《深度学习-keras基础》

大家好，本期我们将开始一个新的专题的写作，因为有一些小伙伴想了解一下深度学习框架Keras的知识，恰好本人也会一点这个知识，因此就开始尝试着写一写吧。本着和大家一起学习的态度，有什么写的不是很好的地方还请大家多多指教。这里我默认大家已经搭建好了深度学习的实践环境了。

**Keras介绍**

关于什么是深度学习，我这里就不多说明了，大家Google就能知道答案。关于深度学习的框架有很多:Tensorflow、Keras、PyTorch、 MXNet、PaddlePaddle等等，那么为什么我这里就开讲Keras呢，因为它简洁好用啊。

**它的简洁在于:**Keras是一个高级深度学习API，使用Python语言进行编写的。Keras能够在TensorFlow、Theano或CNTK上运行，这个意思就是Keras的后端引擎可以是这三者之一，用户可以进行显式的选择。重要的是Keras提供了一个简单和模块化的API来构建和训练我们需要的神经网络，比如卷积神经网络，循环神经网络等等。还有一个优点就是使用Keras可以不用关心大部分函数实现的复杂细节，可真的太棒了。

Keras有四个特性:模块性、易扩展、用户友好和基于Python，以下的介绍来自Keras的中文文档。

* **模块化。**模型被理解为由独立的、完全可配置的模块构成的序列或图。这些模块可以以尽可能少的限制组装在一起。特别是神经网络层、损失函数、优化器、初始化方法、激活函数、正则化方法，它们都是可以结合起来构建新模型的模块。
* **易扩展性。**新的模块是很容易添加的（作为新的类和函数），现有的模块已经提供了充足的示例。由于能够轻松地创建可以提高表现力的新模块，因此其更加适合高级研究。
* **用户友好。**Keras 是为人类而不是为机器设计的 API。它把用户体验放在首要和中心位置。Keras 遵循减少认知困难的最佳实践：它提供一致且简单的 API，将常见用例所需的用户操作数量降至最低，并且在用户错误时提供清晰和可操作的反馈。
* **基于 Python 实现。**Keras 没有特定格式的单独配置文件。模型定义在 Python 代码中，这些代码紧凑，易于调试，并且易于扩展。

温馨提示:如果大家对机器学习不清楚的话，可以先学习机器学习的相关知识，这对于本专题的学习是非常有利的。

**深度学习的“Hello World”**

在跟学习其他语言一样，学习Keras的时候，我们也要学习它的基础知识。深度学习的“Hello World”知识就是使用深度学习来识别手写字识别。在学习之前，我们来看一些其他储备知识：

**一、安装依赖的库和深度学习库**

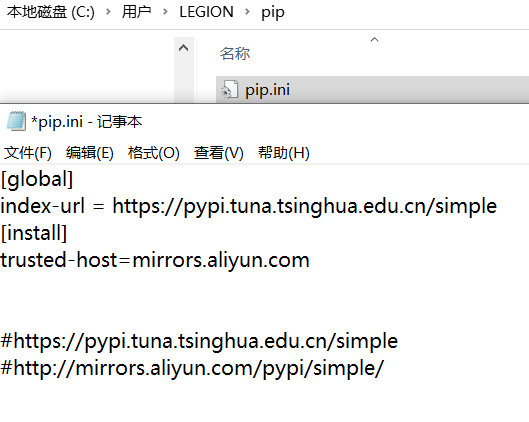
Numpy、scipy、scikit-learn、matplotlib、pandas、graphviz、pydot、h5py、Theano、Tensofflow、Keras。

安装库的方式比较简单，打开cmd，输入pip install packagename即可。

**二、设置下载源**

大家下载库的时候，如果速度很慢，可以修改下载源，本人电脑中文件的

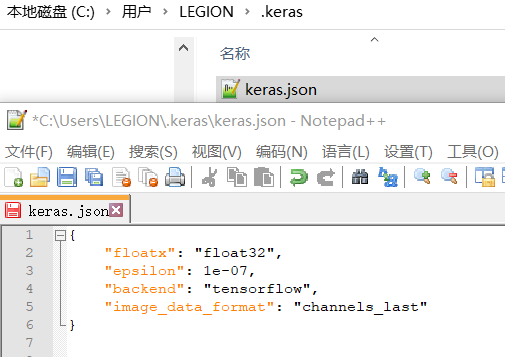
修改路径和修改方式如下,大家根据自己的电脑找到相应的路径：



**三、设置Keras后端**

之前说过，我们可以更换Keras的后端引擎的。本人电脑文件和修改如下：

此时，“backend”:“tensorflow”,可修改tensorflow为Theano. 一般推荐 TensorFlow 后端，大家根据自己的电脑找到相应的路径：



**四、手写字体识别**

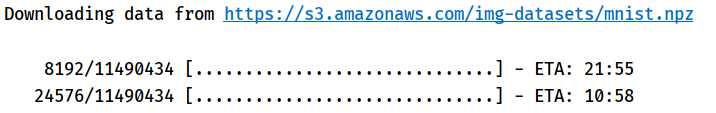
**4.1 加载数据**

深度学习重要的就是数据，那么手写字体的数据是怎样的呢？

我们先看一下这个手写数字体的数据形式：

from keras.datasets import mnist(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()  
print(X\_train.shape)  
print(y\_train.shape)

上述使用mnist.load\_data()加载文件的方式会在从链接：<https://s3.amazonaws.com/img-datasets/mnist.npz>上下载数据集，控制台的部分输出如下：



这个下载的过程是很漫长的，甚至下载不了这个数据集，因此我们将采用其他的方式进行数据下载:

方法一：先从网络上下载好这个数据集，然后传入文件路径：

from keras.datasets import mnist  
path = **r'C:\Users\LEGION\Desktop\datasets\mnist.npz'**(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path)  
print(X\_train.shape)  
print(y\_train.shape)

输出(60000, 28, 28)和(60000,)

方法二：通过numpy来进行搭桥，当然了也要事先下载好数据集：

*# 加载keras包含的mnist的数据集*import numpy as np  
from keras.datasets import mnist  
def load\_data(path):  
 numpy\_load= np.load(path)  
 X\_train, y\_train = numpy\_load[**'x\_train'**], numpy\_load[**'y\_train'**]  
 X\_test, y\_test = numpy\_load[**'x\_test'**], numpy\_load[**'y\_test'**]  
 numpy\_load.close()  
 return (X\_train, y\_train), (X\_test, y\_test)  
  
path = **r'C:\Users\LEGION\Desktop\datasets\mnist.npz'**(X\_train, y\_train), (X\_test, y\_test) = load\_data(path)  
print(X\_train.shape)  
print(y\_train.shape)  
print(X\_test.shape)  
print(y\_test.shape)

X\_train.shape 的形状为(60000, 28, 28)  
y\_train.shape的形状为(60000,)  
X\_test.shape的形状为(10000, 28, 28)  
y\_test.shape的形状为(10000,)

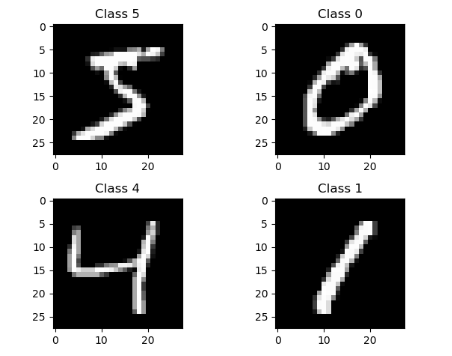
可以看到每条样本是一个28\*28的矩阵(毕竟是图片)，共有训练数据60000个，测试数据10000个，y\_train和y\_test是诸如这样的列表[7 2 1 ... 4 5 6]，其中数字代表每个数据样本的真实值。

**4.2 数据样本展示**

我们可以通过代码来查看一下样本的真实样貌:

from keras.datasets import mnist  
import matplotlib.pyplot as plt  
path = **r'C:\Users\LEGION\Desktop\datasets\mnist.npz'**(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path)  
for each in range(4):  
 plt.subplot(2,2,each+1)  
 plt.imshow(X\_train[each], cmap=plt.get\_cmap(**'gray'**), interpolation=**'none'**)  
 plt.title(**"Class {}"**.format(y\_train[each]))  
plt.show()

输出显示如下：可以看出每一个样本的是gray形式的图片，像素的灰度值在[0,255]



接下来我们的工作就是在训练集(X\_train和y\_train)上训练我们识别手写字体的模型，在测试集上进行测试。

**4.3 重塑维度并归一化**

由于我们的任务是进行识别，实际上也就是进行分类，因此我们有必要对数据进行归一化的操作。另外由于X\_train.shape 的形状为(60000, 28, 28)，也就是说样本是二维形状的数据，在识别的时候是不好处理的，因此我们将数据转换一维，并进行归一化的处理。具体做法如下:

from keras.datasets import mnist  
path = **r'C:\Users\LEGION\Desktop\keras内置datasets\mnist.npz'**(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path)  
X\_train = X\_train.reshape(len(X\_train), -1)  
X\_test = X\_test.reshape(len(X\_test), -1)  
print(X\_train.shape)  
print(X\_test.shape)

输出: (60000, 784)和(10000, 784)

如果大家知道这个样本(图像的大小，如28\*28)，那么饿哦们可以使用另外一个方法进行reshape操作：

X\_train = X\_train.reshape(len(X\_train),784)  
X\_test = X\_test.reshape(len(X\_test), 784)

两者效果是等效的，大家可以查看reshape函数的功能，就能明白这是为什么了。

归一化的操作很简单，我们根据像素值的范围进行归一化，我们先看看数据的类型：

print(X\_train.dtype) *# uint8*

print(X\_test.dtype) *# uint8*

他们是uint8型的，由于要进行归一化，归一化的数据是float32类型的，因此我们将使用astype()进行转换，转换后进行归一化：

X\_train = X\_train.astype(**'float32'**)/255  
X\_test = X\_test.astype(**'float32'**)/255

