神经网络上机实验

基于神经网络解决分类问题

姓名：丰华彬
学号：SA17011135

问题描述

- 使用的数据集：Z-Alizadeh Sani Data Set
- 任务描述：根据病人的检测指标判断是否患有冠状动脉疾病
- 特征数量：55个，各项生理测试指标
- 标签：1个，是否患有冠状动脉疾病
- 数据集情况：303 个实例，无缺失值

数据加载与预处理

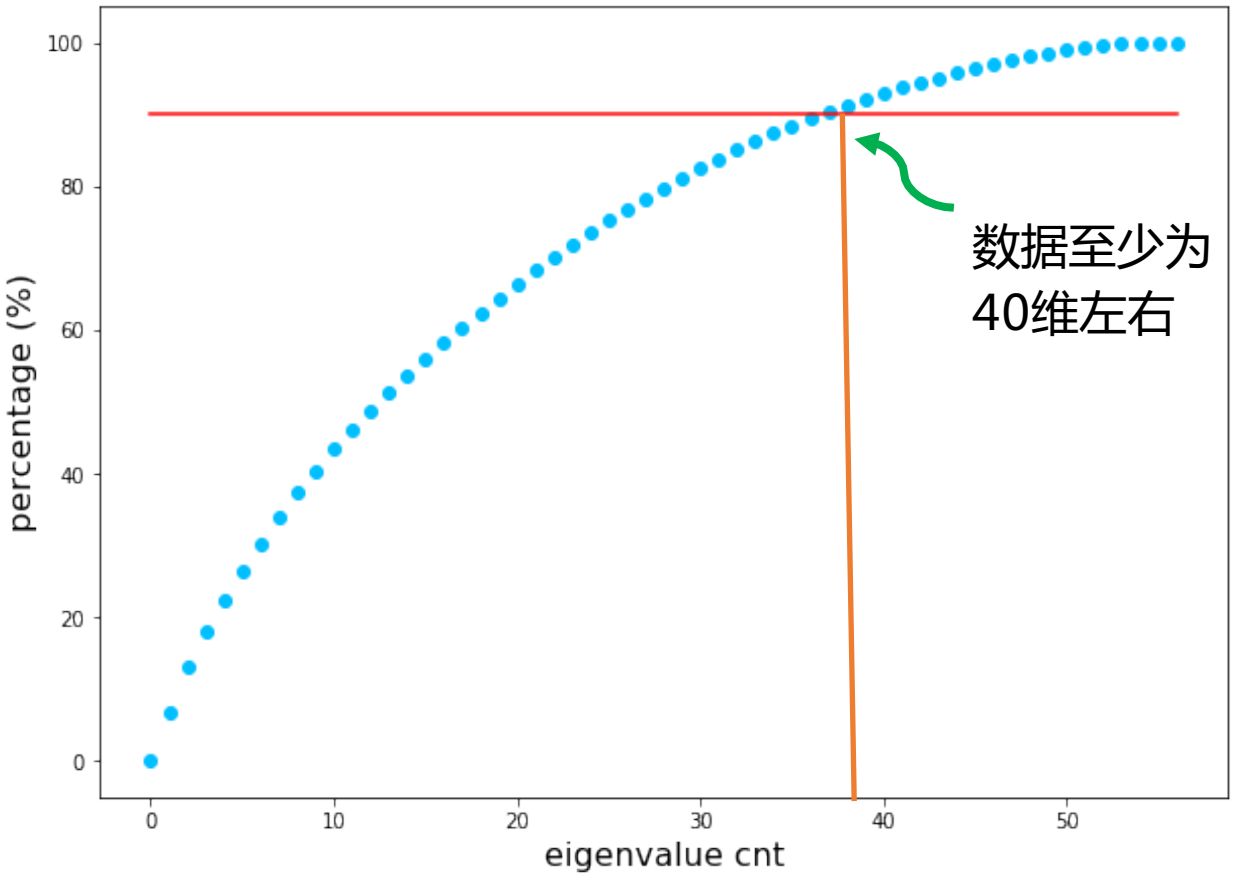
- 加载数据：Pandas 的 read_excel 函数
- 转换文本数据：例如
  - Sex 的 ‘Female’ 和 ‘Male’ 两种不同的特征转换为 0/1 数字特征；
  - BBB (瓣膜性心脏病) 有 LBBB，N，RBBB 3种情况，并且之间没有顺序，需要进行 one-hot 编码
- 数据标准化

数据预处理后

	Age	Weight	Length	Sex	BMI	DM	...	LBBB	RBBB	Cath
0	-0.567507	1.348781	1.102509	0.848062	0.521953	-0.648954	...	-0.211376	-0.164405	1
1	0.779647	-0.319644	-0.827235	-1.175267	0.280658	-0.648954	...	-0.211376	-0.164405	1
2	-0.471282	-1.654383	-0.076779	0.848062	-1.749510	-0.648954	...	-0.211376	-0.164405	1
3	0.683422	-0.569907	-0.720027	-1.175267	-0.099952	-0.648954	...	-0.211376	-0.164405	0
4	-0.856183	1.098517	-1.256067	-1.175267	2.419415	-0.648954	...	-0.211376	-0.164405	0
5	-0.856183	0.097462	1.102509	0.848062	-0.673002	-0.648954	...	-0.211376	-0.164405	1
6	-0.375056	0.514569	0.030429	0.848062	0.521222	-0.648954	...	-0.211376	-0.164405	1
7	1.260774	0.514569	1.102509	0.848062	-0.274683	1.535857	...	-0.211376	-0.164405	1
8	-0.086380	0.848253	-0.183987	-1.175267	1.065528	-0.648954	...	-0.211376	-0.164405	0
9	0.106070	-0.236222	0.566469	0.848062	-0.654051	1.535857	...	4.715302	-0.164405	1
10	-0.086380	0.097462	0.352053	0.848062	-0.164731	-0.648954	...	-0.211376	-0.164405	1
11	2.030576	-0.569907	-1.256067	-1.175267	0.335003	-0.648954	...	-0.211376	-0.164405	1
12	1.068323	-0.319644	-1.470483	-1.175267	0.842201	1.535857	...	-0.211376	-0.164405	1
13	0.779647	-0.903592	-1.148859	-1.175267	-0.166874	1.535857	...	-0.211376	-0.164405	1
14	0.683422	-0.903592	-1.041651	-1.175267	-0.250229	1.535857	...	-0.211376	-0.164405	1
15	0.009845	0.597990	0.244845	0.848062	0.438016	1.535857	...	-0.211376	-0.164405	1

测试用PCA进行数据降维

- 对数据集的特征部分X 对应的 $X^T X$ 进行特征分解，特征值的累加情况为：

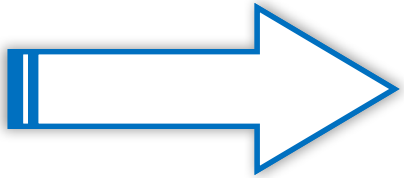


输出用one-hot编码

- 为了使所编写的神经网络能够用于任意的分类，我们需要将输出也进行 one-hot 编码

- 标签 Cath 变为两列

	Cath
0	1
1	1
2	1
3	0
4	0



	cath0	cath1
0	0	1
1	0	1
2	0	1
3	1	0
4	1	0

创建神经网络类

- 神经网络结构
  - 初始化接口

```
# 神经网络结构的初始化
# sizes: 序列，每个值代表一层神经网络节点个数，第1层为输入层，最后1层为输出层
# activate: 激活函数，默认使用 sigmoid函数
# activate_prime: 激活函数对应的微分
# cost_derivative: 损失函数导数，默认使用 MSE 损失函数的导数
def __init__(self, sizes, activate_fun=None,
             activate_prime=None, cost_derivative=None):
```

