使用Tensorflow进行手写数字识别

一、实验内容

> 利用Tensorflow、Python进行手写数字识别

keras是什么

- 基于Python的高级神经网络API
- 以Tensorflow、CNTK等后端才可以运行

Tensorflow-keras是什么

- Tensorflow对Keras API规范的实现
- 相对于以TensorFlow为后端的keras，Tensorflow-keras与Tensorflow结合更加紧密
- 实现在tf.keras空间下

tf.keras和keras联系

- 基于同一套API
  - keras程序可以通过改变导入方式轻松转换为tf.keras程序
  - 反之可能不成立，因为tf.keras有其他特性
- 相同的JSON和HDF5模型序列化格式和语义

tf.keras和keras区别

- tf.keras全面支持eager mode
  - 只是用keras.Sequential和keras.Model时没影响
  - 自定义model内部运算逻辑时会有影响
    - tf底层API可以使用keras的model.fit等抽象
    - 适合研究人员
- tf.keras支持基于tf.data的模型训练
- tf.keras支持TPU训练
- tf.keras支持tf.distribution的分布式策略
- tf.keras可以与tensorflow中的estimator集成
- tf.keras可以保存为SavedModel

分类与回归

- 分类问题预测的是类型，模型的输出是概率分布
- 回归问题预测是值。模型的输出是一个实数值

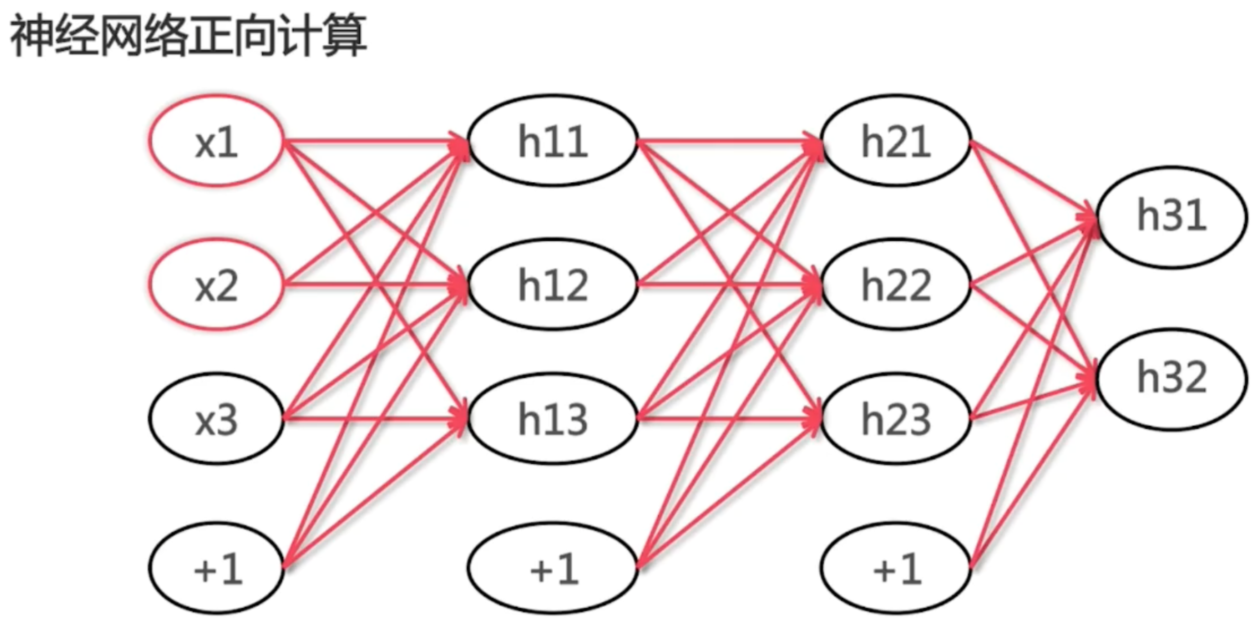
