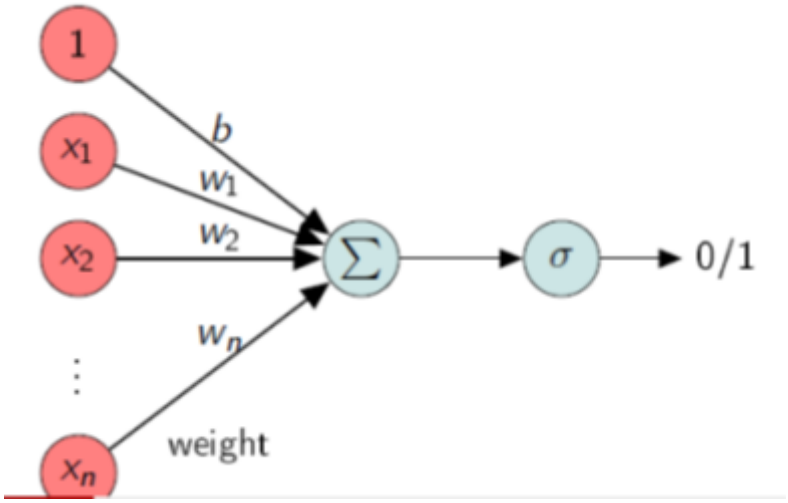
Experiment 1 Perceptron

Perceptron

- Given a set of data, $D=\{\boldsymbol{x}_i, y_i\}, \boldsymbol{x}_i \in R^n, y_i \in \{0,1\}, i=1,2,\cdots,N$. The updating rule of Perceptron is:
  - Calculate the activation output:
    - $p_i = f\left(\boldsymbol{w}^t \cdot \boldsymbol{x}_i + b\right), f(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$
  - Update the weights:
    - $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \left(y_i - p_i\right)\boldsymbol{x}_i, b^{t+1} = b^t + \left(y_i - p_i\right)$
- 

Experiment

- Define a two-class problem, including 30 positive data and 30 negative data
- 首先使用随机函数随机生成一条直线 $y = wx + b$，然后在这条直线附近生成样本点
- 代码实现如下

```
# 生成数据集
import numpy as np
import matplotlib.pyplot as plt

# 训练数据个数
train_N = 60

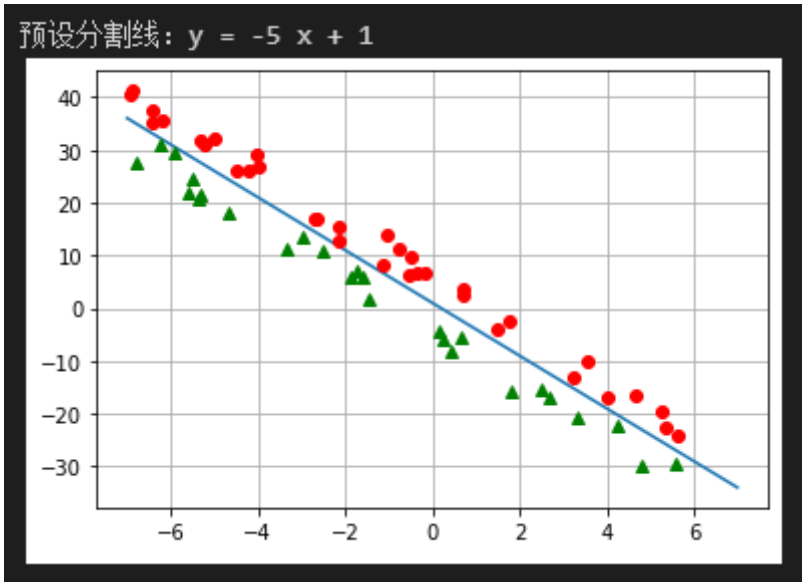
# y = wx + b
w = np.random.randint(-5, 5)
b = np.random.randint(-5, 5)
print('预设分割线: y =',w,'x +',b)
plt.plot([-7,7],[-7*w+b,7*w+b]) # 绘制直线

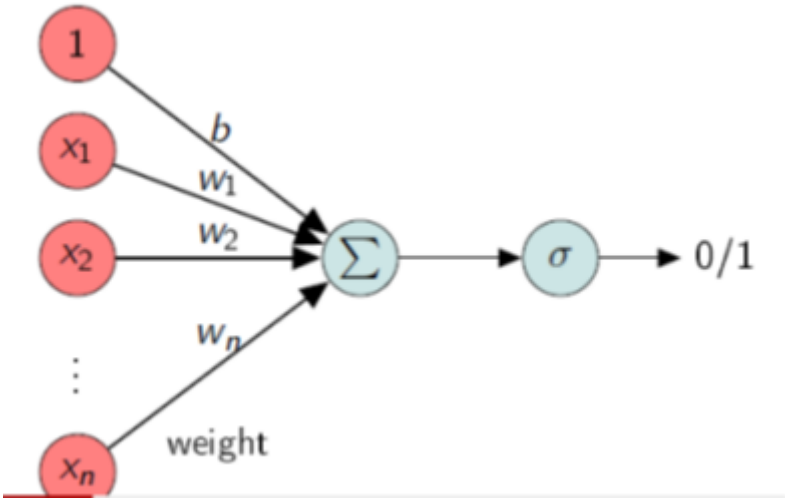
# data_x : [[x,y],label]
data_x = []

i = 0
while i < train_N:
    tmp_x = np.random.uniform(-7,7)
```

```
    tmp_y = tmp_x * w + b + np.random.uniform(-8,8)
    if abs(tmp_y - (tmp_x * w + b)) <= 1:
        continue
    # 正样本
    if tmp_y > tmp_x * w + b:
        plt.plot(tmp_x,tmp_y,'ro')  # 绘制点
        data_x.append([[tmp_x,tmp_y],1])
    # 负样本
    elif tmp_y < tmp_x * w + b:
        plt.plot(tmp_x,tmp_y,'g^')  # 绘制点
        data_x.append([[tmp_x,tmp_y],0])
    i += 1

plt.grid(True)  # 绘制
```

