

پیاده‌سازی یک مدل سه–خروجی

Functional API implementation of a three-output model

```
from keras import layers
from keras import Input
from keras.models import Model

vocabulary_size = 50000
num_income_groups = 10

posts_input = Input(shape=(None,), dtype='int32', name='posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

age_prediction = layers.Dense(1, name='age')(x)
income_prediction = layers.Dense(num_income_groups,
                                 activation='softmax',
                                 name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)

model = Model(posts_input,
              [age_prediction, income_prediction, gender_prediction])
```

Note that the output layers are given names.

مدل‌های چند-خروجی

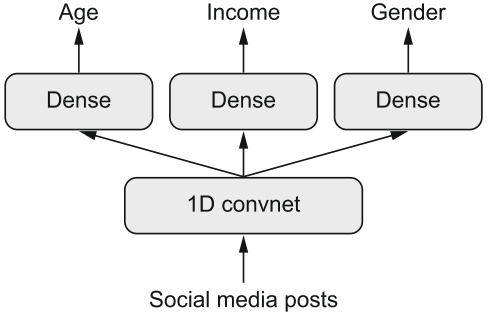
توابع اتلاف چندگانه

Compilation options of a multi-output model: multiple losses

```
model.compile(optimizer='rmsprop',
              loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

```
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'})
```

Equivalent (possible only if you give names to the output layers)



آموزش این مدل نیازمند آن است که برای خروجی‌های مختلف شبکه، توابع اتلاف متفاوتی تعیین شود.

برای مثال:

پیش‌بینی سن (رگرسیون اسکالر): MSE

پیش‌بینی جنسیت (طبقه‌بندی دودویی): CCE

الگوریتم کاهش گرادیان نیازمند می‌نیمم‌سازی یک اسکالر است.

پس باید توابع اتلاف سرهای شبکه در قالب یک اسکالر ترکیب شود.

مانند جمع کردن آنها (روش پیش‌فرض کراس برای اتلاف‌های چندگانه)

مدل‌های چند-خروجی

توابع اتلاف چندگانه: وزن‌دهی

Compilation options of a multi-output model: loss weighting

```
model.compile(optimizer='rmsprop',
              loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'],
              loss_weights=[0.25, 1., 10.])
```

```
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'},
              loss_weights={'age': 0.25,
                            'income': 1.,
                            'gender': 10.})
```

Equivalent (possible only if you give names to the output layers)

اگر مقادیر توابع اتلاف بسیار نامتوازن باشد،
باعث می‌شود که بازنمایی‌های مدل به وظیفه‌ی دارای بیشترین هزینه ترجیح بیشتری دهد و آن را بهینه‌سازی کند.
برای جبران این موضوع، می‌توانیم سطح اهمیت را به‌صورت وزن به توابع هزینه‌ی مربوطه نسبت دهیم.
(به‌طور ویژه برای مواردی که مقادیر اتلاف از مقیاس‌های مختلف تبعیت می‌کند.)

برای مثال، MSE برای رگرسیون سن معمولاً مقادیری بین 3 تا 5 به خود می گیرد،
اما CCE برای طبقه‌بندی جنسیت می‌تواند مقدار پایینی مثل 0.1 باشد.
برای متوازن‌سازی مشارکت هزینه‌های مختلف: وزن 10 برای CCE و وزن 0.25 برای MSE

مدل‌های چند–خروجی

خوراندن داده‌ها به یک مدل چند–خروجی

Feeding data to a multi-output model

```
model.fit(posts, [age_targets, income_targets, gender_targets],
          epochs=10, batch_size=64)
```

```
model.fit(posts, {'age': age_targets,
                  'income': income_targets,
                  'gender': gender_targets},
          epochs=10, batch_size=64)
```

Equivalent (possible only if you give names to the output layers)

age_targets, income_targets, and gender_targets are assumed to be Numpy arrays.

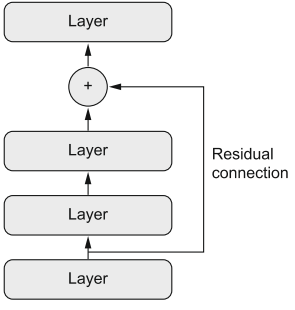
دو گزینه برای آموزش مدل دارای دو خروجی: ۱) از طریق لیستی از آرایه‌های numPy به‌عنوان خروجی
۲) می‌توانیم یک دیکشنری در اختیار آن قرار بدهیم که نام‌های خروجی را به آرایه‌های numPy نگاشت دهد.

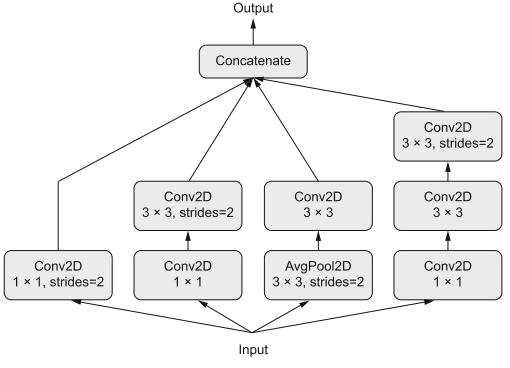
گراف‌های جهت‌دار بدون دور از لایه‌ها

DIRECTED ACYCLIC GRAPHS OF LAYERS

با استفاده از functional API می‌توان شبکه‌هایی با یک توپولوژی داخلی پیچیده را نیز پیاده‌سازی کرد.
شبکه‌های عصبی می‌توانند گراف‌های جهت‌دار بدون دور از لایه‌ها باشند.

قید «بدون دور»: این گراف‌ها نباید چرخه داشته باشند:
تانسور x نمی‌تواند ورودی یکی از لایه‌هایی شود که x را تولید می‌کند.
تنها حلقه‌های پردازشی مجاز، اتصالات بازگشتی داخلی لایه‌های بازگشتی است.





مثال:

ماژول‌های آغازش / inception modules و اتصالات مانده‌ای / residual connections

The purpose of 1 × 1 convolutions

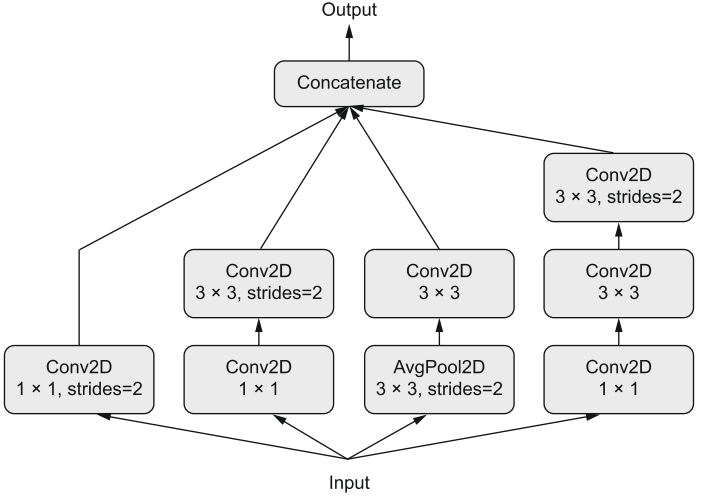
You already know that convolutions extract spatial patches around every tile in an input tensor and apply the same transformation to each patch. An edge case is when the patches extracted consist of a single tile. The convolution operation then becomes equivalent to running each tile vector through a `Dense` layer: it will compute features that mix together information from the channels of the input tensor, but it won't mix information across space (because it's looking at one tile at a time). Such 1 × 1 convolutions (also called *pointwise convolutions*) are featured in Inception modules, where they contribute to factoring out channel-wise feature learning and space-wise feature learning—a reasonable thing to do if you assume that each channel is highly autocorrelated across space, but different channels may not be highly correlated with each other.

گراف‌های جهت‌دار بدون دور از لایه‌ها

ماژول‌های آغازش

INCEPTION MODULES

Inception3 یک نوع متداول از معماری شبکه برای شبکه‌های عصبی کانوولوشنال؛
الهام گرفته شده از معماری قدیمی‌تر «شبکه–در–شبکه»؛
متشکل از یک پشته از ماژول‌هایی که خودشان مانند شبکه‌های مستقل کوچک به‌نظر می‌رسند
و به چند شاخه‌ی موازی تقسیم شده اند.



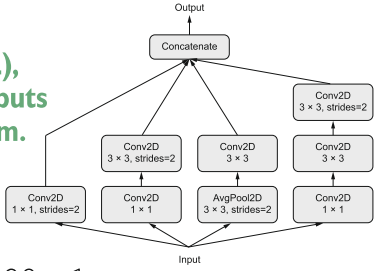
The most basic form of an Inception module has three to four branches starting with a 1 × 1 convolution, followed by a 3 × 3 convolution, and ending with the concatenation of the resulting features.
This setup helps the network separately learn spatial features and channel-wise features,
which is more efficient than learning them jointly.
More-complex versions of an Inception module are also possible, typically involving pooling operations, different spatial convolution sizes
(for example, 5 × 5 instead of 3 × 3 on some branches), and branches without a spatial convolution
(only a 1 × 1 convolution).

An example of such a module, taken from Inception V3

Min Lin, Qiang Chen, and Shuicheng Yan, “Network in Network,” International Conference on Learning Representations (2013), https://arxiv.org/abs/1312.4400.

گراف‌های جهت‌دار بدون دور از لایه‌ها

پیاده‌سازی ماژول آغازش Inception V3



Every branch has the same stride value (2), which is necessary to keep all branch outputs the same size so you can concatenate them.

In this branch, the striding occurs in the spatial convolution layer.

