第6章 TensorFlow高级框架

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 人工智能基础与应用 | | 章名 | | | TensorFlow高级框架 | |
| 教学内容 | TensorFlow高级框架 | | | | 课时 | 2 | |
| 项目性质 | □演示性 □验证性 □设计性 √综合性 | | | | | | |
| 授课班级 |  | 授课日期 | |  | | 授课地点 |  |
| 教学目标 | 熟悉TFLearn深度学习库的使用  熟悉Keras人工神经网络库的使用 | | | | | | |
| 教学内容 | 1. TFLearn 2. Keras | | | | | | |
| 教学重点 | TFLearn例程、Keras例程 | | | | | | |
| 教学难点 | TFLearn例程、Keras例程 | | | | | | |
| 教学准备 | 装有Python的计算机  教学课件PPT  教材：《人工智能基础与应用（微课版）》 | | | | | | |
| 作业设计 |  | | | | | | |

教学过程

|  |  |
| --- | --- |
| **教学环节** | **教学内容与过程**  **（教学内容、教学方法、组织形式、教学手段）** |
| **课前组织** | 做好上课前的各项准备工作（打开计算机、打开课件、打开软件、打开授课计划、教案等），吸引学生注意力。 |
| **课程说明** | 【课前说明】  分别从TFLearn、Keras的特点等知识点进行初步的了解。  【目的】  使学生从了解本节课的学习目标、学习重点、考评方式等方面明确课程学习的要求和目标。 |
| **课程内容描述** | 6.1 TFLearn  1．TFLearn介绍  TFLearn是一个基于TensorFlow构建的模块化的、透明的深度学习库，它可以更快、更方便地搭建一个深度的网络。  5.4节介绍了如何使用TensorFlow搭建一个类似于LeNet-5模型的两层卷积神经网络，以完成手写数字识别，本节使用TFLearn搭建一个类似的网络，完成相同的功能。  TFLearn官方网站上描述其特点如下。  （1）可以通过高度模块化的内置神经网络层、优化器等进行快速模型设计，并可以实现正则化操作。  （2）可以训练任何TensorFlow的Graph，支持多个输入、输出和优化器。  （3）图形可视化，图形中包含权重、激活等详细信息。  （4）可以在CPU、GPU等多个设备上部署。  2．TFLearn安装  在交互界面（联网状态）完成TFLearn的安装。  pip install tflearn==0.3.2  进入Python环境，输入“import tflearn”查看是否安装成功。  3．TFLearn例程  在Python目录下新建MetaFramework目录，在MetaFramework下新建TFLearn目录，并将MNIST数据集放到TFLearn目录下的mnist目录下。  【例6-1】 在TFLearn目录下新建CNN\_MNIST.py，在PyCharm中编写代码。  使用TFLearn搭建一个两层的卷积神经网络，数据集是MNIST手写数字的数据集，TFLearn将卷积、池化、正则化等操作都封装成了类，所以需要先导入这些类。  from \_\_future\_\_ import division, print\_function, absolute\_import  import tflearn  from tflearn.layers.core import input\_data, dropout, fully\_connected  from tflearn.layers.conv import conv\_2d, max\_pool\_2d  from tflearn.layers.normalization import local\_response\_normalization  from tflearn.layers.estimator import regression  导入类之后，需要构建一个拥有两个卷积层的神经网络。使用TFLearn的卷积、池化、正则化、全连接、Dropout等操作完成网络构建，TFLearn在卷积的时候，参数包含激活函数，所以不必单独构建激活函数。  # MNIST数据集加载  import tflearn.datasets.mnist as mnist  X, Y, testX, testY = mnist.load\_data(one\_hot=True)  X = X.reshape([-1, 28, 28, 1])  testX = testX.reshape([-1, 28, 28, 1])  # 搭建卷积神经网络，两层卷积  network = input\_data(shape=[None, 28, 28, 1], name='input')  network = conv\_2d(network, 32, 3, activation='relu', regularizer="L2")  network = max\_pool\_2d(network, 2)  network = local\_response\_normalization(network)  network = conv\_2d(network, 64, 3, activation='relu', regularizer="L2")  network = max\_pool\_2d(network, 2)  network = local\_response\_normalization(network)  network = fully\_connected(network, 128, activation='tanh')  network = dropout(network, 0.8)  network = fully\_connected(network, 256, activation='tanh')  network = dropout(network, 0.8)  network = fully\_connected(network, 10, activation='softmax')  network = regression(network, optimizer='adam', learning\_rate=0.01,  loss='categorical\_crossentropy', name='target')  regression()函数中需要规定优化器类型、学习率和损失函数类型。  完成网络构建后，开始训练模型，在训练过程中可以看到损失以及准确率。  # 训练  model = tflearn.DNN(network, tensorboard\_verbose=0)  model.fit({'input': X}, {'target': Y}, n\_epoch=20,  validation\_set=({'input': testX}, {'target': testY}),  snapshot\_step=100, show\_metric=True, run\_id='convnet\_mnist')  使用TFLearn构建神经网络时，由于封装度更高，所以整体的代码非常简洁。  4．TFLearn的API介绍  TFLearn目前支持大多数的深度学习模型，如Convolutions、LSTM、BiRNN、BatchNorm、PReLU、残差网络、生成对抗网络等。  更多API详见TFLearn官网。  6.2 Keras  1．Keras介绍  Keras是一个由Python编写的开源人工神经网络库，可以作为TensorFlow、Microsoft-CNTK和Theano的后端。现在Keras已经被添加到TensorFlow中，成为了TensorFlow的默认框架。相对于TensorFlow，Keras更加适合快速实验和开始一个项目。  Keras官方网站上描述其特点如下。  （1）对用户友好：Keras提供一致而简洁的API，能够极大减少一般应用下用户的工作量。  （2）模块化：网络中的每一个部分，如网络层、损失函数、优化器、初始化策略、激活函数、正则化方法等，都是独立的模块，可以使用它们来构建模型。  （3）易扩展：非常容易添加新模块，只需要仿照现有的模块编写新的类或函数即可。  （4）更易于与Python协作：Keras没有单独的模型配置文件类型，模型由Python代码描述，使Keras创建的模型更紧凑和更容易调试，并且提供了扩展的便利性。注意，Keras兼容的是Python 2.7～3.6版本。  2．Keras安装  在交互界面（联网状态）完成Keras的安装。  pip install keras==2.2.4  进入Python环境，输入“import keras”查看是否安装成功。  3．Keras例程  Keras的GitHub提供了很多实验例程，包括视觉模型、文本和序列、生成模型等。下表列举了Keras部分例程。  Keras部分例程   |  |  |  | | --- | --- | --- | | 类型 | 名称 | 功能 | | 视觉模型 | mnist\_mlp.py | 在MNIST数据集上训练一个简单的多层感知器 | | mnist\_cnn.py | 在MNIST数据集上训练一个简单的convnet | | cifar10\_cnn.py | 在CIFAR-10小图像数据集上训练一个简单的CNN | | cifar10\_resnet.py | 在CIFAR-10小图像数据集上训练ResNet | | conv\_lstm.py | 使用卷积LSTM网络 | | mnist\_acgan.py | 在MNIST数据集上实现AC-GAN（辅助分类器GAN） | | 文本和序列 | addition\_rnn.py | 执行序列学习以添加两个数字（字符串） | | babi\_rnn.py | 在babi数据集上训练一个双分支的循环网络 | | imdb\_bidirectional\_lstm.py | 在IMDB情绪分类任务上训练双向LSTM | | imdb\_cnn.py | 使用Convolution1D进行文本分类 | | imdb\_lstm.py | 在IMDB情绪分类任务上训练LSTM模型 | | 生成模型 | lstm\_text\_generation.py | 从Nietzsche的著作中生成文本 | | neural\_doodle.py | 通过神经网络制作涂鸦画 |   Keras的核心数据结构是一种模型、一种组织层的方式。最简单的是Sequential模型，即线性堆叠层。  Sequential模型的使用过程包括构建模型、编译模型以及训练评估模型。  先进行模型构建。  from keras.layers import Dense  model.add(Dense(units=64, activation='relu', input\_dim=100))  model.add(Dense(units=10, activation='softmax'))  再进行模型编译。  model.compile(loss='categorical\_crossentropy', optimizer='sgd', metrics= ['accuracy'])  最后进行模型训练，假定数据集输入是x\_train和y\_train，并加入评估和预测。  model.fit(x\_train, y\_train, epochs=5, batch\_size=32)  loss\_and\_metrics = model.evaluate(x\_test, y\_test, batch\_size=32)  classes = model.predict(x\_test, batch\_size=32)  【例6-2】 在MetaFramework下新建Keras目录，在Keras目录下新建CNN\_MNIST.py，在PyCharm中编写代码。  搭建一个神经网络需要经过加载数据、模型构建、模型编译、模型训练、模型评估等几个步骤。利用Keras实现一个双层的卷积神经网络，需要先导入类、设置超参数并加载数据。  from \_\_future\_\_ import print\_function  import keras  from keras.datasets import mnist  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten  from keras.layers import Conv2D, MaxPooling2D  from keras import backend as K  batch\_size = 128  num\_classes = 10  epochs = 12  # 输入照片维度  img\_rows, img\_cols = 28, 28  # 加载MNIST数据集进行训练和数据测试  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  接下来判断使用Theano还是使用TensorFlow，它们的参数输入顺序不同。  if K.image\_data\_format() == 'channels\_first':  x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)  x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)  input\_shape = (1, img\_rows, img\_cols)  else:  x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)  x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)  input\_shape = (img\_rows, img\_cols, 1)  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  x\_train /= 255  x\_test /= 255  print('x\_train shape:', x\_train.shape)  print(x\_train.shape[0], 'train samples')  print(x\_test.shape[0], 'test samples')  # 将类向量转换为二进制类矩阵  y\_train = keras.utils.to\_categorical(y\_train, num\_classes)  y\_test = keras.utils.to\_categorical(y\_test, num\_classes)  然后需要构建模型，这里构建一个两层卷积的神经网络，例程中有两个卷积层、一个池化层、两个全连接层。  model = Sequential()  model.add(Conv2D(32, kernel\_size=(3, 3),  activation='relu',  input\_shape=input\_shape))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))  model.add(Dropout(0.25))  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(num\_classes, activation='softmax'))  随后编译模型，采用交叉熵作为损失函数，优化器为keras.optimizers.Adadelta()。  model.compile(loss=keras.losses.categorical\_crossentropy,  optimizer=keras.optimizers.Adadelta(),  metrics=['accuracy'])  接下来进行模型训练，输入训练数据集和测试数据集的数据，还需要输入批次（batch\_size）和训练轮数（epochs），这两个参数在之前已经由全局变量设定完成。  model.fit(x\_train, y\_train,  batch\_size=batch\_size,  epochs=epochs,  verbose=1,  validation\_data=(x\_test, y\_test))  最后进行模型评估，评估模型的损失以及准确率，并打印出来。  score = model.evaluate(x\_test, y\_test, verbose=0)  print('Test loss:', score[0])  print('Test accuracy:', score[1])  除了直接预测外，Keras还可以保存模型。与TensorFlow不同的是，Keras保存模型和权重的文件是HDF5。  4．Keras的API介绍  由于TensorFlow将Keras添加到框架中，所以Keras可以搭建很多网络，具体API详见其API网站。 |
| **总结评价** | 本节课主要学习TFLearn和Keras元框架的使用。 |