NN

2021 年 4 月 25 日

0.1 实验一、利用神经网络模型进行 MNIST 数据的分类

[1]:
```
import keras
from keras.datasets import mnist
from keras import models
from keras import layers
from keras.utils import to_categorical
# keras.__version__
```

0.1.1 加载数据分为训练数据与测试数据并进行数据归一化

[2]:
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print('训练数据形状: ',train_images.shape,'测试数据形状: ',test_images.shape)
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

```
训练数据形状: (60000, 28, 28) 测试数据形状: (10000, 28, 28)
```

0.1.2 构建神经网络模型

[4]:
```
network = models.Sequential()
network.add(layers.Dense(400, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(100, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='SGD',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

network.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 400)               314000
_________________________________________________________________
dense_1 (Dense)              (None, 100)               40100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 355,110
Trainable params: 355,110
Non-trainable params: 0
_________________________________________________________________
```

0.1.3 利用训练数据进行模型的训练

[5]:
```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 2s 5ms/step - loss: 1.1619 -
accuracy: 0.7197
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.4692 -
accuracy: 0.8777
```

```
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.3659 -
accuracy: 0.8995
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.3218 -
accuracy: 0.9101
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.2937 -
accuracy: 0.9171
```

[5]:
```
<tensorflow.python.keras.callbacks.History at 0x1c709588490>
```

0.1.4 利用测试集进行模型的测试

[6]:
```python
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('测试数据准确率', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2724 -
accuracy: 0.9238
测试数据准确率 0.923799991607666
```

0.1.5 进行数据的可视化处理，加大 **epoch** 观察模型在训练与测试数据集上的情况

[7]:
```python
import tensorflow as tf
import os
import numpy as np
from matplotlib import pyplot as plt


checkpoint_save_path = "./checkpoint_1/mnist.ckpt"
'''
if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)
'''


cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
```

```
                                                 save_weights_only=True,
                                                 save_best_only=True)

history = network.fit(train_images,
                      train_labels,
                      batch_size=128,
                      epochs=55,
                      validation_data=(test_images, test_labels),
                      callbacks=[cp_callback])
network.summary()

# print(model.trainable_variables)
file = open('./weights_1.txt', 'w')
for v in network.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

```
Epoch 1/55
469/469 [==============================] - 3s 5ms/step - loss: 0.2734 -
accuracy: 0.9226 - val_loss: 0.2567 - val_accuracy: 0.9281
Epoch 2/55
469/469 [==============================] - 2s 5ms/step - loss: 0.2567 -
accuracy: 0.9270 - val_loss: 0.2448 - val_accuracy: 0.9309
Epoch 3/55
469/469 [==============================] - 3s 5ms/step - loss: 0.2428 -
accuracy: 0.9311 - val_loss: 0.2311 - val_accuracy: 0.9343
Epoch 4/55
469/469 [==============================] - 2s 5ms/step - loss: 0.2305 -
accuracy: 0.9347 - val_loss: 0.2209 - val_accuracy: 0.9358
Epoch 5/55
469/469 [==============================] - 2s 5ms/step - loss: 0.2195 -
accuracy: 0.9378 - val_loss: 0.2123 - val_accuracy: 0.9388
Epoch 6/55
469/469 [==============================] - 2s 5ms/step - loss: 0.2098 -
```

```
accuracy: 0.9404 - val_loss: 0.2034 - val_accuracy: 0.9410
Epoch 7/55
469/469 [==============================] - 2s 5ms/step - loss: 0.2011 -
accuracy: 0.9428 - val_loss: 0.1955 - val_accuracy: 0.9438
Epoch 8/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1928 -
accuracy: 0.9452 - val_loss: 0.1904 - val_accuracy: 0.9449
Epoch 9/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1854 -
accuracy: 0.9470 - val_loss: 0.1829 - val_accuracy: 0.9480
Epoch 10/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1785 -
accuracy: 0.9496 - val_loss: 0.1762 - val_accuracy: 0.9488
Epoch 11/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1720 -
accuracy: 0.9512 - val_loss: 0.1721 - val_accuracy: 0.9513
Epoch 12/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1661 -
accuracy: 0.9528 - val_loss: 0.1663 - val_accuracy: 0.9513
Epoch 13/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1604 -
accuracy: 0.9547 - val_loss: 0.1606 - val_accuracy: 0.9537
Epoch 14/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1550 -
accuracy: 0.9559 - val_loss: 0.1570 - val_accuracy: 0.9535
Epoch 15/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1500 -
accuracy: 0.9576 - val_loss: 0.1542 - val_accuracy: 0.9540
Epoch 16/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1454 -
accuracy: 0.9586 - val_loss: 0.1503 - val_accuracy: 0.9561
Epoch 17/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1408 -
accuracy: 0.9604 - val_loss: 0.1455 - val_accuracy: 0.9578
Epoch 18/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1367 -
accuracy: 0.9618 - val_loss: 0.1424 - val_accuracy: 0.9575
```