```
from keras import layers

branch_a = layers.Conv2D(128, 1,
                         activation='relu', strides=2)(x)
branch_b = layers.Conv2D(128, 1, activation='relu')(x)
branch_b = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3, strides=2)(x)
branch_c = layers.Conv2D(128, 3, activation='relu')(branch_c)

branch_d = layers.Conv2D(128, 1, activation='relu')(x)
branch_d = layers.Conv2D(128, 3, activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_d)

output = layers.concatenate(
    [branch_a, branch_b, branch_c, branch_d], axis=-1)
```

In this branch, the striding occurs in the average pooling layer.

Concatenates the branch outputs to obtain the module output

گراف‌های جهت‌دار بدون دور از لایه‌ها

ماژول آغازش شدید Xception

معماری کامل Inception V3 در کراس به‌صورت
keras.applications.inception_v3
موجود است که وزن‌های آن بر روی مجموعه داده‌ی ImageNet پیش‌آموزش دیده است.

یک مدل بسیار مرتبط که بخشی از ماژول کاربردهای کراس است Xception می‌باشد.
Xception = extreme inception
یک معماری کانوولوشنال که الهامی آزاد از Inception است.

این معماری ایده‌ی جداسازی یادگیری ویژگی‌های سطح کانال و سطح فضا را تا حد نهایی منطقی آن پی می‌گیرد
و ماژول‌های آغازش را با کانوولوشن‌های قابل جداسازی در سطح عمق جایگزین می‌کند
که شامل یک کانوولوشن عمقی (یک کانوولوشن فضایی که در آن هر کانال ورودی به‌صورت مجزا مدیریت می‌شود) و
بعد از آن یک کانوولوشن نقطه‌ای (یک کانوولوشن ۱×۱) است.
به‌طور مؤثر، این یک شکل اکستریم/افراطی از یک ماژول آغازش است
که در آن ویژگی‌های مکانی و ویژگی‌های سطح کانال کاملاً جدا شده‌اند.
تعداد پارامترهای Xception تقریباً مساوی با تعداد پارامترهای Inception V3 است،
اما کارآیی زمان اجرای آن بهتر است و دقت بالاتری روی ImageNet (همچون دیگر مجموعه‌داده‌های بزرگ–مقیاس) دارد.
(به‌دلیل استفاده‌ی کارآمدتر از پارامترهای مدل)

François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” Conference on Computer Vision and Pattern Recognition (2017), https://arxiv.org/abs/1610.02357.

گراف‌های جهت‌دار بدون دور از لایه‌ها

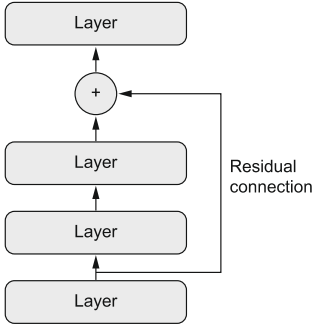
اتصالات مانده‌ای

RESIDUAL CONNECTIONS

اتصالات مانده‌ای یک جزء متداول در شبکه‌های گراف–مانند است
که در بسیاری از معماری‌های شبکه‌ای مربوط به پس از سال ۲۰۱۵ یافت می‌شوند (از جمله Xception).
(معرفی شده توسط He و همکاران از مایکروسافت در اواخر ۲۰۱۵ طی موفقیت در مسابقات ILSVRC ImageNet)

اتصالات مانده‌ای با دو مشکل متداول در مدل‌های یادگیری عمیق در مقیاس بزرگ مقابله می‌کنند:
گرادیان‌های مضمحل شونده و گلوگاه‌های بازنمایی

به‌طور کلی: اضافه کردن اتصالات مانده‌ای به هر مدلی که دارای بیش از ۱۰ لایه باشد، احتمالاً مفید خواهد بود.



A residual connection consists of making the output of an earlier layer available as input to a later layer, effectively creating a shortcut in a sequential network. Rather than being concatenated to the later activation, the earlier output is summed with the later activation, which assumes that both activations are the same size. If they're different sizes, you can use a linear transformation to reshape the earlier activation into the target shape (for example, a Dense layer without an activation or, for convolutional feature maps, a 1 × 1 convolution without an activation).

He et al., "Deep Residual Learning for Image Recognition," https://arxiv.org/abs/1512.03385.

گراف‌های جهت‌دار بدون دور از لایه‌ها

اتصالات مانده‌ای: نحوه‌ی پیاده‌سازی

RESIDUAL CONNECTIONS

هنگامی که اندازه‌ی نقشه‌های ویژگی **یکسان** است: استفاده از اتصالات مانده‌ای همانی:

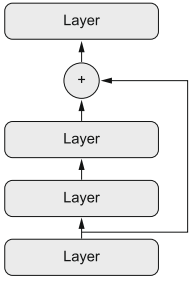
```
from keras import layers

x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)

y = layers.add([y, x])
```

Applies a transformation to x

Adds the original x back to the output features



هنگامی که اندازه‌ی نقشه‌های ویژگی **متفاوت** است: استفاده از اتصالات مانده‌ای خطی:

```
from keras import layers

x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.MaxPooling2D(2, strides=2)(y)

residual = layers.Conv2D(128, 1, strides=2, padding='same')(x)

y = layers.add([y, residual])
```

Uses a 1 × 1 convolution to linearly downsample the original x tensor to the same shape as y

Adds the residual tensor back to the output features

در هر دو مورد فرض می‌شود که تانسور ورودی چهاربعدی موجود است.

Representational bottlenecks in deep learning

In a `Sequential` model, each successive representation layer is built on top of the previous one, which means it only has access to information contained in the activation of the previous layer. If one layer is too small (for example, it has features that are too low-dimensional), then the model will be constrained by how much information can be crammed into the activations of this layer.

You can grasp this concept with a signal-processing analogy: if you have an audio-processing pipeline that consists of a series of operations, each of which takes as input the output of the previous operation, then if one operation crops your signal to a low-frequency range (for example, 0–15 kHz), the operations downstream will never be able to recover the dropped frequencies. Any loss of information is permanent. Residual connections, by reinjecting earlier information downstream, partially solve this issue for deep-learning models.

Vanishing gradients in deep learning

Backpropagation, the master algorithm used to train deep neural networks, works by propagating a feedback signal from the output loss down to earlier layers. If this feedback signal has to be propagated through a deep stack of layers, the signal may become tenuous or even be lost entirely, rendering the network untrainable. This issue is known as *vanishing gradients*.

This problem occurs both with deep networks and with recurrent networks over very long sequences—in both cases, a feedback signal must be propagated through a long series of operations. You're already familiar with the solution that the `LSTM` layer uses to address this problem in recurrent networks: it introduces a *carry track* that propagates information parallel to the main processing track. Residual connections work in a similar way in feedforward deep networks, but they're even simpler: they introduce a purely linear information carry track parallel to the main layer stack, thus helping to propagate gradients through arbitrarily deep stacks of layers.

به‌اشتراک‌گذاری وزن لایه

LAYER WEIGHT SHARING

یک ویژگی مهم تر API تابعی، توانایی به‌کارگیری چندباره‌ی یک نمونه لایه است.
هنگامی که یک نمونه لایه را دو بار فراخوانی می‌کنیم،
به‌جای آن که برای هر فراخوانی یک نمونه‌ی جدید از آن لایه ایجاد شود،
با هر فراخوانی، همان وزن‌ها را مجدداً به‌کار می‌گیریم.

⇓

امکان ایجاد مدل‌هایی دارای شاخه‌های به‌اشتراک‌گذاری شده

(یعنی چند لایه که همگی دانایی یکسانی را به‌اشتراک می‌گذارند و عملیات مشترکی را اجرا می‌کنند:
یعنی؛ پارامترهای یکسانی را به‌اشتراک می‌گذارند و
این بازنمایی‌ها را به‌طور همزمان برای مجموعه‌های مختلفی از ورودی‌ها یاد می‌گیرند.)

مثال: مدلی که تلاش می‌کند تشابه معنایی میان دو جمله را ارزیابی کند.
این مدل دو ورودی دارد (دو جمله‌ی مورد مقایسه)
و یک امتیاز بین صفر و یک را به‌عنوان خروجی برمی‌گرداند
صفر: نامرتبط بودن جملات **یک**: دو جمله یکسان هستند یا صورت‌بندی مجدد یکدیگر هستند.

در این پیکربندی، دو جمله‌ی ورودی تعویض‌پذیر با یکدیگر هستند: تشابه معنایی یک رابطه‌ی متقارن است.
پس لازم نیست دو مدل مستقل برای پردازش هر جمله‌ی ورودی یاد گرفته شود.

به‌اشتراک‌گذاری وزن لایه

مدل Siamese LSTM یا LSTM اشتراکی

SIAMESE LSTM MODEL / SHARED LSTM

می‌خواهیم هر دو جمله را با یک لایه‌ی LSTM واحد پردازش کنیم.
بازنمایی‌های این لایه‌ی LSTM (وزن‌های آن) بر اساس هر دو ورودی به‌صورت همزمان یادگیری می‌شوند.

نحوه‌ی پیاده‌سازی چنین مدلی با استفاده از به‌اشتراک‌گذاری لایه (استفاده‌ی مجدد از لایه) با API تابعی کراس:

```
from keras import layers
from keras import Input
from keras.models import Model

lstm = layers.LSTM(32)

left_input = Input(shape=(None, 128))
left_output = lstm(left_input)

right_input = Input(shape=(None, 128))
right_output = lstm(right_input)

merged = layers.concatenate([left_output, right_output], axis=-1)
predictions = layers.Dense(1, activation='sigmoid')(merged)

model = Model([left_input, right_input], predictions)
model.fit([left_data, right_data], targets)
```

Instantiates a single LSTM layer, once

Building the left branch of the model: inputs are variable-length sequences of vectors of size 128.

Building the right branch of the model: when you call an existing layer instance, you reuse its weights.

Builds the classifier on top

Instantiating and training the model: when you train such a model, the weights of the LSTM layer are updated based on both inputs.

مدل‌ها به‌عنوان لایه‌ها

MODELS AS LAYERS

در API تابعی، می‌توان مدل‌ها را به‌همان شیوه‌ای که لایه‌ها را مورد استفاده قرار می‌دهیم، به‌کار ببریم.
یک مدل را می‌توانیم به‌عنوان یک لایه‌ی بزرگ‌تر در نظر بگیریم.

می‌توانیم مدل را برای یک تانسور ورودی فراخوانی کنیم و یک تانسور خروجی را بازیابی کنیم:

```
y = model(x)
```

اگر مدل دارای چند تانسور ورودی و چند تانسور خروجی باشد، باید با لیستی از تانسورها فراخوانی شود:

```
y1, y2 = model([x1, x2])
```

هنگامی‌که نمونه‌ای از یک مدل را فراخوانی می‌کنیم،
در واقع، در حال استفاده‌ی مجدد از وزن‌های مدل هستیم
(دقیقاً مشابه آنچه هنگام فراخوانی نمونه‌ای از لایه رخ می‌دهد).
فراخوانی یک نمونه (چه نمونه‌ای از لایه، چه نمونه‌ای از مدل)
همواره بازنمایی‌های یادگیری شده‌ی موجود آن نمونه را مجدداً به‌کار خواهد گرفت.

مدل‌ها به‌عنوان لایه‌ها

مثال: مدل بینایی سیامی (با پایه‌ی کانوولوشنال مشترک)

SIAMESE VISION MODEL (SHARED CONVOLUTIONAL BASE)

مثال: یک مدل بینایی که از یک دوربین دوتایی به‌عنوان ورودی خود استفاده می‌کند:
دو دوربین موازی، در فاصله‌ی یک اینچی از یکدیگر.
این مدل می‌تواند عمق را درک کند.
در اینجا به دو مدل مستقل برای استخراج ویژگی‌های بصری از دوربین‌های چپ و راست نیاز نداریم.
این پردازش سطح پایین را می‌توانیم بین دو ورودی به‌اشتراک بگذاریم: از طریق لایه‌هایی با وزن‌های یکسان

```
from keras import layers
from keras import applications
from keras import Input

xception_base = applications.Xception(weights=None,
                                      include_top=False)

left_input = Input(shape=(250, 250, 3))
right_input = Input(shape=(250, 250, 3))

left_features = xception_base(left_input)
right_input = xception_base(right_input)

merged_features = layers.concatenate(
    [left_features, right_input], axis=-1)
```

The base image-processing model is the Xception network (convolutional base only).

The inputs are 250 × 250 RGB images.

Calls the same vision model twice

The merged features contain information from the right visual feed and the left visual feed.

جمع‌بندی

WRAPPING UP

This concludes our introduction to the Keras functional API—an essential tool for building advanced deep neural network architectures.
Now you know the following:

- To step out of the Sequential API whenever you need anything more than a linear stack of layers
- How to build Keras models with several inputs, several outputs, and complex internal network topology, using the Keras functional API
- How to reuse the weights of a layer or model across different processing branches, by calling the same layer or model instance several times

بهترین روش‌های پیشرفته
برای یادگیری عمیق

۲

معاینه و نظارت بر مدل‌های یادگیری عمیق با استفاده از پس‌خوانی‌های کراس و تنسوربورد

معاینه و نظارت بر مدل‌های یادگیری عمیق با استفاده از پس‌خوانی‌های کراس و تنسوربورد

INSPECTING AND MONITORING DEEP-LEARNING MODELS USING KERAS CALLBACKS AND TENSORBOARD

روش‌هایی برای دسترسی بیشتر و کنترل
آنچه در حین **آموزش** درون مدل رخ می‌دهد.

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

USING CALLBACKS TO ACT ON A MODEL DURING TRAINING

در هنگام آموزش مدل، موارد متعددی وجود دارند که نمی‌توان آنها را از ابتدا پیش‌بینی کرد.

خصوصاً: نمی‌توان گفت برای رسیدن به یک مقدار اتلاف اعتبارسنجی بهینه چند اپک باید طی شود.
روش مناسب:
هنگامی که متوجه شدیم اتلاف اعتبارسنجی دیگر رو به بهبود نیست، آموزش را متوقف کنیم.

پس‌خوانی *Callback*	یک پس‌خوانی، یک شیئ (نمونه‌ای از یک کلاس) است که در فراخوانی fit به مدل ارسال می‌شود و در نقاط مختلف طی فرآیند آموزش توسط مدل فراخوانی می‌شود.

پس‌خوانی به تمام داده‌های موجود مربوط به وضعیت مدل و عملکرد آن دسترسی دارد
و می‌تواند اقدام‌های زیر را انجام دهد:
متوقف‌سازی فرآیند آموزش، ذخیره‌سازی مدل، بارگذاری مجموعه‌ی متفاوتی از وزن‌ها، تغییر وضعیت مدل

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

مثال‌هایی از روش‌های به‌کارگیری پس‌خوانی‌ها

نقطه‌بررسی‌گذاری مدل *Model checkpointing*	ذخیره‌سازی وزن‌های کنونی مدل در نقاط مختلف طی فرآیند آموزش
توقف زودهنگام *Early stopping*	متوقف ساختن آموزش هنگامی که در مقدار اتلاف اعتبارسنجی بهبودی دیده نمی‌شود (+ ذخیره‌سازی بهترین مدل)
تنظیم پویای پارامترها *Dynamic parameter adjusting*	تنظیم پویای مقدار پارامترهای خاص در طی آموزش؛ مانند نرخ یادگیری برای بهینه‌ساز
ثبت متریک‌های آموزش و اعتبارسنجی *Logging training and validation metrics*	لاگ‌گیری از متریک‌ها طی آموزش یا مصورسازی بازنمایی‌های یادگیری شده توسط مدل هنگام به‌روزرسانی آنها

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

پس‌خوانی‌های پیش‌ساخته

ماژول keras.callbacks شامل تعدادی پس‌خوانی از پیش‌ساخته است:

```
keras.callbacks.ModelCheckpoint
keras.callbacks.EarlyStopping
keras.callbacks.LearningRateScheduler
keras.callbacks.ReduceLROnPlateau
keras.callbacks.CSVLogger
```

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

پس‌خوانی‌های ModelCheckpoint و EarlyStopping

پس‌خوانی EarlyStopping:

برای متوقف ساختن فرآیند آموزش در زمانی که
یک معیار هدف تحت نظارت به‌ازای تعداد ثابتی از اپک‌ها از بهبود بازمانده باشد.
(برای مثال این پس‌خوانی این امکان را ایجاد می‌کند که به‌محض آنکه با بیش‌برازش مواجه شدیم،
بیش‌برازش را متوقف کند و از الزام به آموزش مجدد مدل به‌ازای تعداد کمتری از اپک‌ها جلوگیری می‌کند)

پس‌خوانی ModelCheckpoint:

این امکان را می‌دهد که مدل را در حین فرآیند آموزش دائماً ذخیره کنیم.
(و به‌دلخواه، فقط بهترین مدل کنونی را تا زمان جاری ذخیره کنیم:
نسخه‌ای از مدل که در پایان یک اپک به بهترین کارآیی دست یافته است.)

پس‌خوانی EarlyStopping معمولاً در ترکیب با ModelCheckpoint استفاده می‌شود.

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

پس‌خوانی‌های ModelCheckpoint و EarlyStopping

Callbacks are passed to the model via the callbacks argument in fit, which takes a list of callbacks. You can pass any number of callbacks.

Interrupts training when improvement stops

Monitors the model's validation accuracy

Interrupts training when accuracy has stopped improving for more than one epoch (that is, two epochs)

Saves the current weights after every epoch

Path to the destination model file

These two arguments mean you won't overwrite the model file unless val_loss has improved, which allows you to keep the best model seen during training.

You monitor accuracy, so it should be part of the model's metrics.

Note that because the callback will monitor validation loss and validation accuracy, you need to pass validation_data to the call to fit.

```
import keras

callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=1,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath='my_model.h5',
        monitor='val_loss',
        save_best_only=True,
    )
]

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

پس‌خوانی ReduceLROnPlateau

پس‌خوانی ReduceLROnPlateau:

برای کاهش نرخ یادگیری هنگام متوقف شدن بهبود اتلاف اعتبارسنجی.

(کاهش یا افزایش نرخ یادگیری در صورت بروز وضعیت ثابت اتلاف،
یک استراتژی مؤثر برای خارج شدن از می‌نیمم‌های محلی در طی فرآیند آموزش است.)

```
callbacks_list = [
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss'
        factor=0.1,
        patience=10,
    )
]

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```

Monitors the model's validation loss

Divides the learning rate by 10 when triggered

The callback is triggered after the validation loss has stopped improving for 10 epochs.

Because the callback will monitor the validation loss, you need to pass validation_data to the call to fit.

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

نوشتن پس‌خوانی‌های اختصاصی

WRITING YOUR OWN CALLBACK

اگر لازم است در حین فرآیند آموزش، عملی خاص انجام شود که در پس‌خوانی‌های پیش‌ساخته وجود ندارد، می‌توانیم پس‌خوانی مختص به خود را بنویسیم.
پس‌خوانی‌ها از طریق ایجاد زیرکلاس‌هایی از کلاس keras.callbacks.Callback پیاده‌سازی می‌شوند.
سپس متدهای پیاده‌سازی شده در طی آموزش در نقاط مختلف می‌توانند فراخوانی شوند:

```
on_epoch_begin      <-- Called at the start of every epoch
on_epoch_end        <-- Called at the end of every epoch

on_batch_begin      <-- Called right before processing each batch
on_batch_end        <-- Called right after processing each batch

on_train_begin      <-- Called at the start of training
on_train_end        <-- Called at the end of training
```

این متدها همگی با یک آرگومان logs فراخوانی می‌شوند که یک دیکشنری حاوی موارد زیر است:
اطلاعاتی در مورد دسته/batch قبلی، اپک، اجرای آموزش قبلی، متریک‌های آموزش و اعتبارسنجی، ...
پس‌خوانی به خصیصه‌های زیر نیز دسترسی دارد:

- self.model: نمونه‌ای از مدل که فراخوانی پس‌خوانی از آن شروع شده است.
- self.validation_data: مقداری از آنچه به‌عنوان داده‌ی اعتبارسنجی به fit گذر داده شده است.

استفاده از پس‌خوانی‌ها برای اقدام بر روی مدل در حین آموزش

نوشتن پس‌خوانی‌های اختصاصی: مثال

ذخیره‌سازی فعالیت‌های هر لایه‌ی مدل در انتهای هر اپک
به‌صورت یک آرایه‌ی numPy بر روی دیسک
که برای اولین نمونه از مجموعه‌ی اعتبارسنجی محاسبه می‌شود.

```
import keras
import numpy as np

class ActivationLogger(keras.callbacks.Callback):

    def set_model(self, model):
        self.model = model
        layer_outputs = [layer.output for layer in model.layers]
        self.activations_model = keras.models.Model(model.input,
                                                    layer_outputs)

    def on_epoch_end(self, epoch, logs=None):
        if self.validation_data is None:
            raise RuntimeError('Requires validation_data.')

validation_sample = self.validation_data[0][0:1]
activations = self.activations_model.predict(validation_sample)
f = open('activations_at_epoch_' + str(epoch) + '.npz', 'w')
np.savez(f, activations)
f.close()
```

Called by the parent model before training, to inform the callback of what model will be calling it

Model instance that returns the activations of every layer

Obtains the first input sample of the validation data

Saves arrays to disk

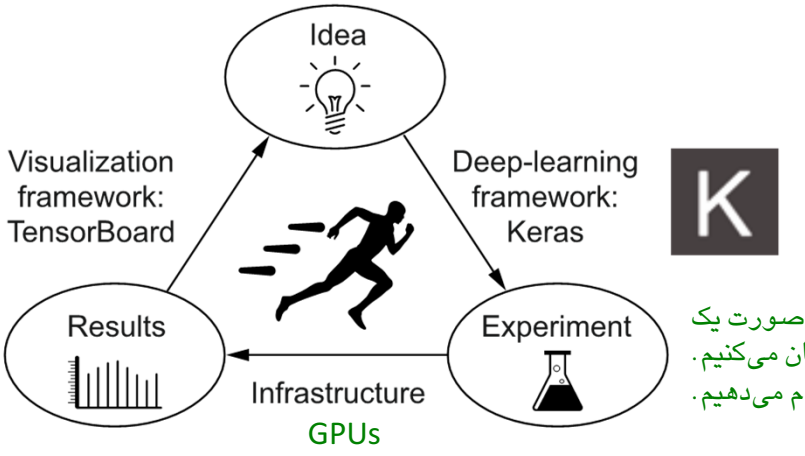
مقدمه‌ای بر تنسوربورد: چارچوب کاری مصورسازی تنسورفلو

INTRODUCTION TO TENSORBOARD: THE TENSORFLOW VISUALIZATION FRAMEWORK

برای انجام یک پژوهش خوب یا توسعه‌ی مدل‌های خوب،
به فیدبک‌های پربار و مکرر درباره‌ی آنچه در حین آزمایش‌ها درون مدل‌های ما رخ می‌دهد، نیاز داریم.

دستیابی به پیشرفت یک فرآیند تکراری یا حلقه است:

۱) با یک ایده شروع می‌کنیم.



۲) ایده را به‌صورت یک آزمایش بیان می‌کنیم.
آزمایش را انجام می‌دهیم.

۳) اطلاعات حاصل از آزمایش را پردازش می‌کنیم که الهام‌بخش ایده‌ی بعدی است.

مقدمه‌ای بر تنسوربورد

چارچوب کاری مصورسازی تنسورفلو

TensorBoard

یک ابزار مصورسازی مبتنی بر مرورگر که در بسته‌ای همراه با TensorFlow در دسترس است.

از این ابزار تنها زمانی می‌توان برای مدل‌های کراس استفاده کرد که از کراس با بَک‌اِند تنسورفلو استفاده کنیم.

هدف: کمک به پایش تمام رویدادهای درون مدل به‌صورت مصَور

- ❑ پایش مصور متریک‌ها در طی فرآیند آموزش
- ❑ مصورسازی معماری مدل شما
- ❑ مصورسازی نمودارهای هیستوگرام فعالیت‌ها و گرادیان‌ها
- ❑ بررسی و کاوش جاسازی‌های کلمات به‌صورت سه–بعدی

مقدمه‌ای بر تنسوربورد

مثال: آموزش یک شبکه‌ی کانوولوشنال یک–بعدی برای وظیفه‌ی تحلیل احساسات IMDB

Text-classification model to use with TensorBoard

استفاده از ۲۰۰۰ کلمه‌ی اول با هدف مصورسازی بهتر جاسازی کلمات

```
import keras
from keras import layers
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 2000
max_len = 500

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

model = keras.models.Sequential()
model.add(layers.Embedding(max_features, 128,
                           input_length=max_len,
                           name='embed'))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
model.summary()
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
```

Number of words to consider as features

Cuts off texts after this number of words (among max_features most common words)

مقدمه‌ای بر تنسوربورد

مثال: آموزش یک شبکه‌ی کانوولوشنال یک-بعدی برای وظیفه‌ی تحلیل احساسات IMDB: استفاده از پس‌خوانی تنسوربورد

پیش از شروع به استفاده از تنسوربورد، باید یک دایرکتوری ایجاد کنیم تا فایل‌های لاگ در آنجا ذخیره شود:

Creating a directory for TensorBoard log files

```
$ mkdir my_log_dir
```

آموزش را با یک نمونه پس‌خوانی تنسوربورد شروع می‌کنیم (نوشتن رویدادهای لاگ بر روی دیسک):

Training the model with a TensorBoard callback