为什么需要目标函数

- 参数是逐步调整的
- 目标函数可以帮助衡量模型的好坏

损失函数

- 平方差损失
- 交叉熵损失

神经网络



神经网络训练

- 下山算法
    - 找到方向
    - 走一步
- 梯度下降
    - 求导
    - 更新参数

深度神经网络

一般的神经网络只有三到五层，而深度神经网络顾名思义就是层次非常深的深度神经网络，多达几十层上百层

激活函数

- Sigmoid
- tanh
- relu
- Leaky relu
- maxout
- Elu

归一化

> 对数据进行规整，使得其方差为0，均值为1

- Min-Max归一化：x*=(x-min)/(max-min)
- Z-score归一化：x*=(x-u)/o

二、实验设计

1. 使用tensorflow自带的mnist数据集，首先将数据划分为训练集、验证集与测试集。
2. 随后对数据集进行归一化处理。
3. 使用3层全连接神经网络训练模型，检验训练效果
4. 使用20层全连接的深度神经网络进行训练，检验训练效果

三、详细实验过程

数据读取与展示

导入必要的包

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import sklearn
import pandas as pd
import os
import sys
import time
import tensorflow as tf
from tensorflow import keras
```

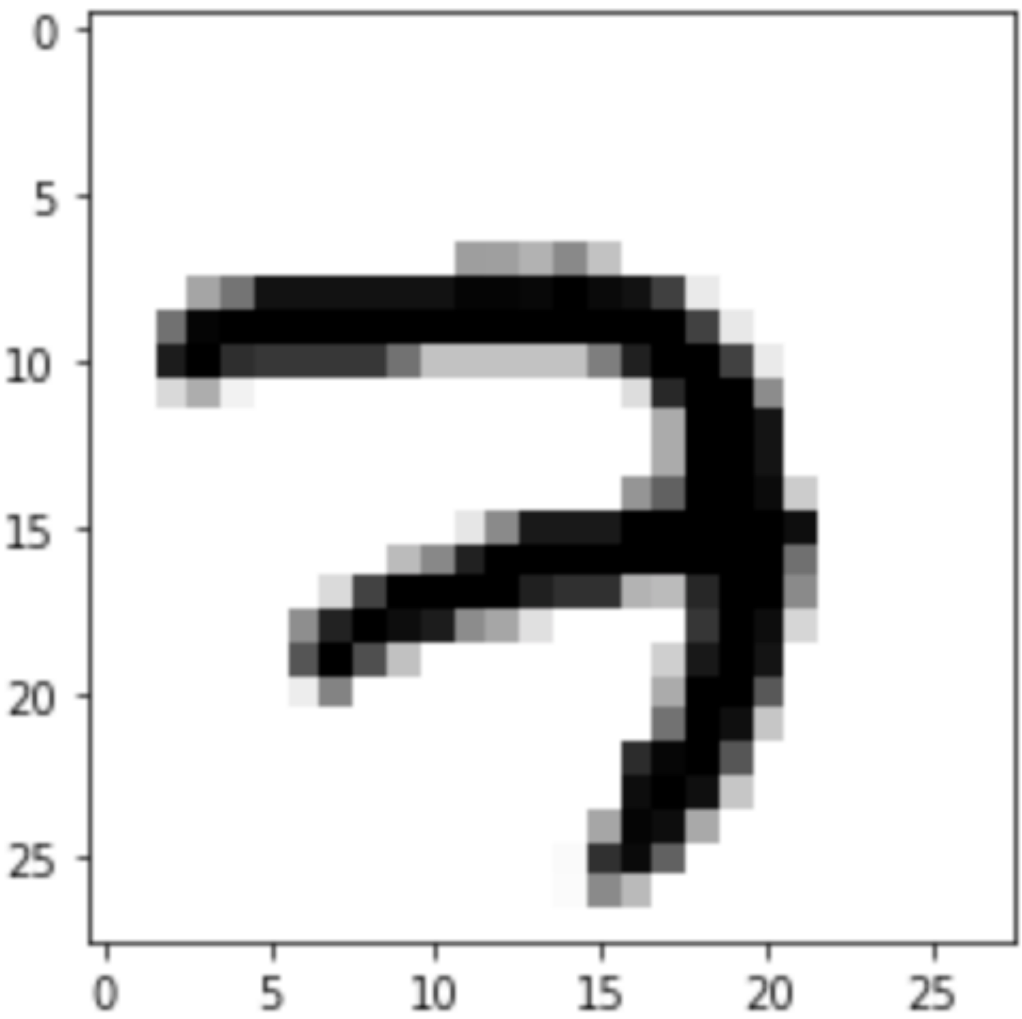
读入数据集

