**فصل 20. شبکه‌های عصبی**

**20. 0 مقدمه**

در مرکز شبکه‌های عصبی واحدی به نام گره یا بخش مرکزی نورون قرار دارد. گره یک یا چند ورودی دریافت می‌کند، هر ورودی را به یک پارامتر (که به وزن نیز معروف است) دیگر ضرب می‌کند، مقادیر ورودی وزن‌دار را به همراه مقدار تعیین‌شده‌ای به نام انحراف (معمولاً 1) جمع می‌کند، سپس این مقدار به تابع فعال‌ساز وارد می‌شود. سپس این خروجی به سمت نورون‌های دیگر در اعماق شبکه عصبی هستند فرستاده می‌شود (اگر وجود داشته باشند).

شبکه‌های عصبی را می‌توان به‌عنوان مجموعه‌ای از لایه‌های متصل تجسم کرد که شبکه‌ای را تشکیل می‌دهند که مقادیر ویژگی‌های یک مشاهده را در یک انتها و مقدار هدف (به عنوان مثال، کلاس مشاهده) را در انتهای دیگر به هم متصل می‌کند. شبکه‌های عصبی پیشخور - که پرسپترون چندلایه نیز نامیده می‌شوند، ساده‌ترین شبکه‌های عصبی مصنوعی هستند که در هر محیط واقعی مورد استفاده قرار می‌گیرند. نام (feedforward) از این واقعیت ناشی می شود که مقادیر ویژگی مشاهده از طریق شبکه "به جلو" تغذیه می شود، با هر لایه به طور متوالی مقادیر ویژگی را با این هدف که خروجی مشابه (یا نزدیک به) مقدار هدف باشد، تغییر می دهد.

به طور خاص، شبکه‌های عصبی پیشخور شامل سه نوع لایه از واحدها هستند. در ابتدای این شبکه عصبی، لایه ورودی قرار دارد که هر واحد شامل مقدار مشاهده برای یک ویژگی است. به عنوان مثال، اگر مشاهده دارای 100 ویژگی باشد، لایه ورودی 100 گره دارد. در انتهای شبکه عصبی لایه خروجی قرار دارد که خروجی لایه‌های پنهان را به مقادیری مفید برای وظیفه مورد نظر تبدیل می‌کند. به عنوان مثال، اگر هدف ما دسته‌بندی دودویی باشد، می‌توانیم از لایه خروجی با یک واحد استفاده کنیم که از تابع سیگموئید برای مقیاس دادن خروجی خود به مقداری بین 0 و 1 استفاده کند، که احتمال کلاس پیش‌بینی‌شده را نمایان می‌سازد. بین لایه ورودی و خروجی، لایه‌های مخفی به نام hidden layers (که به هیچ وجه مخفی نیستند) قرار دارند. این لایه‌های مخفی به ترتیب مقادیر ویژگی را از لایه ورودی به چیزی تغییر می‌دهند که پس از پردازش توسط لایه خروجی، به شکل کلاس هدف شبیه شود.

شبکه‌های عصبی‌ای که دارای لایه‌های مخفی زیادی (مانند 10، 100، 1000) هستند به عنوان شبکه‌های "عمیق" محسوب می‌شوند و کاربرد آنها به عنوان یادگیری عمیق شناخته می‌شود.

شبکه‌های عصبی به طور معمول با تمام پارامترها با مقادیر تصادفی کوچک از توزیع گوسی یا توزیع یکنواخت نرمال مقداردهی اولیه می‌شوند. پس از اینکه مشاهده‌ای (یا بیشتر، معمولاً تعداد تعیین‌شده‌ای از مشاهدات به نام دسته) از طریق شبکه انتقال داده می‌شود، مقدار خروجی با مقدار واقعیِ مشاهده شده با استفاده از تابع خطا مقایسه می‌شود. این به عنوان پیش‌رَوی انجام می‌شود. سپس یک الگوریتم به طور پَسرو از طریق شبکه حرکت می‌کند و می‌شناسد که هر پارامتر چقدر به اشتباه بین مقدار پیش‌بینی شده و مقدار واقعی کمک کرده است که به عمل فرآیند برگشت مشهور است. در هر پارامتر، الگوریتم بهینه‌سازی تعیین می‌کند که هر وزن باید چقدر تغییر یابد تا خروجی بهبود یابد. شبکه‌های عصبی با تکرار این فرآیند از پیش‌رَویی و برگشت برای هر مشاهده در داده‌های آموزش چندین بار (هر بار که همه مشاهدات از طریق شبکه ارسال شده باشند، به عنوان یک دوره شناخته می‌شود و آموزش معمولاً از چندین دوره تشکیل شده است) مقادیر پارامترها را به‌روز می‌کنند. در این فصل، ما از کتابخانه محبوب پایتون به نام Keras برای ساخت، آموزش و ارزیابی انواع مختلف شبکه‌های عصبی استفاده خواهیم کرد. Keras یک کتابخانه سطح بالا است و از کتابخانه‌های دیگری مانند TensorFlow و Theano به عنوان موتور خود استفاده می‌کند. برای ما مزیت Keras این است که می‌توانیم بر روی طراحی و آموزش شبکه تمرکز کنیم و جزئیات عملیات تانسورها را به کتابخانه‌های دیگر واگذار کنیم. شبکه‌های عصبی ساخته‌شده با استفاده از کد Keras می‌توانند با استفاده از همه‌ی پردازنده‌ها (یعنی حتی در لپ‌تاپ شما) و همچنین با پردازنده‌های گرافیکی (یعنی در یک کامپیوتر یادگیری عمیق ویژه) آموزش داده شوند. در دنیای واقعی با داده‌های واقعی، به شدت توصیه می‌شود که شبکه‌های عصبی با استفاده از پردازنده‌های گرافیکی آموزش داده شوند؛ با این حال، به منظور یادگیری، همه شبکه‌های عصبی در این کتاب به اندازه کافی کوچک و ساده هستند تا در چند دقیقه در لپ‌تاپ شما آموزش داده شوند. فقط به خاطر داشته باشید که زمانی که شبکه‌های بزرگتر و داده‌های آموزش بیشتری داشته باشیم، آموزش با استفاده از پردازنده‌های مرکزی به طرز قابل توجهی کندتر از آموزش با استفاده از پردازنده‌های گرافیکی است.

**20. 1 پیش پردازش داده برای شبکه‌های عصبی**

**مسئله**

شما می‌خواهید داده‌ها را برای استفاده در یک شبکه عصبی، پردازش کنید.

**راه حل**

هر ویژگی را با استفاده از استاندارد Scaler در scikit-learn استاندارد کنید:

# Load libraries

from sklearn import preprocessing

import numpy as np

# Create feature

features = np.array([[-100.1, 3240.1],

[-200.2, -234.1],

[5000.5, 150.1],

[6000.6, -125.1],

[9000.9, -673.1]])

# Create scaler

scaler = preprocessing.StandardScaler()

# Transform the feature

features\_standardized = scaler.fit\_transform(features)

# Show feature

features\_standardized

array([[-1.12541308, 1.96429418],

[-1.15329466, -0.50068741],

[ 0.29529406, -0.22809346],

[ 0.57385917, -0.42335076],

[ 1.40955451, -0.81216255]])

**بحث**

اگرچه این دستورالعمل بسیار شبیه به دستورالعمل 4.3 است، ارزش تکرار آن به دلیل اهمیت آن برای شبکه‌های عصبی حائز اهمیت است. به طور معمول، پارامترهای یک شبکه عصبی به عنوان اعداد تصادفی کوچک، مقداردهی اولیه می‌شوند. شبکه‌های عصبی معمولاً زمانی که مقادیر ویژگی‌ها به طور قابل توجهی بزرگتر از مقادیر پارامترها هستند به شکل ناپسندی عمل می‌کنند. علاوه بر این، از آنجا که مقادیر ویژگی‌های یک مشاهده به عنوان آنها از طریق واحدهای تکی ترکیب می‌شوند، اهمیت دارد که تمام ویژگی‌ها از همان مقیاس استفاده کنند.

با توجه به این دلایل، بهترین شیوه (اگرچه همیشه ضروری نیست؛ به عنوان مثال، زمانی که تمام ویژگی‌ها دارای ویژگی‌های دودویی هستند) توصیه می‌شود که هر ویژگی را به نحوی استاندارد کنیم که مقادیر ویژگی‌ها میانگین 0 و انحراف معیار 1 داشته باشند. این کار به راحتی با استفاده از ابزار StandardScaler در scikit-learn قابل انجام است.

تأثیر استانداردسازی را می‌توانید با بررسی میانگین و انحراف معیار ویژگی‌های اولیه ما مشاهده کنید:

# Print mean and standard deviation

**print**("Mean:", round(features\_standardized[:,0].mean()))

**print**('"Standard deviation:", features\_standardized[:,0].std())

Mean: 0.0

Standard deviation: 1.0

**20. 2 طراحی شبکه عصبی**

**مسئله**

شما می­خواهید یک شبکه عصبی طراحی کنید.

**راه حل**

از مدل Sequential در Keras استفاده کنید:

# Load libraries

from keras import models

from keras import layers

# Start neural network

network = models.Sequential()

# Add fully connected layer with a ReLU activation function

network.add(layers.Dense(units=16, activation="relu", input\_shape=(10,)))

# Add fully connected layer with a ReLU activation function

network.add(layers.Dense(units=16, activation="relu"))

# Add fully connected layer with a sigmoid activation function

network.add(layers.Dense(units=1, activation="sigmoid"))

# Compile neural network

network.compile(loss="binary\_crossentropy", # Cross-entropy

optimizer="rmsprop", # Root Mean Square Propagation

metrics=["accuracy"]) # Accuracy performance metric

Using TensorFlow backend.

شبکه‌های عصبی از لایه‌هایی متشکل از یونیت‌ها یا همان گره‌ها تشکیل شده‌اند. با این حال، تنوع شگفت‌انگیزی در انواع لایه‌ها و چگونگی ترکیب آن‌ها برای ایجاد معماری شبکه وجود دارد. در حال حاضر، با وجود الگوهای معماری معمولی (که در این فصل به آن‌ها خواهیم پرداخت)، حقیقت این است که انتخاب معماری مناسب در اغلب موارد یک هنر است و موضوع تحقیقات فراوانی است. برای ساخت یک شبکه عصبی پیش‌خور در Keras، باید در مورد معماری شبکه و فرآیند آموزش، تعدادی انتخاب انجام دهیم. به خاطر داشته باشید که هر واحد در لایه‌های مخفی:

تعدادی ورودی دریافت می‌کند.

هر ورودی را با مقدار یک پارامتر وزن‌دهی می‌کند.

تمام ورودی‌های وزن‌دار را به همراه تعدادی تغییر بایاس (معمولاً 1) جمع می‌کند.

بیشتر اوقات تابعی (که تابع فعال‌ساز نامیده می‌شود) را روی ورودی اعمال می‌کند.