```
callbacks = [
    keras.callbacks.TensorBoard(
        log_dir='my_log_dir',
        histogram_freq=1,
        embeddings_freq=1,
    )
]
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=128,
                    validation_split=0.2,
                    callbacks=callbacks)
```

Log files will be written at this location.

Records activation histograms every 1 epoch

Records embedding data every 1 epoch

مقدمه‌ای بر تنسوربورد

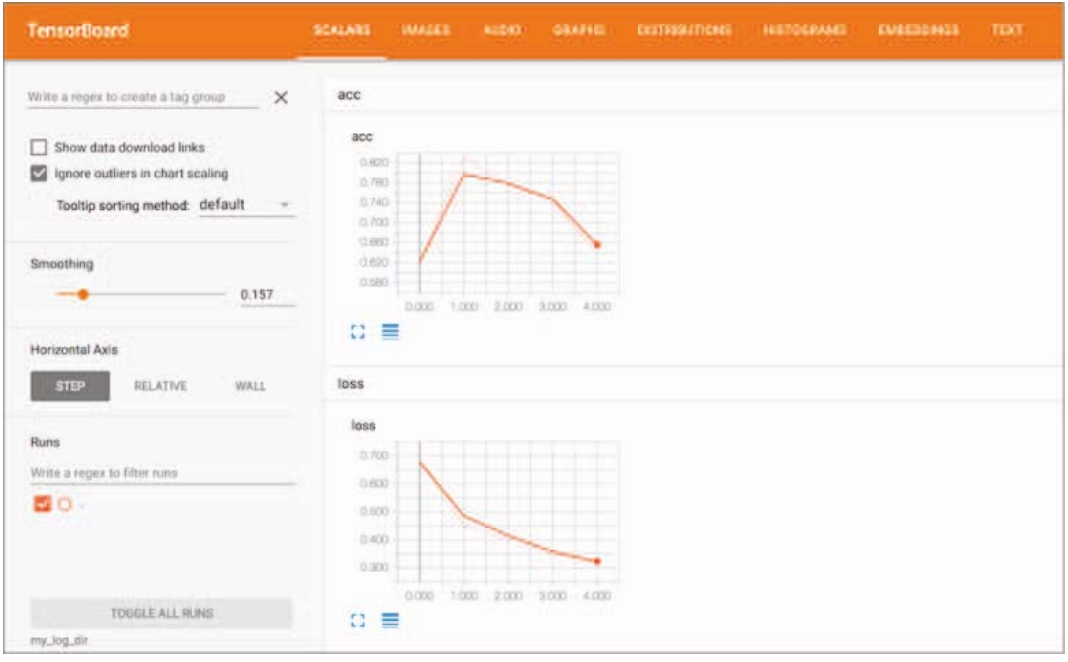
مثال: آموزش یک شبکه‌ی کانوولوشنال یک-بعدی برای وظیفه‌ی تحلیل احساسات IMDB: استفاده از پس‌خوانی تنسوربورد

سرور تنسوربورد را از خط فرمان راه‌اندازی می‌کنیم
و به آن دستور می‌دهیم که لاگ‌هایی که هم‌اکنون توسط پس‌خوانی نوشته می‌شود را بخواند.

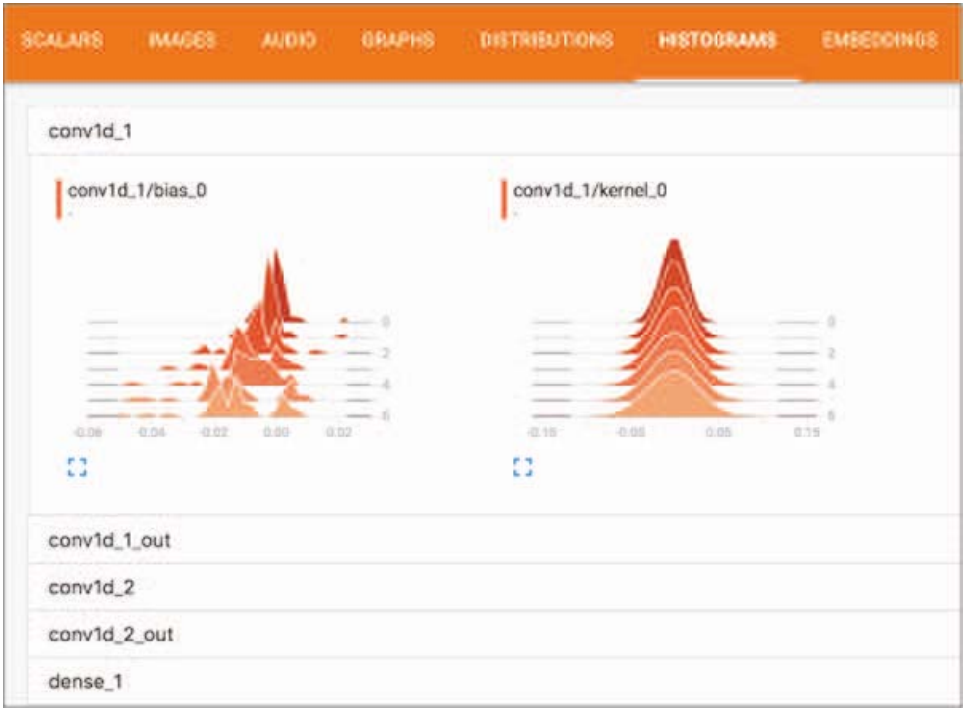
```
$ tensorboard --logdir=my_log_dir
```

آدرس زیر را در مرورگر وارد می‌کنیم و به فرآیند آموزش مدل خود نگاه می‌کنیم:
http://localhost:6006

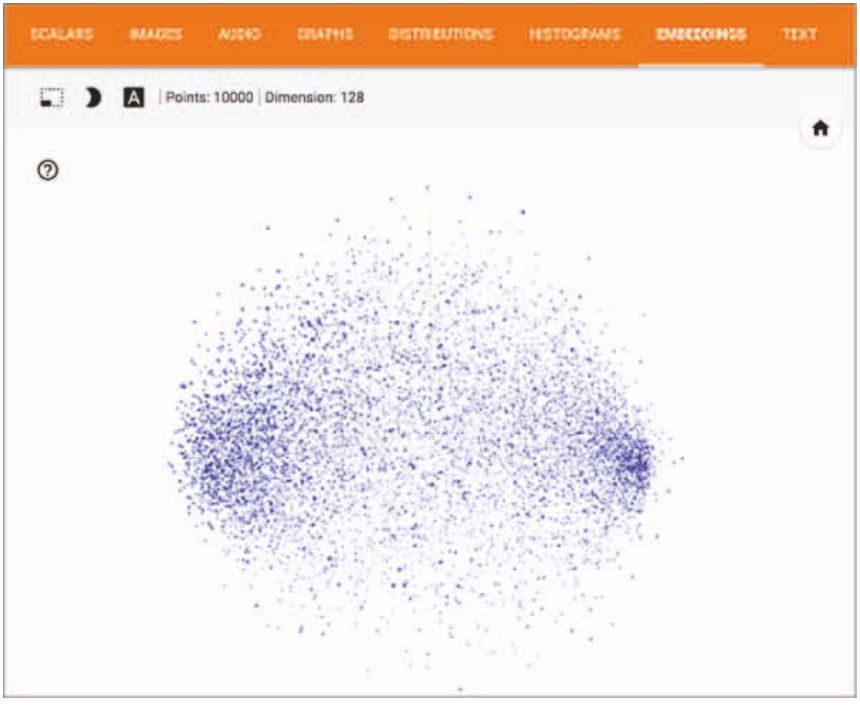
حاوی: گراف‌ها، نمودارهای لحظه‌ای (زنده) از متریک‌های آموزش و اعتبارسنجی،
دسترسی به برگه‌ی هیستوگرام (حاوی مصورسازی‌های زیبا از هیستوگرام‌های مقادیر فعالیت هر لایه)،
دسترسی به برگه‌ی جاسازی،
دسترسی به برگه‌ی گراف‌ها،
....



نظارت بر متریک‌ها با TensorBoard

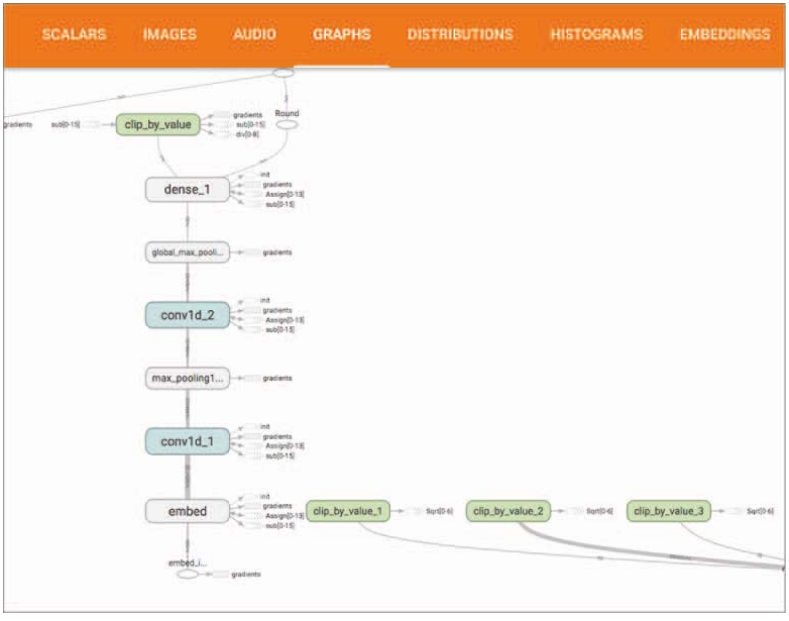


هیستوگرام‌های فعالیت‌های لایه‌ها در TensorBoard



مصورسازی جاسازی کلمات سه–بعدی تعاملی در TensorBoard

برگه‌ی Embedding امکان بررسی مکان‌های جاسازی و روابط فضایی ۱۰,۰۰۰ کلمه‌ی موجود در فهرست واژگان ورودی که توسط لایه‌ی جاسازی اولیه یادگیری شدند را فراهم می‌کند.
فضای جاسازی با الگوریتم کاهش بعد PCA یا t-SNE به دو یا سه بعد کاهش بعد می‌یابد.
در مثال ما، کلمات با مفهوم مثبت و کلمات با مفهوم منفی (متناسب با کاربرد ما) خوشه‌بندی شده‌اند.



مصورسازی گراف تنسورفلو در TensorBoard

برگه‌ی Graphs یک نمای تصویری تعاملی از گراف عملیات سطح پایین تنسورفلو که زیربنای مدل کراس ماست را نشان می‌دهد.
مدل ساده‌ای که در کراس پشته‌ی کوچکی از لایه‌هاست، در پشت صحنه‌ی تنسورفلو یک ساختار گرافی نسبتاً پیچیده دارد.
بخش زیادی از این مدل مربوط به فرآیند کاهش گرادیان است.
کراس گردش کار را بسیار ساده‌تر می‌کند.

رسم مدل‌ها به‌صورت گراف‌هایی از لایه‌ها

در کراس

کراس روش واضح‌تر دیگری را ارائه می‌دهد که
به‌جای رسم مدل‌ها به‌صورت گراف‌هایی از عملیات تنسورفلو،
مدل‌ها را به‌صورت گراف‌هایی از لایه‌ها ترسیم کنیم:

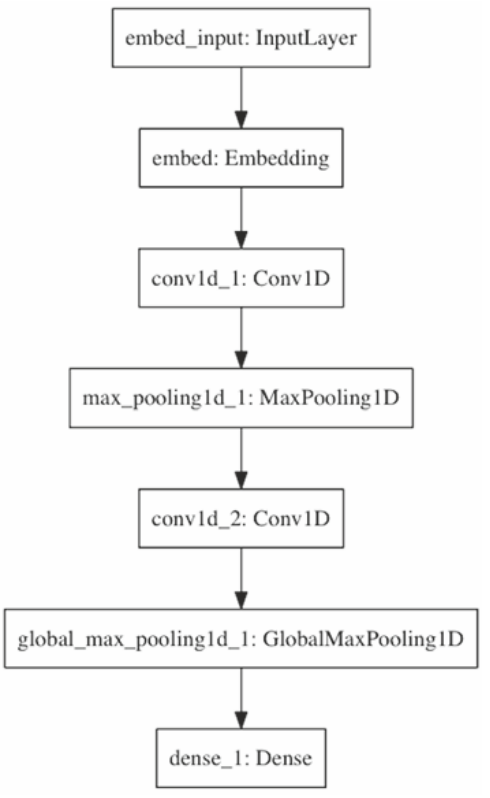
`keras.utils.plot_model`

لازم است کتابخانه‌های pydot، pydot-ng و graph-viz نصب شده باشد.

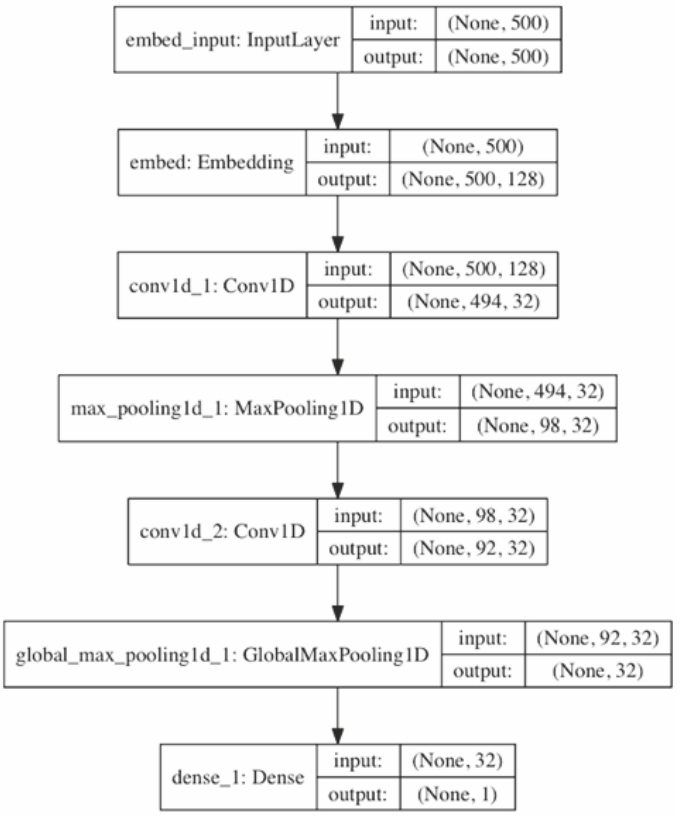
```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

گزینه‌ی انتخابی نمایش اطلاعات شکل تانسورها در گراف لایه‌ها نیز وجود دارد:

```
from keras.utils import plot_model
plot_model(model, show_shapes=True, to_file='model.png')
```




```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```




```
from keras.utils import plot_model
plot_model(model, show_shapes=True, to_file='model.png')
```

جمع‌بندی

WRAPPING UP

- Keras callbacks provide a simple way to monitor models during training and automatically take action based on the state of the model.
- When you're using TensorFlow, TensorBoard is a great way to visualize model activity in your browser. You can use it in Keras models via the TensorBoard callback.

بهترین روش‌های پیشرفته
برای یادگیری عمیق

۳

برداشت بیشترین بهره از مدل‌ها

برداشت بیشترین بهره از مدل‌ها

GETTING THE MOST OUT OF YOUR MODELS

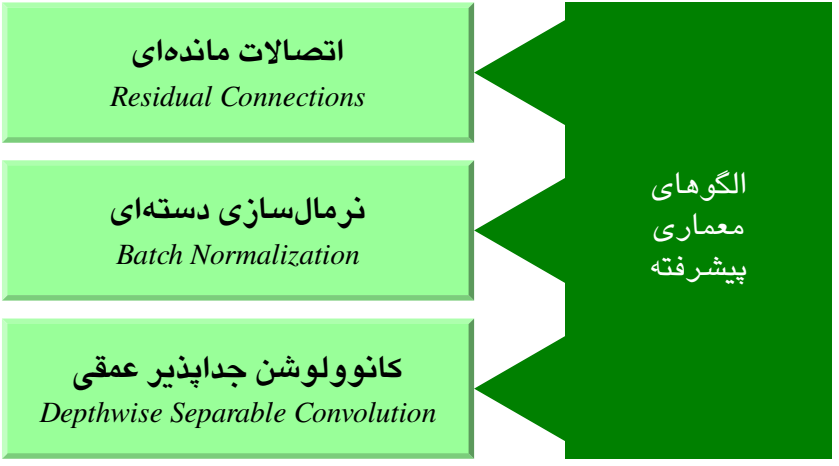
اگر به‌دنبال یک معماری هستیم که خوب کار کند،
آزمودن کورکورانه‌ی معماری‌ها برای این کار کافی است؛

اما

اگر به‌دنبال یک معماری هسیتم که عالی کار کند
و در رقابت‌های یادگیری ماشینی پیروز شود،
به مجموعه‌ای از تکنیک‌ها برای ساخت مدل‌های جدید یادگیری عمیق نیاز داریم.

الگوهای معماری پیشرفته

ADVANCED ARCHITECTURE PATTERNS



این الگوها خصوصاً برای ساخت شبکه‌های عصبی کانوولوشنال عمیق با کارآیی بالا مناسب هستند، اما معمولاً در سایر انواع معماری‌ها نیز یافت می‌شوند.

الگوهای معماری پیشرفته

نرمال‌سازی دسته‌ای

BATCH NORMALIZATION

نرمال‌سازی، مجموعه‌ی گسترده‌ای از روش‌هاست
که هدف آنها شبیه‌تر ساختن نمونه‌های متفاوتی است که توسط یک مدل یادگیری ماشینی دیده شده است.
نرمال‌سازی به مدل کمک می‌کند که داده‌ی جدید را به‌خوبی یاد گرفته و تعمیم بدهد.

متداول‌ترین شکل نرمال‌سازی داده‌ها: متمرکزسازی داده‌ها حول صفر (میانگین صفر–واریانس یک)
در اصل: فرض می‌شود که داده‌ها از یک توزیع نرمال (گاوسی) پیروی می‌کنند

