2

TensorFlow 中的数据流编程

Keras 是一个紧凑且易于学习的深度学习高级 Python 库，可以在 TensorFlow（或 Theano 或 CNTK）之上运行。它允许开发人员专注于深度学习的主要概念，如为神经网络创建层，同时照顾到张量、其形状和数学细节的琐碎细节。TensorFlow（或 Theano 或 CNTK）必须是 Keras 的后端。你可以在不与相对复杂的 TensorFlow（或 Theano 或 CNTK）互动的情况下将 Keras 用于深度学习应用。有两种主要的框架：顺序性 API 和功能性 API。顺序 API 是基于层的序列的想法；这是 Keras 最常见的用法，也是 Keras 最简单的部分。顺序模型可以被认为是一个线性的层堆叠。

简而言之，你创建一个序列模型，你可以很容易地添加层，每个层都可以有卷积、最大池化、激活、丢弃和批量规范化。让我们来看看在 Keras 中开发深度学习模型的主要步骤。

2.1. 深度学习模型的主要步骤

Keras 中的深度学习模型的四个核心部分如下:

1. 定义模型：在这个阶段，你创建一个序列模型并添加层。每个层可以包含一个或多个卷积、池化、批量归一化和激活函数。
2. 编译模型：这里你在调用模型的 compile() 函数之前应用损失函数和优化器。
3. 用训练数据拟合模型。在这里，你通过调用模型的 fit() 函数在测试数据上训练模型。
4. 进行预测。在这里，你通过调用 evaluate() 和 predict() 等函数，使用模型在新的数据上生成预测。

Keras 中的深度学习过程有八个步骤。

1. 加载数据。
2. 预处理数据。
3. 定义模型。
4. 编译模型。
5. 拟合模型。

6. 评估模型。
7. 进行预测。
8. 保存模型。

2.1.1. 加载数据

如何加载数据的过程如下：

Listing 2.1: 导入模块

```
# Importing modules
import numpy as np
import os
from keras. datasets import cifar10
from keras. models import Sequential
from keras. layers. core import Dense, Dropout, Activation
from keras.optimizers import adam
from keras.utils import np_utils
```

Listing 2.2: 加载数据

```
#Load Data
np.random. seed(100) # for reproducibility
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
#cifar-10 has images of airplane, automobile, bird, cat,
# deer, dog, frog, horse, ship and truck ( 10 unique Labels)
# For each image. width = 32, height =32, Number of channels (RGB) = 3
```

2.1.2. 预处理数据

以下是你如何预处理数据：

Listing 2.3: 预处理数据

```
#预处理数据
#FLatten the data, MLP doesn't use the 2D structure of the data. 3072 = 3*32*32
X_train = X_train. reshape(50000, 3072) # 50,000 images for training
X_test = X_test. reshape(10000, 3072) # 10,000 images for test
# Gaussian Normalization( Z- score)
X train
(X_train- np.mean(X_train))/np.std(X _train)
X_test = (X_test- np.mean(X_test))/np.std(X_test)
```

Listing 2.4: 将类向量转换为二进制类矩阵 (如 one-hot 向量)

```
# 将类向量转换为二进制类矩阵 (如 one-hot 向量)
labels
10 #10 unique Labels(0-9)
Y train
np_utils. to_categorical(y_train, labels)
Y_test = np_utils. to_categorical(y_test, labels)
```

2.1.3. 定义模型

Keras 中的序列模型被定义为一个层的序列。你创建一个序列模型，然后添加层。你需要确保输入层有正确的输入数量。假设你有 3,072 个输入变量；那么你需要用 512 个节点/神经元创建第一个隐藏层。在第二个隐藏层中，你有 120 个节点/神经元。最后，你在输出层有十个节点。例如，一幅图像映射到十个节点上，显示出成为 label1（airplane）、label2（automobile）、label3（cat）、…、label10（truck）的概率。概率最高的节点是预测的类别/标签。

Listing 2.5: 定义模型的架构

```
# 定义模型的架构
model = Sequential()
model. add(Dense(512, input_shape-(3072,))) # 3*32*32
3072
model.add(Activation('relu'))
model.add(Dropout(0.4)) # Regularization
model. add(Dense(120))
model.add (Activation('relu'))
model.add (Dropout (0.2))# Regularization
model. add (Dense(labels)) #Last Layer with 10 outputs, each output per class
model. add(Activation('sigmoid'))
```

一幅图像有三个通道（RGB），在每个通道中，图像有 32×32=1024 像素。因此，每幅图像有 3×1024=3072 个像素（特征/X/输入）。在 3072 个特征的帮助下，你需要预测标签 1（数字 0）、标签 2（数字 1）的概率，以此类推。这意味着该模型预测了 10 个输出（数字 0-9），每个输出代表相应标签的概率。最后一个激活函数（sigmoid，如前所示）对九个输出给出 0，对一个输出给出 1。这个标签就是图像的预测类别（图 2-1）。例如，3,072 个特征 ⇒ 512 节点 ⇒120 节点 ⇒10 节点。

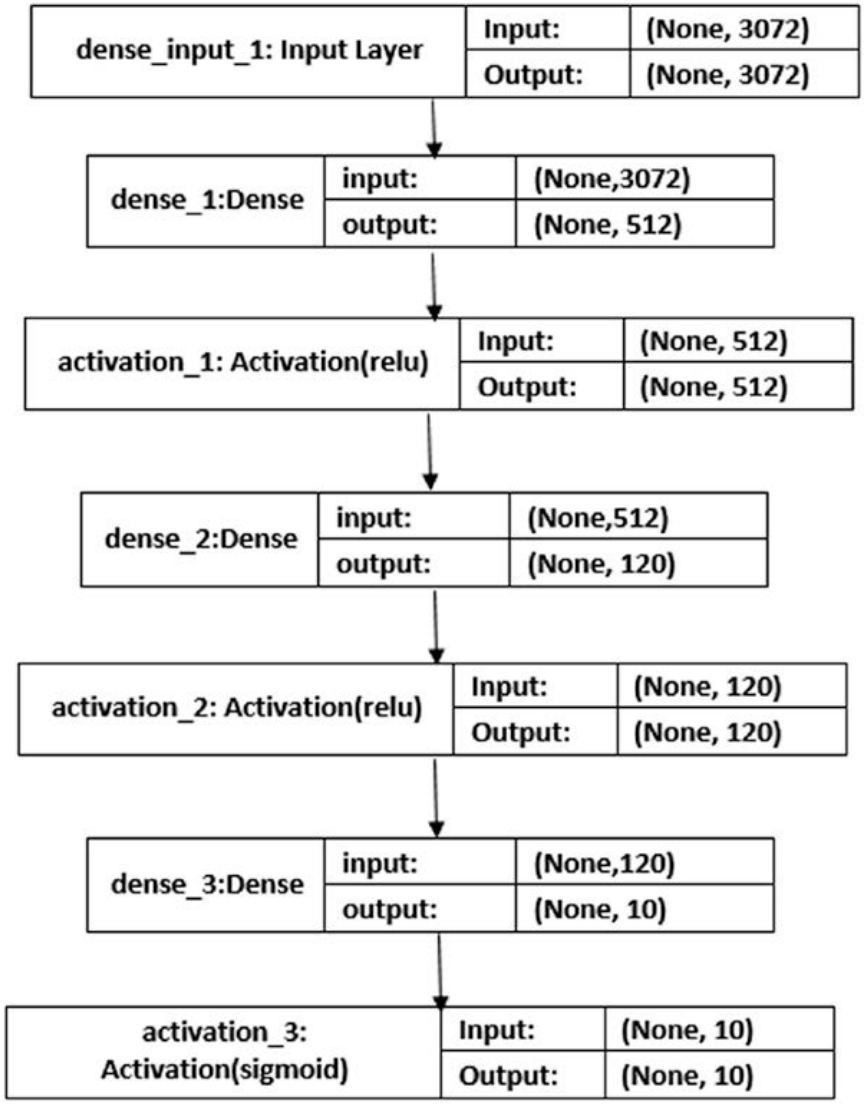

图 2.1: 定义模型

下一个问题是，你如何知道要使用的层数和它们的类型？没有人有确切的答案。对于评估指标来说，最好的办法是由你决定最佳的层数以及每层的参数和步骤。也有采用启发式方法的。最佳的网络结构是通过试错实验的过程找到的。一般来说，你需要一个足够大的网络来捕捉问题的结构。

在这个例子中，你将使用一个有三层的全连接网络结构。一个密集的类定义了全连接层。在这种情况下，你将网络权重初始化为一个从均匀分布（uniform）中生成的小的随机数，在这种情况下是在 0 和 0.05 之间，因为这是 Keras 中默认的均匀权重初始化。另一个传统的选择是正常的从高斯分布生成的小随机数。你使用或扣到默认阈值为 0.5 的任一类别的硬分类。你可以通过添加每一层来拼凑它。

2.1.4. 编译模型

Listing 2.6: 编译模型

```
# Compile the model
# Use adam as an optimizer
adam - adam(0.01)
# the cross entropy between the true Label and the output (softmax) of the model
model.compile(loss-'categorical crossentropy', optimizer-adam, metrics-["accuracy"])
```

流行的损失函数有二元交叉熵、分类交叉熵、平均-平方-对数-误差（mean_squared_logarithmic_error）和铰链损失。流行的优化器有随机梯度下降（SGD）、RMSProp、adam、adagrad 和 adadelta。流行的评估指标是准确性、召回率和 F1 分值。

简而言之，这一步的目的是通过迭代来调整基于损失函数的权重和偏置，基于优化器的评估指标，如准确性。

2.1.5. 适配模型

在定义和编译了模型之后，你需要通过在一些数据上执行模型来进行预测。在这里，你需要指定 epochs；这些是训练过程在数据集中运行的迭代次数，以及批次大小，也就是在权重更新前评估的实例数量。对于这个问题，程序将运行少量的历时（10），在每个历时中，它将完成 50（=50,000/1,000）次迭代，其中批次大小为 1,000，训练数据集有 50000 个实例/图像。同样，没有硬性规定来选择批次大小。但它不应该非常小，而且应该远远小于训练数据集的大小，以减少内存消耗。

```
#Make the model learn ( Fit the model)
model.fit(X_train, Y_train,batch_size=1000, nb_epoch=10,validation_data=(X_test, Y_test))

Train on 50000 samples, validate on 10000 samples
Epoch 1/10
 1000/50000 [.............................] - ETA: 6s - loss: 2.3028 - acc: 0.1060

C:\ProgramData\Anaconda3\lib\site-packages\keras\models.py:848: UserWarning: The `nb_epoch` argument in `fit` has been renamed
`epochs`.
  warnings.warn('The `nb_epoch` argument in `fit` '

50000/50000 [==============================] - 6s - loss: 2.3030 - acc: 0.0974 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 2/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.1012 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 3/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0972 - val_loss: 2.3026 - val_acc: 0.1000
Epoch 4/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0997 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 5/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0975 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 6/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0986 - val_loss: 2.3028 - val_acc: 0.1000
Epoch 7/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0995 - val_loss: 2.3028 - val_acc: 0.1000
Epoch 8/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0983 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 9/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0998 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 10/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0972 - val_loss: 2.3027 - val_acc: 0.1000

<keras.callbacks.History at 0x2870136eef0>
```

图 **2.2:** Fit the Modell

2.1.6. 评估模型

在训练数据集上训练好神经网络后，你需要评估网络的性能。请注意，这只能让你知道你对数据集的建模效果如何（例如，训练准确率），但你不会知道算法在新数据上的表现可能如何。这是为了简单起见，但理想情况下，你可以将你的数据分成训练和测试数据集，用于训练和评估你的模型。你可以在你的训练数据集上使用 evaluation() 函数来评估你的模型，并将用于训练模型的相同输入和输出传递给它。这将为每个输入和输出对生成一个预测，并收集分数，包括平均损失和你配置的任何指标，如准确性。

Listing 2.7: 评估模型在测试集上的表现

```
#Evaluate how the model does on the test set
score = model. evaluate (X_test, Y_test, verbose=0)
#Accuracy Score
print ('Test accuracy:', score[1])
```

2.1.7. 预测

一旦你建立并评估了模型，你需要对未知数据进行预测。

Listing 2.8: 对未知数据进行预测

```
#Predict digit(0-9) for test Data
model.predict_classes(x_test)
```

输出

Listing 2.9: 结果

```
9888/10000 [============================》 .] - ETA: 0S
array([3, 8, 8,3, 4, 71, dtype=int64)
```

2.1.8. 保存并重新加载模型

这里是最后一步。

Listing 2.10: 保存模型

```
%#Saving the model
model.save('model.h5')
jsonModel = model.to_json()
model. save_weights (*modelweight.h5')
```

Listing 2.11: 加载模型

```
%#Load weight of the saved model
modelwt = model. load_weights ('modelweight.h5')
```

2.1.9. 总结模型

现在让我们来看看如何总结模型。

Listing 2.12: 模型的摘要

```
%#Summary of the model
model. summary()
```

```
#Summary of the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               1573376
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 120)               61560
_________________________________________________________________
activation_8 (Activation)    (None, 120)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1210
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 1,636,146
Trainable params: 1,636,146
Non-trainable params: 0
_________________________________________________________________
```

图 **2.3:** Fit the Modell

2.2. 改进 Keras 模型的其他步骤

这里还有一些改进你的模型的步骤。1. 有时，由于梯度消失或爆炸，模型建立过程没有完成。如果是这种情况，你应该采取以下措施。

Listing 2.13: 防止梯度消失或者爆炸

```
1 from keras.callbacks import Earlystopping
2 early\_ stopping monitor = Earlystopping (patience=2)
3 model. fit (x\_train, y\_train, batch\_size-1000, epochs=10,
4 validation data= (x test, y test),
5 callbacks- [early\_stopping\_monitor])
```

2. 对输出形状进行建模。

Shape of the n-dim array (output of the model

at the current position)

model.output_shape

3. 建立总结性表述的模型。

model.summary()

4. 对配置进行建模。

model.get_config()

5. 列出该模型中的所有权重张量。

model.get_weights()

以下是 Keras 模型的完整代码。

用 Keras 建立的深度学习模型

import numpy as np

from keras. models import Sequential
from keras. layers import Dense
读取数据 data = np.random.random((500,100))
labels = np.random.randint(2,size=(500,1))
建立模型
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'*))
model.add(Dense(1, activation='sigmoid'))
编译模型
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
拟合模型
model. fit(X/train], Y|[train], epochs-150, batch_size-10, verbose=0)
评估该模型
scores = model. evaluate(X|test], Y|test], verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
cvscores.append(scores[1] * 100) print("%.2f%% (+/- %.2f%%)" %(numpy.mean(cvscores), numpy.std(cvscores))) 预测
predictions = model.predict(data)

2.3. TensorFlowyu 与 Keras

Keras 通过利用 TensorFlow/Theano 之上的强大而清晰的深度学习库来提供高级神经网络。Keras 是 TensorFlow 的一个重要补充，因为它的层和模型与纯 TensorFlow 的张力兼容。此外，它可以与其他 TensorFlow 库一起使用。

以下是在 TensorFlow 中使用 Keras 的步骤：

1. 首先创建一个 TensorFlow 会话，并将其注册到 Keras。这意味着 Keras 将使用你注册的会话来初始化它内部创建的所有变量。

Listing 2.14: 创建一个 TensorFlow 会话，并将其注册到 Keras

```
import TensorFlow as tf
sess = tf.Session()
from keras import backend as K
K.set_session(sess)
```

2. Keras 模块，如模型、层和激活，被用来建立模型。Keras 引擎会自动将这些模块转换为相当于 TensorFlow 的脚本。

3. 除了 TensorFlow，Theano 和 CNTK 也可以作为 Keras 的后端。

4. TensorFlow 后端有一个惯例，即按照深度、高度、宽度的顺序制作输入形状（给你的网络的第一层），其中深度可以指通道的数量。

5. 你需要正确配置 keras.json 文件，以便它使用 TensorFlow 后端。它应该看起来像这样。

Listing 2.15: 使用 TensorFlow 后端

```
{
```

```
    "backend": "theano",
    "epsilon": 1e-07,
    "image_data_format": "channels_first",
    "floatx": "float32"
}
```