خروجی را به یونیت‌ها یا همان گره‌ها در لایه‌ی بعدی ارسال می‌کند. ابتدا، برای هر لایه در لایه‌های مخفی و خروجی، باید تعداد واحدهایی که در لایه قرار می‌دهیم و تابع فعال‌ساز را تعیین کنیم. به طور کلی، هر چه تعداد واحدها در یک لایه بیشتر باشد، شبکه ما قادر به یادگیری الگوهای پیچیده‌تر خواهد بود. با این حال، تعداد بیشتر یونیت‌ها ممکن است باعث بیش‌برازش شبکه به داده‌های آموزشی شده و به نحوی مخرب، بر عملکرد داده‌های آزمایشی تأثیر بگذارد. برای لایه‌های مخفی، تابع فعال‌ساز محبوب واحد خطی تصحیح شده (ReLU) است:

که z مجموع ورودی‌های وزن‌دار و انحراف است. همان‌طور که می‌بینیم، اگر z از 0 بزرگتر باشد، تابع فعال‌ساز مقدار z را برمی‌گرداند؛ در غیر اینصورت، تابع مقدار 0 را باز می‌گرداند. این تابع فعال‌ساز ساده دارای تعدادی خاصیت مطلوب است (که بحث درباره‌ی آن خارج از دامنه‌ی این کتاب است) و این امر باعث شده ‌است که انتخاب محبوبی در شبکه‌های عصبی باشد. با این حال، باید آگاه باشیم که تعداد زیادی تابع فعال‌ساز وجود دارد. ثانیاً، باید تعداد لایه‌های مخفی را در شبکه تعیین کنیم. لایه‌های بیشتر، به شبکه امکان یادگیری روابط پیچیده‌تر را می‌دهد، اما با هزینه‌ی محاسباتی بیشتری همراه است. سوماً، ما باید ساختار تابع فعال‌ساز (در صورت وجود) لایه خروجی را تعیین کنیم. ماهیت تابع خروجی اغلب توسط هدف شبکه تعیین می‌شود. در ادامه، برخی از الگوهای رایج لایه خروجی آمده است:

تشخیص دوتایی (Binary classification)

یک واحد با تابع فعال‌ساز سیگموئید.

تشخیص چنددرجه‌ای (Multiclass classification)

k واحد (که k تعداد کلاس‌های هدف است) و تابع فعال‌ساز سافت‌مکس.

رگرسیون (Regression)

یک واحد بدون تابع فعال‌ساز.

چهارم، ما باید یک تابع خطا (تابعی که میزان تطابق مقدار پیش‌بینی شده با مقدار واقعی را اندازه‌گیری می‌کند) تعریف کنیم؛ این تابع معمولاً با توجه به نوع مسئله تعیین می‌شود:

تشخیص دوتایی

آنتروپی متقابل دوتایی. (Binary cross-entropy)

تشخیص چنددرجه‌ای

آنتروپی دسته‌ای. (Categorical cross-entropy)

رگرسیون

خطای میانگین مربعات. (Mean square error)

پنجم، ما باید یک بهینه‌ساز تعریف کنیم، که به طور ساده می‌توان آن را به عنوان استراتژی‌مان برای پیمایش تابع خطا برای یافتن مقادیر پارامترهایی که کمترین خطا را ایجاد می‌کنند، تصور کرد. انتخاب‌های رایج برای بهینه‌سازها شامل نزول تصادفی گرادیانی، نزول تصادفی گرادیانی با آسیب همراهی، پخش مربعات میانگین و برآورد لحظه‌ای تطبیقی هستند (اطلاعات بیشتر در همچنین ببینید).

ششم، ما می‌توانیم یک یا چند معیار برای ارزیابی عملکرد انتخاب کنیم، مانند دقت.

Keras دو روش برای ایجاد شبکه‌های عصبی ارائه می‌دهد. مدل ترتیبی Keras با انتشار لایه‌ها، شبکه‌های عصبی ایجاد می‌کند. یک روش جایگزین برای ایجاد شبکه‌های عصبی به نام API کارایی وجود دارد، اما این روش بیشتر برای پژوهشگران مناسب است تا عمل‌کنندگان.

در این راه‌حل، ما یک شبکه عصبی دو لایه‌ای ایجاد کردیم (وقتی لایه‌ها را می‌شماریم، لایه ورودی را به دلیل عدم داشتن پارامترهایی برای یادگیری محاسبه نمی‌کنیم) با استفاده از مدل ترتیبی Keras. هر لایه چگال است (همچنین به عنوان اتصال کامل شناخته می‌شود)، به این معنی که تمام واحدهای لایه قبلی به تمام نورون‌های لایه بعدی متصل هستند.

در لایه مخفی اول، ما units=16 را تنظیم کردیم، به این معنی که این لایه، شامل 16 واحد با توابع فعال‌ساز ReLU است: activation='relu'. در Keras، لایه مخفی اول هر شبکه باید شامل یک پارامتر input\_shape باشد، که شکل داده‌های ویژگی را مشخص می‌کند. به عنوان مثال، (10,) به لایه اول می‌گوید که انتظار دارد هر مشاهده دارای ۱۰ مقدار ویژگی باشد. لایه دوم ما همانند لایه اول است، بدون نیاز به پارامتر input\_shape. این شبکه برای تشخیص دوتایی طراحی شده است، بنابراین لایه خروجی فقط یک واحد با تابع فعال‌ساز سیگموئید دارد، که خروجی را بین ۰ و ۱ محدود می‌کند (نمایانگر احتمال مشاهده در کلاس ۱).

در نهایت، قبل از اینکه بتوانیم مدل خود را آموزش دهیم، باید به Keras بگوییم چگونه می‌خواهیم شبکه ما یاد بگیرد. این کار را با استفاده از متد compile انجام می‌دهیم، با الگوریتم بهینه‌سازی ما (RMSProp)، تابع خطا (binary\_crossentropy) و یک یا چند معیار عملکرد.

**منابع بیشتر**

* [Losses, Keras](https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)
* [توابع خطا برای دسته‌بندی، ویکی‌پدیا](https://en.wikipedia.org/wiki/Loss_functions_for_classification)
* [در مورد توابع خطا برای شبکه‌های عصبی عمیق در دسته‌بندی، کاتارژینا یانوخا، وویچک ماریان چارنتسکی](https://arxiv.org/abs/1702.05659)

**20. 3 آموزش یک تشخیص‌دهنده‌ی دوتایی**

**مسئله**

شما می‌خواهید یک شبکه عصبی دسته‌بند(تشخیص دهنده‌ی) دوتایی آموزش دهید.

**راه‌حل**

از Keras برای ساخت یک شبکه عصبی فیدفوروارد(Feed Forward) استفاده کرده و آن را با استفاده از متد fit آموزش دهید.

# Load libraries

**import** numpy **as** np

**from** keras.datasets **import** imdb

**from** keras.preprocessing.text **import** Tokenizer

**from** keras **import** models

**from** keras **import** layers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 1000

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to one-hot encoded feature matrix*

tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network* network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu", input\_shape=(

number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square Propagation* metrics=["accuracy"]) *# Accuracy performance metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector*

epochs=3, *# Number of epochs*

verbose=1, *# Print description after each epoch*

batch\_size=100, *# Number of*  *observations per batch*

validation\_data=(features\_test, target\_test)) *# Test data*

Using TensorFlow backend.

Train on 25000 samples, validate on 25000 samples

Epoch 1/3

25000/25000 [==============================]

-1s – loss: 0.4215 – acc: 0.8105 – val\_loss: 0.3386 – val\_acc: 0.8555

Epoch 2/3

25000/25000 [==============================]

-1s - loss: 0.3242 - acc: 0.8645 - val\_loss: 0.3258 - val\_acc: 0.8633

Epoch 3/3

25000/25000 [==============================]

- 1s - loss: 0.3121 - acc: 0.8699 - val\_loss: 0.3265 - val\_acc: 0.8599

**بحث**

در دستور 20.2، نحوه ساخت یک شبکه عصبی با استفاده از مدل ترتیبی Keras را مورد بحث قرار دادیم. در این دستورالعمل، ما آن شبکه عصبی را با استفاده از داده­های واقعی آموزش می­دهیم. به طور خاص، ما از 50000 نقد فیلم (25000 به عنوان داده آموزشی، 25000 مورد برای آزمایش) استفاده می­کنیم که به عنوان مثبت یا منفی دسته‌بندی می‌شوند. ما متن بررسی­ها را به 5000 ویژگی باینری تبدیل می­کنیم که نشان دهنده وجود یکی از 1000 کلمه رایج است. به بیان ساده­تر، شبکه‌های عصبی ما از 25000 مشاهده استفاده می­کنند که هر کدام دارای 1000 ویژگی است تا مثبت یا منفی بودن نقد فیلم را پیش بینی کنند.

شبکه عصبی مورد استفاده‌ی ما همان شبکه­ای است که در دستور العمل 20.2 وجود دارد (برای توضیح دقیق به آنجا مراجعه کنید). تنها نکته‌ی اضافه این است که در آن دستور، ما فقط شبکه عصبی را ایجاد کردیم، اما آن را آموزش ندادیم.

در Keras، شبکه عصبی خود را با استفاده از تابع fit آموزش می­دهیم. شش پارامتر قابل توجه برای تعریف وجود دارد. دو پارامتر اول ویژگی­ها و بردار هدف، داده‌های آموزشی است. ما می­توانیم شکل ماتریس ویژگی را با استفاده از تابع shape مشاهده کنیم:

# View shape of feature matrix

features\_train.shape

(25000, 1000)

پارامتر epochs تعیین می­کند که از آموزش مدل توسط داده‌ها چند دوره باشد. Verbose تعیین می‌کند که چه مقدار اطلاعات در طول فرآیند آموزش خروجی داده می‌شود، با 0 بدون خروجی، 1 خروجی نوار پیشرفت، و 2 یک خط ورود به سیستم در هر دوره. batch\_size تعداد مشاهدات را برای انتشار در شبکه قبل از به روز رسانی پارامترها تنظیم می­کند.

در نهایت، مجموعه آزمایشی از داده­ها را برای ارزیابی مدل ارائه کردیم. این ویژگی­های تست و بردار هدف آزمایشی می­توانند آرگومان‌های validation\_data باشند که از آنها برای ارزیابی استفاده ­کنند. از طرف دیگر، می‌توانستیم از validation\_split برای تعریف اینکه چه کسری از داده‌های آموزشی را می‌خواهیم برای ارزیابی نگهداری کنیم، استفاده کنیم.

در scikit-learn، تابع fit یک مدل آموزش‌دیده را به ما می‌دهد، اما در Keras، تابع fit یک شیء History حاوی مقادیر تلفات و معیارهای عملکرد در هر دوره را برمی‌گرداند.

**4.20 آموزش یک دسته‌بندی کننده‌ی چند کلاسه**

**مسئله**

شما می­خواهید یک شبکه عصبی دسته‌بندی کننده‌ی چند کلاسه را آموزش دهید.

**راه حل**

از Keras برای ساخت یک شبکه عصبی پیشخور با یک لایه خروجی با توابع فعال ساز softmax استفاده کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** reuters

**from** **keras.utils.np\_utils** **import** to\_categorical

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 5000

*# Load feature and target data*

data = reuters.load\_data(num\_words=number\_of\_features)

data\_train, target\_vector\_train), (data\_test, target\_vector\_test) = data

*# Convert feature data to a one-hot encoded feature matrix*

tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# One-hot encode target vector to create a target matrix*

target\_train = to\_categorical(target\_vector\_train)

target\_test = to\_categorical(target\_vector\_test)

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=100,

activation="relu",

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=100, activation="relu"))

*# Add fully connected layer with a softmax activation function*

network.add(layers.Dense(units=46, activation="softmax"))

*# Compile neural network*

network.compile(loss="categorical\_crossentropy", *# Crossentropy*

optimizer="rmsprop", *# Root Mean Square Propagation* metrics=["accuracy"]) *# Accuracy performance metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target* epochs=3, *# Three epochs* verbose=0, *# No output* batch\_size=100, *# Number of*

*observations per batch*

validation\_data=(features\_test, target\_test)) *# Test data*

Using TensorFlow backend.