```
minist = keras.datasets.mnist
(x_train_all,y_train_all),(x_test,y_test)=minist.load_data()
x_valid,x_train=x_train_all[:5000],x_train_all[5000:]
y_valid,y_train=y_train_all[:5000],y_train_all[5000:]

print(x_valid.shape,y_valid.shape)
print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

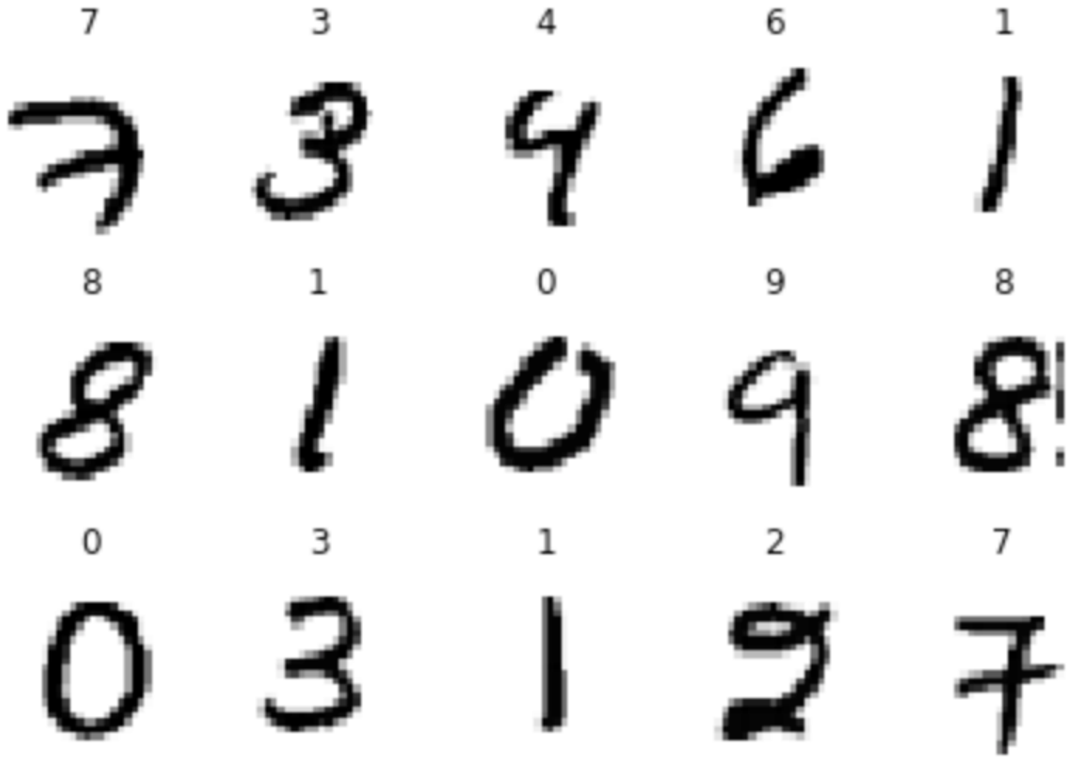
```
(5000, 28, 28) (5000,)
(55000, 28, 28) (55000,)
(10000, 28, 28) (10000,)
```

展示数据集

```
def show_single_image(img_arr):
    plt.imshow(img_arr,cmap="binary")
    plt.show()

show_single_image(x_train[0])
```




```
def show_images(n_rows,n_cols,x_data,y_data,class_names):
    assert len(x_data)==len(y_data)
    assert n_rows*n_cols<len(x_data)
    plt.figure(figsize=(n_cols*1.4,n_rows*1.6))
    for row in range(n_rows):
        for col in range(n_cols):
            index=n_cols*row+col
            plt.subplot(n_rows,n_cols,index+1)
            plt.imshow(x_data[index],cmap="binary",interpolation="nearest")
            plt.axis("off")
            plt.title(class_names[y_data[index]])
    plt.show()
class_names=['0','1','2','3','4','5','6','7','8','9']
show_images(3,5,x_train,y_train,class_names)
```



数据归一化

```python
print(np.max(x_train),np.min(x_train))

# x=(x-u)/std

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train_scaled=scaler.fit_transform(x_train.astype(np.float32).reshape(-1,1)).reshape(-1,28,28)

x_valid_scaled=scaler.transform(x_valid.astype(np.float32).reshape(-1,1)).reshape(-1,28,28)

x_test_scaled=scaler.transform(x_test.astype(np.float32).reshape(-1,1)).reshape(-1,28,28)

print(np.max(x_train_scaled),np.min(x_train_scaled))
```

```
255 0
```

```
2.8209424 -0.4241323
```

三层神经网络

模型构建

```
model=keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(300,activation="relu"))
model.add(keras.layers.Dense(100,activation="relu"))
model.add(keras.layers.Dense(10,activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy",optimizer = "sgd",metrics=["accuracy"])
```

查看模型

```
model.layers
```

```
[<tensorflow.python.keras.layers.core.Flatten at 0x7fa90e6d6be0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e3215b0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90dac7640>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90dac7a30>]
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense (Dense)                (None, 300)               235500    
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010      
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

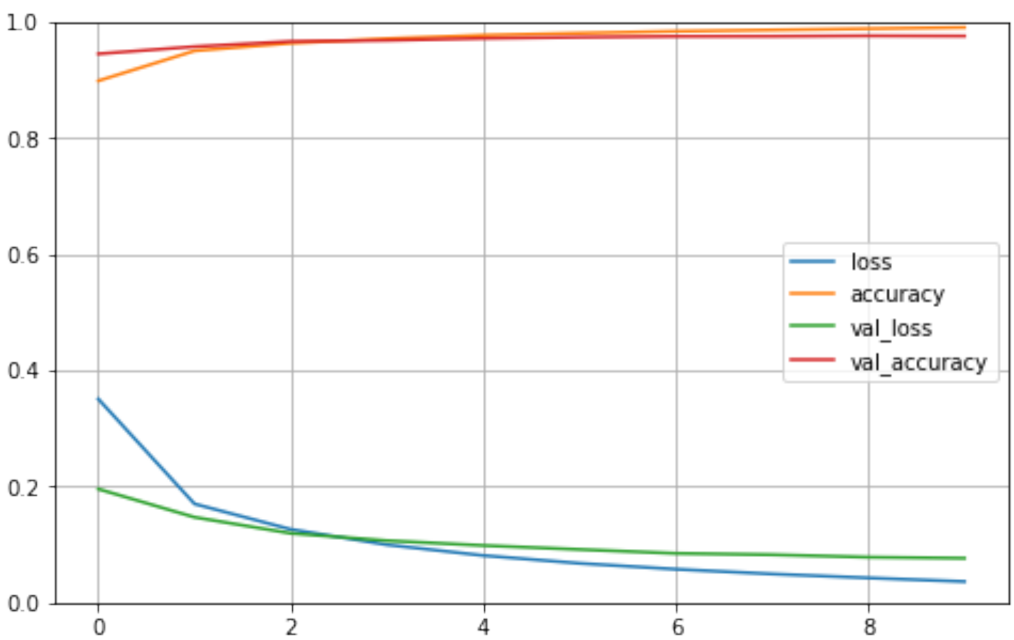
训练模型

```
history=model.fit(x_train_scaled,y_train,epochs=10,validation_data=
(x_valid_scaled,y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3510 - accuracy: 0.8987 - val_loss: 0.1959 - val_a
ccuracy: 0.9452
Epoch 2/10
1719/1719 [==============================] - 8s 5ms/step - loss: 0.1703 - accuracy: 0.9504 - val_loss: 0.1473 - val_a
ccuracy: 0.9576
Epoch 3/10
1719/1719 [==============================] - 9s 5ms/step - loss: 0.1265 - accuracy: 0.9635 - val_loss: 0.1196 - val_a
ccuracy: 0.9666
Epoch 4/10
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0999 - accuracy: 0.9714 - val_loss: 0.1070 - val_a
ccuracy: 0.9684
Epoch 5/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0814 - accuracy: 0.9772 - val_loss: 0.0987 - val_a
ccuracy: 0.9718
Epoch 6/10
1719/1719 [==============================] - 8s 5ms/step - loss: 0.0679 - accuracy: 0.9807 - val_loss: 0.0918 - val_a
ccuracy: 0.9738
Epoch 7/10
1719/1719 [==============================] - 9s 5ms/step - loss: 0.0578 - accuracy: 0.9841 - val_loss: 0.0851 - val_a
ccuracy: 0.9752
Epoch 8/10
1719/1719 [==============================] - 9s 5ms/step - loss: 0.0496 - accuracy: 0.9863 - val_loss: 0.0828 - val_a
ccuracy: 0.9754
Epoch 9/10
1719/1719 [==============================] - 8s 5ms/step - loss: 0.0428 - accuracy: 0.9885 - val_loss: 0.0785 - val_a
ccuracy: 0.9762
Epoch 10/10
1719/1719 [==============================] - 8s 5ms/step - loss: 0.0365 - accuracy: 0.9906 - val_loss: 0.0767 - val_a
ccuracy: 0.9758
```

查看学习曲线图

```
def plot_learning_curves(history):
    pd.DataFrame(history.history).plot(figsize=(8,5))
    plt.grid(True)
    plt.gca().set_ylim(0,1)
    plt.show()
plot_learning_curves(history)
```



模型预测

```
model.evaluate(x_test_scaled,y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0756 - accuracy: 0.9756
[0.0755981057882309, 0.975600004196167]
```

深度神经网络

模型构建

```
model=keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))

for i in range(20):
    model.add(keras.layers.Dense(100,activation="relu"))
model.add(keras.layers.Dense(10,activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy",optimizer = "sgd",metrics=["accuracy"])
```

查看模型

```
model.layers
```

```
[<tensorflow.python.keras.layers.core.Flatten at 0x7fa90e5019d0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa93ac9f8e0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e50db20>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e50dfa0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa911eaddf0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa9132c21c0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4e1e80>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4d5520>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4d5f70>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4d4880>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4c6250>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4c6d00>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4f65b0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e4f6520>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5de910>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5d32e0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5d3d90>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5c6640>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5c65b0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5fca90>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e5f3370>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa90e50d610>]
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 100)               78500     
_________________________________________________________________
dense_4 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_5 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_6 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_7 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_8 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_9 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_10 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_11 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_12 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_13 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_14 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_15 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_16 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_17 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_18 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_19 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_20 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_21 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_22 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1010      
=================================================================
Total params: 271,410
Trainable params: 271,410
Non-trainable params: 0
```

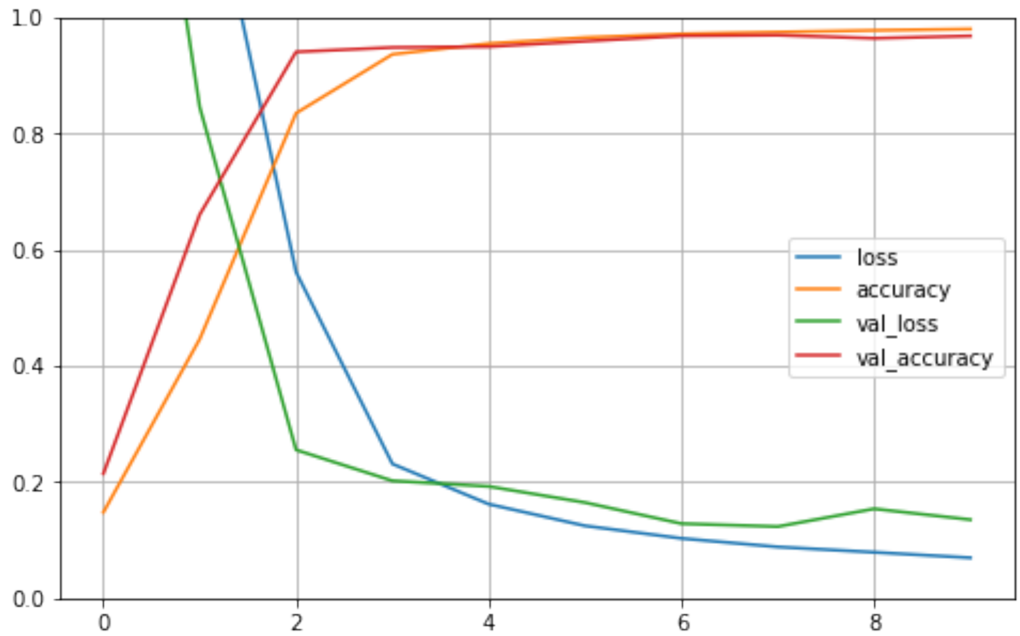
训练模型

