第五章 构建多层感知模型进行手写数字图像识别

5.1 场景描述

小明是某快递公司工作人员，每天都要收取大量的快递单，然后将每个快递单上的电话号码录入到系统中，更加痛苦的是，每个人的“手写体”实在是太有个性了，往往在电话号码的数字确认时头疼不已，如图 5-1 所示。小明想，要是有个机器能帮我直接识别这些电话号码并录入系统中就好了。

图 5-1 快递单中的手写数字

5.2 任务描述

创建一个手写数字识别“大脑”，用来对输入的手写数字图像进行识别，如图 5-2 所示。本章将使用一个多层感知器模型来创建一个空白“大脑，然后利用 MNIST 数据集，对空白“大脑”进行训练，使得该“大脑”成为一个手写数字识别“大脑”。同时，我们对手写数字识别“大脑”进行测试，看看手写数字识别“大脑”是否足够聪明。

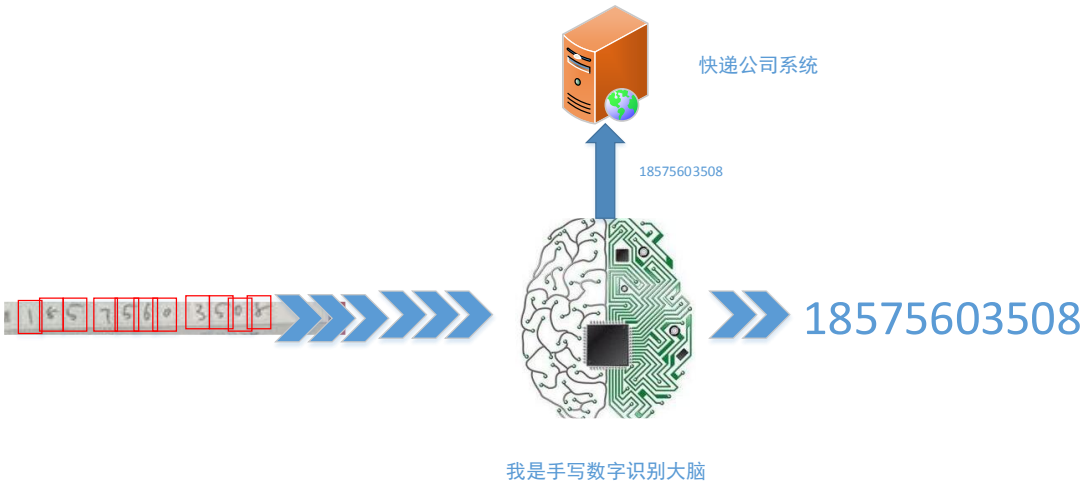


图 5-2 创建智能“大脑”识别手写数字

5.3 任务分解

按照任务要求，我们对手写数字识别大脑的创建过程描述如图 5-3 所示。

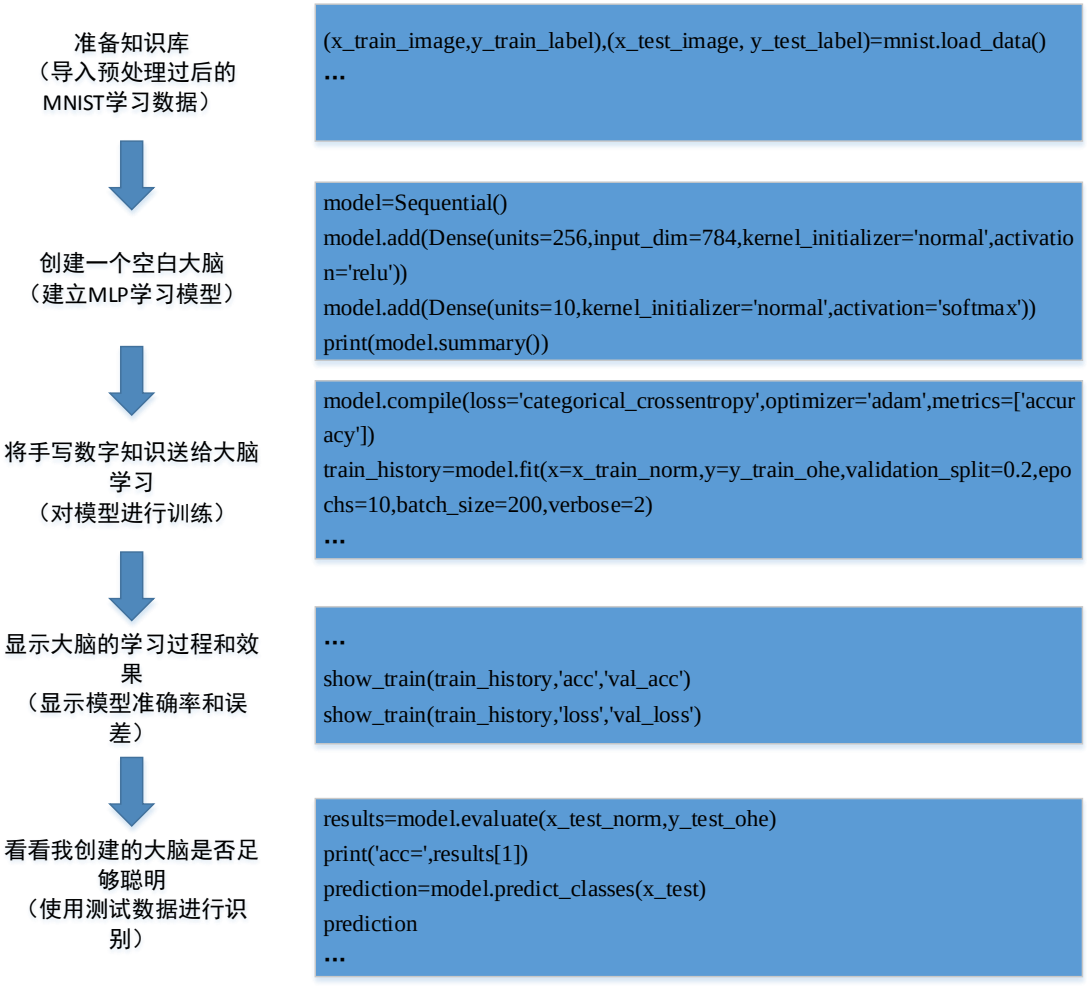


图 5-3 手写数字识别大脑的创建流程图

5.4 知识储备

5.4.1 多层感知模型介绍

深度学习近年来在各个领域都取得了十分巨大的影响力和效果，特别是对于原始未加工，且单独不可解释的特征尤为有效，传统的方法依赖手工选取特征，而神经网络可以进行学习，通过层次结构学习到更利于任务的特征。深度学习的发展得益于近年来互联网充足的数据、计算机硬件的发展以及大规模并行计算的普及。

