神经网络实验

内容提要

- 神经网络中的分类模型
- 神经网络中的回归模型
- 卷积神经网络

神经网络中的分类模型

- from sklearn.neural_network import MLPClassifier

- *class* sklearn.neural_network.MLPClassifier(*hidden_layer_sizes=(100,), activation='relu', solver='adam',alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,n_iter_no_change=10*)

神经网络中的分类模型

- 常用参数说明

参数名	说明
activation	激活函数（identity,logistic,tanh,relu）,默认为relu
alpha	正则化程度，L2正则化，默认为0.0001
hidden_layer_sizes	隐藏层的规模，默认是1个隐藏层100个节点
solver	权重优化的求解器（lbfgs,sgd,adam），默认为*adam*

神经网络中的分类模型

- activation参数

identity:对特征不做处理，返回值是f(x)=x

logistic:返回f(x)=1/[1+exp(-x)]

tanh:双曲正切处理，返回f(x)=tanh(x)

relu:线性整流函数又称修正线性单元，返回 f(x) = max(0, x)

神经网络中的分类模型

- solver参数

lbfgs:是准牛顿方法族的优化器

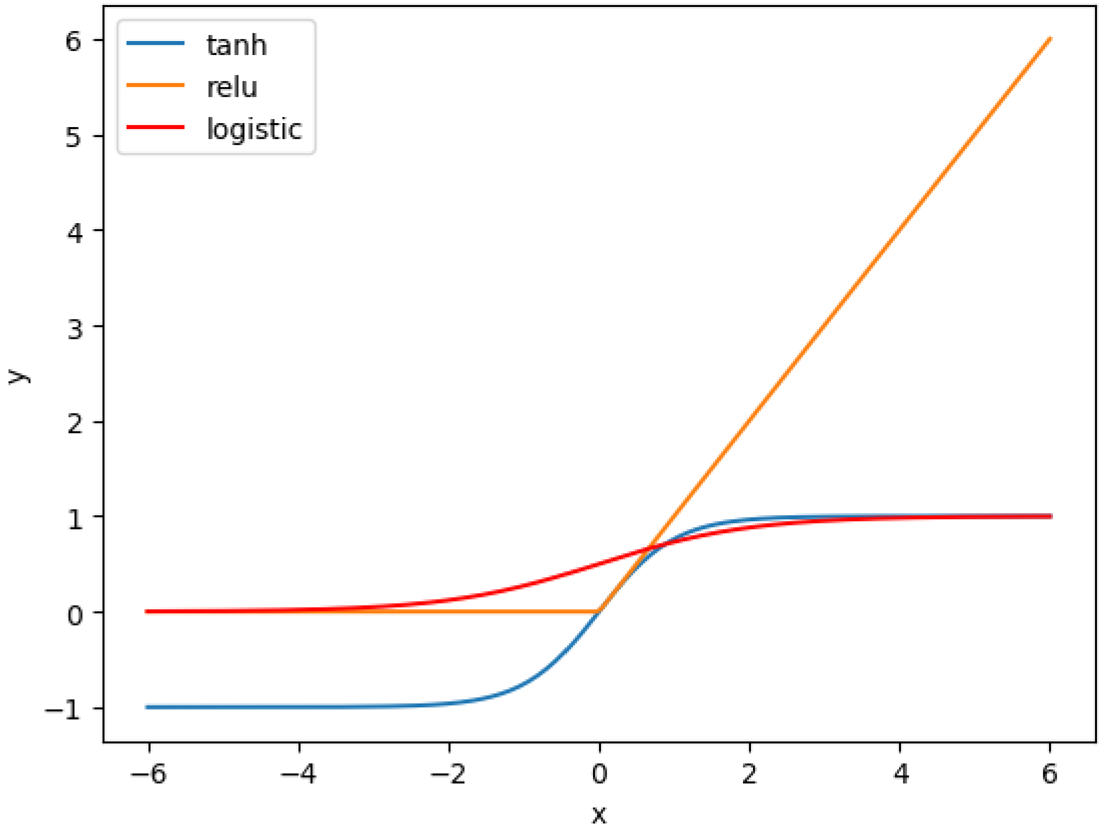
sgd:随机梯度下降

adma：基于随机梯度的优化器

神经网络中的分类模型

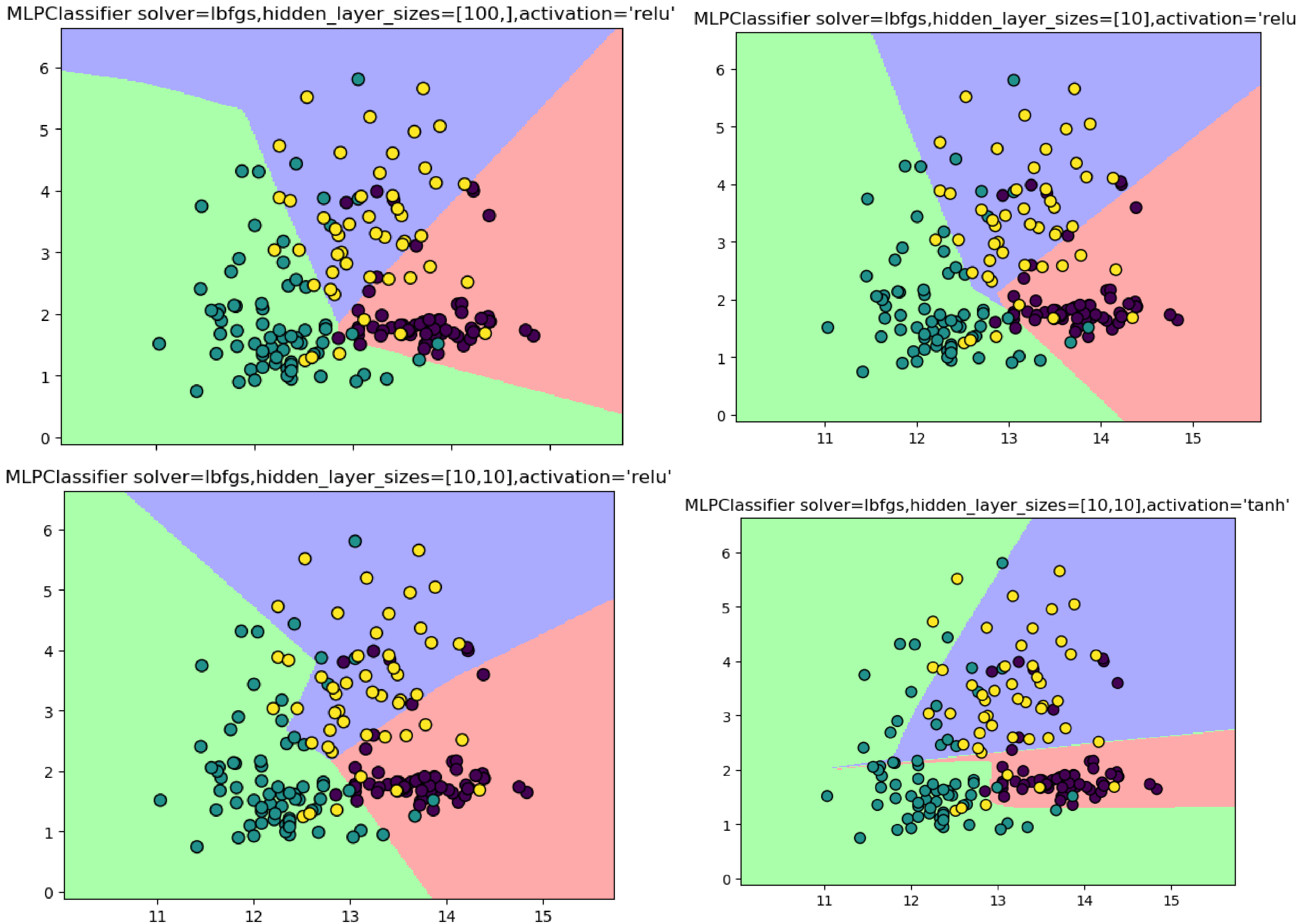
- import numpy as np
- import matplotlib.pyplot as plt
- line=np.linspace(-6,6,200)
- #tanh
- plt.plot(line,np.tanh(line),label='tanh')
- #relu
- plt.plot(line,np.maximum(line,0),label='relu')
- #logistic
- plt.plot(line,1/(1+np.exp(-line)),label='logistic')
- plt.legend(loc='best')
- plt.xlabel('x')
- plt.ylabel('y')
- plt.show()

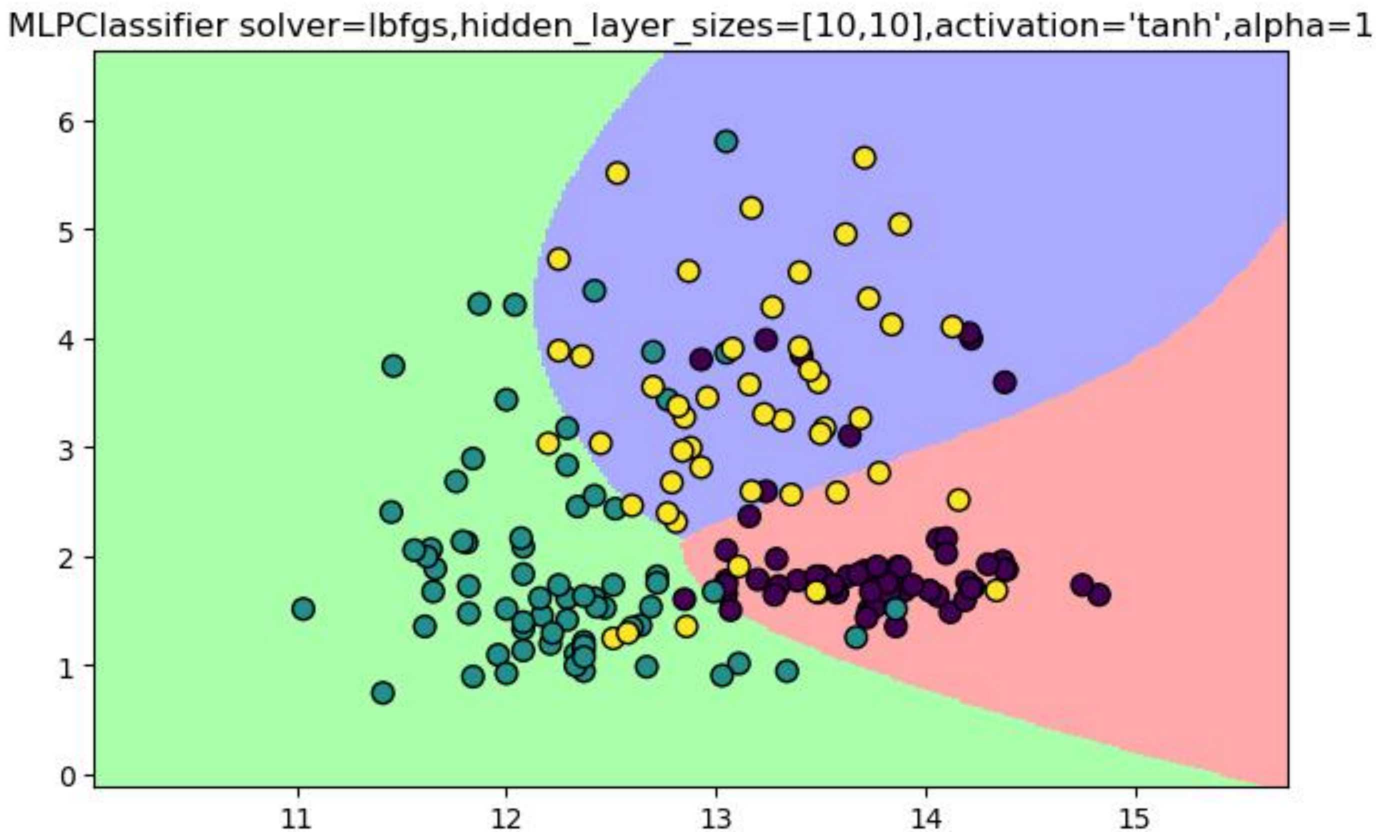
神经网络中的分类模型



神经网络中的分类模型

- from sklearn.neural_network import MLPClassifier
- from sklearn.datasets import load_wine
- from sklearn.model_selection import train_test_split
- import matplotlib.pyplot as plt
- from matplotlib.colors import ListedColormap
- import numpy as np
- wine=load_wine()
- x=wine.data[:,:2]
- y=wine.target
- x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)
- mlp=MLPClassifier(solver='lbfgs', random_state=0)
- mlp.fit(x_train,y_train)
- print("train_score=",mlp.score(x_train,y_train))
- #可视化
- cmap_light=ListedColormap(['#FFAAAA','#AAFFAA','#AAAAFF'])
- cmap_bold=ListedColormap(['#FF0000','00FF00','#0000FF'])
- x_min,x_max=x_train[:,0].min()-1,x_train[:,0].max()+1
- y_min,y_max=x_train[:,1].min()-1,x_train[:,1].max()+1
- xx,yy=np.meshgrid(np.arange(x_min,x_max,.02),np.arange(y_min,y_max,.02))
- z=mlp.predict(np.c_[xx.ravel(),yy.ravel()])
- z=z.reshape(xx.shape)
- plt.figure()
- plt.pcolormesh(xx,yy,z,cmap=cmap_light)
- plt.scatter(x[:,0],x[:,1],c=y,edgecolor='k',s=60)
- plt.xlim(xx.min(),xx.max())
- plt.ylim(yy.min(),yy.max())
- plt.title("MLPClassifier")
- plt.show()


MLPClassifier solver=lbfgs,hidden_layer_sizes=[100,],activation='relu'
MLPClassifier solver=lbfgs,hidden_layer_sizes=[10],activation='relu'
MLPClassifier solver=lbfgs,hidden_layer_sizes=[10,10],activation='relu'
MLPClassifier solver=lbfgs,hidden_layer_sizes=[10,10],activation='tanh'


MLPClassifier solver=lbfgs,hidden_layer_sizes=[10,10],activation='tanh',alpha=1
0
1
2
3
4
5
6
11
12
13
14
15

神经网络中的回归模型

- from sklearn.neural_network import MLPRegressor

- *class* sklearn.neural_network.MLPRegressor(*hidden_layer_sizes=(100,), activation=’relu’, solver=’adam’, alpha=0.0001,batch_size=’auto’, learning_rate=’constant’, learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,n_iter_no_change=10*)

