[1]:
```
import numpy as np
import pandas as pd
from scipy.stats import zscore
```

[2]:
```
def sigmoid(z):
    """The sigmoid function."""
    return 1.0 / (1.0 + np.exp(-z))

def sigmoid_d(z):
    """Derivative of the sigmoid function."""
    s = sigmoid(z)
    return s * (1 - s)
```

[3]:
```
class NeuralNetwork:
    def __init__(self, input_dim, hidden_dim, out_dim, g=sigmoid, g_d=sigmoid_d):
        self.W_ih = 0.1 * np.random.rand(hidden_dim, input_dim)  # 输入层到隐含层的权重矩阵
        self.b_ih = 0.1 * np.random.rand(hidden_dim)             # 输入层到隐含层的偏置
        self.W_ho = 0.1 * np.random.rand(out_dim, hidden_dim)    # 隐含层到输出层的权重矩阵
        self.b_ho = 0.1 * np.random.rand(out_dim)                # 隐含层到输出层的偏置

        self.g = g      # 激活函数
        self.g_d = g_d  # 激活函数的梯度

    def feedForward(self, x):
        """ 输入 x，前馈产生输出。"""
        self.x = x                                      # 输入
        self.in_h = self.W_ih @ self.x + self.b_ih      # 隐含层输入
        self.out_h = self.g(self.in_h)                  # 隐含层输出
        self.in_o = self.W_ho @ self.out_h + self.b_ho  # 输出层输入
        self.out_o = self.g(self.in_o)                  # 输出层输出，即网络最终输出
        return self.out_o

    def backPropagate(self, target):
        """ 反向传播并产生各层敏感度。"""
        self.delta_o = (self.out_o - target) * self.g_d(self.in_o)         # 输出层敏感度
        self.delta_h = (self.W_ho.T @ self.delta_o) * self.g_d(self.in_h)  # 隐含层敏感度

    def update(self, rate):
        """ 更新各个参数。"""
        self.W_ho -= rate * (np.mat(self.delta_o).T @ np.mat(self.out_h))
        self.b_ho -= rate * self.delta_o
        self.W_ih -= rate * (np.mat(self.delta_h).T @ np.mat(self.x))
        self.b_ih -= rate * self.delta_h

    def predict(self, x):
        """ 对输入特征 x 进行预测，返回预测结果的下标。"""
        in_h = self.W_ih @ x + self.b_ih
        out_h = self.g(in_h)
        in_o = self.W_ho @ out_h + self.b_ho
        out_o = self.g(in_o)
        return out_o.argmax()  # 返回结果中最大值的下标
```

---

[4]:

```
data_df = pd.read_csv('dataset/horse-colic-data.csv')
data_df['outcome'] = data_df.pop('outcome')
for column in data_df.columns:
    if column == 'outcome' or data_df[column].var() == 0:
        continue
#     data_df[column] = zscore(data_df[column])
    data_df[column] = (data_df[column] - data_df[column].min()) / (data_df[column].max() - data_df[column].min())  # 归一化
# data_df

test_df = pd.read_csv('dataset/horse-colic-test.csv')
test_df['outcome'] = test_df.pop('outcome')
for column in test_df.columns:
    if column == 'outcome' or test_df[column].var() == 0:
        continue
#     test_df[column] = zscore(test_df[column])
    test_df[column] = (test_df[column] - test_df[column].min()) / (test_df[column].max() - test_df[column].min())  # 归一化
# test_df
```

[5]:

```
EPOCHS = 400
RATE = 0.01  # 学习率
nn = NeuralNetwork(35, 12, 3)

for epoch in range(EPOCHS):
    for i in range(len(data_df)):
        sample = data_df.iloc[i].to_numpy()
        x = sample[:-1]
        target = np.zeros(3)
        target[int(sample[-1])-1] = 1

        output = nn.feedForward(x)
        nn.backPropagate(target)
        nn.update(RATE)

    data_df = data_df.sample(frac=1)  # 打乱样本顺序
    RATE *= 0.99  # 减少学习率
```

[6]:

```
count = 0
for i in range(len(test_df)):
    sample = test_df.iloc[i].to_numpy()
    x, target = sample[:-1], sample[-1]
    if nn.predict(x) + 1 == target:
        count += 1
print('{} / {} = {:.2%}'.format(count, len(test_df), count/len(test_df)))
```

```
51 / 68 = 75.00%
```

[ ]:

```

```