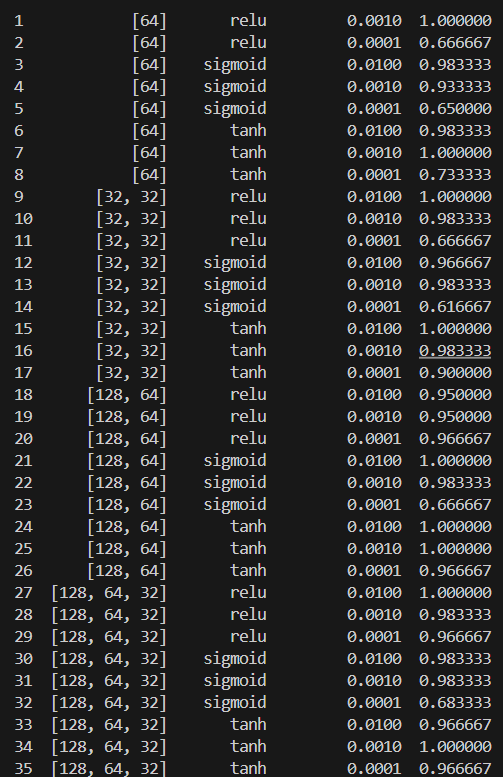
**模式识别报告L10**

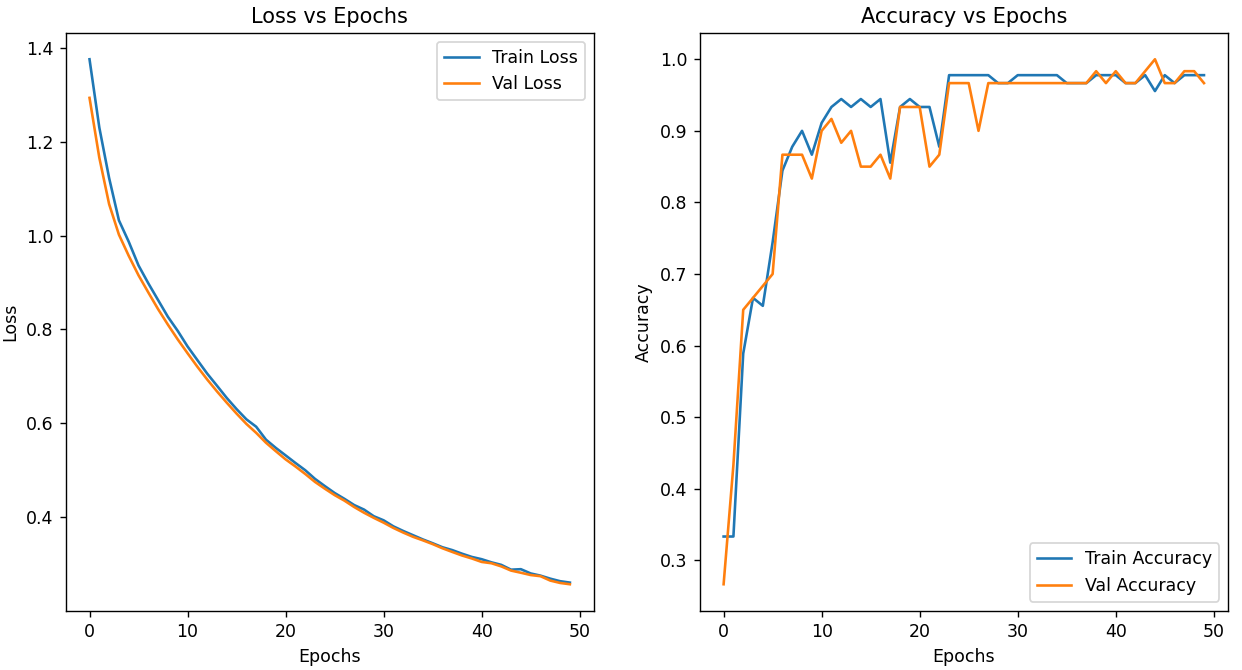
自卓2201 杨欣怡 U202215067

1，IRIS数据集有三类目标，每个类别有50个样本，每个样本有四维特征。自行设计神经网络实现对这三个目标的识别，实验时每个类别随机选30个样本进行训练，另外20个样本用于测试。希望能通过设计不同的隐含层数、每层的节点数、不同的学习率、不同的激活函数等对实验结果进行讨论。