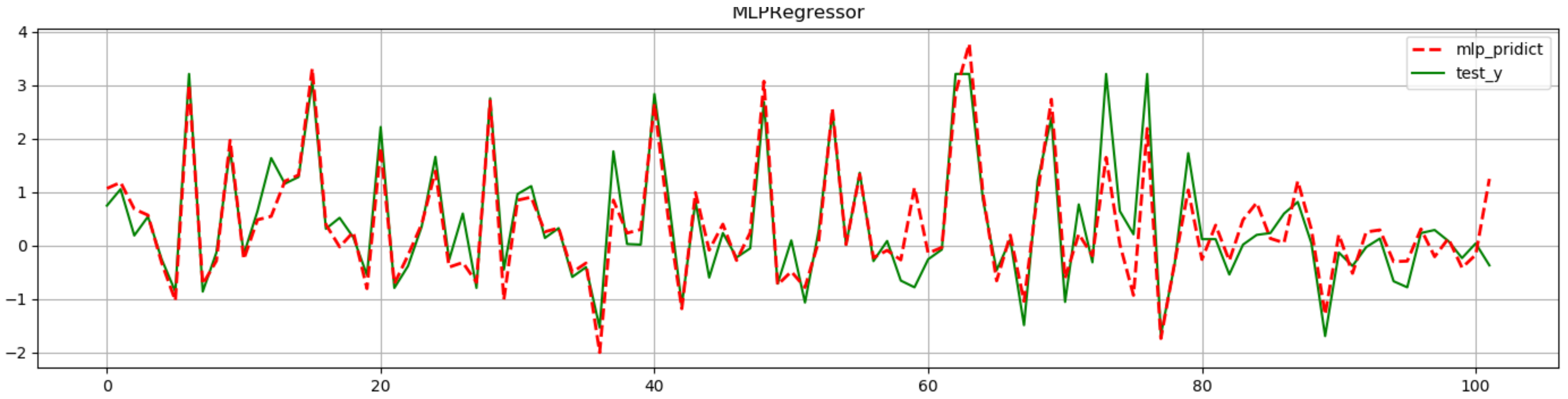
神经网络中的回归模型

- from sklearn.datasets import load_boston
- from sklearn.model_selection import train_test_split
- from sklearn import preprocessing
- from sklearn.neural_network import MLPRegressor
- #波士顿房价数据
- boston=load_boston()
- x=boston.data
- y=boston.target
- train_x, test_x, train_y, test_y = train_test_split(x, y,train_size=0.8, random_state=10)
- #数据标准化
- ss_x = preprocessing.StandardScaler()
- train_x = ss_x.fit_transform(train_x)
- test_x = ss_x.transform(test_x)
- ss_y = preprocessing.StandardScaler()
- train_y = ss_y.fit_transform(train_y.reshape(-1, 1))
- test_y=ss_y.transform(test_y.reshape(-1, 1))
- # 多层感知器-回归模型
- model_mlp = MLPRegressor(solver='lbfgs', hidden_layer_sizes=(20, 20, 20), random_state=10)
- model_mlp.fit(train_x,train_y)
- mlp_score=model_mlp.score(test_x,test_y)
- print('sklearn多层感知器-回归模型得分',mlp_score)
- #多层感知器预测
- mlp_pridict=model_mlp.predict(test_x)

神经网络中的回归模型

- import matplotlib.pyplot as plt
- fig = plt.figure(figsize=(20, 3))
- axes = fig.add_subplot(1, 1, 1)
- line1,=axes.plot(range(len(test_y)), test_y, 'g',label='test_y')
- line2,=axes.plot(range(len(mlp_pridict)), mlp_pridict, 'r--',label='mlp_pridict',linewidth=2)
- axes.grid()
- fig.tight_layout()
- plt.legend(handles=[line2,line1])
- plt.title('MLPRegressor')
- plt.show()

神经网络中的回归模型



卷积神经网络（CNN）

- 卷积层（用来提取局部区域的特征）
- 池化层（用在连续的卷积层之间，减少特征和参数数量）
- 全连接层

Keras卷积神经网络

- 识别手写数字

卷积层1（28*28图像共16层）

池化层1（14*14图像共16层）

卷积层2（14*14图像共36层）

池化层2（7*7图像共36层）

平坦层（36*7*7=1764神经元）

隐藏层（128个神经元）

输出层（10个神经元）

Keras卷积神经网络

- from keras.datasets import mnist
- from keras.utils import np_utils
- import numpy as np
- from keras.models import Sequential
- from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
- np.random.seed(10)
- (x_train,y_train),(x_test,y_test)=mnist.load_data()
- #将数字图像特征转换为四维矩阵
- x_train4d=x_train.reshape(x_train.shape[0],28,28,1).astype('float32')
- x_test4d=x_test.reshape(x_test.shape[0],28,28,1).astype('float32')