```
history=model.fit(x_train_scaled,y_train,epochs=10,validation_data=(x_valid_scaled,y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 14s 8ms/step - loss: 2.2575 - accuracy: 0.1480 - val_loss: 1.9294 - val_
accuracy: 0.2146
Epoch 2/10
1719/1719 [==============================] - 11s 7ms/step - loss: 1.3380 - accuracy: 0.4463 - val_loss: 0.8457 - val_
accuracy: 0.6608
Epoch 3/10
1719/1719 [==============================] - 12s 7ms/step - loss: 0.5617 - accuracy: 0.8352 - val_loss: 0.2555 - val_
accuracy: 0.9408
Epoch 4/10
1719/1719 [==============================] - 14s 8ms/step - loss: 0.2314 - accuracy: 0.9366 - val_loss: 0.2023 - val_
accuracy: 0.9484
Epoch 5/10
1719/1719 [==============================] - 14s 8ms/step - loss: 0.1621 - accuracy: 0.9551 - val_loss: 0.1926 - val_
accuracy: 0.9498
Epoch 6/10
1719/1719 [==============================] - 14s 8ms/step - loss: 0.1248 - accuracy: 0.9655 - val_loss: 0.1648 - val_
accuracy: 0.9592
Epoch 7/10
1719/1719 [==============================] - 13s 7ms/step - loss: 0.1031 - accuracy: 0.9717 - val_loss: 0.1285 - val_
accuracy: 0.9684
Epoch 8/10
1719/1719 [==============================] - 14s 8ms/step - loss: 0.0884 - accuracy: 0.9749 - val_loss: 0.1235 - val_
accuracy: 0.9696
Epoch 9/10
1719/1719 [==============================] - 11s 6ms/step - loss: 0.0792 - accuracy: 0.9778 - val_loss: 0.1541 - val_
accuracy: 0.9642
Epoch 10/10
1719/1719 [==============================] - 12s 7ms/step - loss: 0.0697 - accuracy: 0.9802 - val_loss: 0.1354 - val_
accuracy: 0.9682
```

查看学习曲线图

```
def plot_learning_curves(history):
    pd.DataFrame(history.history).plot(figsize=(8,5))
    plt.grid(True)
    plt.gca().set_ylim(0,1)
    plt.show()
plot_learning_curves(history)
```



模型预测

```
model.evaluate(x_test_scaled,y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1488 - accuracy: 0.9652
[0.14883717894554138, 0.9652000069618225]
```

四、实验结果

1. 神经网络模型可以较好的解决分类问题，在本次实验中，手写体的识别准确率最高达到了99%以上
2. 神经网络层数并非越多越好，在本次实验中20层的深度神经网络表现略落后仅三层的神经网络

五、实验心得体会

通过本次实验我掌握了Python和Tensorflow的用法，对神经网络的原理和应用有了更加深刻的理解，了解前言的深度学习的知识，同时也提高了编程能力，本次的实验使我有了很多的进步。