```
normalized_data = (data - np.mean(data, axis=...)) / np.std(data, axis=...)
```

این نوع نرمال‌سازی پیش از خوراندن داده‌ها به مدل آنها را نرمال‌سازی می‌کند؛
اما نرمال‌سازی داده‌ها پس از هر تبدیل انجام شده توسط شبکه، باید یک دغدغه باشد:
حتی اگر داده‌های ورودی به شبکه دارای میانگین صفر و واریانس یک باشد،
دلیلی وجود ندارد که این شرایط برای داده‌های خروجی (هر لایه) نیز صدق کند.

⇓

نیاز به نرمال‌سازی پس از عملیات هر لایه
(نرمال‌سازی دسته‌ای)

الگوهای معماری پیشرفته

نرمال‌سازی دسته‌ای: روش کار و اثر

BATCH NORMALIZATION

نرمال‌سازی دسته‌ای، نوعی لایه است که می‌تواند داده‌ها را به‌صورت وفقی نرمال‌سازی کند؛
حتی در زمانی که میانگین و واریانس در فرآیند آموزش طی زمان تغییر کند.

روش کار:
یک میانگین متحرک نمایی از میانگین و واریانس دسته‌ای/batch-wise
از داده‌هایی که در طول آموزش مشاهده شده‌اند، به‌صورت داخلی نگهداری می‌شود.

اثر اصلی نرمال‌سازی دسته‌ای، کمک به انتشار گرادیان (مشابه اتصالات مانده‌ای) است،
و بنابراین امکان ایجاد شبکه‌های عمیق‌تر را فراهم می‌کند

(برخی شبکه‌های عمیق را تنها در صورتی می‌توان آموزش داد که شامل چند لایه‌ی BN باشند؛
مثل ResNet50، InceptionV3، Xception)

Sergey Ioffe and Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Proceedings of the 32nd International Conference on Machine Learning (2015), https://arxiv.org/abs/1502.03167.

الگوهای معماری پیشرفته

نرمال‌سازی دسته‌ای: به‌کارگیری

لایه‌ی BatchNormalization معمولاً پس از یک لایه‌ی کانوولوشنال یا یک لایه‌ی متصل متراکم استفاده می‌شود:

```
conv_model.add(layers.Conv2D(32, 3, activation='relu'))   <-- After a Conv layer
conv_model.add(layers.BatchNormalization())

dense_model.add(layers.Dense(32, activation='relu'))      <-- After a Dense layer
dense_model.add(layers.BatchNormalization())
```

لایه‌ی BatchNormalization یک آرگومان axis را دریافت می‌کند،
که محور مربوط به آن ویژگی که باید نرمال‌سازی شود را مشخص می‌کند.
مقدار پیش‌فرض آن برابر با 1- است: آخرین محوردر تانسور ورودی.

مقدار پیش‌فرض 1- برای استفاده با لایه‌های Dense، RNN و Conv2D که در آنها
data_format به‌مقدار "channels-last" تنظیم شده است، صحیح است.
اما
برای استفاده با لایه‌های Conv2D که در آنها
data_format به‌مقدار "channels-first" تنظیم شده است، محور ویژگی، محور 1 است،
پس باید axis = 1 تنظیم شود.

Batch renormalization

A recent improvement over regular batch normalization is *batch renormalization*, introduced by Ioffe in 2017.[a] It offers clears benefits over batch normalization, at no apparent cost. At the time of writing, it's too early to tell whether it will supplant batch normalization—but I think it's likely. Even more recently, Klambauer et al. introduced *self-normalizing neural networks*,[b] which manage to keep data normalized after going through any `Dense` layer by using a specific activation function (`selu`) and a specific initializer (`lecun_normal`). This scheme, although highly interesting, is limited to densely connected networks for now, and its usefulness hasn't yet been broadly replicated.

---

[a] Sergey Ioffe, "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models" (2017), https://arxiv.org/abs/1702.03275.

[b] Günter Klambauer et al., "Self-Normalizing Neural Networks," Conference on Neural Information Processing Systems (2017), https://arxiv.org/abs/1706.02515.

الگوهای معماری پیشرفته

کانوولوشن جداپذیر عمقی

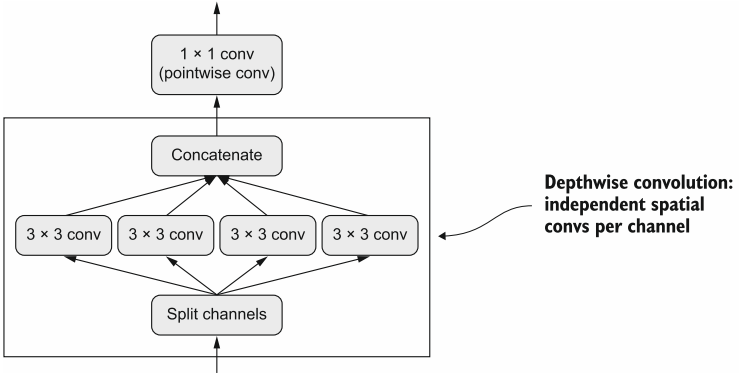
DEPTHWISE SEPARABLE CONVOLUTION

لایه‌ی SeparableConv2D لایه‌ای است که به‌عنوان جایگزین Conv2D قابل استفاده است؛
و باعث می‌شود مدل سبک‌تر (تعدادن وزن قابل آموزش کمتر) و سریع‌تر (تعداد عملیات ممیزشناور کمتر) شود
و به‌علاوه موجب بهبود میزان کارآیی در حد چند درصد می‌شود.

این لایه، پیش از مخلوط کردن کانال‌های خروجی از طریق یک کانوولوشن نقطه‌ای (۱×۱)،
اقدام به اجرای یک کانوولوشن فضایی بر روی هر کانال ورودی خود به‌طور مستقل می‌کند.

این معادل است با: جداسازی یادگیری ویژگی‌های فضایی و یادگیری ویژگی‌های کانالی.
این کار مفهوم بسیاری دارد اگر فرض کنیم مکان‌های فضایی در ورودی شدیداً همبسته هستند، اما کانال‌های مختلف نسبتاً مستقل هستند.

این لایه تعداد بسیار کمتری پارامتر نیاز دارد و شامل محاسبات کمتری است، در نتیجه مدل‌های سریع‌تر و کوچک‌تری را نتیجه می‌دهد.
از آنجا که این روش اجرای کانوولوشن از لحاظ بازنمایی کارآمدتر است، بنابراین با داده‌های کمتر، بازنمایی‌های بهتری را یاد می‌گیرد و به مدل‌هایی با کارآیی بهتر منجر می‌شود.



الگوهای معماری پیشرفته

کانوولوشن جداپذیر عمقی

DEPTHWISE SEPARABLE CONVOLUTION

مزیت‌های لایه‌ی SeparableConv2D خصوصاً هنگامی اهمیت می‌یابند
که قصد داریم مدل‌های کوچکی را از ابتدا با داده‌های محدود آموزش بدهیم.

وقتی به‌سراغ مدل‌هایی با مقیاس بزرگ‌تر می‌رویم،
کانوولوشن جداپذیر عمقی، پایه و اساس معماری Xception است.

François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” Conference on Computer Vision and Pattern Recognition (2017), https://arxiv.org/abs/1610.02357.

الگوهای معماری پیشرفته

کانوولوشن جداپذیر عمقی

DEPTHWISE SEPARABLE CONVOLUTION

مثال: ساخت یک شبکه‌ی کانوولوشنال جداپذیر عمیق
سبک برای یک وظیفه‌ی طبقه‌بندی تصویر
در یک مجموعه‌داده‌ی کوچک
(softmax categorical classification)

```
from keras.models import Sequential, Model
from keras import layers

height = 64
width = 64
channels = 3
num_classes = 10

model = Sequential()
model.add(layers.SeparableConv2D(32, 3,
                                 activation='relu',
                                 input_shape=(height, width, channels,)))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

بهینه‌سازی هایپرپارامترها

HYPERPARAMETER OPTIMIZATION

هنگام ساخت یک مدل یادگیری عمیق، باید در مورد تعداد زیادی از گزینه‌های ظاهراً اختیاری تصمیم‌گیری کنیم:

تعداد لایه در پشته‌سازی؟
تعداد واحد/ فیلتر برای هر لایه؟
تابع فعالیت؟ ReLU یا ...؟
استفاده از BatchNormalization یا خیر؟
استفاده از Dropout تا چه میزان؟
...

به این پارامترهای سطح معماری **هایپرپارامتر** می‌گوییم.
(برای تمایز با نام پارامتر که همان وزن‌های شبکه است و از طریق پس‌انتشار آموزش داده می‌شوند.)

پژوهشگران و مهندسان، در عمل از طریق تجربه به بینشی در مورد این انتخاب‌ها دست پیدا می‌کنند.
هیچ قانون رسمی برای این انتخاب‌ها وجود ندارد.
تصمیم‌های اولیه تقریباً همیشه زیربهینه است.
این انتخاب‌ها باید به‌صورت تکراری تغییر داده شوند و مدل مجدداً آموزش پیدا کند.
این کار بسیار وقت‌گیر است و بهتر است که به یک ماشین واگذار شود.

بهینه‌سازی هایپرپارامترها

بهینه‌سازی خودکار هایپرپارامترها

AUTOMATIC HYPERPARAMETER OPTIMIZATION

جستجوی خودکار و سیستماتیک فضای تصمیم‌های ممکن به‌شیوه‌ای اصولی:
جستجو در فضای معماری‌ها و یافتن بهترین معماری از نظر کارآیی

فرآیند: بهینه‌سازی خودکار هایپرپارامترها

۱) انتخاب مجموعه‌ای از هایپرپارامترها (به‌طور خودکار)
۲) ایجاد مدل متناظر
۳) برازش مدل با داده‌های آموزشی و اندازه‌گیری کارآیی نهایی بر روی داده‌های اعتبارسنجی
۴) انتخاب مجموعه‌ی بعدی هایپرپارامترها برای آزمایش (به‌طور خودکار)
۵) تکرار (از مرحله‌ی ۲)
۶) اندازه‌گیری کارآیی بر روی داده‌های آزمایشی در پایان

کلید این فرآیند الگوریتمی است که از تاریخچه‌ی کارآیی اعتبارسنجی استفاده می‌کند
و از مجموعه‌های هایپرپارامترهای داده‌شده، مجموعه‌ی بعدی هایپرپارامترها را برای ارزیابی انتخاب می‌کند.
برای این کار تکنیک‌های مختلف متعددی امکان‌پذیر هستند:
بهینه‌سازی بیزی، الگوریتم‌های ژنتیک، جستجوی تصادفی ساده و ...

بهینه‌سازی هایپرپارامترها

بهینه‌سازی خودکار هایپرپارامترها: چالش‌ها

AUTOMATIC HYPERPARAMETER OPTIMIZATION

در فرآیند بهینه‌سازی خودکار هایپرپارامترها،
آموزش پارامترها/وزن‌های یک مدل نسبتاً ساده است
(محاسبه‌ی تابع اتلاف بر روی دسته‌ی کوچکی از داده‌ها و سپس استفاده از پس‌انتشار برای جابجایی وزن‌ها)
اما
به‌روزرسانی هایپرپارامترها شدیداً چالش‌برانگیز است:

- **محاسبه‌ی سیگنال فیدبک می‌تواند بسیار پرهزینه باشد:**
  (برای تعیین اینکه آیا این مجموعه از هایپرپارامترها برای این وظیفه به یک مدل با کارآیی بالا منجر می‌شود یا خیر؟) نیازمند ایجاد و آموزش یک مدل جدید از صفر بر روی مجموعه‌ی داده است.

- **فضای پارامترها معمولاً از تصمیم‌های گسسته تشکیل شده است در نتیجه مشتق‌پذیر نیست.**
  برای همین نمی‌توان از کاهش گرادیان در فضای هایپرپارامترها استفاده کرد.
  در نتیجه باید به تکنیک‌های بهینه‌سازی فاقد گرادیان تکیه کنیم که طبیعتاً دارای کارآمدی بسیار کمتری نسبت به کاهش گرادیان هستند.

بهینه‌سازی هایپرپارامترها

بهینه‌سازی خودکار هایپرپارامترها: ابزارها

AUTOMATIC HYPERPARAMETER OPTIMIZATION

به‌دلیل نوپا بودن و چالش‌های حوزه‌ی بهینه‌سازی خودکار هایپرپارامترها
در حال حاضر برای بهینه‌سازی این مدل‌ها ابزارهای بسیار محدودی در دسترس داریم.

اغلب مشخص می‌شود که جستجوی تصادفی (انتخاب تصادفی هایپرپارامترها برای ارزیابی به‌طور تکراری)
بهترین راه‌حل است؛ با وجود اینکه ساده‌ترین راه‌حل است.

ابزاری که به‌طور قابل اعتمادی بهتر از جستجوی تصادفی عمل می‌کند، Hyperopt است.
(https://github.com/hyperopt/hyperopt)
یک کتابخانه‌ی پایتون برای بهینه‌سازی هایپرپارامترها که در داخل خود از
درخت‌هایی از تخمین‌گرهای پارزن
برای پیش‌بینی مجموعه‌هایی از هایپرپارامترهایی که احتمالاً خوب عمل می‌کنند، استفاده می‌کند.

ابزار دیگر، Hyperas است.
(https://github.com/maxpumperla/hyperas)
یک کتابخانه‌ی پایتون یکپارچه‌سازی Hyperopt برای استفاده با Keras.

نکته:
در هنگام استفاده از بهینه‌سازی خودکار هایپرپارامترها باید مراقب بیش‌برازش به مجموعه‌ی اعتبارسنجی باشیم.

بهینه‌سازی هایپرپارامترها

HYPERPARAMETER OPTIMIZATION

در مجموع، بهینه‌سازی هایپرپارامترها تکنیک قدرتمندی است
که برای رسیدن به مدل‌های مرز دانش در هر وظیفه
و یا پیروزی در مسابقات یادگیری ماشینی ضروری است.

بهینه‌سازی خودکار هایپرپارامترها مانند یادگیری خودکار ویژگی‌ها در یادگیری عمیق
چیزی است که امیدواریم در آینده به بلوغ برسد و رایج شود.

مدل‌های دسته‌جمعی

ترکیب مدل‌ها

MODEL ENSEMBLING

یک تکنیک قدرتمند دیگر برای دستیابی به بهترین نتایج ممکن در یک وظیفه، **ترکیب مدل‌ها**ست.

ترکیب مدل‌ها
به‌معنی تلفیق پیش‌بینی‌های یک مجموعه از مدل‌های گوناگون برای تولید پیش‌بینی‌های بهتر است.

در مسابقات یادگیری ماشینی (مانند Kaggle)
مشاهده می‌شود که برندگان از ترکیب دسته‌های بسیار بزرگی از مدل‌ها استفاده می‌کنند
که ناگزیر هر مدل تنها را بدون توجه به کیفیت و خوبی آن شکست می‌دهد.

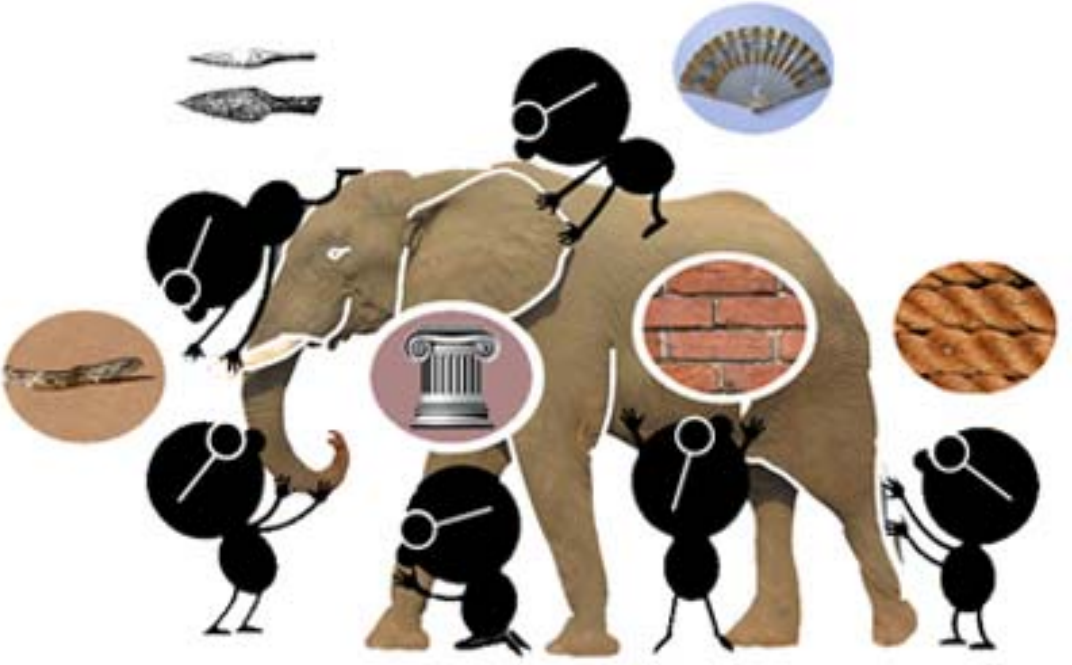
مدل‌های دسته‌جمعی

منطق

MODEL ENSEMBLING

ترکیب / Ensembling مبتنی بر این فرض است که
مدل‌های خوب متفاوتی که به‌صورت مستقل آموزش یافته‌اند،
احتمالاً به دلایل متفاوتی خوب هستند:
هر مدل برای ایجاد پیش‌بینی‌های خود به‌جنبه‌هایی از داده‌ها نگاه می‌کند
که می‌تواند با نگاه مدل دیگر اندکی متفاوت باشد.

هر مدل، بخشی از «حقیقت» و نه تمام آن را درمی‌یابد!
(مثال ماجرای مردان نابینا و فیل)
فیل، ترکیبی از بخش‌هاست:
هر نابینا به‌تنهایی حقیقت را کشف نمی‌کند،
اما وقتی با هم گفتگو می‌کنند (ترکیب مدل‌ها)، می‌توانند داستان نسبتاً دقیقی را تعریف کنند.



مدل‌های دسته‌جمعی

مثال: طبقه‌بندی (ترکیب با متوسط گیری)

از طبقه‌بندی به‌عنوان یک مثال استفاده می‌کنیم:
ساده‌ترین روش برای تلفیق پیش‌بینی‌های یک مجموعه از طبقه‌بندی‌کننده‌ها:
متوسط گیری از پیش‌بینی‌های آنها در زمان استنتاج است.

Use four different models to compute initial predictions.

```
preds_a = model_a.predict(x_val)
preds_b = model_b.predict(x_val)
preds_c = model_c.predict(x_val)
preds_d = model_d.predict(x_val)

final_preds = 0.25 * (preds_a + preds_b + preds_c + preds_d)
```

This new prediction array should be more accurate than any of the initial ones.

این روش در صورتی کار می‌کند که طبقه‌بندی‌کننده‌ها از کیفیت خوب کم و بیش یکسانی برخوردار باشند.
اگر یکی از آنها خیلی بدتر از بقیه باشد، ممکن است پیش‌بینی نهایی به‌خوبی بهترین طبقه‌بندی‌کننده‌ی گروه نباشد.

مدل‌های دسته‌جمعی

مثال: طبقه‌بندی (ترکیب با متوسط گیری وزن‌دار)

از طبقه‌بندی به‌عنوان یک مثال استفاده می‌کنیم:
روش هوشمندانه تر برای تلفیق پیش‌بینی‌های یک مجموعه از طبقه‌بندی‌کننده‌ها:
متوسط گیری وزن‌دار از پیش‌بینی‌های آنها در زمان استنتاج است.

```
preds_a = model_a.predict(x_val)
preds_b = model_b.predict(x_val)
preds_c = model_c.predict(x_val)
preds_d = model_d.predict(x_val)

final_preds = 0.5 * preds_a + 0.25 * preds_b + 0.1 * preds_c + 0.15 * preds_d
```

These weights (0.5, 0.25, 0.1, 0.15) are assumed to be learned empirically.

معمولاً به طبقه‌بندی‌کننده‌های بهتر، وزن‌های بالاتری نسبت داده می‌شود و
به طبقه‌بندی‌کننده‌های بدتر، وزن‌های پایین‌تری نسبت داده می‌شود.

برای یافتن مجموعه‌ی خوبی از وزن‌های ترکیب، می‌توان از
جستجوی تصادفی یا یک الگوریتم بهینه‌سازی ساده مانند Nelder-Mead استفاده کرد.

می‌توان متوسط گیری را روی یک تابع نمایی از پیش‌بینی‌ها نیز انجام داد.
در کل، یک میانگین وزن‌دار ساده که بر روی داده‌های اعتبارسنجی بهینه شده باشد، خط پایه‌ی بسیار قدرتمندی است.

مدل‌های دسته‌جمعی

کلید کار

کلید کار ترکیب مدل‌ها، **تنوع/گوناگونی** طبقه‌بندی کننده‌هاست.

تنوع قدرت است!

اگر تمام مردان نابینا فقط خرطوم فیل را لمس می‌کردند، موافقت می‌کردند که فیل‌ها شبیه مار هستند
و تا ابد در مورد حقیقت و ماهیت فیل در جهل باقی می‌ماندند.

تنوع چیزی است که باعث می‌شود ترکیب به‌درستی کار کند.

از منظر یادگیری ماشینی،
اگر تمام مدل‌های ما به‌شیوه‌ای یکسان دچار بایاس/انحراف باشند،
این بایاس/انحراف در ترکیب مدل‌ها نیز باقی خواهد ماند.
اگر این مدل‌ها دارای بایاس/انحراف‌های گوناگون باشند،
این انحراف‌ها یکدیگر را حنثی خواهند کرد
و ترکیب آنها قدرتمندتر و دقیق‌تر خواهد بود.

به همین دلیل، باید مدل‌هایی را ترکیب کنیم که **تا حد ممکن خوب** باشند و در عین حال تا **حد ممکن متفاوت** باشند.

(این به‌معنی استفاده از معماری‌های بسیار متفاوت و یا حتی شاخه‌های مختلف یادگیری ماشینی است.)
ناکارآمدترین کار، ترکیب یک شبکه‌ی واحد است که چندین مرتبه به‌صورت مستقل و با مقداردهی اولیه‌ی تصادفی متفاوت آموزش یافته است. در این صورت ترکیب مدل‌ها دارای تنوع اندکی خواهد بود و تنها یک پیشرفت اندک نسبت به هر مدل منفرد حاصل می‌شود.

مدل‌های دسته‌جمعی

یک تجربه

یکی از روش‌هایی که در عمل خوب کار می‌کند (برای دامنه‌ی مسائل ممکن)،
استفاده‌ی ترکیبی از **مدل‌های مبتنی بر درخت**
(مانند جنگل‌های تصادفی یا درخت‌های تقویت شده با گرادیان)
و **شبکه‌های عصبی عمیق** است.

آنچه در ترکیب مدل‌ها مهم است،
کیفیت بهترین مدل موجود نیست، بلکه تنوع مجموعه مدل‌های انتخابی است.

در دوره‌های اخیر، یکی از شیوه‌های ترکیب پایه که در عمل هم بسیار موفق بوده است،
دسته‌ی عریض و عمیق (wide and deep) از مدل‌هاست: ادغام یادگیری عمیق و یادگیری کم‌عمق.
این مدل‌ها شامل آموزش یک شبکه‌ی عصبی عمیق با یک مدل خطی بزرگ است.

آموزش همزمان خانواده‌ای از مدل‌های متنوع نیز گزینه‌ی دیگری برای دستیابی به ترکیب مدل‌هاست.

برداشت بیشترین بهره از مدل‌ها

جمع‌بندی

WRAPPING UP

- When building high-performing deep convnets, you'll need to use residual connections, batch normalization, and depthwise separable convolutions. In the future, it's likely that depthwise separable convolutions will completely replace regular convolutions, whether for 1D, 2D, or 3D applications, due to their higher representational efficiency.
- Building deep networks requires making many small hyperparameter and architecture choices, which together define how good your model will be. Rather than basing these choices on intuition or random chance, it's better to systematically search hyperparameter space to find optimal choices. At this time, the process is expensive, and the tools to do it aren't very good. But the Hyperopt and Hyperas libraries may be able to help you. When doing hyperparameter optimization, be mindful of validation-set overfitting!
- Winning machine-learning competitions or otherwise obtaining the best possible results on a task can only be done with large ensembles of models. Ensembling via a well-optimized weighted average is usually good enough. Remember: diversity is strength. It's largely pointless to ensemble very similar models; the best ensembles are sets of models that are as dissimilar as possible (while having as much predictive power as possible, naturally).

بهترین روش‌های پیشرفته برای یادگیری عمیق

خلاصه

- In this chapter, you learned the following:
  - How to build models as arbitrary graphs of layers, reuse layers (layer weight sharing), and use models as Python functions (model templating).
  - You can use Keras callbacks to monitor your models during training and take action based on model state.
  - TensorBoard allows you to visualize metrics, activation histograms, and even embedding spaces.
  - What batch normalization, depthwise separable convolution, and residual connections are.
  - Why you should use hyperparameter optimization and model ensembling.
- With these new tools, you're better equipped to use deep learning in the real world and start building highly competitive deep-learning models.

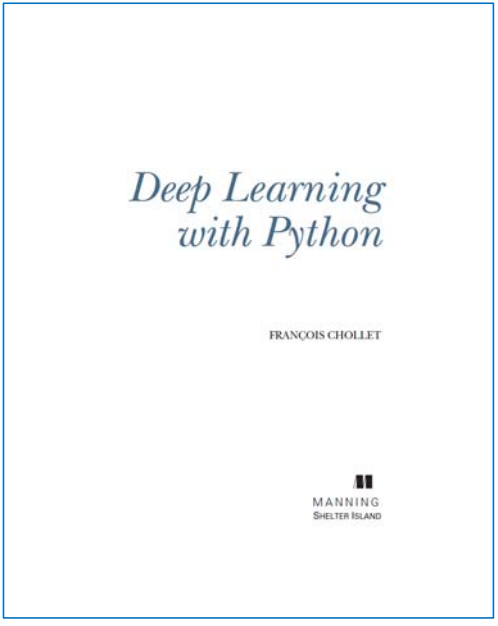
یادگیری عمیق

بهترین روش‌های پیشرفته
برای یادگیری عمیق

۴

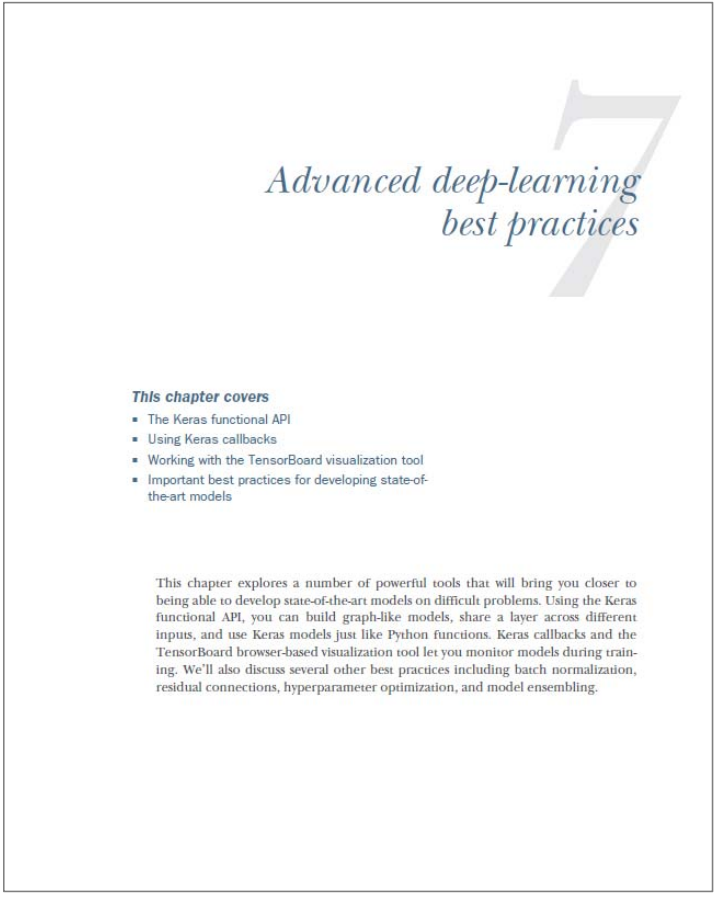
منابع

منبع اصلی



François Chollet,
Deep Learning with Python,
Manning Publications, 2018.

Chapter 7


7
Advanced deep-learning best practices

This chapter covers

- The Keras functional API
- Using Keras callbacks
- Working with the TensorBoard visualization tool
- Important best practices for developing state-of-the-art models

This chapter explores a number of powerful tools that will bring you closer to being able to develop state-of-the-art models on difficult problems. Using the Keras functional API, you can build graph-like models, share a layer across different inputs, and use Keras models just like Python functions. Keras callbacks and the TensorBoard browser-based visualization tool let you monitor models during training. We'll also discuss several other best practices including batch normalization, residual connections, hyperparameter optimization, and model ensembling.