#《数学软件与实践》实验报告/结果说明文档 (MS-Proj1)

中国人民大学 信息学院 崔冠宇 2018202147

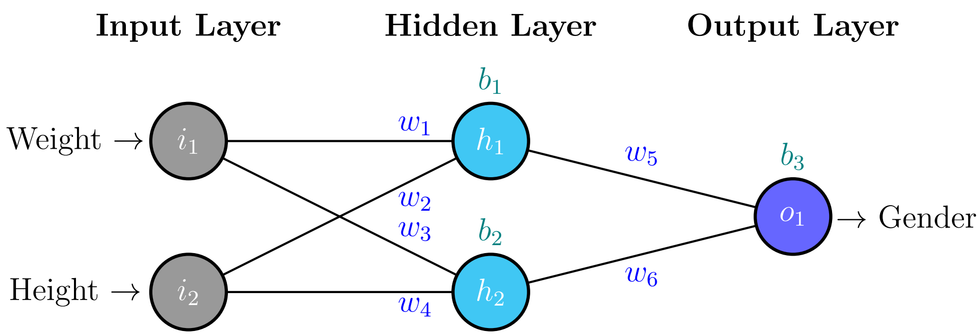
一、实验题目：机器学习实例——基于身高体重的性别预测.

二、实验目的:

- 了解机器学习中神经网络的数学原理;
- 了解随机梯度下降法在神经网络训练中的应用;
- 实践 Python 编程, 以同学的身高体重数据预测性别.

三、实验方法与步骤:

1. 首先使用 matplotlib 绘制训练集数据的体重－身高散点图, 观察数据分布, 对训练集的情况做一个初步了解.
2. 然后以老师提供的 Python 脚本为基础, 建立如下图所示的神经网络.



身高体重－性别神经网络 示意图.

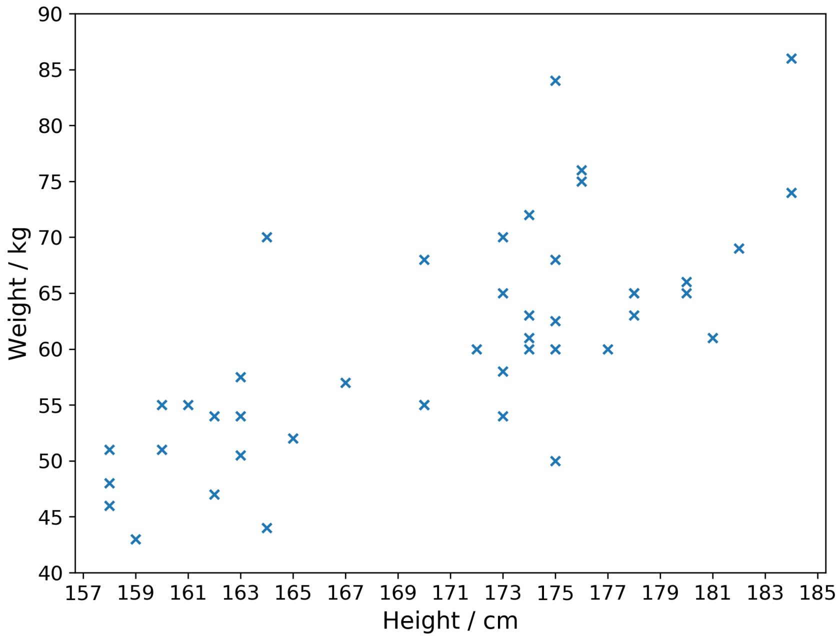
3. 之后在训练集上用随机梯度下降法 (见下) 训练神经网络, 绘制损失－训练轮数 (Loss-Epoch) 以及预测正确率－训练轮数 (Correctness-Epoch) 图线.

Algorithm 随机梯度下降法 (SGD)