**بحث**  
در این راه حل، ما یک شبکه عصبی مشابه با دسته‌بندی کننده‌ی باینری از آخرین دستور، اما با تغییرات قابل توجه ایجاد کردیم. اول، داده‌های ما 11228 خبر رویترز است. هر خبر در 46 موضوع دسته بندی شده است. ما داده‌های ویژگی خود را با تبدیل به خط‌های خبری به 5000 ویژگی باینری (که نشان دهنده‌ی وجود یک کلمه خاص در خط‌های خبری است) آماده کردیم. ما داده‌های هدف را با یک کدگذاری داغ آماده کردیم تا یک ماتریس هدف به دست آوریم که نشان می‌دهد یک مشاهده متعلق به کدام یک از 46 کلاس است:

*# View target matrix*

target\_train

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.]])

دوم، ما تعداد واحدها را در هر یک از لایه‌های پنهان افزایش دادیم تا به شبکه عصبی کمک کنیم تا روابط پیچیده‌تر بین 46 کلاس را نشان دهد.

سوم، از آنجایی که این یک مشکل دسته‌بندی چند کلاسه است، ما از یک لایه خروجی با 46 واحد (یک واحد در هر کلاس) استفاده کردیم که حاوی یک تابع فعال‌ساز softmax است. تابع فعال‌ساز softmax آرایه‌ای از 46 مقدار را برمی‌گرداند که جمع آن 1 است. این 46 مقدار احتمال یک مشاهده را برای عضویت در هر یک از 46 کلاس نشان می‌دهند.

چهارم، ما از یک تابع ضرر مناسب برای دسته‌بندی چند کلاسه، تابع از دست دادن متقابل آنتروپی، categorical\_crossentropy استفاده کردیم.

**20. 5 آموزش یک واپسگرا (Regressor)**

**مسئله**  
شما می­خواهید یک شبکه عصبی را برای رگرسیون آموزش دهید.

**راه حل**

از Keras برای ساخت یک شبکه عصبی پیش‌خور با یک واحد خروجی و بدون تابع فعال‌ساز استفاده کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **sklearn.datasets** **import** make\_regression

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** preprocessing

*# Set random seed*

np.random.seed(0)

*# Generate features matrix and target vector*

features, target = make\_regression (n\_samples = 10000,

n\_features = 3, n\_informative = 3, n\_targets = 1, noise = 0.0, random\_state = 0)

*# Divide our data into training and test sets*

features\_train, features\_test, target\_train, target\_test = train\_test\_split( features, target, test\_size=0.33, random\_state=0)

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=32,

activation="relu",

input\_shape=(features\_train.shape[1],)))

*# Add fully connected layer with a ReLU activation function*

network.add(layers.Dense(units=32, activation="relu"))

*# Add fully connected layer with no activation function* network.add(layers.Dense(units=1))

*# Compile neural network*

network.compile (loss="mse", *# Mean squared error*

optimizer="RMSprop", *# Optimization algorithm* metrics=["mse"]) *# Mean squared error*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector* epochs=10, *# Number of epochs* verbose=0, *# No output* batch\_size=100, *# Number of observations per batch* validation\_data=(features\_test, target\_test)) *# Test data*

Using TensorFlow backend.

**بحث**  
ایجاد یک شبکه عصبی برای پیش بینی مقادیر پیوسته به جای احتمالات کلاس کاملاً امکان پذیر است. در مورد طبقه‌بندی‌کننده‌ی باینری (دستور العمل 20.3) ما از یک لایه خروجی با یک واحد و یک تابع فعال‌ساز سیگموئید استفاده کردیم تا احتمال اینکه مشاهده‌ای کلاس 1 باشد را ایجاد کنیم. نکته مهم این است که تابع فعال‌ساز سیگموئید مقدار خروجی را بین 0 و 1 محدود می‌کند. اگر آن محدودیت را با نداشتن تابع فعال سازی حذف کنیم، اجازه می­دهیم خروجی یک مقدار پیوسته باشد.

علاوه بر این، از آنجایی که ما در حال آموزش رگرسیون هستیم، باید از یک تابع ضرر و معیار ارزیابی مناسب استفاده کنیم. در مورد ما میانگین مربعات خطا:

که در آن n تعداد مشاهدات است. مقدار واقعی هدفی است که می‌خواهیم پیش‌بینی کنیم، y، برای مشاهده iام و مقدار پیش بینی شده مدل برای است.

در نهایت، چون ما از داده­های شبیه‌سازی شده با استفاده از make\_regression در scikit-learn استفاده می­کنیم، مجبور نبودیم ویژگی­ها را استاندارد کنیم. با این حال، باید توجه داشت که تقریباً در تمام موارد دنیای واقعی استانداردسازی ضروری است.

**20. 6 پیش بینی**

**مسئله**

شما می‌خواهید از یک شبکه عصبی برای پیش بینی استفاده کنید.

**راه حل**

از Keras برای ساخت یک شبکه عصبی پیش‌خور استفاده کنید، سپس با استفاده از تابع predict پیش‌بینی را انجام دهید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets**

**import** imdb **from** **keras.preprocessing.text**

**import** Tokenizer **from** **keras**

**import** models **from** **keras**

**import** layers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 10000

*# Load data and target vector from IMDB movie data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert IMDB data to a one-hot encoded feature matrix* tokenizer = Tokenizer(num\_words=number\_of\_features) features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network* network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

activation="relu",

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function* network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square Propagation*

metrics=["accuracy"]) *# Accuracy performance metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector* epochs=3, *# Number of epochs* verbose=0, *# No output*

batch\_size=100, *# Number of observations per batch*

validation\_data=(features\_test, target\_test)) *# Test data*

*# Predict classes of test set*

predicted\_target = network.predict(features\_test)

Using TensorFlow backend.

بحث  
پیش بینی در Keras آسان است. هنگامی که شبکه عصبی خود را آموزش دادیم، می‌توانیم از روش پیش‌بینی استفاده کنیم، که به عنوان آرگومان، مجموعه‌ای از ویژگی‌ها را می‌گیرد و خروجی پیش‌بینی‌شده را برای هر مشاهده برمی‌گرداند. در راه حل ما شبکه عصبی ما برای دسته‌بندی باینری تنظیم شده است، بنابراین خروجی پیش بینی شده احتمال کلاس 1 بودن است. مشاهدات با مقادیر پیش بینی شده‌ی بسیار نزدیک به 1، به احتمال زیاد کلاس 1 هستند، در حالی که مشاهدات با مقادیر پیش بینی شده‌ی بسیار نزدیک به 0، به احتمال زیاد کلاس 0 است. برای مثال، این احتمال پیش بینی شده است که اولین مشاهده در ماتریس ویژگی آزمون ما کلاس 1 است:

*# View the probability the first observation is class 1*

predicted\_target[0]

array([ 0.83937484], dtype=float32)

**20. 7 تجسم تاریخچه‌ی آموزش**

**مسئله**

شما می­خواهید "نقطه شیرین" را در میزان اشتباه (loss score) و یا دقت (accuracy score) شبکه عصبی پیدا کنید.

**راه حل**

از Matplotlib برای به تصویر کشیدن از دست دادن مجموعه تست و آموزشی در هر دوره استفاده کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

**import** **matplotlib.pyplot** **as** **plt**

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 10000

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) =

imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to a one-hot encoded feature matrix*

tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary") features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

activation="relu",

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target*

epochs=15, *# Number of epochs*

verbose=0, *# No output*

batch\_size=1000, *# Number of*

*observations per batch*

validation\_data=(features\_test,

target\_test)) *# Test data*

*# Get training and test loss histories*

training\_loss = history.history["loss"]

test\_loss = history.history["val\_loss"]

*# Create count of the number of epochs*

epoch\_count = range(1, len(training\_loss) + 1)

*# Visualize loss history*

plt.plot(epoch\_count, training\_loss, "r--")

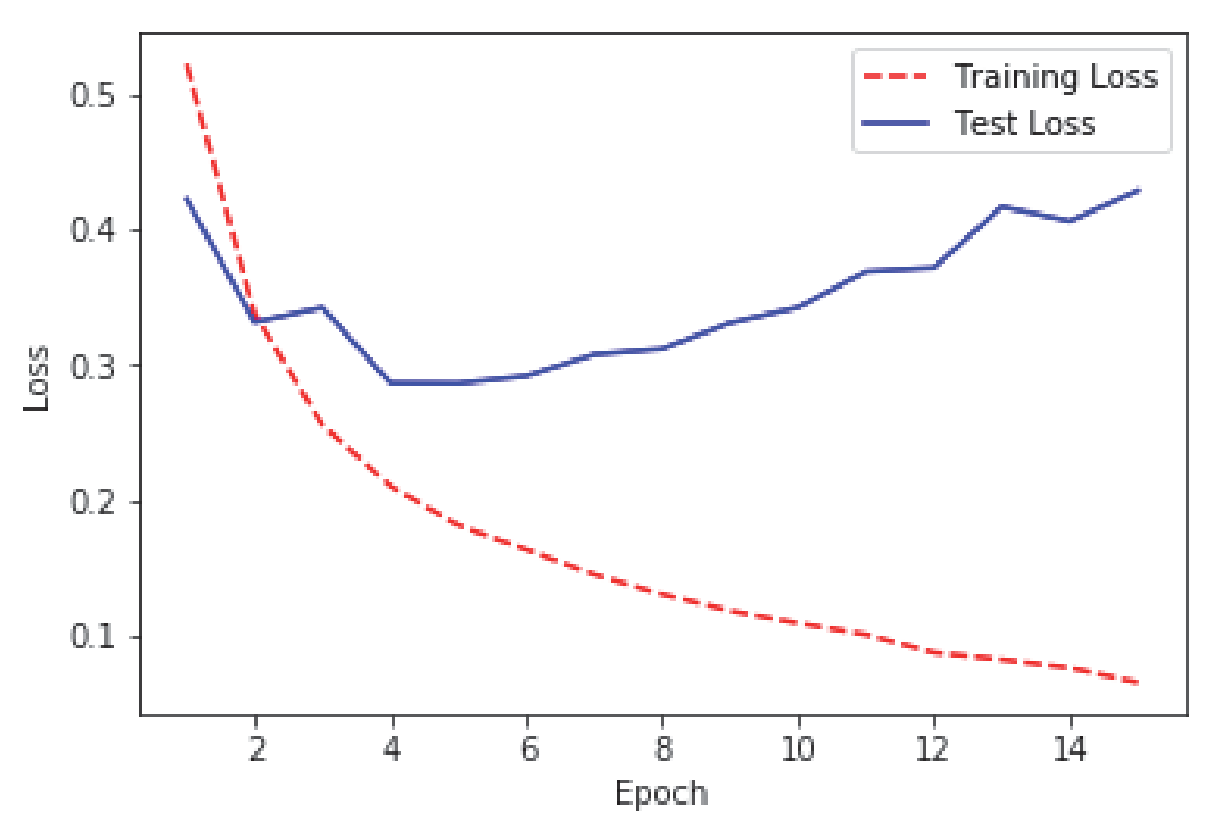
plt.plot(epoch\_count, test\_loss, "b-")

plt.legend(["Training Loss", "Test Loss"])

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.show();

Using TensorFlow backend.