感知器：感知器接受每个感知元（神经元）传输过来的数据，当数据到达某个阀值的时候，就会产生对应的行为，如图 5-4 所示。每个神经感知元有一个对应的权重，当所有神经感知元加权后超过某个激活函数的阈值时，输出执行对应的行为。如图 5-4 为单个感知器结构图。

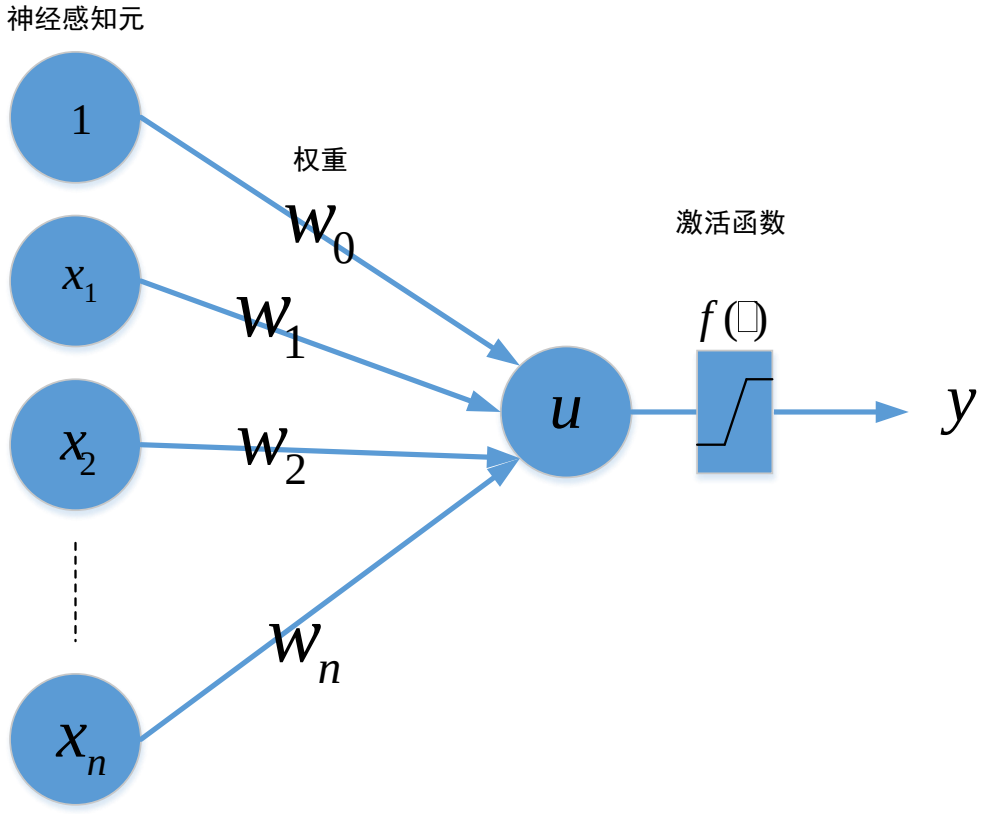


图 5-4 单个神经元感知器结构

$$u = w_0 + x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

$$y = f(u)$$

例如，当我们计算学生成绩是否能获得最高一等奖学金的时候，会按照学生

的每门课的成绩再乘上每门课的权重，最后获得的加权成绩看是否超过 90，如果超过 90，则获得一等奖学金。这个时候，我们可以把 x_n 当做第 n 门课的分数，w_n 当成第 n 门课所占的权重，假设有 5 门课{数学，英语，体育，政治，人工智能开发}，每门课所占权重为{0.2,0.2,0.1,0.1,0.4}，某位学生 5 门课的成绩分别为{90,80,78,90,88}，则最后的加权成绩为：

$$u = 90\times 0.2 + 80\times 0.2 + 78\times 0.1 + 90\times 0.1 + 88\times 0.4 = 86$$