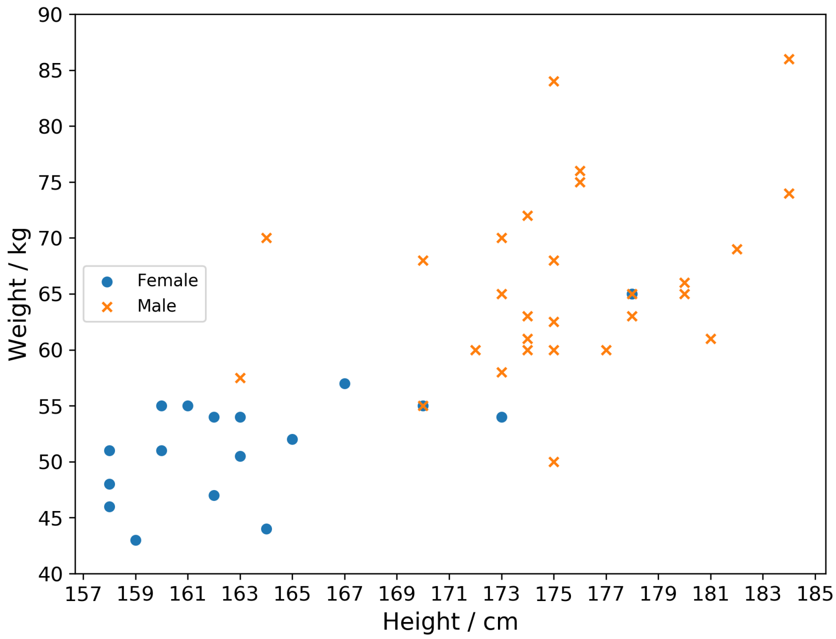
$\mathbf{w}, \mathbf{b} \leftarrow$ Random values
$\eta \leftarrow \eta_0, N \leftarrow N_0$
repeat
  for $(\mathbf{x}_i, y_i)$ in dataset **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}, \mathbf{b})$
    $\mathbf{b} \leftarrow \mathbf{b} - \eta \nabla_{\mathbf{b}} L(\mathbf{w}, \mathbf{b})$
  end for
until Converge or reach given times N.

四、实验结果及分析说明:

运行 [**附录A**] 中的 **scatter.py**, 得到训练数据集的两张体重—身高散点图, 如下图所示:



(a) 不分性别的体重—身高散点图.



(b) 分性别着色的体重—身高散点图.

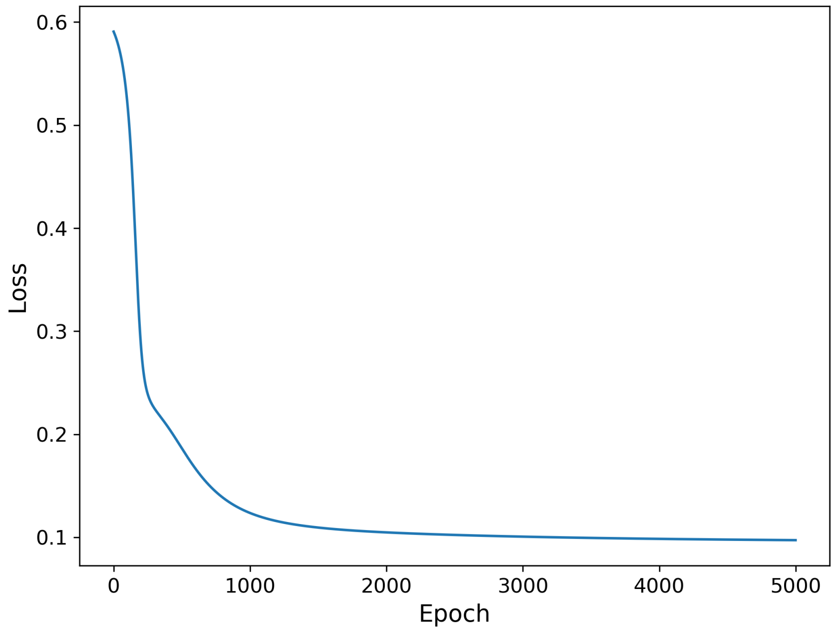
Figure 1: **scatter.py** 运行结果.

通过散点图可以看出, 在我们收集的样本中 ——

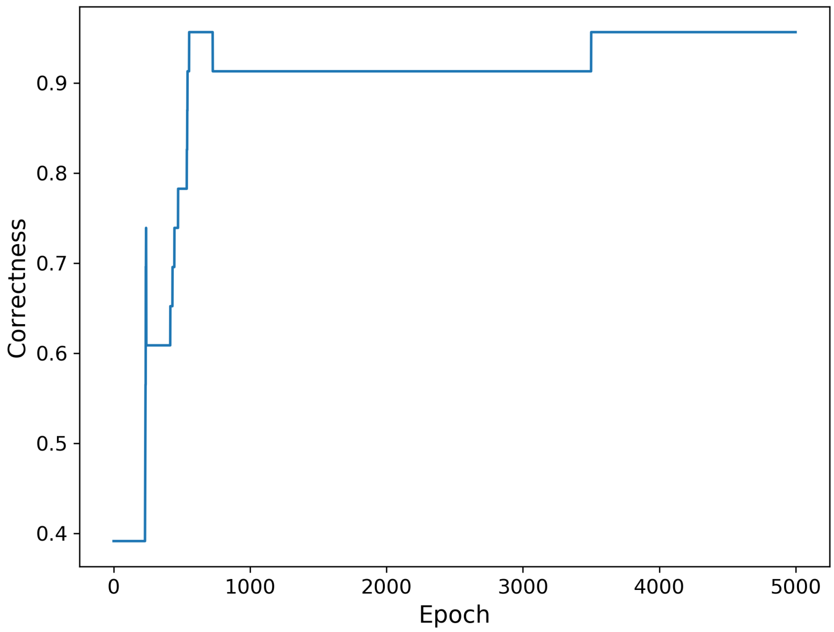
- 无论性别, 身高和体重有明显的正相关关系;
- 存在两个"可疑点"(大约在 (170, 55) 和 (178, 65)), 有不同性别的数据重合了;