  - 训练接口

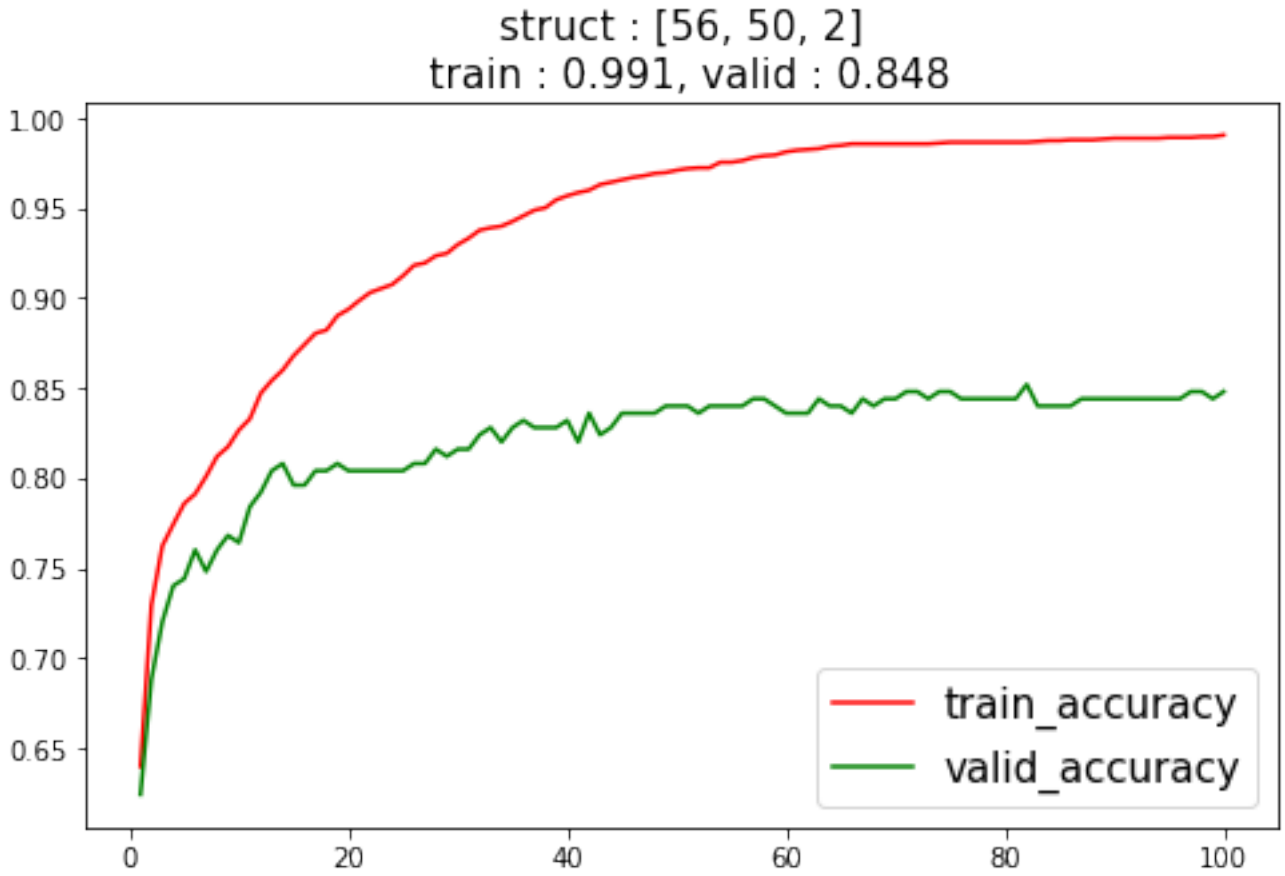
```
# epoches: 训练的轮数
# mini_batch_size: SGD 批量的大小
# eta: 学习率
# 返回: 一个序列，每个值是一次训练之后测试的准确率
def train(self, train_x, train_y, epochs, mini_batch_size, eta,
          valid_x=None, valid_y=None):
```