但是，由于 $u = 86 < 90$，我们令：

$$y = f(u) = 0$$

因此，这名学生不能获得一等奖学金。

多层感知器（Multi-Layer Perceptron，MLP）：也叫全连接神经网络模型，网络结构如图 5-5 所示，分为输入层，隐含层与输出层。

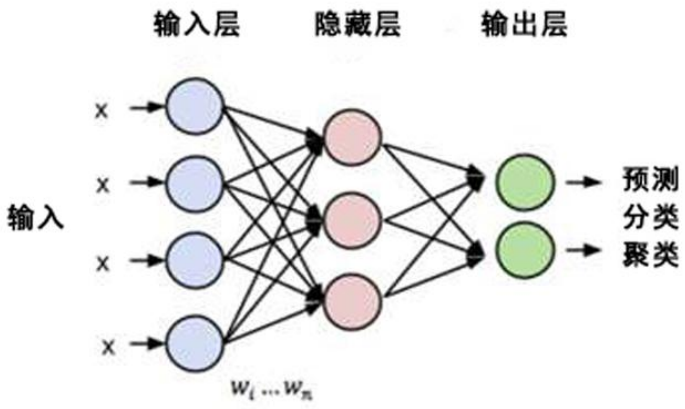


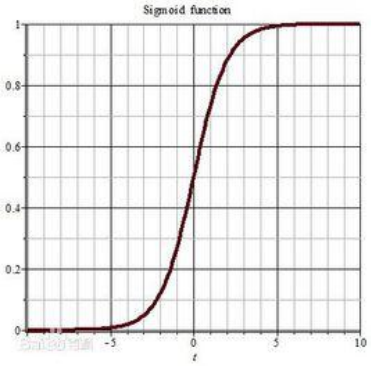
图 5-5 多层感知器模型结构（只含有一层隐藏层）

输入层(input layer)：这一层是神经网络的输入。在这一层，有多少个输入就有多少个神经元。

隐藏层(hidden layers)：隐藏层在输入层和输出层之间，隐藏层的层数是可变的。隐藏层的功能就是把输入映射到输出。神奇的是，如果存在一个函数连接输入和输出，使得 输出=f(输入)，则一个只有一个隐藏层的多层感知器可以完全模仿这个函数。

输出层：这层的神经元的多少取决于我们要解决的问题。例如前面所说的值判断是否获得一等奖学金，则输出只有是或者否，采用一个神经元就可以了。

除了输入层外，其余的每层激活函数均采用 sigmod 函数，如图 5-6 所示。多层感知器就如它的名字一样，由很多个神经元，分成很多层。可以实现更好的性能，但是 MLP 容易受到局部极小值与梯度弥散的困扰。



$$S(x) = \frac{1}{1+e^{-x}}$$

图 5-6 sigmod 函数输入输出曲线

多层感知器与简单的感知器有很多的不同。相同的是它们的权重都是随机的，所有的权重通常都是[-0.5,0.5]之间的随机数。多层感知器有很多应用，统计分析学、模式识别、光学符号识别只是其中的一些应用。

MLP 存在的不足：

（1）网络的隐含节点个数选取问题至今仍是一个世界难题；

（2）停止阈值、学习率、动量常数需要采用 trial-and-error 法（试错法），极其耗时；

（3）学习速度慢；

（4）容易陷入局部极值，学习不够充分。

5.5 任务实现步骤

按照任务流程，我们可以将该任务分解成如下几个子任务，依次完成：

第一步：准备知识库，导入手写数字图像数据集，我们使用已经 MNIST 数据集，并对数据集进行处理以适应 MLP 模型的输入数据格式要求。

第二步：创建空白大脑，建立 MLP 学习模型。使用 keras 模型中的函数来建立 MLP 学习模型，完成输入层、隐藏层与输出层的参数配置。

第三步：将手写数字图像知识送给大脑学习，设置模型的训练参数，启动模型进行训练，并动态查看模型的训练状态。

第四步：通过查看模型训练过程中的准确率和误差变化，了解大脑的学习过程和效果。

第五步：使用训练好的“大脑”模型，对 MNIST 中的测试数据进行预测和识别。

5.5.1 创建 jupyter notebook 项目

（1）打开 jupyter notebook，如图 5-7 所示。

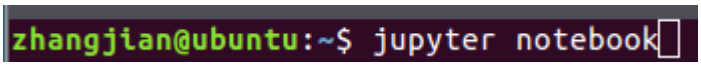


图 5-7 启动 jupyter notebook

（2）在 python3 下新建一个 notebook 项目，命名为 rask5-1，如图 5-8 所示。

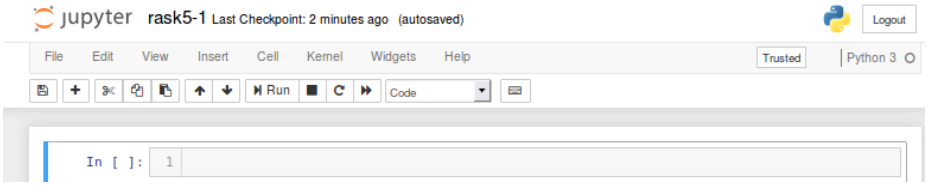


图 5-8 新建 notebook 项目示意图

5.5.2 处理手写数字图像数据集

（1）在 jupyter notebook 中输入图 5-9 中显示的代码，并确认代码无错误。

```python
from keras.utils import np_utils #导入keras模块中的utils函数内容
import numpy as np            #导入numpy模块
#用于指定随机数生成时所用算法开始的整数值，不指定则每次随机数都一样
np.random.seed(10)
from keras.datasets import mnist #导入keras模块中的datasets函数内容
#下载mnist数据集
(x_train_image, y_train_label),(x_test_image, y_test_label)=mnist.load_data()
#将二维图像数据转换成一维向量并改变类型为浮点数类型
x_train=x_train_image.reshape(60000,784).astype('float32')
x_test=x_test_image.reshape(10000,784).astype('float32')
#将一维向量数据归一化使得数据处于[0,1]区间
x_train_norm=x_train/255
x_test_norm=x_test/255
#将测试数据进行one-hot-encode处理
y_train_ohe=np_utils.to_categorical(y_train_label)
y_test_ohe=np_utils.to_categorical(y_test_label)
print('第一张训练照片数据为:')
print(x_train_image[0])
print('第一张训练照片转换成一维向量后数据为:')
print(x_train[0])
print('前3个训练label进行one-hot-encoding转换后的数据为:')
print(y_train_ohe[:3])
```