- 除去“可疑点”之外, 男生女生之间有着较为明确的分类边界;
- 女生样本较为集中地分布在图表左下方, 而男生样本较为分散地分布在图表右上方.

运行 [**附录A**] 中的 **NNPython.py**, 同时得到训练集损失－训练轮数 (Loss-Epoch) 曲线图以及测试集预测正确率－训练轮数 (Correctness-Epoch) 曲线图, 如下图所示:



(a) 训练集上的损失－训练轮数(Loss-Epoch)曲线图.



(b) 测试集上的预测正确率－训练轮数(Correctness-Epoch)曲线图.

Figure 2: **NNPython.py** 运行结果.

在程序中, 为了更公平、更准确地测试我们的模型, 随机选取一半样本作为训练集, 另一半作为测试集.

首先说明一下程序运行时几个参数的值：这两张图是在总训练轮数 (epochs) 为 5000，学习率 (learn_rate) 为 0.005，每隔 1 轮进行一次统计 (epoch_gap) 的条件下运行得到的.

其次还要指出，由于数据集样本数太少 (45个样本)，以及数据质量不高，有疑似错误的样本 (2个)，导致预测结果较不稳定，最终预测准确度在 0.83-0.96 之间徘徊.

然后我们分析一下这两张图 ——

- 通过第一幅曲线图可以看出，损失对于训练轮数来说，是一个减函数，而且初期下降的比较快，后来逐渐变慢.
- 通过第二幅曲线图可以看出，预测正确率随着训练轮数的增加整体呈上升趋势，但是每次结果并不完全一致，而且还会出现不升反降的情况. 这主要是由于样本数量较少，导致程序结果不稳定，预测结果错误一个都会导致正确率发生较大改变.

最后，尽管我们的模型并没有达到 100% 的准确率，而且准确率也不能稳定的达到 90% 以上. 但我相信，如果在一个比较大的数据集上，我们的程序很可能可以达到 90% 以上的准确率.

附录A. 程序清单：

- **绘制数据散点图 —— scatter.py:**

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_excel('data.xlsx')

# Scatter
plt.figure(figsize = (8, 6))
plt.scatter(data['身高/cm'], data['体重/kg'], s = 30, marker = 'x')
#plt.title('Weight-Height Scatter', size = 16)
plt.xlabel('Height / cm', size = 14)
plt.ylabel('Weight / kg', size = 14)
plt.xticks(range(157, 187, 2), size = 12)
plt.yticks(range(40, 95, 5), size = 12)
plt.savefig("scatter0.png", dpi = 300)
plt.show()

```

```python
# Colored scatter
grouped = data.groupby("性别")
girls = grouped.get_group('女')
boys = grouped.get_group('男')