از طرف دیگر، می‌توانیم از همین رویکرد برای تجسم دقت آموزش و تست در هر دوره استفاده کنیم:

*# Get training and test accuracy histories*

training\_accuracy = history.history["acc"]

test\_accuracy = history.history["val\_acc"]

plt.plot(epoch\_count, training\_accuracy, "r--")

plt.plot(epoch\_count, test\_accuracy, "b-")

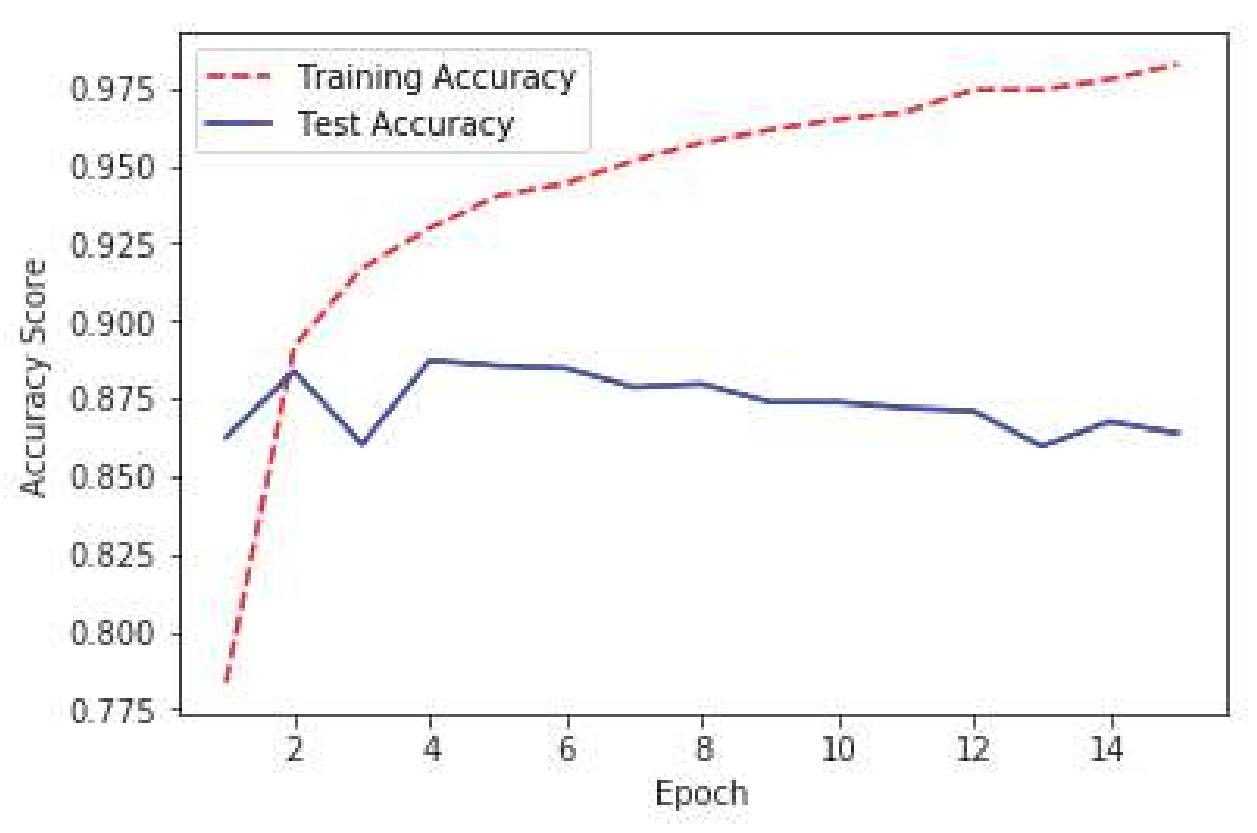
*# Visualize accuracy history*

plt.legend(["Training Accuracy", "Test Accuracy"])

plt.xlabel("Epoch")

plt.ylabel("Accuracy Score")

plt.show();



**بحث**

وقتی شبکه عصبی ما جدید باشد، عملکرد ضعیفی خواهد داشت. همانطور که شبکه عصبی از داده­های آموزشی یاد می­گیرد، خطای مدل در هر دو مجموعه آموزشی و تست افزایش می­یابد. با این حال، در یک نقطه خاص، شبکه عصبی شروع به "به خاطر سپردن" داده‌های آموزشی می­کند و بیش از حد برازش می­کند. هنگامی که این اتفاق می­افتد، خطای آموزش کاهش می­یابد در حالی که خطای تست شروع به افزایش می­کند. بنابراین، در بسیاری از موارد یک "نقطه شیرین" وجود دارد که در آن خطای تست (خطایی است که ما عمدتاً به آن اهمیت می‌دهیم) در پایین‌ترین نقطه خود قرار دارد. این تأثیر را می­توان به وضوح در راه حلی مشاهده کرد که در آن آموزش و شکست آزمون را در هر دوره به نمودار برده و تصویر می­کنیم. توجه داشته باشید که خطای آزمون در حوالی دوره پنجم کمتر است، پس از آن میزان تلفات تمرینی همچنان افزایش می‌یابد در حالی که ضرر تست شروع به افزایش می‌کند. در این مرحله به بعد، مدل بیش از حد مناسب است.

**20. 8 کاهش بیش از حد تناسب با تنظیم وزن**

**مسئله**

شما می­خواهید اوورفیت (overfit) را کاهش دهید.

**راه حل**

سعی کنید پارامترهایی از شبکه را که تنظیم وزن (weight regularization) نیز نامیده می‌شوند، کنترل کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **keras** **import** regularizers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 100

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to a one-hot encoded feature matrix*

tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

activation="relu",

kernel\_regularizer=regularizers.l2(0.01),

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

kernel\_regularizer=regularizers.l2(0.01),

activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector* epochs=3, *# Number of epochs* verbose=0, *# No output* batch\_size=100, *# Number of*

*observations per batch*

validation\_data=(features\_test,

target\_test)) *# Test data*

Using TensorFlow backend.

بحث  
یک استراتژی برای مبارزه با شبکه­های عصبی اوورفیت‌شده، جریمه کردن پارامترها (به عنوان مثال، وزن) شبکه عصبی به گونه­ای است که آنها را به مقادیر کوچک هدایت می­کند - ایجاد یک مدل ساده­تر که کمتر مستعد اضافه‌ی برازش است. این روش تنظیم وزن یا کاهش وزن نامیده می­شود. به طور خاص، در تنظیم وزن، یک جریمه به تابع کاهش اضافه می­شود، مانند هنجار L2.

در Keras، می‌توانیم با استفاده از

kernel\_regularizer=regularizers.l2(0.01) یک تنظیم وزن اضافه کنیم. پارامترهای لایه در این مثال، با مقدار 0.01 تعیین می­کند که چقدر مقادیر پارامترهای بالاتر را جریمه می­کنیم.

**20. 9 کاهش بیش از حد برازش با توقف زودهنگام**

**مسئله**

شما می­خواهید اوورفیت را کاهش دهید.

**راه حل**

سعی کنید تمرین را زمانی متوقف کنید که کاهش ضرر تست متوقف شود، این استراتژی به نام توقف زودهنگام شناخته می‌شود:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **keras.callbacks** **import** EarlyStopping, ModelCheckpoint

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 1000

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to a one-hot encoded feature matrix* tokenizer = Tokenizer(num\_words=number\_of\_features) features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary") features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

activation="relu",

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Set callback functions to early stop training and save the best model so far*

callbacks = [EarlyStopping(monitor="val\_loss", patience=2),

ModelCheckpoint(filepath="best\_model.h5",

monitor="val\_loss",

save\_best\_only=True)]

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector* epochs=20, *# Number of epochs* callbacks=callbacks, *# Early stopping* verbose=0, *# Print description after*

*each epoch*

batch\_size=100, *# Number of*

*observations per batch*

validation\_data=(features\_test, target\_test)) *# Test data*

Using TensorFlow backend.

**بحث**

همانطور که در دستور العمل 20.7 بحث کردیم، معمولاً در دوره­های آموزشی اول، خطاهای آموزش و آزمون کاهش می­یابد، اما در یک نقطه شبکه شروع به "به خاطر سپردن" داده­های آموزشی می­کند و باعث می­شود که خطای آموزشی کاهش یابد حتی در حالی که خطای تست وجود دارد. شروع به افزایش می­کند به دلیل این پدیده، یکی از متداول‌ترین و بسیار مؤثرترین روش­ها برای مقابله با اضافه تناسب، نظارت بر روند تمرین و توقف تمرین، درست در زمانی است که خطای تست شروع به افزایش می­کند. این استراتژی توقف زودهنگام نامیده می­شود.

در Keras، ما می­توانیم توقف اولیه را به عنوان یک تابع پاسخ به تماس(Callback) پیاده‌سازی کنیم. Callback‌ها توابعی هستند که می­توانند در مراحل خاصی از فرآیند آموزشی مانند پایان هر دوره، اعمال شوند. به طور خاص، ما در راه حل خود گنجانده­ایم

EarlyStopping(monitor='val\_loss', patience=2)

این موضوع بیان می­دارد که ما می­خواستیم ضرر تست (اعتبارسنجی) را در هر دوره نظارت کنیم و پس از اینکه ضرر تست، بعد از دو دوره بهبود نیافت، آموزش قطع می­شود. با این حال، از آنجایی که patience=2 را تنظیم کردیم، بهترین مدل را نخواهیم گرفت، بلکه مدل را دو دوره بعد از بهترین مدل خواهیم داشت. بنابراین، به صورت اختیاری، می‌توانیم در عملیات دوم، ModelCheckpoint را اضافه کنیم، که مدل را پس از هر بازرسی در یک فایل ذخیره می‌کند (که می‌تواند در مواردی که یک دوره‌ی آموزشی چند روزه به دلایلی قطع شود، مفید باشد). اگر save\_best\_only=True را تنظیم کنیم برای ما مفید خواهد بود، زیرا در این صورت ModelCheckpoint فقط بهترین مدل را ذخیره می­کند.

**20. 10 کاهش بیش از حد برازش با انصراف**

**مسئله**

شما می­خواهید اوورفیت را کاهش دهید.

**راه حل**

با استفاده از dropout نویز را به معماری شبکه خود وارد کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 1000

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to a one-hot encoded feature matrix*

tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train,

mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network*

network = models.Sequential()

*# Add a dropout layer for input layer*

network.add(layers.Dropout(0.2, input\_shape= (number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add a dropout layer for previous hidden layer* network.add(layers.Dropout(0.5))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add a dropout layer for previous hidden layer* network.add(layers.Dropout(0.5))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector*

epochs=3, *# Number of epochs*

verbose=0, *# No output*

batch\_size=100, *# Number of*

*observations per batch*

validation\_data=(features\_test,

target\_test)) *# Test data*

Using TensorFlow backend.

**بحث**

Dropout یک روش محبوب و قدرتمند برای منظم کردن شبکه­های عصبی است. در Dropout، هر بار که دسته­ای از مشاهدات برای آموزش ایجاد می­شود، نسبتی از واحدها در یک یا چند لایه در صفر ضرب می­شود (یعنی حذف می‌شود). در این تنظیمات، هر دسته در یک شبکه آموزش داده می‌شود (به عنوان مثال، پارامترهای یکسان)، اما هر دسته با نسخه‌ی کمی متفاوت‌تر از معماری آن شبکه مواجه می­شود.

انصراف موثر است زیرا با حذف مداوم و تصادفی واحدها در هر دسته، واحدها را مجبور می‌کند تا مقادیر پارامترهایی را که می‌توانند تحت طیف گسترده‌ای از معماری‌های شبکه عمل کنند، یاد بگیرند. به این معنی که آنها یاد می­گیرند که در برابر اختلالات (به عنوان مثال، نویز) در سایر واحدهای پنهان، مقاوم باشند و این مانع از حفظ داده­های آموزشی توسط شبکه می­شود.

امکان اضافه کردن ریزش به هر دو لایه‌ی مخفی و ورودی وجود دارد. هنگامی که یک لایه ورودی حذف می­شود، مقدار ویژگی آن برای آن دسته به شبکه وارد نمی­شود. یک انتخاب متداول برای قسمتی از واحدها که حذف می­شوند، 0.2 برای واحدهای ورودی و 0.5 برای واحدهای پنهان است.

در Keras، می‌توانیم با افزودن لایه‌های Dropout به معماری شبکه خود، dropout را پیاده‌سازی کنیم. هر لایه Dropout یک اَبَرپارامتر تعریف شده از واحدها را در لایه قبلی در هر دسته حذف می­کند. به یاد داشته باشید که در Keras لایه ورودی به عنوان اولین لایه فرض می­شود و با استفاده از add اضافه نمی­شود. بنابراین، اگر بخواهیم dropout را به لایه ورودی اضافه کنیم، اولین لایه ای که در معماری شبکه خود اضافه می­کنیم یک لایه‌ی dropout است. این لایه شامل نسبت واحدهای لایه‌ی ورودی به افت 0.2 و input\_shape است که شکل داده­های مشاهده را مشخص می­کند. بعد از هر یک از لایه­های پنهان، یک لایه‌ی حذفی با 0.5 اضافه می­کنیم.

**20. 11 ذخیره پیشرفت آموزش مدل**

**مسئله**

با توجه به یک شبکه عصبی که آموزش آن زمان زیادی طول می­کشد، می­خواهید پیشرفت خود را در صورت قطع شدن فرآیند آموزش ذخیره کنید.

**راه حل**

از تابع callback ModelCheckpoint برای ذخیره مدل بعد از هر دوره استفاده کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing.text** **import** Tokenizer

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **keras.callbacks** **import** ModelCheckpoint

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 1000

*# Load data and target vector from movie review data* (data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Convert movie review data to a one-hot encoded feature matrix* tokenizer = Tokenizer(num\_words=number\_of\_features)

features\_train = tokenizer.sequences\_to\_matrix(data\_train, mode="binary")

features\_test = tokenizer.sequences\_to\_matrix(data\_test, mode="binary")

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16,

activation="relu",

input\_shape=(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Set callback functions to early stop training and save the best model so far*

checkpoint = [ModelCheckpoint(filepath="models.hdf5")]

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target vector* epochs=3, *# Number of epochs* callbacks=checkpoint, *# Checkpoint* verbose=0, *# No output* batch\_size=100, *# Number of*

*observations per batch*

validation\_data=(features\_test,

target\_test)) *# Test data*

Using TensorFlow backend.

بحث

در دستور العمل 20.8 ما از تابع callback ModelCheckpoint همراه با EarlyStopping برای پایان دادن به نظارت و پایان دادن به آموزش، زمانی که خطای تست بهبود نمی­یابد استفاده کردیم. با این حال، دلیل دیگری، پیش پا افتاده­تر برای استفاده از ModelCheckpoint وجود دارد. در دنیای واقعی، معمول است که شبکه­های عصبی ساعت­ها یا حتی روزها آموزش ببینند. در این مدت بسیاری از مشکلات ممکن است پیش بیایند: کامپیوترها ممکن است برق خود را از دست بدهند، سرورها از کار بیفتند یا دانشجویان فارغ‌التحصیل بی ملاحظه می­توانند لپ تاپ شما را ببندند.

ModelCheckpoint با ذخیره‌ی مدل بعد از هر دوره، این مشکل را کاهش می­دهد. به طور خاص، بعد از هر دوره ModelCheckpoint یک مدل را در مکانی که توسط پارامتر filepath مشخص شده است ذخیره می­کند.

اگر فقط یک نام فایل (به عنوان مثال models.hdf 5) را وارد کنیم، آن فایل در هر دوره با آخرین مدل بازنویسی(حذف فایل قبلی و نوشتن فایل جدید) می­شود. اگر فقط بخواهیم بهترین مدل را با توجه به عملکرد برخی از توابع ضرر ذخیره کنیم، می­توانیم به این صورت تنظیم کنیم:

save\_best\_only=True و monitor='val\_loss'

برای اینکه اگر مدل تست بدتری نسبت به مدل قبلی داشته باشد، فایلی را بازنویسی نکند. از طرف دیگر، ما می‌توانیم مدل هر دوره را به ‌عنوان فایل خودش با گنجاندن شماره دوره و نمره‌ی ضرر تست در نام فایل ذخیره کنیم. برای مثال، اگر مسیر file را روی آن تنظیم کنیم به این صورت میشود:

model\_{epoch:02d}\_{val\_loss:.2f}.hdf5

نام فایل حاوی مدل ذخیره شده‌ی پس از دوره‌ی یازدهم با مقدار تلفات آزمایشی 0.33 برای model\_10\_0.35.hdf5 خواهد بود (توجه داشته باشید که شماره دوره 0-شاخص شده).

**12.20 k-Fold شبکه­های عصبی متقاطع اعتبار**

**مسئله**

شما می­خواهید یک شبکه عصبی را با استفاده از اعتبارسنجی متقاطع k-fold ارزیابی کنید.

**راه حل**

در اغلب شبکه‌های عصبی متقابل، اعتبارسنجی k برابری نه ضروری است و نه توصیه می‌شود. با این حال، اگر مناسب باشد: از پوشش Sikit-Learn Keras استفاده کنید تا به مدل‌های متوالی Keras اجازه دهید از scikit-learn API استفاده کنند:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.datasets** **import** make\_classification

*# Set random seed*

np.random.seed(0)

*# Number of features*

number\_of\_features = 100

*# Generate features matrix and target vector*

features, target = make\_classification(n\_samples = 10000,

n\_features =

number\_of\_features,

n\_informative = 3, n\_redundant = 0, n\_classes = 2, weights = [.5, .5], random\_state = 0)

*# Create function returning a compiled network*

**def** create\_network():

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation*

*function*

network.add(layers.Dense(units=16, activation="relu",

input\_shape=(

number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function*

network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-*

*entropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy*

*performance metric*

*# Return compiled network*

**return** network

*# Wrap Keras model so it can be used by scikit-learn*

neural\_network = KerasClassifier(build\_fn=create\_network,

epochs=10,

batch\_size=100,

verbose=0)

*# Evaluate neural network using three-fold cross-validation* cross\_val\_score(neural\_network, features, target, cv=3)

Using TensorFlow backend.

array([ 0.90461907, 0.77437743, 0.87068707])

**بحث**

از نظر تئوری، دلیلی وجود ندارد که نتوانیم از اعتبارسنجی متقاطع برای ارزیابی شبکه­های عصبی استفاده کنیم. با این حال، شبکه‌های عصبی اغلب بر روی داده‌های بسیار بزرگ استفاده می‌شوند و ممکن است ساعت‌ها یا حتی روزها طول بکشد تا آموزش ببیند. به همین دلیل، اگر زمان آموزش طولانی باشد، اضافه کردن هزینه محاسباتی اعتبارسنجی متقاطع k برابر توصیه نمی­شود. برای مثال، ارزیابی مدلی که معمولاً یک روز طول می‌کشد، با استفاده از اعتبارسنجی متقابل 10 برابری ارزیابی شود. اگر داده­های بزرگی داریم، اغلب مناسب است که به سادگی شبکه عصبی را در برخی از مجموعه‌های آزمایشی ارزیابی کنیم.

اگر داده‌های کوچک‌تری داشته باشیم، اعتبارسنجی متقاطع k-fold می‌تواند برای به حداکثر رساندن توانایی ما در ارزیابی عملکرد شبکه عصبی مفید باشد. این در Keras امکان‌پذیر است، زیرا ما می‌توانیم هر شبکه عصبی را به گونه‌ای بپیچانیم که بتواند از ویژگی‌های ارزیابی موجود در scikit-learn، از جمله اعتبارسنجی متقاطع k-fold استفاده کند. برای انجام این کار، ابتدا باید تابعی ایجاد کنیم که یک شبکه عصبی کامپایل شده را بر می­گرداند. در مرحله بعد، از KerasClassifier (اگر طبقه‌بندی‌کننده داشته باشیم ازن همین تابع و اگر رگرسیور داشته باشیم، می‌توانیم از KerasRegressor استفاده کنیم) برای بسته‌بندی مدل استفاده می‌کنیم تا بتوان از آن توسط scikit-learn استفاده کرد. پس از این، ما می‌توانیم از شبکه عصبی خود مانند هر الگوریتم یادگیری Sicit-Learn دیگر استفاده کنیم (به عنوان مثال، جنگل‌های تصادفی، رگرسیون لجستیک). در راه حل خود، از cross\_val\_score برای اجرای یک اعتبارسنجی متقاطع سه برابری در شبکه عصبی خود استفاده کردیم.

**13.20 تنظیم شبکه­های عصبی**

**مسئله**

شما می­خواهید به طور خودکار بهترین هایپرپارامترها را برای شبکه عصبی خود انتخاب کنید.

راه حل

یک شبکه عصبی Keras را با ابزارهای انتخاب مدل scikit-learn مانند GridSearchCV ترکیب کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** GridSearchCV

**from** **sklearn.datasets** **import** make\_classification

*# Set random seed*

np.random.seed(0)

*# Number of features*

number\_of\_features = 100

*# Generate features matrix and target vector*

features, target = make\_classification(n\_samples = 10000,

n\_features =

number\_of\_features,

n\_informative = 3, n\_redundant = 0, n\_classes = 2, weights = [.5, .5], random\_state = 0)

*# Create function returning a compiled network*

**def** create\_network(optimizer="rmsprop"):

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation*

*function*

network.add(layers.Dense (units=16,

activation="relu", input\_shape=

(number\_of\_features,)))

*# Add fully connected layer with a ReLU activation function*

network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-*

*entropy*

optimizer=optimizer, *# Optimizer* metrics=["accuracy"]) *# Accuracy*

*performance metric*

*# Return compiled network*

**return** network

*# Wrap Keras model so it can be used by scikit-learn* neural\_network = KerasClassifier(build\_fn=create\_network, verbose=0)

*# Create hyperparameter space*

epochs = [5, 10]

batches = [5, 10, 100]

optimizers = ["rmsprop", "adam"]

*# Create hyperparameter options*

hyperparameters = dict(optimizer=optimizers, epochs=epochs, batch\_size=batches)

*# Create grid search*

grid = GridSearchCV(estimator=neural\_network,

param\_grid=hyperparameters)

*# Fit grid search*

grid\_result = grid.fit(features, target)

Using TensorFlow backend.

**بحث**

در دستور العمل­های 12.1 و 12.2، ما استفاده از تکنیک­های انتخاب مدل scikit-learn برای شناسایی بهترین فراپارامترهای یک مدل scikitlearn را پوشش دادیم. در دستور العمل 20.12 ما یاد گرفتیم که می­توانیم شبکه عصبی خود را بپیچانیم تا بتواند از API scikit-learn استفاده کند. در این دستور ما این دو تکنیک را برای شناسایی بهترین هایپرپارامترهای یک شبکه عصبی ترکیب می­کنیم.

فراپارامترهای یک مدل مهم هستند و باید با دقت انتخاب شوند. با این حال، قبل از اینکه به ذهنمان خطور کند که استراتژی‌های انتخاب مدل مانند جستجوی شبکه‌ای ایده‌ی خوبی است، باید بدانیم که اگر آموزش مدل ما معمولاً ۱۲ ساعت یا یک روز طول می‌کشید، این فرآیند جستجوی شبکه‌ای ممکن است یک هفته یا بیشتر طول بکشد. بنابراین، تنظیم خودکار فراپارامتر شبکه‌های عصبی، گلوله‌ی نقره‌ای نیست، اما در شرایط خاص ابزار مفیدی است.

در راه حل خود، ما یک جست و جوی شبکه‌ی اعتبارسنجی متقابل بر روی تعدادی گزینه برای الگوریتم بهینه سازی، تعداد دوره‌ها و اندازه‌ی دسته انجام دادیم. حتی اجرای این نمونه‌ی اسباب‌بازی چند دقیقه طول کشید، اما پس از انجام آن، می‌توانیم از best\_params\_ برای مشاهده هایپرپارامترهای شبکه عصبی با بهترین نتایج استفاده کنیم:

*# View hyperparameters of best neural network*

grid\_result. best\_params\_

{'batch\_size': 10, 'epochs': 5, 'optimizer': 'adam'}

**14.20 تجسم شبکه­های عصبی**

**مسئله**

شما می­خواهید به سرعت، معماری یک شبکه عصبی را به تصویر بکشید.

**راه حل**

از model\_to\_dot در Keras یا plot\_model استفاده کنید:

*# Load libraries*

**from** **keras** **import** models

**from** **keras** **import** layers

**from** **IPython.display** **import** SVG

**from** **keras.utils.vis\_utils** **import** model\_to\_dot

**from** **keras.utils** **import** plot\_model

*# Start neural network*

network = models.Sequential()

*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu", input\_shape=(10,)))

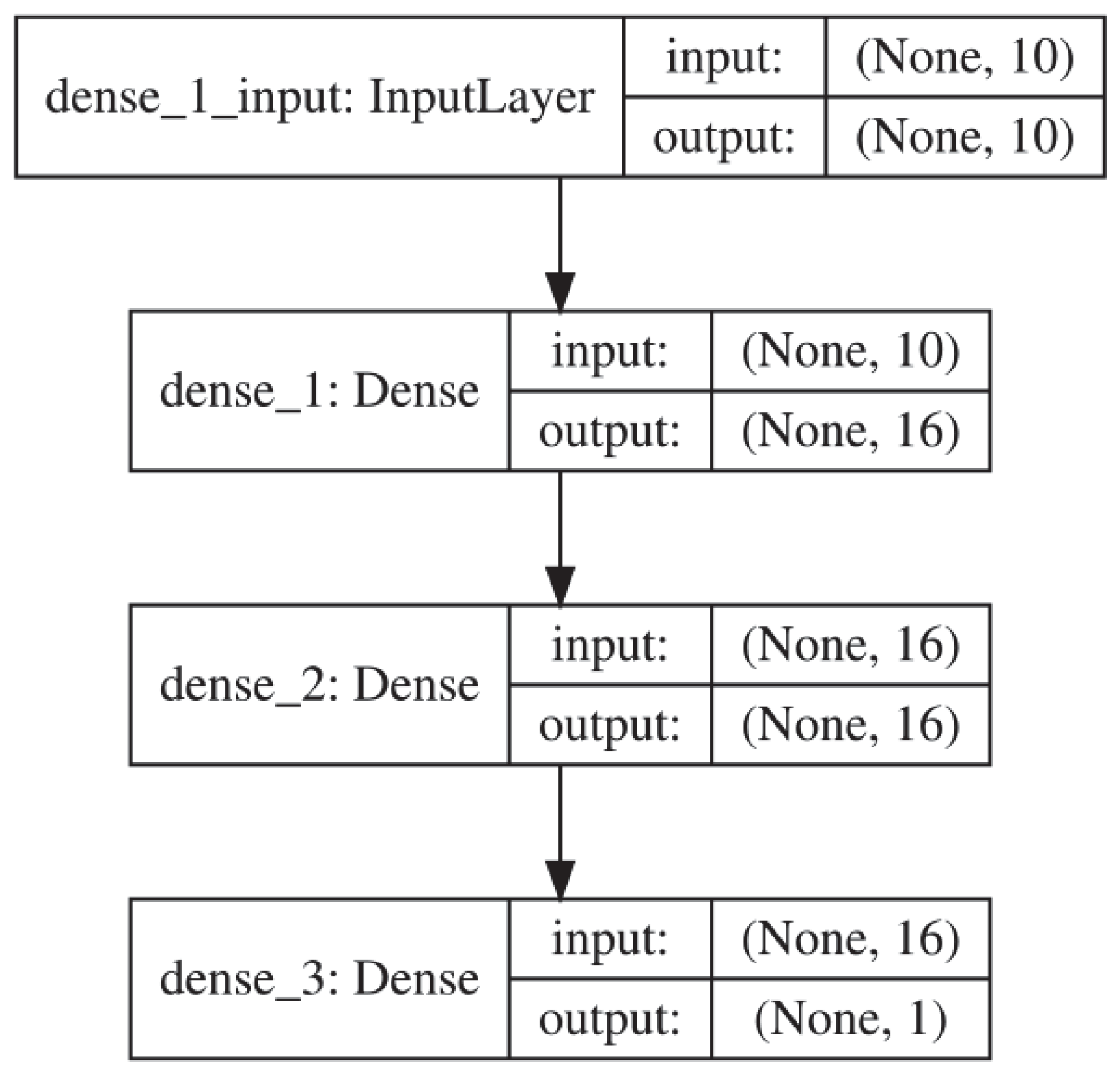
*# Add fully connected layer with a ReLU activation function* network.add(layers.Dense(units=16, activation="relu"))

*# Add fully connected layer with a sigmoid activation function* network.add(layers.Dense(units=1, activation="sigmoid"))

*# Visualize network architecture* SVG(model\_to\_dot(network,

show\_shapes=True).create(prog="dot", format="svg"))

Using TensorFlow backend.



یا اگر بخواهیم تصویرسازی را به صورت فایل ذخیره کنیم، می­توانیم از plot\_model استفاده کنیم:

*# Save the visualization as a file* plot\_model(network, show\_shapes=True, to\_file="network.png")

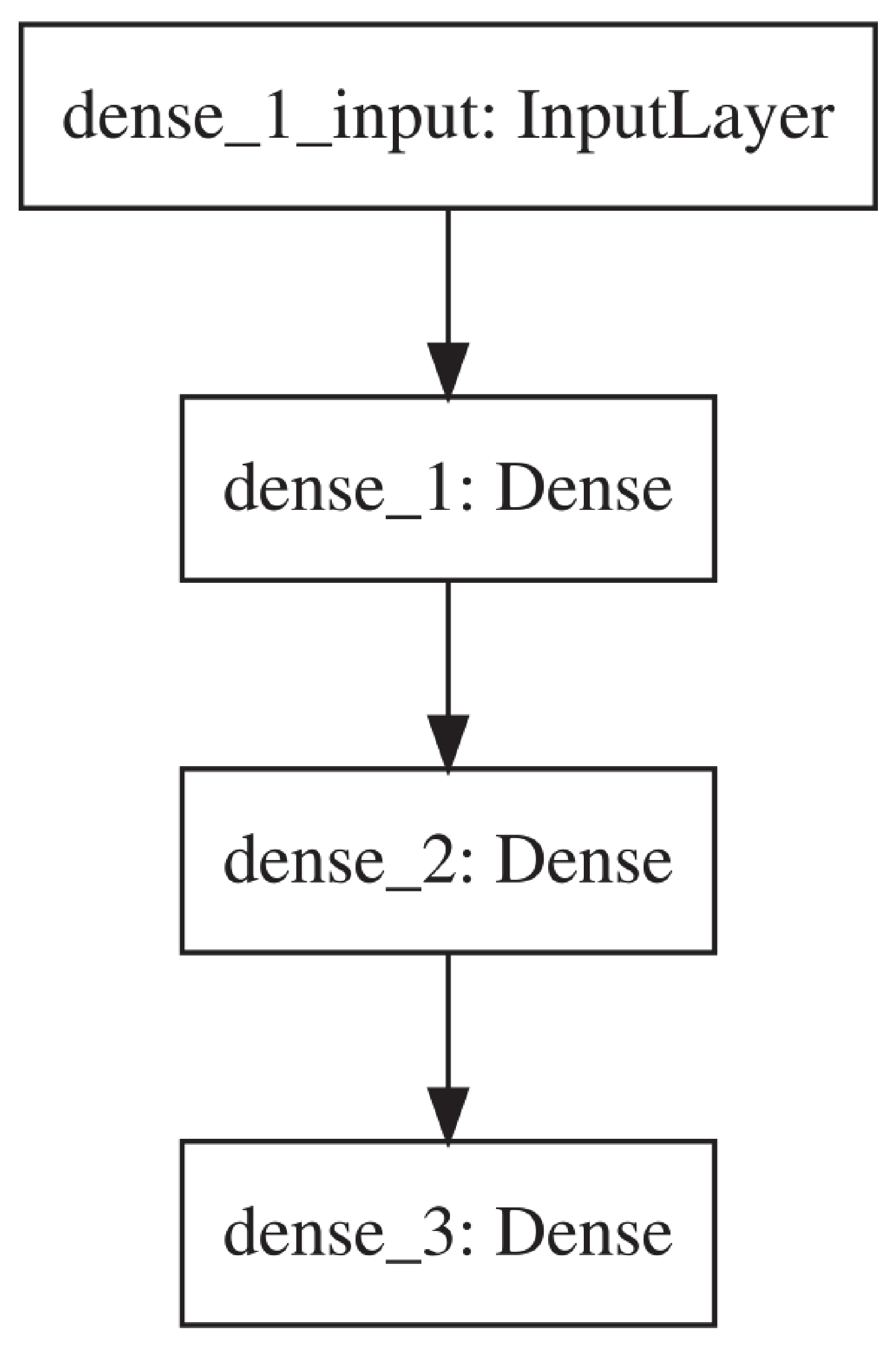
**بحث**

Keras توابع مفیدی را برای به تصویر کشیدن سریع شبکه­های عصبی فراهم می­کند.

اگر بخواهیم یک شبکه عصبی را در یک نوت بوک Jupyter نمایش دهیم، می‌توانیم از model\_to\_dot استفاده کنیم. پارامتر show\_shapes نشان می­دهد که شکل ورودی­ها و خروجی­ها را دارد و می­تواند به اشکال زدایی کمک کند. برای یک مدل ساده­تر، می­توانیم show\_shapes=True را تنظیم کنیم:

*# Visualize network architecture* SVG(model\_to\_dot(network,

show\_shapes=False).create(prog="dot", format="svg"))



**15.20 دسته‌بندی تصاویر**

**مسئله**

شما می­خواهید تصاویر را با استفاده از یک شبکه عصبی کانولوشن دسته‌بندی کنید.