图 5-9 处理手写数字图像数据集代码

（2）按 Ctrl+Enter 组合键执行代码，代码没有错误提示后按 Shift+Enter 组合键新建下一个单元格，结果显示如图 5-10 所示：

```
/home/zhangjian/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is dep
recated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.

第一张训练照片数据为:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
   ...

第一张训练照片转换成一维向量后数据为:
[   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    3.   18.
   18.   18.  126.  136.  175.   26.  166.  255.  247.  127.    0.    0.    0.    0.
   ...

 前3个训练label进行one-hot-encoding转换后的数据为:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

图 5-10 导入手写数字图像数据集后结果显示图

（3）代码解析可参考第 3 章内容。

5.5.3 建立 MLP 学习模型

（1）在 jupyter notebook 中输入图 5-11 中显示的代码，并确认代码无错误。

```
from keras.models import Sequential #导入keras模块中的models函数内容
from keras.layers import Dense#导入keras模块中的layers函数内容
model=Sequential() # 建立线性堆叠模型
model.add(Dense(units=256,
                input_dim=784,
                kernel_initializer='normal',
                activation='relu'))  #添加隐藏层
model.add(Dense(units=10,
                kernel_initializer='normal',
                activation='softmax')) #添加输出层
print(model.summary())#打印建立的模型内容
```

图 5-11 建立 MLP 学习模型代码

（2）按 Ctrl+Enter 组合键执行代码，结果显示如图 5-12：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
None
```

图 5-12 建立的 MLP 学习模型参数显示

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对 MLP 学习模型中的代码进行逐行解析，对整个模型的创建进行更深入的了解。

- from keras.models import Sequential

导入 keras 框架中的模型库。

- from keras.layers import Dense

导入 keras 框架中全连接层模型。

- model=Sequential()

建立一个贯序学习模型。Keras 有两种类型的模型，序贯模型（Sequential）和函数式模型（Model），函数式模型应用更为广泛，序贯模型是函数式模型的一种特殊情况，是函数式模型的简略版，为最简单的线性、从头到尾的结构顺序，不分叉。贯序模型是线性的层结构（layers），可以任意添加层数，并且需要对模

型中每层进行配置，模型每层中也可以添加其他的模型，如图 5-13 所示。

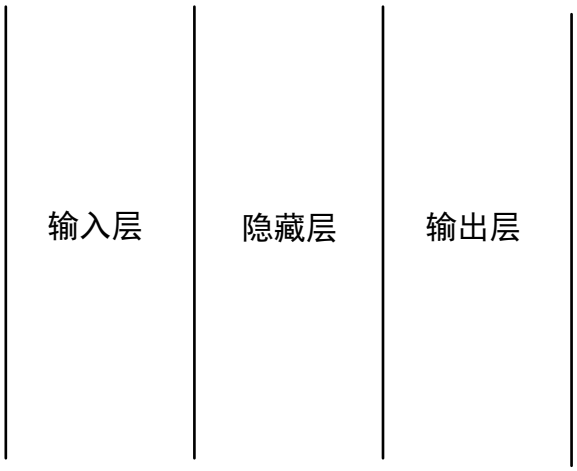


图 5-13 keras 贯序模型中的层结构图

Sequential 模型包含很多组建，但我们常用的基本组件有如下几个：

- model.add( )，在模型中添加层；
- model.compile( ),对模型训练的模式设置；
- model.fit( )，对模型进行训练参数设置并启动训练；
- model.evaluate( )，对模型进行评估；
- model.predict_classes( )，模型进行分类预测；
- model.predict ( )，对分类概率进行预测。

- model.add(Dense(units=256,input_dim=784,kernel_initializer='normal',activation='relu'))

在输入层建立 784 个神经元，隐藏层建立 256 个神经元的全连接结构的层。程序使用 model.add()方法将 Dense 神经网络层结构引入到框架中。Dense 就是常用的全连接层，它将上一层和下一层的神经元都完全相连，如图 5-14 所示。

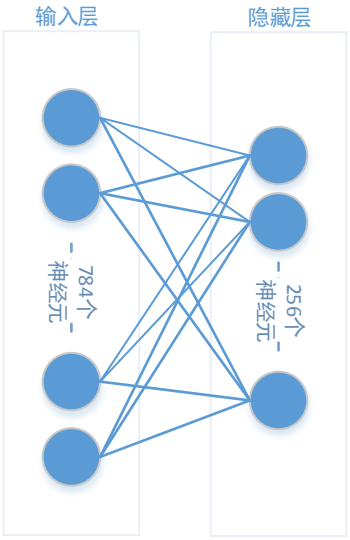


图 5-14 输入层与隐藏层图

Dense()神经网络层的配置函数主要用来对该层结构进行配置，其函数原型为：

keras.layers.core.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)

参数说明：

- units：大于 0 的整数，代表该层的输出维度。
- activation：激活函数，为预定义的激活函数名（参考激活函数），或逐元素（element-wise）的 Theano 函数。如果不指定该参数，将不会使用任何激活函数（即使用线性激活函数：a(x)=x）。
- use_bias: 布尔值，是否使用偏置项。
- kernel_initializer：权值初始化方法，为预定义初始化方法名的字符串，或用于初始化权重的初始化器。
- bias_initializer：偏置向量初始化方法，为预定义初始化方法名的字符串，或用于初始化偏置向量的初始化器。
- kernel_regularizer：施加在权重上的正则项，为 Regularizer 对象。
- bias_regularizer：施加在偏置向量上的正则项，为 Regularizer 对象。
- activity_regularizer：施加在输出上的正则项，为 Regularizer 对象。

- kernel_constraints：施加在权重上的约束项，为 Constraints 对象。
- bias_constraints：施加在偏置上的约束项，为 Constraints 对象。

➢ model.add(Dense(units=10,kernel_initializer='normal',activation='softmax'))

建立输出层，该层有 10 个神经元，对应 0~9 个数字的输出，并使用 softmax 激活函数，使用 softmax 激活函数可以将神经元输出转换为预测每一个数字的概率。由于我们定义的是三层结构，输出层上一层对应的就是隐藏层，keras 会自动将上一个定义的 model.add()作为本层的输入。

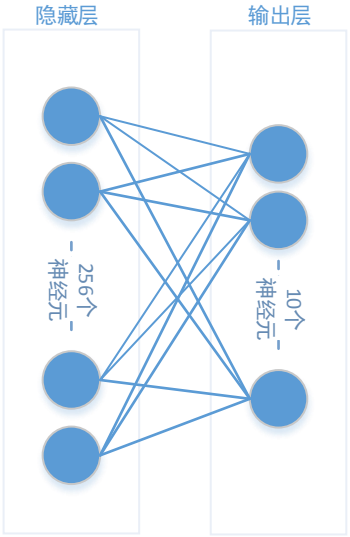


图 5-15 隐藏层与输出层图

➢ print(model.summary())

打印出模型概况，如图 5-12 所示。它实际调用的是 keras.utils.print_summary。dense_1 为隐藏层，有 200960 个参数，因为输入层有 784 个单元，隐藏层有 256 个单元，按照全连接模式，一共需要（784+1）x256=200960 个权重参数进行训练。同理，dense_2 为输出层，按照全连接模式，一共有参数（256+1）x10=2570 个参数。两层参数一共有 203530 个参数需要通过数据集进行训练获得。

5.5.4 对模型进行训练

（1）在 jupyter notebook 中输入图 5-16 中显示的代码，并确认代码无错误。

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy']) #对训练模型进行参数设置
train_history=model.fit(x=x_train_norm,
                        y=y_train_ohe,
                        validation_split=0.2,
                        epochs=10,
                        batch_size=200,
                        verbose=2) #设置训练参数,并启动训练

```

图 5-16 对模型进行训练的代码

（2）按下 Ctrl+Enter 组合键,显示代码的运行结果

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
 - 3s - loss: 0.4491 - acc: 0.8788 - val_loss: 0.2248 - val_acc: 0.9393
Epoch 2/10
 - 2s - loss: 0.1963 - acc: 0.9439 - val_loss: 0.1619 - val_acc: 0.9554
Epoch 3/10
 - 3s - loss: 0.1377 - acc: 0.9612 - val_loss: 0.1286 - val_acc: 0.9629
Epoch 4/10
 - 2s - loss: 0.1038 - acc: 0.9703 - val_loss: 0.1153 - val_acc: 0.9653
Epoch 5/10
 - 2s - loss: 0.0817 - acc: 0.9770 - val_loss: 0.1010 - val_acc: 0.9691
Epoch 6/10
 - 2s - loss: 0.0663 - acc: 0.9816 - val_loss: 0.0988 - val_acc: 0.9693
Epoch 7/10
 - 2s - loss: 0.0543 - acc: 0.9848 - val_loss: 0.0878 - val_acc: 0.9729
Epoch 8/10
 - 2s - loss: 0.0453 - acc: 0.9880 - val_loss: 0.0834 - val_acc: 0.9753
Epoch 9/10
 - 2s - loss: 0.0365 - acc: 0.9906 - val_loss: 0.0814 - val_acc: 0.9742
Epoch 10/10
 - 2s - loss: 0.0305 - acc: 0.9925 - val_loss: 0.0821 - val_acc: 0.9744
```

图 5-17 模型训练过程中的准确率和误差动态数据

（3）按下 Shift+Enter 组合键新建下一个单元格。

（4）下面对模型进行训练的代码进行逐行解析，能对模型的学习设置和学习过程进行更深入的了解。

- model.compile（loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy']）

  调用 model.compile()函数对训练模型进行设置，参数设置为：

  loss='categorical_crossentropy'：loss（损失函数）设置为交叉熵模式，在深度学习中用交叉熵模式训练效果会比较好。

  optimizer='adam'：optimizer（优化器）设置为 adam，在深度学习中可以让训练更快收敛，并提高准确率。

  metrics=['accuracy']：评估模式设置为准确度评估模式。

- **模型配置函数** compile()用来对模型进行训练模式的配置，其函数原型

为：

compile(self, optimizer, loss, metrics=None, sample_weight_mode=None)，其中各参数分别为：

- loss：字符串（预定义损失函数名）或目标函数，参考损失函数
- optimizer：字符串（预定义优化器名）或优化器对象，参考优化器
- metrics：列表，包含评估模型在训练和测试时的网络性能的指标，典型用法是 metrics=['accuracy']
- sample_weight_mode：如果你需要按时间步为样本赋权（2D 权矩阵），将该值设为“temporal”。默认为“None”，代表按样本赋权（1D 权）。在下面 fit 函数的解释中有相关的参考内容。

loss（目标函数，或称损失函数）是编译一个模型必须的两个参数之一，可以通过传递预定义目标函数名字指定目标函数，也可以传递一个 Theano/TensroFlow 的符号函数作为目标函数。表 5-1 是 loss 参数常用的损失函数。