K-折 交叉验证选择超参数

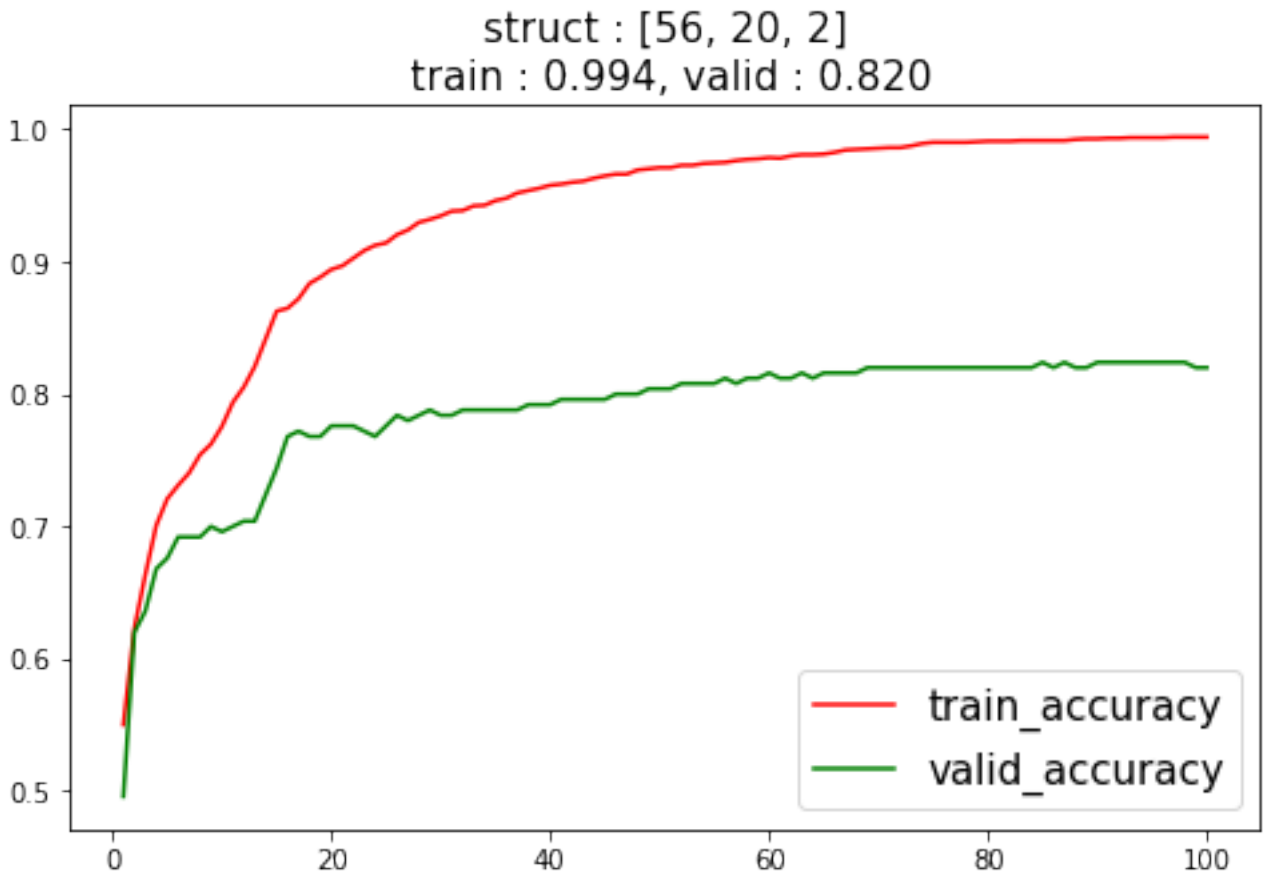
- 记录每一折迭代中的训练集上准确率和验证集上准确率的变化
- 若一共进行 10 折，每一折训练100次，将产生 1000 对的准确率，将每一折的准确率取平均值并进行绘制

```
# net_struct 神经网络结构
# data_x, data_y 训练集数据
# eta 学习率
# epoches 训练次数
# n_fold 折数
# 返回每次训练后 K 折的训练集上的准确率和验证集上的准确率
def kfold(net_struct, data_x, data_y, eta, epoches ,n_fold=10):
```