```
Epoch 19/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1326 -
accuracy: 0.9625 - val_loss: 0.1376 - val_accuracy: 0.9589
Epoch 20/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1288 -
accuracy: 0.9638 - val_loss: 0.1351 - val_accuracy: 0.9597
Epoch 21/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1252 -
accuracy: 0.9649 - val_loss: 0.1328 - val_accuracy: 0.9614
Epoch 22/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1218 -
accuracy: 0.9659 - val_loss: 0.1293 - val_accuracy: 0.9618
Epoch 23/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1185 -
accuracy: 0.9666 - val_loss: 0.1261 - val_accuracy: 0.9623
Epoch 24/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1152 -
accuracy: 0.9679 - val_loss: 0.1242 - val_accuracy: 0.9626
Epoch 25/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1123 -
accuracy: 0.9685 - val_loss: 0.1221 - val_accuracy: 0.9633
Epoch 26/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1093 -
accuracy: 0.9694 - val_loss: 0.1191 - val_accuracy: 0.9633
Epoch 27/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1066 -
accuracy: 0.9701 - val_loss: 0.1165 - val_accuracy: 0.9651
Epoch 28/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1040 -
accuracy: 0.9708 - val_loss: 0.1155 - val_accuracy: 0.9642
Epoch 29/55
469/469 [==============================] - 2s 5ms/step - loss: 0.1015 -
accuracy: 0.9718 - val_loss: 0.1134 - val_accuracy: 0.9656
Epoch 30/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0990 -
accuracy: 0.9727 - val_loss: 0.1111 - val_accuracy: 0.9657
Epoch 31/55
```

```
469/469 [==============================] - 2s 5ms/step - loss: 0.0965 -
accuracy: 0.9735 - val_loss: 0.1096 - val_accuracy: 0.9668
Epoch 32/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0944 -
accuracy: 0.9739 - val_loss: 0.1087 - val_accuracy: 0.9662
Epoch 33/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0923 -
accuracy: 0.9748 - val_loss: 0.1081 - val_accuracy: 0.9667
Epoch 34/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0902 -
accuracy: 0.9753 - val_loss: 0.1043 - val_accuracy: 0.9688
Epoch 35/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0880 -
accuracy: 0.9760 - val_loss: 0.1029 - val_accuracy: 0.9680
Epoch 36/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0861 -
accuracy: 0.9764 - val_loss: 0.1018 - val_accuracy: 0.9690
Epoch 37/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0842 -
accuracy: 0.9770 - val_loss: 0.1002 - val_accuracy: 0.9695
Epoch 38/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0824 -
accuracy: 0.9772 - val_loss: 0.0987 - val_accuracy: 0.9710
Epoch 39/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0806 -
accuracy: 0.9781 - val_loss: 0.0981 - val_accuracy: 0.9703
Epoch 40/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0789 -
accuracy: 0.9787 - val_loss: 0.0972 - val_accuracy: 0.9697
Epoch 41/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0773 -
accuracy: 0.9789 - val_loss: 0.0959 - val_accuracy: 0.9708
Epoch 42/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0756 -
accuracy: 0.9795 - val_loss: 0.0951 - val_accuracy: 0.9705
Epoch 43/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0740 -
```