表 5-1 loss 参数常用的损失函数

目标函数名	含义
binary_crossentropy	亦称作对数损失，logloss
categorical_crossentropy	亦称作多类的对数损失，注意使用该目标函数时，需要将标签转化为形如(nb_samples, nb_classes)的二值序列
sparse_categorical_crossentrop	如上，但接受稀疏标签。注意，使用该函数时仍然需要你的标签与输出值的维度相同，你可能需要在标签数据上增加一个维度：np.expand_dims(y,-1)
kullback_leibler_divergence	从预测值概率分布 Q 到真值概率分布 P 的信息增益,用以度量两个分布的差异.
poisson	即(predictions - targets * log(predictions))的均值
cosine_proximity	即预测值与真实标签的余弦距离平均值的相反数

注意：当使用"categorical_crossentropy"作为目标函数时，标签应该为多类模式，即 one-hot 编码的向量，而不是单个数值。可以使用工具中的 to_categorical 函数完成该转换。

optimizer（优化器）是编译 Keras 模型必要的两个参数之一。可以在调用 model.compile()之前初始化一个优化器对象，然后传入该函数，如下所示：

```
#自定义一个优化器对象
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#配置模型时使用自定义的优化器对象
model.compile(loss='mean_squared_error', optimizer=sgd)
```

也可以在调用 model.compile()时传递一个预定义优化器名。表 5-2 是系统提供的常见预定义优化器类型。

表 5-2 优化器类型表

SGD	随机梯度下降法，支持动量参数，支持学习衰减率，支持 Nesterov 动量 keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
RMSprop	该优化器通常是面对递归神经网络时的一个良好选择 keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-06)
Adagrad	keras.optimizers.Adagrad(lr=0.01, epsilon=1e-06)
Adadelta	keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=1e-06)
Adam	keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
Adamax	Adamax 优化器来自于 Adam 的论文的 Section7，该方法是基于无穷范数的 Adam 方法的变体。 keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
Nadam	Nesterov Adam optimizer: Adam 本质上像是带有动量项的 RMSprop，Nadam 就是带有 Nesterov 动量的 Adam RMSprop 默认参数来自于论文，推荐不要对默认参数进行更改。 keras.optimizers.Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
TFOptimizer	TF 优化器的包装器 keras.optimizers.TFOptimizer(optimizer)

metrics（性能评估）性能评估模块提供了一系列用于模型性能评估的函数，这些函数在模型编译时由 metrics 关键字设置，性能评估函数类似于目标函数，只不过该性能的评估结果将不会用于训练。可以通过字符串来使用域定义的性能评估函数，也可以自定义一个 Theano/TensorFlow 函数并使用之。表 5-3 是常见的系统提供的性能评估函数。

表 5-3 系统提供的性能评估函数表

binary_accuracy	对二分类问题,计算在所有预测值上的平均正确率
categorical_accuracy	对多分类问题,计算在所有预测值上的平均正确率
sparse_categorical_accuracy	与 categorical_accuracy 相同,在对稀疏的目标值预测时有用
top_k_categorical_accracy	计算 top-k 正确率,当预测值的前 k 个值中存在目标类别即认为预测正确
sparse_top_k_categorical_accuracy	与 top_k_categorical_accracy 作用相同，但适用于稀疏情况

- train_history=model.fit(x=x_train_normalize,y=y_label_ohe,validation_split=0.2, epochs=10,batch_size=200,verbose=2)

调用 model.fit 配置训练参数，开始训练，并保存训练结果。

x=x_train_normalize：MNIST 数据集中已经经过预处理的训练集图像

y=y_label_ohe：MNIST 数据集中已经经过预处理的训练集 label

validation_split=0.2：训练之前将输入的训练数据集中 80%作为训练数据，20%作为测试数据。

epochs=10：设置训练周期为 10 次。

batch_size=200：设置每一次训练周期中，训练数据每次输入多少个。

verbose=2：设置成显示训练过程。

- **启动训练函数 model.fit()**用来对模型进行训练参数的配置以及启动训练模型，model.fit()的函数原型为：

fit(self, x, y, batch_size=32, epochs=10, verbose=1, callbacks=**None**,

validation_split=0.0, validation_data=**None**, shuffle=**True**,

class_weight=**None**, sample_weight=**None**, initial_epoch=0)，

fit 函数返回一个 History 的对象，其 History.history 属性记录了损失函数和其他指标的数值随 epoch 变化的情况，如果有验证集的话，也包含了验证集的这些指标变化情况。其输入参数如下：

- x：输入数据。如果模型只有一个输入，那么 x 的类型是 numpy array，如果模型有多个输入，那么 x 的类型应当为 list，list 的元素是对应于各个输入的 numpy array
- y：标签，numpy array
- batch_size：整数，指定进行梯度下降时每个 batch 包含的样本数。训练时一个 batch 的样本会被计算一次梯度下降，使目标函数优化一步。
- epochs：整数，训练终止时的 epoch 值，训练将在达到该 epoch 值时停止，当没有设置 initial_epoch 时，它就是训练的总轮数，否则训练的总轮数为 epochs - inital_epoch
- verbose：日志显示，0 为不在标准输出流输出日志信息，1 为输出进度条记录，2 为每个 epoch 输出一行记录
- callbacks：list，其中的元素是 keras.callbacks.Callback 的对象。这个 list 中的回调函数将会在训练过程中的适当时机被调用。
- validation_split：0~1 之间的浮点数，用来指定训练集的一定比例数据作为验证集。验证集将不参与训练，并在每个 epoch 结束后测试的模型的指标，如损失函数、精确度等。注意，validation_split 的划分在 shuffle 之前，因此如果你的数据本身是有序的，需要先手工打乱再指定 validation_split，否则可能会出现验证集样本不均匀。
- validation_data：形式为（x，y）的 tuple，是指定的验证集。此参数将覆盖 validation_spilt。
- shuffle：布尔值或字符串，一般为布尔值，表示是否在训练过程中随机打乱输入样本的顺序。若为字符串“batch”，则是用来处理 HDF5 数据的特殊情况，它将在 batch 内部将数据打乱。
- class_weight：字典，将不同的类别映射为不同的权值，该参数用来在训练过