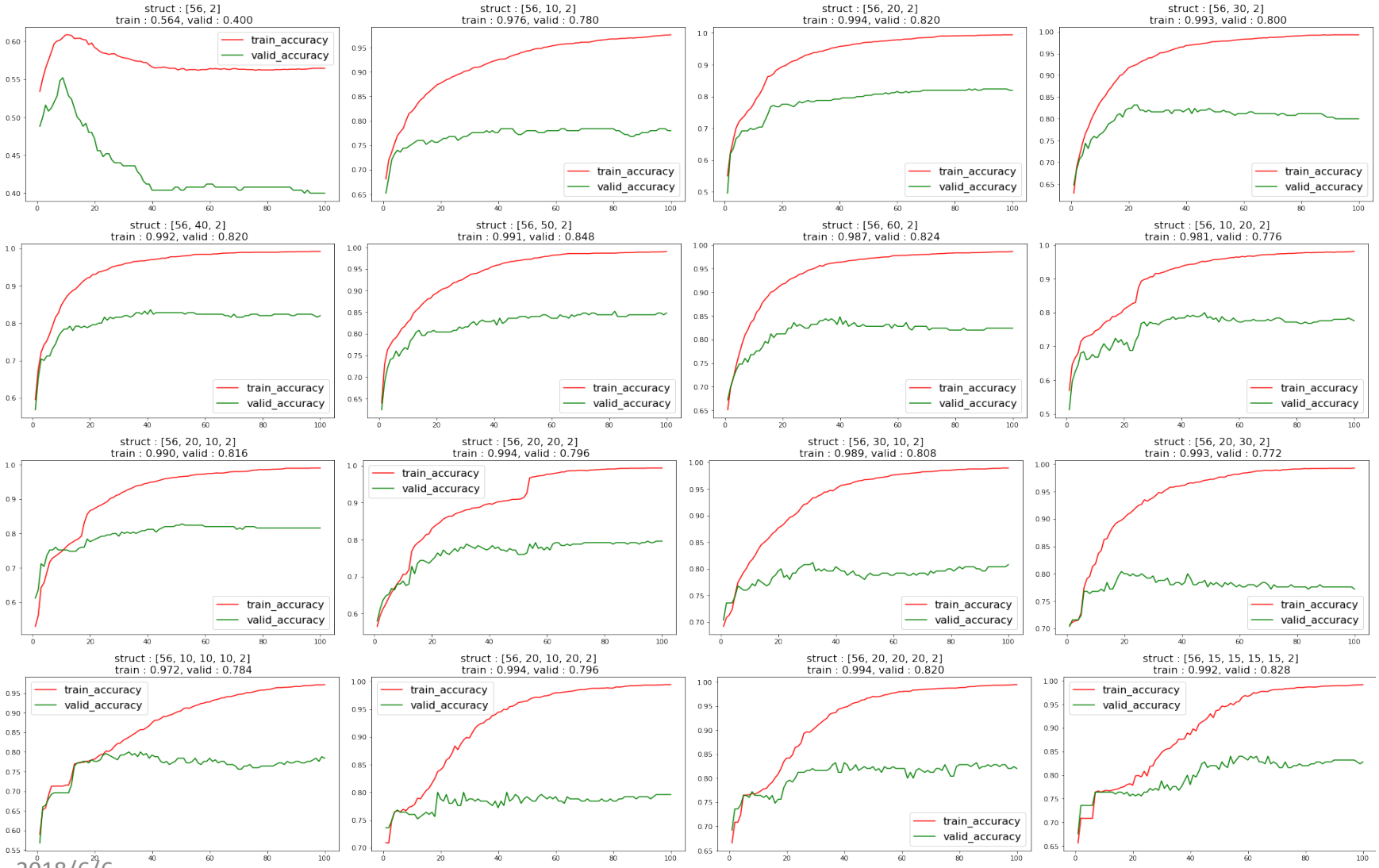
50 个隐层单元


struct : [56, 50, 2]
train : 0.991, valid : 0.848
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
20
40
60
80
100
train_accuracy
valid_accuracy