之前我们说到标签数据(也就是样本的真实类别)y\_train和y\_test是诸如这样的列表[7 2 1 ... 4 5 6]，我们在此需要进行One-hot encoding的操作，不懂One-hot encoding编码的同学可以查阅下资料：

from keras.utils import np\_utils  
y\_train = np\_utils.to\_categorical(y\_train)  
y\_test = np\_utils.to\_categorical(y\_test)

至此我们的数据就处理结束了，接下来我们来搭建神经网络模型并训练。

**4.4 构建模型**

关于这一步骤，我先给出构建模型的代码，然后在进行解释：

from keras.models import Sequential  
from keras.layers import Dense,Activation  
model = Sequential()  
model.add(Dense(512, input\_shape=(28\*28,),activation=**'relu'**))  
model.add(Dense(10,activation=**'softmax'**))  
model.compile(optimizer=**'adam'**, loss=**'categorical\_crossentropy'**, metrics=[**'accuracy'**])  
model.fit(X\_train, y\_train, epochs=10, batch\_size=64, verbose=1, validation\_split=0.05)Testloss, Testaccuracy = model.evaluate(X\_test, y\_test)  
print(**'**Testloss **'**, Testloss)  
print(‘Testaccuracy**:'**, Testaccuracy)

上述代码中：

1. 第三行代码表示使用Keras中的Sequential 顺序模型。
2. 第四行代码表示增加一个全连接层(密集层)，这层有512个神经元，输入形状是(28\*28,)此处的输入形状也可以写为：input\_dim=28\*28。Activation表示这层的激活函数使用是relu，不同的激活函数有不同的效果，大家可以自己学习。
3. 第五行代码增加一个全连接层，这层有10个神经元，这于我们的期望输出个数一致(10个数字)，激活函数使用softmax。
4. 第六行代码表示编译我们之前构建的模型，优化器选择adam,损失函数为categorical\_crossentropy，指标度量选择accuracy精度。
5. 第七行代码表示使用训练集数据来训练我们的模型，epochs表示模型的训练的时期数，每个epoch是对x，y的整个迭代，这里迭代10次。batch\_size代表每个梯度更新的样本数，默认值为32. verbose：日志显示verbose = 0为不在标准输出流输出日志信息，verbose = 1为输出进度条记录，verbose=2为每个epoch输出一行记录，默认为1。validation\_split按一定比例从训练集中取出一部分作为验证集。最后一行控制台输出：loss: 0.0109 - acc: 0.9964 - val\_loss: 0.0862 - val\_acc: 0.9823
6. 第八行代码表示对已经训练好的模型在验证集上进行评价，打印出验证集上的损失和精度为：loss 0.0794 ，Accuracy: 0.9784。

可以看出在测试集上的精度要比训练集上的精度高，这说明我们训练的模型过拟合了，过拟合是指机器学习模型在新数据上的性能往往比在训练数据上要差。

至此，我们就完成了使用Keras完成了深度学习的“Hello World”项目，后期我们将构建卷积神经网络来进一步提升模型的精度。有兴趣的小伙伴可以自己画一个样本，然后进行识别。

构建训练模型的整个代码如下：

from keras.datasets import mnist  
from keras.utils import np\_utils  
from keras.models import Sequential  
from keras.layers import Dense,Activation  
path = **r'C:\Users\LEGION\Desktop\keras内置datasets\mnist.npz'**(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path)  
  
X\_train = X\_train.reshape(len(X\_train),-1)  
X\_test = X\_test.reshape(len(X\_test), -1)  
X\_train = X\_train.astype(**'float32'**)/255  
X\_test = X\_test.astype(**'float32'**)/255  
  
y\_train = np\_utils.to\_categorical(y\_train)  
y\_test = np\_utils.to\_categorical(y\_test)  
  
model = Sequential()  
model.add(Dense(512, input\_shape=(28\*28,),activation=**'relu'**))  
model.add(Dense(10,activation=**'softmax'**))  
model.compile(optimizer=**'adam'**, loss=**'categorical\_crossentropy'**, metrics=[**'accuracy'**])  
model.fit(X\_train, y\_train, epochs=10, batch\_size=64, verbose=1, validation\_split=0.05)  
loss, accuracy = model.evaluate(X\_test, y\_test)  
Testloss, Testaccuracy = model.evaluate(X\_test, y\_test)  
print(**'Testloss:'**, Testloss)  
print(**'Testaccuracy:'**, Testaccuracy)

**总结**

Keras实现一个深度学习的模型还是非常简单的，对于Keras大家是不是有点了解了呢？学习Keras最重要的就是需要多写，多看官方的API。当然了最重要的是大家要有机器学习或者深度学习的一些理论基础。下次推文我们来介绍Keras中的模型的种类，即[Sequential 顺序模型](https://blog.csdn.net/weixin_42886817/article/details/99831718#Sequential__16)和[Model 模型](https://blog.csdn.net/weixin_42886817/article/details/99831718#Model__162)。