plt.figure(figsize = (8, 6))
plt.scatter(girls['身高/cm'], girls['体
    重/kg'], s = 30, marker = 'o', label = 'Female')
plt.scatter(boys['身高/cm'], boys['体
    重/kg'], s = 30, marker = 'x', label = 'Male')
plt.legend(loc = 6)
#plt.title('Weight-Height Scatter', size = 16)
plt.xlabel('Height / cm', size = 14)
plt.ylabel('Weight / kg', size = 14)
plt.xticks(range(157, 187, 2), size = 12)
plt.yticks(range(40, 95, 5), size = 12)
plt.savefig("scatter1.png", dpi = 300)
plt.show()
```

- 神经网络建构、训练与测试 —— **NNPython.py:**

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy.interpolate import make_interp_spline, BSpline

def sigmoid(x):
    # Sigmoid activation function: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
    # Derivative of sigmoid: f'(x) = f(x) * (1 - f(x))
    fx = sigmoid(x)
    return fx * (1 - fx)

def mse_loss(y_true, y_pred):
    # y_true and y_pred are numpy arrays of the same length.
```

```python
    return ((y_true - y_pred) ** 2).mean()

class OurNeuralNetwork:
    '''
    A neural network with:
        - 2 inputs
        - a hidden layer with 2 neurons (h1, h2)
        - an output layer with 1 neuron (o1)

    *** DISCLAIMER ***:
    The code below is intended to be simple and educational, NOT optimal.
    Real neural net code looks nothing like this. DO NOT use this code.
    Instead, read/run it to understand how this specific network works.
    '''
    def __init__(self):
        # Weights
        self.w1 = np.random.normal()
        self.w2 = np.random.normal()
        self.w3 = np.random.normal()
        self.w4 = np.random.normal()
        self.w5 = np.random.normal()
        self.w6 = np.random.normal()

        # Biases
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        # x is a numpy array with 2 elements.
        h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
        h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
        o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
```

```python
        return o1

    def train(self, data, all_y_trues, epochs = 1000, learn_rate = 1, epoch_gap = 10):
        '''
        - data is a (n x 2) numpy array, n = # of samples in the dataset.
        - all_y_trues is a numpy array with n elements.
            Elements in all_y_trues correspond to those in data.
        '''

        # Split dataset into train and test
        X_train, X_test, y_train, y_test = train_test_split(data, all_y_trues, test_size = 0.50)

        '''
        X_train = data[0::2]
        X_test = data[1::2]
        y_train = all_y_trues[0::2]
        y_test = all_y_trues[1::2]
        '''

        # Arrays used to plot later
        epoch_arr = []
        loss_arr = []
        corr_arr = []

        for epoch in range(epochs):
            for x, y_true in zip(X_train, y_train):
                # --- Do a feedforward (we'll need these values later)
                sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
                h1 = sigmoid(sum_h1)

                sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.
```

```
b2
            h2 = sigmoid(sum_h2)

            sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
            o1 = sigmoid(sum_o1)
            y_pred = o1

            # --- Calculate partial derivatives.
            # --- Naming: d_L_d_w1 represents "partial L / partial w1"
            d_L_d_ypred = -2 * (y_true - y_pred)

            # Neuron o1
            d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
            d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
            d_ypred_d_b3 = deriv_sigmoid(sum_o1)

            d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
            d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

            # Neuron h1
            d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
            d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
            d_h1_d_b1 = deriv_sigmoid(sum_h1)

            # Neuron h2
            d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
            d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
            d_h2_d_b2 = deriv_sigmoid(sum_h2)

            # --- Update weights and biases
            # Neuron h1
            self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
            self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
```

```
                self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

                # Neuron h2
                self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
                self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
                self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

                # Neuron o1
                self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
                self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
                self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

            # --- Calculate total loss at the end of each epoch
            if epoch % epoch_gap == 0:
                # Loss on train dataset
                y_train_preds = np.apply_along_axis(self.feedforward, 1, X_train)
                loss = mse_loss(y_train, y_train_preds)
                # Correctness on the test dataset
                corr_test = correctness(y_test, self.predict(X_test))
                print("Epoch %d, Train Loss: %.5f, Test Correctness: %.5f" % (epoch, loss, corr_test))

                epoch_arr.append(epoch)
                loss_arr.append(loss)
                corr_arr.append(corr_test)

        # Plot
```

```
        epoch_arr = np.array(epoch_arr)
        loss_arr = np.array(loss_arr)
        corr_arr = np.array(corr_arr)

        plt.figure(figsize = (8, 6))
        plt.plot(epoch_arr, loss_arr)
        plt.xlabel('Epoch', size = 14)
        plt.ylabel('Loss', size = 14)
        plt.xticks(size = 12)
        plt.yticks(size = 12)
        plt.savefig("NNPython1.png", dpi = 300)
        plt.show()

        plt.figure(figsize = (8, 6))
        plt.plot(epoch_arr, corr_arr)
        plt.xlabel('Epoch', size = 14)
        plt.ylabel('Correctness', size = 14)
        plt.xticks(size = 12)
        plt.yticks(size = 12)
        plt.savefig("NNPython2.png", dpi = 300)
        plt.show()

    def predict(self, X_test):
        y_test_preds = np.where(np.apply_along_axis(self.feedforward, 1, X_test) > 0.5, 1, 0)
        return y_test_preds

def normalize(x):
    # Return a normalized vector of x
    return (x - x.mean()) / x.std()

def correctness(y_test, y_test_preds):
    # Return the correctness between y_test and y_test_preds
    return np.mean(y_test == y_test_preds)

df = pd.read_excel('data.xlsx')
```

```
data = np.zeros((df.shape[0], 2))

# Normalize vectors
data[:, 0] = normalize(df['身高/cm'].values)
data[:, 1] = normalize(df['体重/kg'].values)

df['性别'].replace('男', '0', inplace = True)
df['性别'].replace('女', '1', inplace = True)
all_y_trues = np.array(df['性别'].astype(int))


# Train our neural network!
network = OurNeuralNetwork()
network.train(data, all_y_trues, epochs = 5000, learn_rate =
    0.005, epoch_gap = 1)
```