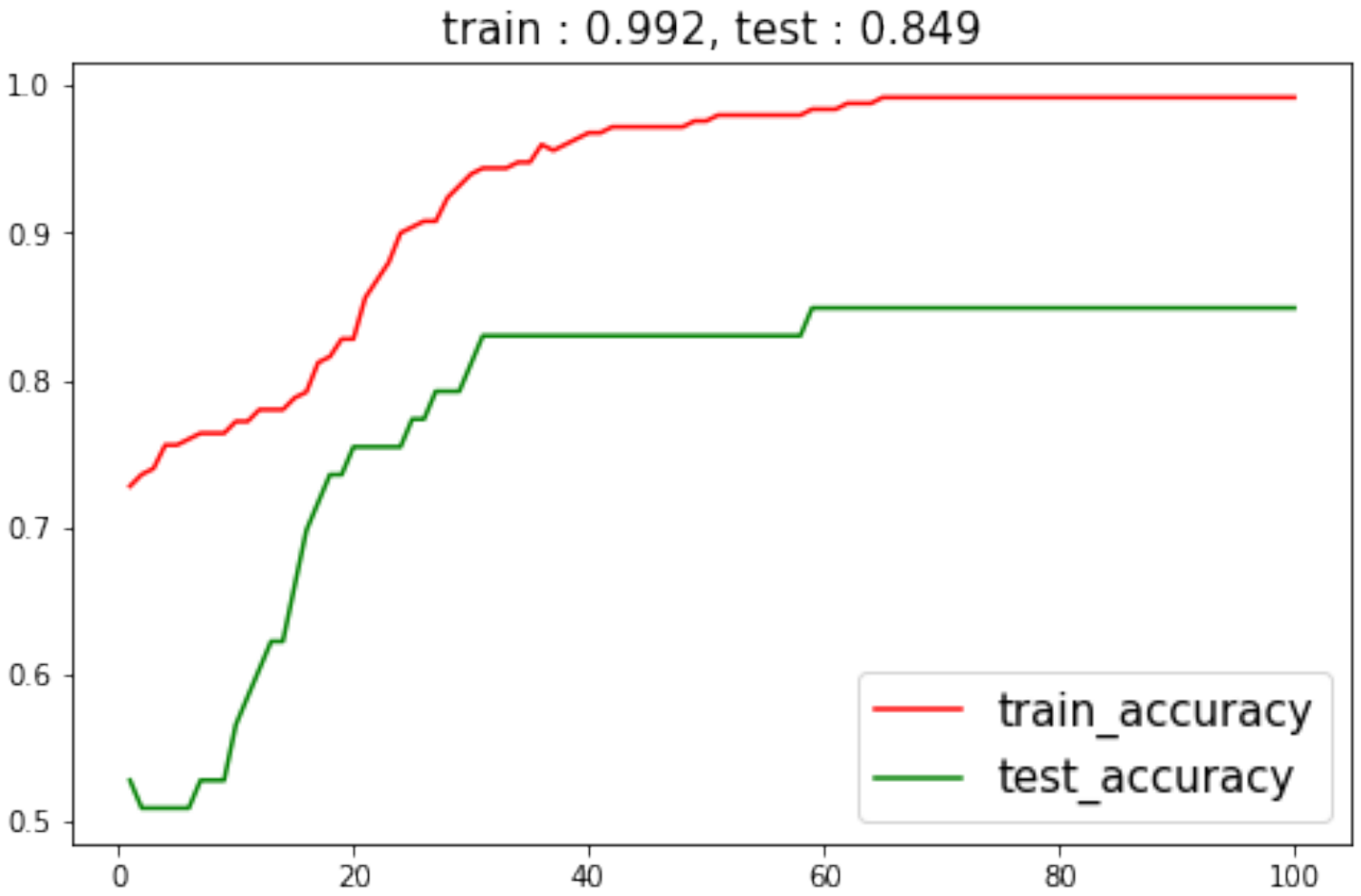
20 个隐层单元


struct : [56, 20, 2]
train : 0.994, valid : 0.820
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100
train_accuracy
valid_accuracy

训练的其他结构


struct : [56, 2]
train : 0.564, valid : 0.400
struct : [56, 10, 2]
train : 0.976, valid : 0.780
struct : [56, 20, 2]
train : 0.994, valid : 0.820
struct : [56, 30, 2]
train : 0.993, valid : 0.800
struct : [56, 40, 2]
train : 0.992, valid : 0.820
struct : [56, 50, 2]
train : 0.991, valid : 0.848
struct : [56, 60, 2]
train : 0.987, valid : 0.824
struct : [56, 10, 20, 2]
train : 0.981, valid : 0.776
struct : [56, 20, 10, 2]
train : 0.990, valid : 0.816
struct : [56, 20, 20, 2]
train : 0.994, valid : 0.796
struct : [56, 30, 10, 2]
train : 0.989, valid : 0.808
struct : [56, 20, 30, 2]
train : 0.993, valid : 0.772
struct : [56, 10, 10, 10, 2]
train : 0.972, valid : 0.784
struct : [56, 20, 10, 20, 2]
train : 0.994, valid : 0.796
struct : [56, 20, 20, 20, 2]
train : 0.994, valid : 0.820
struct : [56, 15, 15, 15, 15, 2]
train : 0.992, valid : 0.828
train_accuracy
valid_accuracy

使用选定的超参数进行训练



使用 MLPClassifier训练

- 构建类似的神经网络

```
from sklearn import neural_network
clf = neural_network.MLPClassifier(hidden_layer_sizes=(50),
        activation='logistic',solver='sgd',alpha=0.5,
        batch_size=10,learning_rate='constant',max_iter=1000)
```

- 训练与测试

```
clf.fit(train_X, train_y)
sum(clf.predict(test_X)==test_y)/len(test_y)
```

准确率为：0.83

测试多组网络结构

```
from sklearn import neural_network
nets = [(10,),(20,),(30,),(40,),(50,),(60,),(10,20),(20,10),(20,20),
       (30,10),(20,30),(10,10,10),(20,10,20),(20,20,20),(15,15,15,15)]
for net_struct in nets:
    clf = neural_network.MLPClassifier(hidden_layer_sizes=net_struct,
                                       activation='logistic',
                                       solver='sgd',
                                       alpha=0.5,
                                       batch_size=10,
                                       learning_rate='constant',
                                       max_iter=1000)
    clf.fit(train_X, train_y)
    print(f'{net_struct} : {sum(clf.predict(test_X)==test_y)/len(test_y)}')
```

(10,) : 0.8490566037735849
(20,) : 0.8490566037735849
(30,) : 0.8490566037735849
(40,) : 0.8490566037735849
(50,) : 0.8490566037735849
(60,) : 0.8490566037735849
(10, 20) : 0.7735849056603774
(20, 10) : 0.7735849056603774
(20, 20) : 0.7735849056603774
(30, 10) : 0.7735849056603774
(20, 30) : 0.7735849056603774
(10, 10, 10) : 0.7735849056603774
(20, 10, 20) : 0.7735849056603774
(20, 20, 20) : 0.7735849056603774
(15, 15, 15, 15) : 0.7735849056603774

使用默认参数参数

```
from sklearn import neural_network
nets = [(10,),(20,),(30,),(40,),(50,),(60,),(10,20),(20,10),(20,20),
        (30,10),(20,30),(10,10,10),(20,10,20),(20,20,20),(15,15,15,15)]
for net_struct in nets:
    clf = neural_network.MLPClassifier()
    clf.fit(train_X, train_y)
    print(f'{net_struct} : {sum(clf.predict(test_X)==test_y)/len(test_y)}')
```

(10,) : 0.9056603773584906
(30,) : 0.9056603773584906
(50,) : 0.8867924528301887
(10, 20) : 0.9245283018867925
(20, 20) : 0.9056603773584906
(20, 30) : 0.9245283018867925
(20, 10, 20) : 0.9056603773584906
(15, 15, 15, 15) : 0.8867924528301887

(20,) : 0.9056603773584906
(40,) : 0.9056603773584906
(60,) : 0.8867924528301887
(20, 10) : 0.9056603773584906
(30, 10) : 0.9245283018867925
(10, 10, 10) : 0.8867924528301887
(20, 20, 20) : 0.8867924528301887

谢谢！