（1）代码实现

1. import numpy as np
2. import pandas as pd
3. import tensorflow as tf
4. from sklearn import datasets
5. from sklearn.model\_selection import train\_test\_split
6. from sklearn.preprocessing import LabelEncoder
7. from sklearn.metrics import accuracy\_score
8. import matplotlib.pyplot as plt
9. # 加载IRIS数据集
10. iris = datasets.load\_iris()
11. X = iris.data # 特征数据
12. y = iris.target # 标签数据
13. # 将标签转换为one-hot编码
14. y = tf.keras.utils.to\_categorical(y, num\_classes=3)
15. # 将数据集分为训练集和测试集
16. # 每个类别随机选30个样本作为训练数据，剩余20个样本作为测试数据
17. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, stratify=y, random\_state=42)
18. def create\_model(hidden\_layers=[64], activation='relu', learning\_rate=0.001):
19. model = tf.keras.Sequential()
21. # 输入层
22. model.add(tf.keras.layers.InputLayer(input\_shape=(X\_train.shape[1],)))
24. # 隐含层
25. for units in hidden\_layers:
26. model.add(tf.keras.layers.Dense(units, activation=activation))
28. # 输出层
29. model.add(tf.keras.layers.Dense(3, activation='softmax')) # softmax用于多分类
31. # 编译模型
32. optimizer = tf.keras.optimizers.Adam(learning\_rate=learning\_rate)
33. model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])
35. return model
36. # 设置不同的网络参数组合进行实验
37. hidden\_layers\_configs = [
38. [64], # 1个隐含层，64个节点
39. [32, 32], # 2个隐含层，每层32个节点
40. [128, 64], # 2个隐含层，128个和64个节点
41. [128, 64, 32], # 3个隐含层，128、64和32个节点
42. ]
43. activation\_functions = ['relu', 'sigmoid', 'tanh']
44. learning\_rates = [0.01, 0.001, 0.0001]
45. # 训练和评估所有配置组合
46. results = []
47. for hidden\_layers in hidden\_layers\_configs:
48. for activation in activation\_functions:
49. for lr in learning\_rates:
50. print(f"Training with hidden\_layers={hidden\_layers}, activation={activation}, learning\_rate={lr}")
52. # 创建模型
53. model = create\_model(hidden\_layers=hidden\_layers, activation=activation, learning\_rate=lr)
55. # 训练模型
56. history = model.fit(X\_train, y\_train, epochs=50, batch\_size=10, validation\_data=(X\_test, y\_test), verbose=0)
58. # 评估模型
59. y\_pred = model.predict(X\_test)
60. accuracy = accuracy\_score(np.argmax(y\_test, axis=1), np.argmax(y\_pred, axis=1))
62. # 记录结果
63. results.append({
64. 'hidden\_layers': hidden\_layers,
65. 'activation': activation,
66. 'learning\_rate': lr,
67. 'accuracy': accuracy
68. })
69. # 输出实验结果
70. results\_df = pd.DataFrame(results)
71. print(results\_df)
72. # 绘制训练过程的损失和准确率曲线
73. def plot\_training\_history(history):
74. plt.figure(figsize=(12, 6))
76. # 绘制训练和验证的损失曲线
77. plt.subplot(1, 2, 1)
78. plt.plot(history.history['loss'], label='Train Loss')
79. plt.plot(history.history['val\_loss'], label='Val Loss')
80. plt.title('Loss vs Epochs')
81. plt.xlabel('Epochs')
82. plt.ylabel('Loss')
83. plt.legend()
84. # 绘制训练和验证的准确率曲线
85. plt.subplot(1, 2, 2)
86. plt.plot(history.history['accuracy'], label='Train Accuracy')
87. plt.plot(history.history['val\_accuracy'], label='Val Accuracy')
88. plt.title('Accuracy vs Epochs')
89. plt.xlabel('Epochs')
90. plt.ylabel('Accuracy')
91. plt.legend()
92. plt.show()





2，LeNet网络结构如下：

（i）第1层卷积层Conv-1： 6个5\*5\*1大小的滤波器， stride=1，padding=2，接Sigmoid做激活函数；

（ii）接下来是池化层AvePool-1，它以2\*2、stride=2做Average Pooling操作；

（iii）第2层卷积层Conv-2： 16个5\*5\*1大小的滤波器， stride=1，padding=0，接Sigmoid做激活函数；

（iv）再接一个池化层AvePool--2，它以2\*2、stride=2做Average Pooling操作；

（v）对AvePool--2层输出做了Flatten操作后，与120个神经元做全连接，构成FC-1，Sigmoid做激活函数；

（vi）再与84个神经元做全连接，构成FC-2，Sigmoid做激活函数；

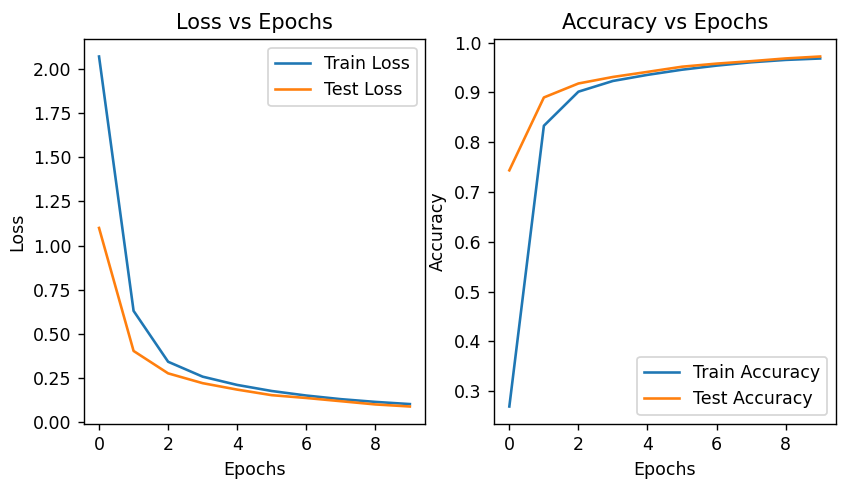
（vii）再全连接10个神经元输出，用Softmax完成10个类别的分类。

编写上述网络结构的代码，对MNIST数据集实现分类，训练时的batch size为256，一共训练10遍epoch，画出训练时的损失函数、训练集上的分类精度和测试集上的分类精度随epoch增加的变化曲线。训练完成后，在测试集上随机抽取10个样本，观察分类结果。

（1）代码实现

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import tensorflow as tf
4. from tensorflow.keras import layers, models
5. from tensorflow.keras.datasets import mnist
6. from tensorflow.keras.utils import to\_categorical
7. # 加载和预处理数据
8. (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()
9. x\_train, x\_test = x\_train / 255.0, x\_test / 255.0
10. x\_train = np.expand\_dims(x\_train, axis=-1) # (60000, 28, 28, 1)
11. x\_test = np.expand\_dims(x\_test, axis=-1) # (10000, 28, 28, 1)
12. y\_train = to\_categorical(y\_train, 10)
13. y\_test = to\_categorical(y\_test, 10)
14. # 构建LeNet模型
15. def build\_lenet\_model():
16. model = models.Sequential()
17. model.add(layers.Conv2D(6, (5, 5), strides=1, padding='same', activation='sigmoid', input\_shape=(28, 28, 1)))
18. model.add(layers.AvgPool2D(pool\_size=(2, 2), strides=2))
19. model.add(layers.Conv2D(16, (5, 5), strides=1, padding='valid', activation='sigmoid'))
20. model.add(layers.AvgPool2D(pool\_size=(2, 2), strides=2))
21. model.add(layers.Flatten())
22. model.add(layers.Dense(120, activation='sigmoid'))
23. model.add(layers.Dense(84, activation='sigmoid'))
24. model.add(layers.Dense(10, activation='softmax'))
25. return model
26. # 编译和训练模型
27. model = build\_lenet\_model()
28. model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])
29. history = model.fit(x\_train, y\_train, epochs=10, batch\_size=256, validation\_data=(x\_test, y\_test))
30. # 绘制训练曲线
31. def plot\_training\_history(history):
32. plt.figure(figsize=(12, 4))
33. plt.subplot(1, 3, 1)
34. plt.plot(history.history['loss'], label='Train Loss')
35. plt.plot(history.history['val\_loss'], label='Test Loss')
36. plt.title('Loss vs Epochs')
37. plt.xlabel('Epochs')
38. plt.ylabel('Loss')
39. plt.legend()
40. plt.subplot(1, 3, 2)
41. plt.plot(history.history['accuracy'], label='Train Accuracy')
42. plt.plot(history.history['val\_accuracy'], label='Test Accuracy')
43. plt.title('Accuracy vs Epochs')
44. plt.xlabel('Epochs')
45. plt.ylabel('Accuracy')
46. plt.legend()
47. plt.show()
48. plot\_training\_history(history)
49. # 随机抽取10个测试集样本并显示预测结果
50. random\_indices = np.random.randint(0, x\_test.shape[0], 10)
51. x\_random\_samples = x\_test[random\_indices]
52. y\_random\_samples = y\_test[random\_indices]
53. y\_pred = model.predict(x\_random\_samples)
54. plt.figure(figsize=(12, 5))
55. for i in range(10):
56. plt.subplot(2, 5, i+1)
57. plt.imshow(x\_random\_samples[i].reshape(28, 28), cmap='gray')
58. plt.title(f"True: {np.argmax(y\_random\_samples[i])}, Pred: {np.argmax(y\_pred[i])}")
59. plt.axis('off')
60. plt.show()

（2）损失函数以及分类精度的曲线



（3）在测试集上抽取10个样本，如图所示：