**راه حل**

از Keras برای ایجاد یک شبکه عصبی با حداقل یک لایه کانولوشنال استفاده کنید:

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** mnist

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Dropout, Flatten

**from** **keras.layers.convolutional** **import** Conv2D, MaxPooling2D

**from** **keras.utils** **import** np\_utils

**from** **keras** **import** backend **as** K

*# Set that the color channel value will be first*

K.set\_image\_data\_format("channels\_first")

*# Set seed*

np.random.seed(0)

*# Set image information*

channels = 1

height = 28

width = 28

*# Load data and target from MNIST data*

(data\_train, target\_train), (data\_test, target\_test) = mnist.load\_data()

*# Reshape training image data into feature*

data\_train = data\_train.reshape(data\_train.shape[0], channels, height, width)

*# Reshape test image data into features*

data\_test = data\_test.reshape(data\_test.shape[0], channels, height, width)

*# Rescale pixel intensity to between 0 and 1*

features\_train = data\_train / 255

features\_test = data\_test / 255

*# One-hot encode target*

target\_train = np\_utils.to\_categorical(target\_train)

target\_test = np\_utils.to\_categorical(target\_test)

number\_of\_classes = target\_test.shape[1]

*# Start neural network*

network = Sequential()

*# Add convolutional layer with 64 filters, a 5x5 window, and ReLU activation function*

network.add(Conv2D(filters=64,

kernel\_size=(5, 5),

input\_shape=(channels, width, height), activation='relu'))

*# Add max pooling layer with a 2x2 window*

network.add(MaxPooling2D(pool\_size=(2, 2)))

*# Add dropout layer*

network.add(Dropout(0.5))

*# Add layer to flatten input*

network.add(Flatten())

*# # Add fully connected layer of 128 units with a ReLU activation function*

network.add(Dense(128, activation="relu"))

*# Add dropout layer*

network.add(Dropout(0.5))

*# Add fully connected layer with a softmax activation function*

network.add(Dense(number\_of\_classes, activation="softmax"))

*# Compile neural network*

network.compile(loss="categorical\_crossentropy", *# Crossentropy*

optimizer="rmsprop", *# Root Mean Square*

*Propagation*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Train neural network*

network.fit(features\_train, *# Features*

target\_train, *# Target*

epochs=2, *# Number of epochs*

verbose=0, *# Don't print description after each*

*epoch*

batch\_size=1000, *# Number of observations per*

*batch*

validation\_data=(features\_test, target\_test)) *#*

*Data for evaluation*

Using TensorFlow backend.

<keras.callbacks.History at 0x133f37e80>

**بحث**

شبکه­های عصبی کانولوشن (که همچنین ConvNets نامیده می‌شوند) یک نوع شبکه محبوب هستند که در بینایی کامپیوتری (به عنوان مثال، تشخیص گربه­ها، سگ­ها، هواپیماها و حتی هات‌داگ­ها) بسیار موثر بوده­اند. استفاده از شبکه­های عصبی پیشخور روی تصاویر کاملاً امکان پذیر است که هر پیکسل یک ویژگی است. با این حال، هنگام انجام این کار با آن مواجه می­شویم.

دو مشکل عمده‌ داریم. اول، شبکه­های عصبی پیشخور ساختار فضایی پیکسل­ها را در نظر نمی­گیرند. به عنوان مثال، در یک تصویر 10 × 10 پیکسل، ما ممکن است آن را به یک بردار از ویژگی­های 100 پیکسل تبدیل کنیم، و در این مورد، فید فوروارد اولین ویژگی (به عنوان مثال، مقدار پیکسل) را همان رابطه‌ی ویژگی دهم با ویژگی یازدهم در نظر می‌گیرد. با این حال، در واقعیت، ویژگی دهم یک پیکسل در سمت دور تصویر را به عنوان اولین ویژگی نشان می‌دهد، در حالی که ویژگی یازدهم، پیکسلی را بلافاصله زیر پیکسل اول نشان می‌دهد. دوم و به همین ترتیب، شبکه­های عصبی پیشخور، روابط کلی را در ویژگی‌ها به جای الگوهای محلی یاد می­گیرند. به عبارت عملی­تر، این بدان معناست که شبکه­های عصبی پیشخور قادر به تشخیص یک شی بدون توجه به جایی که در یک تصویر ظاهر می­شود، نیستند. به عنوان مثال، تصور کنید ما در حال آموزش یک شبکه عصبی برای تشخیص چهره هستیم، و این چهره‌ها ممکن است در هر نقطه از تصویر از سمت راست بالا تا وسط به سمت چپ پایین ظاهر شوند.

قدرت شبکه­های عصبی کانولوشنال توانایی آنها در رسیدگی به هر دوی این مسائل (و سایر مسائل) است. توضیح کامل شبکه‌های عصبی کانولوشنال از حوصله این کتاب خارج است، اما توضیح مختصری مفید خواهد بود. داده­های یک تصویر مجزا شامل دو یا سه بعد است: ارتفاع، عرض و عمق. دو مورد اول باید واضح باشد، اما آخری شایسته‌ی توضیح است. در تصاویر خاکستری، عمق تنها یک (شدت پیکسل) است و بنابراین تصویر یک ماتریس است. با این حال، در تصاویر رنگی، رنگ یک پیکسل با مقادیر متعدد نشان داده می­شود. به عنوان مثال، در یک تصویر RGB، رنگ یک پیکسل با سه مقدار نشان دهنده‌ی قرمز، سبز و آبی نشان داده می­شود. بنابراین، داده­های یک تصویر را می­توان یک تانسور سه بعدی تصور کرد: عرض × ارتفاع × عمق (به نام نقشه­های ویژگی).

در شبکه‌های عصبی کانولوشن، یک پیچیدگی (اگر نمی‌دانید به چه معناست نگران نباشید) را اینطور می‌توان تصور کرد که یک پنجره را روی پیکسل‌های یک تصویر بکشید و هم به پیکسل‌های مجزا و هم به همسایگان آن نگاه کنید. سپس داده‌های تصویر خام را به یک تانسور سه بعدی جدید تبدیل کنید که در آن دو بعد اول تقریباً عرض و ارتفاع هستند، در حالی که بعد سوم (که حاوی مقادیر رنگ بود) اکنون الگوها را نشان می‌دهد - به نام فیلترها - (به عنوان مثال، یک تصویر واضح گوشه یا گرادیان فراگیر) که آن پیکسل به آن تعلق دارد.

دومین مفهوم مهم برای اهداف ما، ادغام لایه‌ها است. لایه‌های ادغام شده، پنجره‌ای را روی داده‌های ما جابه‌جا می‌کنند (اگرچه معمولاً فقط به هر پیکسل n که به آن striding می‌گویند نگاه می‌کنند) داده‌های ما را با خلاصه کردن پنجره به نحوی کوچک می‌کنند. متداول‌ترین روش، حداکثر ادغام است که در آن حداکثر مقدار در هر پنجره به لایه بعدی ارسال می‌شود. یکی از دلایل حداکثر جمع آوری صرفاً عملی است. فرآیند کانولوشنال پارامترهای بسیار بیشتری را برای یادگیری ایجاد می‌کند که می‌تواند خیلی سریع به دست بیاورد. دوم، به طور واضح تر، حداکثر ادغام را می‌توان به عنوان "کوچک کردن" یک تصویر در نظر گرفت.

یک مثال ممکن است در اینجا مفید باشد. تصور کنید تصویری داریم که شامل صورت یک سگ است. اولین لایه‌ی کانولوشن ممکن است الگوهایی مانند لبه‌های شکل پیدا کند. سپس از یک لایه مکس پول(max pool) برای "کوچک کردن" و یک لایه کانولوشنال دوم برای پیدا کردن الگوهایی مانند گوش سگ استفاده می­کنیم. در نهایت از یک لایه حداکثری دیگر برای بزرگنمایی مجدد و یک لایه کانولوشنال نهایی برای یافتن الگوهایی مانند صورت سگ­ها استفاده می‌کنیم.

در نهایت، لایه‌های کاملاً متصل اغلب در انتهای شبکه برای انجام دسته‌بندی واقعی استفاده می‌شوند.

در حالی که راه حل ما ممکن است مانند بسیاری از خطوط کد به نظر برسد، در واقع بسیار شبیه به دسته‌بندی کننده‌ی باینری ما در قبل از این فصل است. در این راه حل، ما از مجموعه داده‌ی معروف MNIST استفاده کردیم که یک مجموعه داده‌ی معیار واقعی در یادگیری ماشین است. مجموعه داده‌ی MNIST شامل 70000 تصویر کوچک (28 × 28) از ارقام دست‌نویس از 0 تا 9 است. این مجموعه داده، برچسب‌گذاری شده است تا رقم واقعی (یعنی کلاس) را برای هر تصویر کوچک بدانیم. تقسیم استاندارد آموزش و آزمون استفاده از 60000 تصویر برای آموزش و 10000 تصویر برای تست است.

ما داده­ها را در قالب مورد انتظار یک شبکه کانولوشن سازماندهی مجدد کردیم. به طور خاص، ما از reshape برای تبدیل داده‌های مشاهده به گونه‌ای استفاده کردیم که شکل مورد انتظار Keras باشد. در مرحله بعد، ما مقادیر را بین 0 و 1 تغییر دادیم، زیرا اگر مقادیر یک مشاهده بسیار بیشتر از پارامترهای شبکه (که به صورت اعداد کوچک اولیه می‌شوند) باشد، عملکرد آموزشی ممکن است آسیب ببیند. در نهایت، ما داده‌های هدف را کدگذاری کردیم تا هدف هر مشاهده دارای 10 کلاس باشد که نشان‌دهنده ارقام 0-9 است.

با فرآیند داده‌های تصویری می­توانیم شبکه کانولوشن خود را بسازیم. ابتدا یک لایه‌ی کانولوشن اضافه می­کنیم و تعداد فیلترها و سایر مشخصات را مشخص می­کنیم. اندازه پنجره یک فراپارامتر است. با این حال، 3 × 3 روش استاندارد برای اکثر تصاویر است، در حالی که پنجره­های بزرگتر اغلب در تصاویر بزرگتر استفاده می‌شوند. دوم، ما یک لایه ادغام حداکثر اضافه می‌کنیم و پیکسل­های نزدیک را خلاصه می­کنیم. سوم، ما یک لایه dropout اضافه می­کنیم تا شانس بیش از حد برازش شدن مدل را کاهش دهیم. چهارم، ما یک لایه‌ی مسطح اضافه می­کنیم تا ورودی­های کانولوشن را به قالبی تبدیل کنیم که می­تواند توسط یک لایه‌ی کاملاً متصل استفاده شود. در نهایت، لایه­های کاملا متصل و یک لایه خروجی را اضافه می­کنیم تا دسته‌بندی واقعی انجام شود.

توجه داشته باشید که چون این یک مشکل دسته‌بندی چند کلاسه است، از تابع فعال سازی softmax در لایه خروجی استفاده می­کنیم.

لازم به ذکر است که این یک شبکه عصبی کانولوشنال بسیار ساده است. مشاهده‌ی یک شبکه بسیار عمیق‌تر با لایه‌های ترکیبی و حداکثری بیشتر در کنار هم معمول است.

**16.20 بهبود عملکرد با تقویت تصویر**

**مسئله**

شما می­خواهید عملکرد شبکه عصبی کانولوشنال خود را بهبود ببخشید.