程中调整损失函数（只能用于训练）

- sample_weight：权值的 numpy array，用于在训练时调整损失函数（仅用于训练）。可以传递一个 1D 的与样本等长的向量用于对样本进行 1 对 1 的加权，或者在面对时序数据时，传递一个的形式为（samples，sequence_length）的矩阵来为每个时间步上的样本赋不同的权。这种情况下请确定在编译模型时添加了 sample_weight_mode='temporal'。
- initial_epoch: 从该参数指定的 epoch 开始训练，在继续之前的训练时有用。

5.5.5 显示模型准确率与误差

（1）在 jupyter notebook 中输入图 5-18 中显示的代码，并确认代码无错误。

```
import matplotlib.pyplot as plt #导入matplotlib模块中的pyplot函数进行图形绘制
def  show_train(train_histroy,train,validation):  #定义训练准确率和误差绘制函数
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.show()
#绘制训练数据准确率变化曲线和测试数据准确率变化曲线对比图
show_train(train_history,'acc','val_acc')
#绘制训练数据误差变化曲线和测试数据误差变化曲线对比图
show_train(train_history,'loss','val_loss')
```

图 5-18 显示模型准确率与误差代码

（2)按下 Ctrl+Enter 组合键, 显示代码的运行结果如图 5-19，并按下 Shift+Enter 组合键新建下一个单元格。

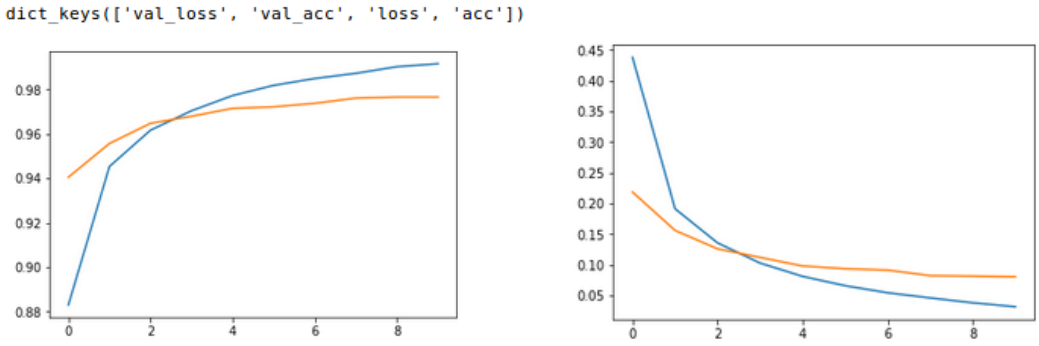


图 5-19 训练数据和测试数据准确率和误差变化曲线对比图

（3）下面对显示模型准确率和误差代码进行详细解析，以便对模型的有更深入的了解。

- import matplotlib.pyplot as plt

导入绘图库。

- def show_train(train_histroy,train,validation):

  定义绘图函数 show_train, 输入参数为,

train_history：训练的历史记录对象，Histroy 数据类型

train：要显示的第一个参数

validation：要显示的第二个参数

- plt.plot(train_history.history[train])
- plt.plot(train_history.history[validation])
- plt.show()

分别画出两个参数的折现图，并进行显示。

- show_train(train_history,'acc','val_acc')

画出准确率和验证准确率的变化折线图，如图 5-19 所示。

- show_train(train_history,'loss','val_loss')

画出误差和验证误差的变化折线图，如图 5-19 所示。

5.5.6 利用测试数据进行预测评估与识别

（1）在 jupyter notebook 中输入图 5-18 中显示的代码，并确认代码无错误。

```
results=model.evaluate(x_test_norm,y_test_ohe)#利用测试数据对模型进行评估
print('acc=',results[1]) #打印测试数据进行评估的准确率
prediction=model.predict_classes(x_test) #使用训练好的模型对测试数据进行分类
prediction[:10] #打印前10个测试数据预测分类的结果
```

图 5-20 利用测试数据进行预测评估与识别代码

（2）按下 Ctrl+Enter 组合键, 显示代码的运行结果，按下 Shift+Enter 组合键新建下一个单元格。

```
10000/10000 [==============================] - 1s 58us/step
acc= 0.9808

array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9])
```

图 5-21 测试数据进行预测的结果显示

（3）下面对使用测试数据进行预测的代码进行详细的解析，以便掌握使用模型进行预测的方法。

- results=model.evaluate(x_test_norm,y_test_ohe)

x_test_norm：输入数据位预处理后的测试数据集

y_test_ohe：标签为预处理后的测试标签集

模型误差估计函数 model.evaluate()按 batch 计算在某些输入数据上模型的误差，其函数原型为：

evaluate(self, x, y, batch_size=32, verbose=1, sample_weight=None)，其中

- x：输入数据，与 fit 一样，是 numpy array 或 numpy array 的 list

- y：标签，numpy array
- batch_size：整数，含义同 fit 的同名参数
- verbose：含义同 fit 的同名参数，但只能取 0 或 1
- sample_weight：numpy array，含义同 fit 的同名参数

本函数返回一个测试误差的标量值（如果模型没有其他评价指标），或一个标量的 list（如果模型还有其他的评价指标）。model.metrics_names 将给出 list 中各个值的含义。

- print('acc=',results[1])

打印返回测试结果中第二个参数的值。

- prediction=model.predict_classes(x_test)

对测试数据集进行预测，测试数据集是未经过归一化处理的数据。测试结果返回到 prediciton 变量中。

模型预测函数 model.predict_classes ()用来对测试数据进行分类预测。其函数原型为：predict_classes(X，batch_size=128，verbose=1)，函数返回测试数据的类预测数组，return：测试数据的标签数组。

- prediction[:10]

显示预测结果的前 10 项，由图 5-21 中可以看到，预测结果保存为一维数组，显示预测到的对应图像的数字。

5.6 项目代码完整示例

```python
### rask5-1.py
#使用MNIST数据集
#采用MLP模型进行训练
#
#**************** 第一步：准备知识库，导入手写数字图像数据集**********************
from keras.utils import np_utils #导入keras模块中的utils函数内容
import numpy as np          #导入numpy模块
#用于指定随机数生成时所用算法开始的整数值，不指定则每次随机数都一样
np.random.seed(10)
from keras.datasets import mnist #导入keras模块中的datasets函数内容
#下载mnist数据集
(x_train_image, y_train_label),(x_test_image, y_test_label)=mnist.load_data()
#将二维图像数据转换成一维向量并改变类型为浮点数类型
x_train=x_train_image.reshape(60000,784).astype('float32')
x_test=x_test_image.reshape(10000,784).astype('float32')
#将一维向量数据归一化使得数据处于[0,1]区间
x_train_norm=x_train/255
x_test_norm=x_test/255
#将测试数据进行one-hot-encode处理
y_train_ohe=np_utils.to_categorical(y_train_label)
y_test_ohe=np_utils.to_categorical(y_test_label)
print('第一张训练照片数据为：')
print(x_train_image[0])
print('第一张训练照片转换成一维向量后数据为：')
print(x_train[0])
print('前3个训练label进行one-hot-encoding转换后的数据为：')
print(y_train_ohe[:3])
```

```python

#*************** 第二步：创建空白大脑，建立MLP学习模型**********************
from keras.models import Sequential #导入keras模块中的models函数内容
from keras.layers import Dense#导入keras模块中的layers函数内容
model=Sequential() # 建立线性堆叠模型
model.add(Dense(units=256,input_dim=784,kernel_initializer='normal',activation='relu'))  #添加隐藏层
model.add(Dense(units=10,kernel_initializer='normal',activation='softmax')) #添加输出层
print(model.summary())#打印建立的模型内容
```

```python

#*************** 第三步：将手写数字图像知识送给大脑学习，对模型进行训练**********************
model.compile(loss='categorical_crossentropy',/
optimizer='adam',metrics=['accuracy']) #对训练模型进行参数设置
train_history=model.fit(x=x_train_normalize,/
    y=y_label_onddd,validation_split=0.2,/
    epochs=10,batch_size=200,verbose=2) #设置训练参数，并启动训练
```

```python

#*************** 第四步：显示大脑的学习过程和效果，显示模型训练过程中的准确率和误差变化***
import matplotlib.pyplot as plt #导入matplotlib模块中的pyplot函数进行图形绘制
def  show_train(train_histroy,train,validation):  #定义训练准确率和误差绘制函数
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.show()
#绘制训练数据准确率变化曲线和测试数据准确率变化曲线对比图
show_train(train_history,'acc','val_acc')
#绘制训练数据误差变化曲线和测试数据误差变化曲线对比图
show_train(train_history,'loss','val_loss')
```

```python

#*************** 第五步：看看我创建的大脑是否足够聪明，利用模型进行预测和识别**********
results=model.evaluate(x_test_norm,y_test_ohe)#利用测试数据对模型进行评估
print('acc=',results[1]) #打印测试数据进行评估的准确率
prediction=model.predict_classes(x_test) #使用训练好的模型对测试数据进行分类
prediction[:10] #打印前10个测试数据预测分类的结果
```

5.7 小结与应用

本章主要介绍了如何构建一个 MLP 多层感知模型，利用手写数字图像数据集对模型进行训练，使得该模型能够识别手写数字。随着信息网络的推广，有大量的数据要输入计算机网络。而且在现代信息社会，方方面面都要与数字打交道。目前手写数字识别主要的应用有以下三个领域。

（1）在邮件分拣中的应用

邮件的自动分拣中，脱机手写数字识别往往与人工辅助识别等手段相结合，完成邮政编码的阅读，然而在一些大城市的中心邮局每天处理高达几百万件，业务量的急剧上升使得邮件的分拣自动化成为大势所趋。

（2）在财税、金融领域中的应用

金融财务、税务、金融是脱机手写数字识别应用的又一重要领域。随着我国经济的迅速发展，每天等待处理的财务、税务报表、支票、付款单等越来越多，如果能把他们用计算机自动处理，无疑可以节约大量的时间、金钱和劳力，更可以提高效率。

（3）在大规模数据统计中的应用

手写数字识别在数据统计、行业年检、人口普查等领域都要进行大规模的数据统计，此时就需要输入大量的数据，以前完全要手工输入需要耗费大量的人力和物力，如果能把他们用计算机自动处理，无疑可以节约大量的时间、金钱和劳力，更可以提高效率。

习题

1. 简述多层感知模型的优缺点？

2. 在本章训练模型中，改变模型的隐藏层神经元个数、模型的损失函数、优化器等，记录测试数据进行预测的准确率。

测试序号	隐藏层神经元个数	模型的损失函数	优化器	测试数据准确率评估
1				
2				
3				
4				
5				

3.在本章中对 10000 个测试数据进行分类预测后，找出预测匹配对的次数。

预测值	0	1	2	3	4	5	6	7	8	9
Label 值										
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										