**Keras读书笔记**

**Keras神经模型生命周期**

1、定义网络Define

2、编绎网络Compile

3、训练网络Fit

4、评估网络Evaluate

5、网络预测Prediction

**定义网络**

mode = Sequential()

mode.add(Dense(2))

或者：

model = Sequential()

#全连接层，5个神经元。

model.add(Dense(5, input\_dim=2))

#全连接层，1个神经元

model.add(Dense(1))

也可以将一个层拆分为多个层定义

model = Sequential()

model.add(Dense(5, input\_dim=2))

model.add(Activation('relu'))

model.add(Dense(1))

model.add(Activation('sigmoid'))

**编绎网络**

model.compile(optimizer='sgd', loss='mse')

**训练网络**

history = model.fit(X, y, batch\_size=10, epochs=100)

**评估网络**

loss, accuracy = model.evaluate(X, y)

**（一）分类问题：印地安人糖尿病预测**

https://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/

数据来源：[download from the UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes)

CSV数据文件内容，总共768行，最后一列指出是否有糖尿病

6,148,72,35,0,33.6,0.627,50,1  
1,85,66,29,0,26.6,0.351,31,0  
8,183,64,0,0,23.3,0.672,32,1

**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense  
**import** numpy  
  
*# load and prepare the dataset*dataset = numpy.loadtxt(**"pima-indians-diabetes.csv"**, delimiter=**","**)  
第9列是答案

X = dataset[:,0:8]  
Y = dataset[:,8]  
  
*# 1. define the network*model = Sequential()

全连接层12个神经元，输入层8列数据，relu激活函数  
model.add(Dense(12, input\_dim=8, activation=**'relu'**))

全连接层1个神经元，激活函数sigmoid，输出是概率分布。  
model.add(Dense(1, activation=**'sigmoid'**))  
  
*# 2. compile the network*

损失函数binary\_crossentropy，优化算法adam  
model.compile(loss=**'binary\_crossentropy'**, optimizer=**'adam'**, metrics=[**'accuracy'**])  
  
*# 3. fit the network*训练模型

history = model.fit(X, Y, epochs=100, batch\_size=10)  
  
*# 4. evaluate the network*评估模型

loss, accuracy = model.evaluate(X, Y)  
print(**"\nLoss: %.2f, Accuracy: %.2f%%"** % (loss, accuracy\*100))  
  
*# 5. make predictions*预测模型

probabilities = model.predict(X)  
predictions = [float(round(x)) **for** x **in** probabilities]  
accuracy = numpy.mean(predictions == Y)  
print(**"Prediction Accuracy: %.2f%%"** % (accuracy\*100))

**（二）简单线性回归**

**import** numpy **as** np  
#初始化随机种子

np.random.seed(1337)  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense  
**import** matplotlib.pyplot **as** plt  
  
*# 生成数据*X = np.linspace(-1, 1, 200) *#在返回（-1, 1）范围内的等差序列200个数字*np.random.shuffle(X) *# 打乱顺序*Y = 0.5 \* X + 2 + np.random.normal(0, 0.05, (200, )) *#生成Y并添加噪声  
# plot*plt.scatter(X, Y)  
plt.show()  
  
X\_train, Y\_train = X[:160], Y[:160] *# 前160组数据为训练数据集*X\_test, Y\_test = X[160:], Y[160:] *#后40组数据为测试数据集  
  
# 构建神经网络模型*model = Sequential()  
model.add(Dense(input\_dim=1, units=1))  
  
*# 回归问题选择方差误差，优化算法选择sgd*model.compile(loss=**'mse'**, optimizer=**'sgd'**)  
  
*# 训练过程*print(**'Training -----------'**)  
**for** step **in** range(501):  
 cost = model.train\_on\_batch(X\_train, Y\_train)  
 **if** step % 50 == 0:  
 print(**"After %d trainings, the cost: %f"** % (step, cost))  
  
*# 测试过程*print(**'\nTesting ------------'**)  
cost = model.evaluate(X\_test, Y\_test, batch\_size=40)  
print(**'test cost:'**, cost)  
W, b = model.layers[0].get\_weights()  
print(**'Weights='**, W, **'\nbiases='**, b)  
  
*# 将训练结果绘出*Y\_pred = model.predict(X\_test)  
plt.scatter(X\_test, Y\_test)  
plt.plot(X\_test, Y\_pred)  
plt.show()

**（三）非线性回归**

**from** keras.optimizers **import** SGD  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense,Activation  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
np.random.seed(0)  
x\_data=np.linspace(-0.5,0.5,200)  
noise=np.random.normal(0,0.02,x\_data.shape)  
y\_data=np.square(x\_data)+noise  
  
*#构建一个顺序模型*model=Sequential()  
  
*#在模型中添加一个全连接层  
#units是输出维度,input\_dim是输入维度(shift+两次tab查看函数参数)  
#输入1个神经元,隐藏层10个神经元,输出层1个神经元*model.add(Dense(units=10,input\_dim=1))  
model.add(Activation(**'tanh'**)) *#增加非线性激活函数*model.add(Dense(units=1)) *#默认连接上一层input\_dim=10*model.add(Activation(**'tanh'**))  
  
*#定义优化算法(修改学习率)*defsgd=SGD(lr=0.3)  
  
*#编译模型*model.compile(optimizer=defsgd,loss=**'mse'**) *#optimizer参数设置优化器,loss设置目标函数  
  
#训练模型***for** step **in** range(3001):  
 *#每次训练一个批次* cost=model.train\_on\_batch(x\_data,y\_data)  
 *#每500个batch打印一个cost值* **if** step%500==0:  
 print(**'cost:'**,cost)  
  
*#打印权值和偏置值*W,b=model.layers[0].get\_weights() *#layers[0]只有一个网络层*print(**'W:'**,W,**'b:'**,b)  
  
*#x\_data输入网络中，得到预测值y\_pred*y\_pred=model.predict(x\_data)  
  
plt.scatter(x\_data,y\_data)  
  
plt.plot(x\_data,y\_pred,**'r-'**,lw=3)  
plt.show()

**numpy常用函数**

1、概率密度函数

numpy.random.normal(loc=0.0, scale=1.0, size=None)

loc: 概率分布的均值

scale: 标准差（对应分布的宽度，超大越胖，越小越瘦高）

size: 一个数字或一个tuple，输出的shape