**راه حل**

برای نتایج بهتر، تصاویر را از قبل پردازش کرده و داده­ها را با استفاده از ImageDataGenerator تقویت کنید:

*# Load library*

**from** **keras.preprocessing.image** **import** ImageDataGenerator

*# Create image augmentation*

augmentation = ImageDataGenerator(featurewise\_center=True, *#*

*Apply ZCA whitening*

zoom\_range=0.3, *# Randomly*

*zoom in on images*

width\_shift\_range=0.2, *#*

*Randomly shift images*

horizontal\_flip=True, *#*

*Randomly flip images*

rotation\_range=90) *#*

*Randomly rotate*

*# Process all images from the directory 'raw/images'* augment\_images = augmentation.flow\_from\_directory("raw/images", *# Image folder*

batch\_size=32, *# Batch size*

class\_mode="binary", *# Classes*

save\_to\_dir="processed/images")

Using TensorFlow backend.

Found 12665 images belonging to 2 classes.

**بحث**

ابتدا یک عذرخواهی - کد این راه حل بلافاصله برای شما اجرا نمی­شود زیرا پوشه‌های مورد نیاز تصاویر را ندارید. با این حال، از آنجایی که رایج‌ترین حال، داشتن فهرستی از تصاویر است، می‌خواستم آن را اضافه کنم. این تکنیک باید به راحتی به تصاویر شما ترجمه شود.

یکی از راه­های بهبود عملکرد یک شبکه عصبی کانولوشن، پیش پردازش تصاویر است. ما تعدادی از تکنیک­ها را در فصل 8 مورد بحث قرار دادیم. با این حال، شایان ذکر است که ImageDataGenerator در کتابخانه‌ی Keras حاوی تعدادی تکنیک اولیه پیش پردازش است.

به عنوان مثال، ما در راه حل خود از featurewise\_center=True برای استانداردسازی پیکسل­ها در کل داده­ها استفاده کردیم.

تکنیک دوم برای بهبود عملکرد، اضافه کردن نویز است. یکی از ویژگی‌های جالب شبکه‌های عصبی این است که به طور غیرمنتظره، عملکرد آنها اغلب با اضافه شدن نویز به داده‌ها بهبود می‌یابد. دلیل آن این است که نویز اضافی می‌تواند شبکه‌های عصبی را در برابر نویزهای دنیای واقعی قوی‌تر کند و از تطبیق بیش از حد داده‌ها جلوگیری کند.

هنگام آموزش شبکه‌های عصبی کانولوشنال برای تصاویر، می‌توانیم با تبدیل تصادفی تصاویر به روش‌های مختلف، مانند ورق زدن تصاویر یا بزرگ‌نمایی روی تصاویر، نویز را به مشاهدات خود اضافه کنیم. حتی تغییرات کوچک می­تواند عملکرد مدل را به طور قابل توجهی بهبود بخشد. ما می­توانیم از همان کلاس ImageDataGenerator برای انجام این تبدیل­ها استفاده کنیم. اسناد Keras (به «همچنین ببینید» اشاره شده است) فهرست کامل تبدیل‌های موجود را مشخص می‌کند. با این حال، مثال ما شامل نمونه­ای از آنها است، از جمله بزرگنمایی تصادفی، جابجایی، چرخش و چرخش.

توجه به این نکته مهم است که خروجی flow\_from\_directory یک شیء مولد پایتون است. این به این دلیل است که در بیشتر موارد ما می­خواهیم تصاویر را به صورت درخواستی پردازش کنیم زیرا آنها برای آموزش به شبکه عصبی ارسال می­شوند. اگر بخواهیم تمام تصاویر را قبل از آموزش پردازش کنیم، می‌توانیم به سادگی روی ژنراتور تکرار کنیم.

در نهایت، از آنجایی که augment\_images یک مولد است، هنگام آموزش شبکه عصبی ما باید به جای fit از fit\_generator استفاده کنیم. مثلاْ:

*# Train neural network*

network.fit\_generator(augment\_images,

*#Number of times to call the generator for each epoch*

steps\_per\_epoch=2000,

*# Number of epochs* epochs=5,

*# Test data generator*

validation\_data=augment\_images\_test,

*# Number of items to call the*

*generator*

*# for each test epoch* validation\_steps=800)

توجه داشته باشید که تمام تصاویر خام استفاده شده در این دستور در GitHub موجود است.

همچنین ببینید

* [پیش پردازش تصویر، Keras](https://keras.io/api/preprocessing/image/)

**20. 17 دسته‌بندی متن**

**مسئله**

شما می­خواهید داده­های متنی را دسته‌بندی کنید.

راه حل

از یک شبکه عصبی بازگشتی حافظه کوتاه مدت استفاده کنید:

*# Load libraries*

**import** **numpy** **as** **np**

**from** **keras.datasets** **import** imdb

**from** **keras.preprocessing** **import** sequence

**from** **keras** **import** models **from** **keras** **import** layers

*# Set random seed*

np.random.seed(0)

*# Set the number of features we want*

number\_of\_features = 1000

*# Load data and target vector from movie review data*

(data\_train, target\_train), (data\_test, target\_test) = imdb.load\_data(

num\_words=number\_of\_features)

*# Use padding or truncation to make each observation have*

*400 features*

features\_train = sequence.pad\_sequences(data\_train, maxlen=400) features\_test = sequence.pad\_sequences(data\_test, maxlen=400)

*# Start neural network*

network = models.Sequential()

*# Add an embedding layer*

network.add(layers.Embedding(input\_dim=number\_of\_features, output\_dim=128))

*# Add a long short-term memory layer with 128 units* network.add(layers.LSTM(units=128))

*# Add fully connected layer with a sigmoid activation function*

network.add(layers.Dense(units=1, activation="sigmoid"))

*# Compile neural network*

network.compile(loss="binary\_crossentropy", *# Cross-entropy*

optimizer="Adam", *# Adam optimization*

metrics=["accuracy"]) *# Accuracy performance*

*metric*

*# Train neural network*

history = network.fit(features\_train, *# Features*

target\_train, *# Target*

epochs=3, *# Number of epochs*

verbose=0, *# Do not print description*

*after each epoch*

batch\_size=1000, *# Number of*

*observations per batch*

validation\_data=(features\_test, target\_test)) *# Test data*

Using TensorFlow backend.

**بحث**

اغلب اوقات ما داده­های متنی‌ای داریم که می‌خواهیم آنها را دسته‌بندی کنیم. در حالی که امکان استفاده از یک نوع شبکه‌ی کانولوشن وجود دارد، ما بر روی گزینه محبوب‌تر تمرکز می­کنیم: شبکه عصبی بازگشتی. ویژگی کلیدی شبکه­های عصبی بازگشتی این است که اطلاعات، دوباره به شبکه بازمی گردد. این مورد به شبکه­های عصبی بازگشتی نوعی حافظه می­دهد که می­توانند برای درک بهتر داده­های متوالی استفاده کنند. یک نوع محبوب شبکه عصبی بازگشتی، شبکه حافظه کوتاه مدت بلند مدت (LSTM) است که به اطلاعات اجازه می­دهد تا در شبکه به عقب بروند. برای توضیح بیشتر، به منابع اضافی مراجعه کنید.

در این راه حل، ما داده‌های بررسی فیلم خود را از دستور 20.3 داریم و می­خواهیم یک شبکه LSTM را آموزش دهیم تا پیش بینی کند که آیا این نقدها مثبت هستند یا منفی. قبل از اینکه بتوانیم شبکه خود را آموزش دهیم، کمی پردازش داده مورد نیاز است. داده­های متنی ما به شکل لیستی از اعداد صحیح است:

*# View first observation*

**Print (**data\_train [0])

[1, 14, 22, 16, 43, 530, 973, 2, 2, 65, 458, 2, 66, 2, 4, 173, 36, 256, 5, 25,

100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150,

4, 172, 112, 167, 2,

336, 385, 39, 4, 172, 2, 2, 17, 546, 38, 13, 447, 4,

192, 50, 16, 6, 147, 2,

19, 14, 22, 4, 2, 2, 469, 4, 22, 71, 87, 12, 16, 43,

530, 38, 76, 15, 13, 2,

4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12,

8, 316, 8, 106, 5, 4,

2, 2, 16, 480, 66, 2, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135,

48, 25, 2, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,

16, 82, 2, 8, 4, 107,

117, 2, 15, 256, 4, 2, 7, 2, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46,

7, 4, 2, 2, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2, 56, 26, 141, 6, 194, 2,

18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224,

92, 25, 104, 4, 226, 65, 16, 38, 2, 88, 12, 16, 283, 5,

16, 2, 113, 103, 32,

15, 16, 2, 19, 178, 32]

هر عدد صحیح در این لیست مربوط به یک کلمه است. با این حال، از آنجا که هر مرور شامل تعداد کلمات یکسانی نیست، طول هر مشاهده یکسان نیست. بنابراین، قبل از اینکه بتوانیم این داده­ها را در شبکه عصبی خود وارد کنیم، باید تمام مشاهدات را به یک اندازه انجام دهیم. ما می­توانیم این کار را با استفاده از pad\_sequences انجام دهیم. pad\_sequences داده‌های هر مشاهده را به گونه‌ای قرار می‌دهد که اندازه همه آنها یکسان باشد. اگر به اولین مشاهده خود پس از پردازش توسط pad\_sequences نگاه کنیم، می­توانیم این را ببینیم:

*# View first observation*

**Print (**data\_train [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 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 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 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 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 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 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 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 0 0 0 0 0

0 0 0 0 1 89 27 2 2 17 199 132 5 2 16 2 24 8

760 4 2 7 4 22 2 2 16 2 17 2 7 2 2 9 4 2

8 14 991 13 877 38 19 27 239 13 100 235 61 483 2 4 7 4

20 131 2 72 8 14 251 27 2 7 308 16 735 2 17

29 144 28

77 2 18 12]

در مرحله‌ی بعد، ما از یکی از امیدوارکننده‌ترین تکنیک­ها در پردازش زبان طبیعی استفاده می­کنیم: جاسازی کلمات. در طبقه‌بندی‌کننده باینری خود از دستور 20.3، مشاهدات را یک‌بار کدگذاری کردیم و از آن به عنوان ورودی شبکه عصبی استفاده کردیم. با این حال، این بار ما هر کلمه را به عنوان یک بردار در یک فضای چند بعدی نشان خواهیم داد و اجازه می­دهیم فاصله بین دو بردار نشان دهنده‌ی شباهت بین کلمات باشد. در Keras می­توانیم این کار را با افزودن یک لایه Embedding انجام دهیم. برای هر مقداری که به لایه‌ی Embedding ارسال می‌شود، یک بردار نشان دهنده، آن کلمه را به صورت خروجی می‌دهد. لایه‌ی زیر لایه‌ی LSTM ما با 128 واحد است که امکان استفاده از اطلاعات ورودی­های قبلی را در عملیات آتی فراهم می­کند و به طور مستقیم به ماهیت متوالی داده­ها می­پردازد. در نهایت، چون این یک مشکل دسته‌بندی باینری است (هر بررسی مثبت یا منفی است)، یک لایه‌ی خروجی کاملاً متصل با یک واحد و یک تابع فعال‌ساز سیگموئید اضافه می‌کنیم.

شایان ذکر است که LSTM‌ها یک موضوع بسیار گسترده و تمرکز تحقیقات زیادی هستند. این دستورالعمل، در حالی که امیدواریم مفید باشد، با آخرین کلمه در مورد اجرای LSTM‌ها فاصله دارد.