```
accuracy: 0.9800 - val_loss: 0.0944 - val_accuracy: 0.9721
Epoch 44/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0726 -
accuracy: 0.9802 - val_loss: 0.0933 - val_accuracy: 0.9715
Epoch 45/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0711 -
accuracy: 0.9807 - val_loss: 0.0916 - val_accuracy: 0.9715
Epoch 46/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0697 -
accuracy: 0.9811 - val_loss: 0.0907 - val_accuracy: 0.9720
Epoch 47/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0683 -
accuracy: 0.9815 - val_loss: 0.0902 - val_accuracy: 0.9722
Epoch 48/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0668 -
accuracy: 0.9821 - val_loss: 0.0892 - val_accuracy: 0.9730
Epoch 49/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0656 -
accuracy: 0.9822 - val_loss: 0.0885 - val_accuracy: 0.9732
Epoch 50/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0641 -
accuracy: 0.9827 - val_loss: 0.0881 - val_accuracy: 0.9735
Epoch 51/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0630 -
accuracy: 0.9830 - val_loss: 0.0868 - val_accuracy: 0.9729
Epoch 52/55
469/469 [==============================] - 3s 6ms/step - loss: 0.0617 -
accuracy: 0.9836 - val_loss: 0.0866 - val_accuracy: 0.9727
Epoch 53/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0606 -
accuracy: 0.9840 - val_loss: 0.0852 - val_accuracy: 0.9740
Epoch 54/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0595 -
accuracy: 0.9844 - val_loss: 0.0858 - val_accuracy: 0.9745
Epoch 55/55
469/469 [==============================] - 2s 5ms/step - loss: 0.0584 -
accuracy: 0.9845 - val_loss: 0.0842 - val_accuracy: 0.9744
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 400)               314000
_________________________________________________________________
dense_1 (Dense)              (None, 100)               40100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 355,110
Trainable params: 355,110
Non-trainable params: 0
_________________________________________________________________
```

0.1.6 可视化绘图

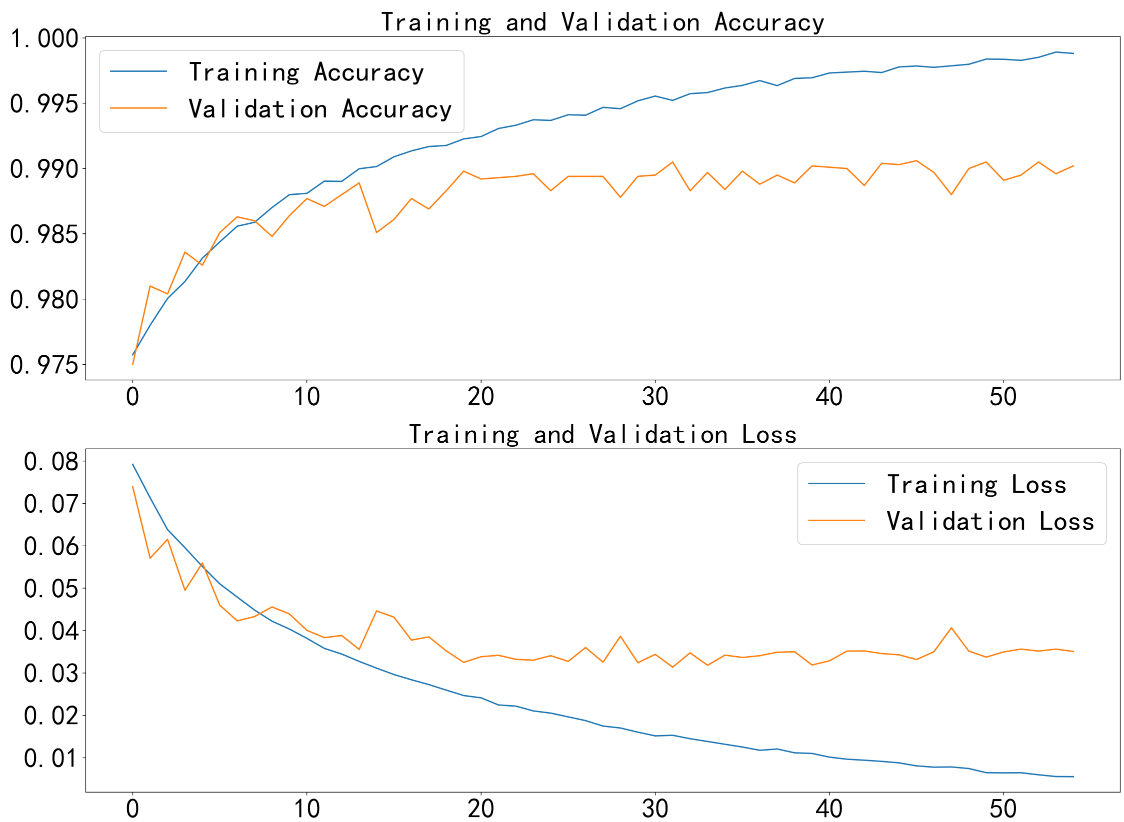
[20]:

```python
##loss 是训练集 loss, val_loss 是测试集 loss
#sparse_categorical_accuracy 是训练集 sparse_categorical_accuracy,
val_sparse_categorical_accuracy 是测试集 sparse_categorical_accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(dpi=500, figsize=[20, 15])
plt.rcParams["font.sans-serif"] = ["SimHei"]
#subplot 函数将图像分为两行一列，这段代码画出第一行
plt.subplot(2, 1, 1)
#plot 描述曲线
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
#title 函数设置图标题
plt.title('Training and Validation Accuracy',fontsize=30)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
# 画出图例
```

```python
plt.legend(fontsize=30)

# 这段代码画出第二行
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss',fontsize=30)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.legend(fontsize=30)
plt.show()
```

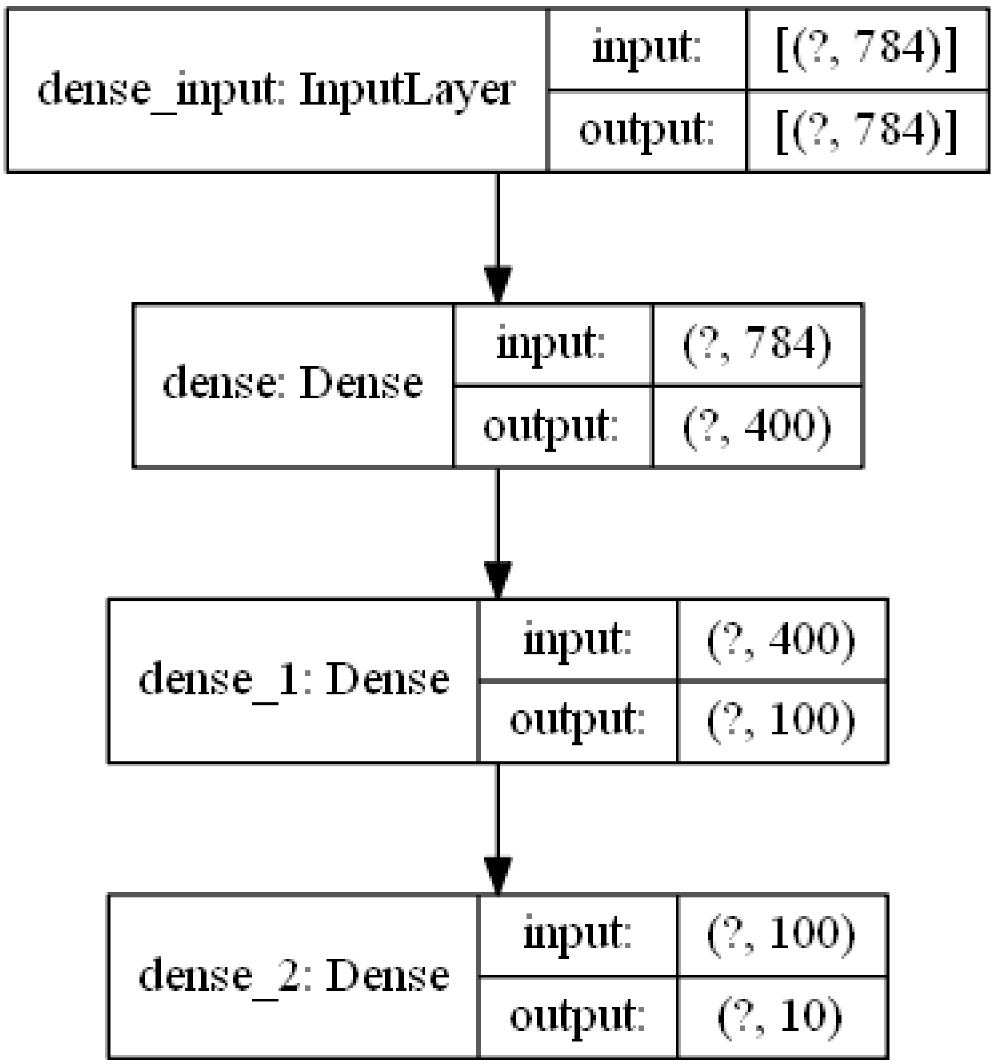


0.1.7 绘制模型的结构图

[9]:
```
from keras.models import load_model
from keras.utils import plot_model


# 输出模型，将结果保存到项目文件夹中
plot_model(network, to_file='model_1.png', show_shapes='True')
```

[9]:



[ ]: