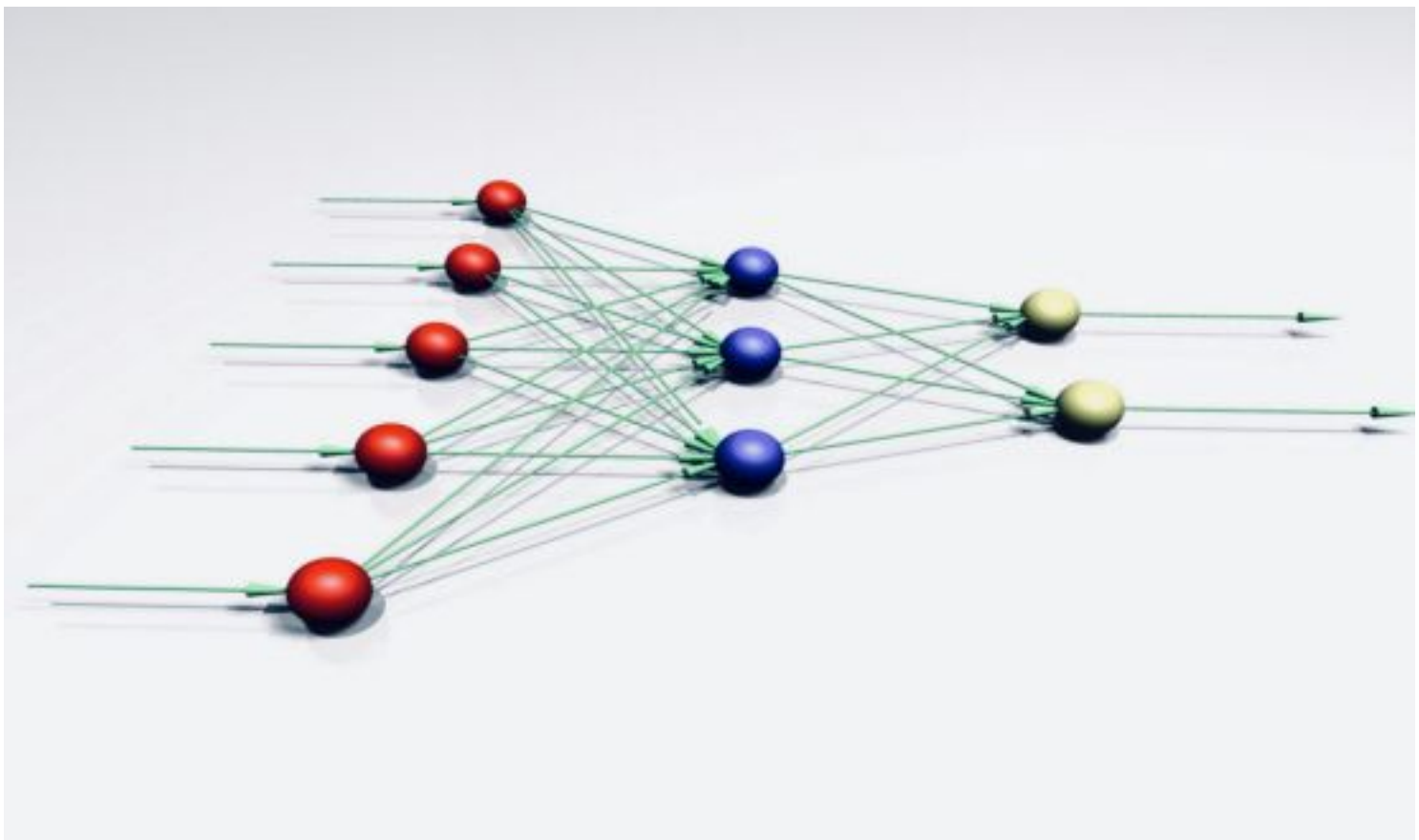
教你使用Keras一步步构建深度神经网络：以情感分析任务为例

2018-04-18 专知

【导读】Keras是深度学习领域一个非常流行的库，通过它可以使用简单的代码构建强大的神经网络。本文介绍基于Keras构建神经网络的基本过程，包括加载数据、分析数据、构建模型，配置模型等。并通过imdb情感分类任务来让读者更加清晰地了解每一步的过程，最终实现一个完整的情感分类实例。

作者 | Niklas Donges
编译 | 专知
参与 | Yingying, Xiaowen



用Keras构建神经网络

Keras是目前最受欢迎的深度学习库之一，对人工智能的商品化做出了巨大贡献。它使用起来非常简单，允许你用几行代码构建强大的神经网络。在这篇文章中，你将了解如何通过Keras构建神经网络，通过将用户评论分为两类：积极或消极评估来预测用户评论的情感。这就是所谓的情感分析，我们会用著名的imdb评论数据集来做实验。我们将构建的模型也可以应用于其他机器学习问题，只需进行一些更改。

请注意，本文我们不会深入Keras或深度学习的细节。本文旨在为你提供Keras神经网络的蓝图，并使你熟悉其实现。

Keras是什么?

Keras是一个开源的python库，可以让你轻松构建神经网络。该库能够在TensorFlow，Microsoft Cognitive Toolkit，Theano和MXNet上运行。 Tensorflow和Theano是Python中用来构建深度学习算法的最常用的数字平台，但它们可能相当复杂且难以使用。相比之下，Keras提供了一种简单方便的方法来构建深度学习模型。它的创造者是FrançoisChollet，使人们能够尽可能快速和简单地构建神经网络。他专注于可扩展性，模块化，极简主义和python的支持。Keras可以使用GPU和CPU，它同时支持Python 2和3。 Google Keras为深度学习和人工智能的商品化做出了巨大贡献，因为它已经商品化了强大的现代深度学习算法，这些算法以前不仅无法访问，而且也不可用。

什么是情感分析?

借助情感分析，我们想要确定说话者或作家对于文档，交互或事件的态度（例如情绪）。因此，这是一个自然语言处理问题，需要理解文本，以及潜在的意图。情绪主要分为积极的，消极的和中立三类。因此，情感分析广泛应用于诸如评论，调查，文档等等。

imdb数据集

imdb情绪分类数据集由来自imdb用户的50,000个电影评论组成，标记为positive（1）或negative（0）。评论是预处理的，每一个都被编码为一个整数形式的单词索引序列。评论中的单词按照它们在数据集中的总体频率进行索引。例如，整数“2”编码数据中第二个最频繁的词。 50,000个评论中，25000个作为训练集，另25000个作为测试集。该数据集由斯坦福大学的研究人员创建并于2011年发布，他们的准确率达到了88.89%。

导入库并获取数据

我们首先导入所需的库来预处理数据。

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import to_categorical
from keras import models
from keras import layers
```

我们继续下载已经内置到Keras中的imdb数据集。由于我们不希望将数据集进行50/50的训练、测试拆分，我们会在下载后立即将数据合并到数据和目标中，因此我们可以稍后再进行80/20拆分。

```
from keras.datasets import imdb
(training_data, training_targets), (testing_data, testing_targets) =
```

```
imdb.load_data(num_words=10000)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)
```

探索数据

现在我们可以开始探索数据集了：

```
print("Categories:", np.unique(targets))
print("Number of unique words:", len(np.unique(np.hstack(data))))

Categories: [0 1]
Number of unique words: 9998
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))

Average Review length: 234.75892
Standard Deviation: 173.0
```

你可以在上面的输出中看到数据集被标记为两个类别，分别代表0或1，表示评论的情感。整个数据集包含9998个字，评论的平均长度为234个字，标准差为173个字。

现在我们来看一个训练样例：

```
print("Label:", targets[0])

Label: 1
print(data[0])

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 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, 4536, 1111, 17, 546,
 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613,
 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17,
 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223,
 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51,
 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16,
 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71,
 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381,
 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21,
 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104,
 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,
 15, 16, 5345, 19, 178, 32]
```

你在上面，你可以看到数据集的第一次预览，它被标记为“正”（1）。下面的代码检索字典映射词索引回到原来的单词，以便我们可以阅读它们。它用“#”替换每个未知的单词。它通过使用

get_word_index（）函数来完成。

```
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (key, value) in index.items()])
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
print(decoded)

# this film was just brilliant casting location scenery story direction everyone's
# really suited the part they played and you could just imagine being there robert
#  is an amazing actor and now the same being director
#  father came from the same scottish island as myself so i loved the fact
# there was a real connection with this film the witty remarks throughout
# the film were great it was just brilliant so much that i bought the film as soon as
it was released for
#  and would recommend it to everyone to watch and the fly fishing was amazing
really cried at the end
# it was so sad and you know what they say if you cry at a film it must have been good
and this definitely was also
#  to the two little boy's that played the
#  of norman and paul they were just brilliant children are often left out of the
#  list i think because the stars that play them all grown up are such a big profile
for the whole film
# but these children are amazing and should be praised for what they have done don't
you think the whole story was so lovely
# because it was true and was someone's life after all that was shared with us all
```

准备

我们将矢量化每个评论并为缺失项填充零，以便它包含正好一万个数字。这意味着我们用零填充每个比10,000字少的评论。我们这样做是因为神经网络的每个输入都需要具有相同的大小。我们也将目标转化为浮点数。

```
def vectorize(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))



for i, sequence in enumerate(sequences):
    results[i, sequence] = 1
return results

data = vectorize(data)
targets = np.array(targets).astype("float32")
```

现在我们将数据分成训练和测试集。训练集将包含40,000条评论，测试设置为10,000条。