Keras卷积神经网络

- #对数据进行标准化
- x_train4d=x_train4d/255
- x_test4d=x_test4d/255
- #使用one-hot encoding转换
- y_train=np_utils.to_categorical(y_train)
- y_test=np_utils.to_categorical(y_test)

Keras卷积神经网络

- #建立线性堆叠模型
- model=Sequential()
- #建立卷积层1,filter滤镜， kernel_size每个滤镜的大小，让卷积运算产生的卷积图像大小不变，
- model.add(Conv2D(filters=16,kernel_size=(5,5),padding='same',input_shape=(28,28,1),activation='relu'))
- #建立池化层1
- model.add(MaxPooling2D(pool_size=(2,2)))

Keras卷积神经网络

- #建立卷积层2
- model.add(Conv2D(filters=36,kernel_size=(5,5),padding='same',activation='relu'))
- #建立池化层2
- model.add(MaxPooling2D(pool_size=(2,2)))
- #加入dropout层
- model.add(Dropout(0.25))

Keras卷积神经网络

- #建立平坦层,将36个7*7的图像换为一维向量，36*7*7=1764
- model.add(Flatten())
- #建立隐藏层，128个神经元
- model.add(Dense(128,activation='relu'))
- model.add(Dropout(0.5))
  #建立输出层
- model.add(Dense(10,activation='softmax'))

Keras卷积神经网络

- print(model.summary())

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 36)        14436
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 36)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 36)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1764)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               225920
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
```

Keras卷积神经网络

- #定义训练方式
- model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
- #开始训练
- train_history=model.fit(x=x_train4d,y=y_train,validation_split=0.2,epochs=10,batch_size=300,verbose=2)

Keras卷积神经网络

```
 - 44s - loss: 0.4901 - acc: 0.8472 - val_loss: 0.0977 - val_acc: 0.9715
Epoch 2/10
 - 40s - loss: 0.1419 - acc: 0.9580 - val_loss: 0.0638 - val_acc: 0.9806
Epoch 3/10
 - 41s - loss: 0.1032 - acc: 0.9692 - val_loss: 0.0511 - val_acc: 0.9844
Epoch 4/10
 - 40s - loss: 0.0851 - acc: 0.9751 - val_loss: 0.0457 - val_acc: 0.9861
Epoch 5/10
 - 41s - loss: 0.0723 - acc: 0.9779 - val_loss: 0.0394 - val_acc: 0.9867
Epoch 6/10
 - 41s - loss: 0.0649 - acc: 0.9806 - val_loss: 0.0389 - val_acc: 0.9883
Epoch 7/10
 - 41s - loss: 0.0573 - acc: 0.9823 - val_loss: 0.0412 - val_acc: 0.9875
Epoch 8/10
 - 40s - loss: 0.0513 - acc: 0.9843 - val_loss: 0.0340 - val_acc: 0.9898
Epoch 9/10
 - 40s - loss: 0.0456 - acc: 0.9865 - val_loss: 0.0339 - val_acc: 0.9898
Epoch 10/10
 - 40s - loss: 0.0430 - acc: 0.9868 - val_loss: 0.0338 - val_acc: 0.9903
```

Keras卷积神经网络

- #评估模型的准确性
- scores=model.evaluate(x_test4d,y_test)
- print(scores)
- #预测
- prediction=model.predict_classes(x_test4d)

- #查看前10项预测结果
- print(prediction[:10])

```
[0.024816589137994016, 0.991]
[7 2 1 0 4 1 4 9 5 9]
```