```
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]
```

建立和训练模型

我们现在可以建立我们简单的神经网络。我们首先定义我们要构建的模型的类型。Keras中有两种类型的模型可供使用：Sequential模型和与API函数一起使用的Model类。
https://keras.io/models/sequential/
https://keras.io/models/model/

然后我们只需添加输入层，隐藏层和输出层。在他们之间，我们使用dropout来防止过拟合。请注意，你应始终使用20%到50%之间的dropout率。在每一层，我们使用“Dense”，这意味着单元全连接。在隐藏层中，我们使用ReLU函数，因为这在大多数情况下会产生令人满意的结果。输出层使用sigmoid函数，它将输出值映射到0和1之间。请注意，我们在输入层将输入大小设置为10,000，因为我们的整数长度为10,000个整数。输入层最多需要10,000个输入，输出层大小为50。

最后，让Keras打印我们刚刚构建的模型的概要。

```
# Input - Layer
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
# Hidden - Layers
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu")
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))model.summary()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 50)                500050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================

Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
```

现在我们需要编译我们的模型，也就是为模型训练进行配置。我们使用“adam”优化器。优化器是在训练期间改变权重和偏差的算法。我们也选择二进制 - 交叉熵作为损失（因为我们处理二进制分类）和准确性作为我们的评估指标。

```
model.compile(
 optimizer = "adam",
 loss = "binary_crossentropy",
 metrics = ["accuracy"]
)
```

我们现在可以训练我们的模型。我们设置batch_size为500并且只训练两个epoch，因为我发现如果我们训练的时间更长，这个模型会变好。批处理大小（batch_size）定义了将通过网络传播的样本的数量，而一个epoch是整个训练数据的迭代。通常情况下，batch_size越大，训练速度越快，但并不总是快速收敛。训练中batch_size较小的训练较慢，但收敛速度更快。这肯定是与问题相关的，你需要尝试一些不同的值。如果你第一次遇到问题，我建议你使用批处理大小为32的标准大小。

```
results = model.fit(
 train_x, train_y,
 epochs= 2,
 batch_size = 500,
 validation_data = (test_x, test_y)
)

Train on 40000 samples, validate on 10000 samples
Epoch 1/2
40000/40000 [==============================] - 5s 129us/step - loss:
0.4051 - acc: 0.8212 - val_loss: 0.2635 - val_acc: 0.8945
Epoch 2/2
40000/40000 [==============================] - 4s 90us/step - loss:
0.2122 - acc: 0.9190 - val_loss: 0.2598 - val_acc: 0.8950
```

评估模型：

```
print(np.mean(results.history["val_acc"]))

0.894750000536
```

你可以在下面看到整个模型的代码：

```
import numpy as np
from keras.utils import to_categorical
from keras import models
from keras import layers
from keras.datasets import imdb

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=10000)
data = np.concatenate((training_data, testing_data), axis=0)
```

```
targets = np.concatenate((training_targets, testing_targets), axis=0)


def vectorize(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results


test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]
model = models.Sequential()
# Input - Layer
model.add(layers.Dense(50, activation="relu", input_shape=(10000,)))
# Hidden - Layers
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
# Output- Layer
model.add(layers.Dense(1, activation="sigmoid"))
model.summary()
# compiling the model
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
results = model.fit(
    train_x, train_y,
    epochs=2,
    batch_size=500,
    validation_data=(test_x, test_y)
)
print("Test-Accuracy:", np.mean(results.history["val_acc"]))
```

总结

本文介绍了情感分析的内容，以及为什么Keras是最常用的深度学习库之一。 最重要的是，你了解到Keras对深度学习和人工智能的商品化做出了重大贡献。你学会了如何建立一个简单的六层神经网络，可以预测电影评论的情感，其准确率达到89%。现在，你可以使用此模型对其他文本来源进行二值情感分析，但需要将其全部更改为10,000的长度，或者更改输入层。你也可以将此模型应用于其他相关机器学习问题，只需进行一些更改。

https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/
https://keras.io/datasets/

参考链接：
https://towardsdatascience.com/how-to-build-a-neural-network-with-keras-e8faa33d0ae4

-END-

专·知

人工智能领域主题知识资料查看获取：【专知荟萃】人工智能领域26个主题知识资料全集（入门/进阶/论文/综述/视频/专家等）

请PC登录www.zhuanzhi.ai或者点击阅读原文，注册登录专知，获取更多AI知识资料！

请扫一扫如下二维码关注我们的公众号，获取人工智能的专业知识！

请加专知小助手微信（Rancho_Fang），加入专知主题人工智能群交流！加入专知主题群（请备注主题类型：AI、NLP、CV、 KG等）交流~

点击“阅读原文”，使用专知

阅读原文

投诉