- 

- 样本数据的存储结构为 $data_x$ ：$[[x,y],label]$
- 网络的训练目标是训练参数 $\boldsymbol{w},b$ ，对于任意输入 $\boldsymbol{x}$ ，能够正确区分其为正样本或是负样本
  - 

- 然后实现Perceptron，将其封装成类
  - 首先对预测参数进行初始化，设置学习率 learningRate = 0.1

- ```
  def __init__(self):
      super(Perceptron, self).__init__()
      # 随机初始化
      self.w = [np.random.randint(-5, 5) for i in range(2)]
      self.b = np.random.randint(-5, 5)
      self.learningRate = 0.1
  ```

- 然后实现正向传播和反向优化
  - Calculate the activation output:
    - $p_i = f\left(\boldsymbol{w}^t \cdot \boldsymbol{x}_i + b\right), f(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$
  - Update the weights:
    - $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \left(y_i - p_i\right)\boldsymbol{x}_i, b^{t+1} = b^t + \left(y_i - p_i\right)$
  - 根据上述公式，进行代码实现
  - ```
    def response(self, x):
        """计算预测结果：求和，激活"""
        y = sum([i * j for i, j in zip(self.w, x)]) + self.b
        if y >= 0:
            return 1
        else:
            return 0

    def updateWeights(self, x, iterError):
        """
            更新参数权重
            w(t+1) = w(t) + (yi - pi) * xi
            b(t+1) = b(t) + (yi - pi)
        """
        # self.w += self.learningRate * iterError * int(x)
        self.w = [i + self.learningRate * iterError * j for i, j in
    zip(self.w, x)]
        self.b += self.learningRate * iterError
    ```

- 在网络训练部分，对每一个训练样本，进行预测，如果预测错误，则进行权重更新。直到本次epoch，所有预测结果都准确
  - ```
    def train(self, data):
        """
            训练网络
            """
        lear_flag = True
        iteration = 0

        while lear_flag:

            globalError = 0.0
            for data in data_x:
                # 计算预测结果
                r = self.response(data[0])
                if data[1] != r: # 预测错误，更新权重
                    iterError = data[1] - r
                    self.updateWeights(data[0], iterError)
                    globalError += 1
            iteration += 1
    ```
    ```

```
            print(f'Epoch {iteration} finished, accuracy is
{str((train_N - globalError) / train_N * 100)}%')
            if globalError == 0.0 or iteration >= 100: # 判定学习结束条件
                print('iterations:',iteration)
                lear_flag = False # 停止学习
```

- ```
  # 网络训练
  p = Perceptron()
  p.train(data_x)

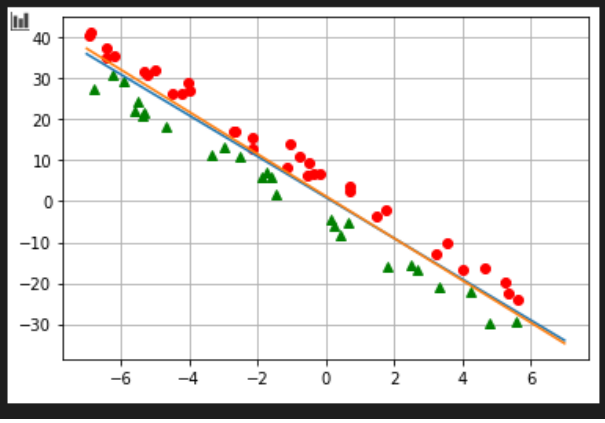
  # Epoch 1 finished, accuracy is 46.666666666666664%
  # Epoch 2 finished, accuracy is 53.333333333333336%
  # Epoch 3 finished, accuracy is 55.00000000000001%
  # Epoch 4 finished, accuracy is 58.333333333333336%
  # Epoch 5 finished, accuracy is 75.0%
  # Epoch 6 finished, accuracy is 80.0%
  # Epoch 7 finished, accuracy is 100.0%
  # iterations: 7
  ```

- ```
  # 可视化
  print('预设分割线：y =',w,'x +',b)
  print('预测分割线：', p.w[0],'* x + ', p.w[1],'* y =', - p.b)

  for data in data_x:
      if data[1] == 0:
          plt.plot(data[0][0],data[0][1],'g^')
      else:
          plt.plot(data[0][0],data[0][1],'ro')

  plt.plot([-7,7],[-7*w+b,7*w+b]) # 预设分割线
  plt.plot([-7,7],[-(-7.0*p.w[0]+p.b)/p.w[1],-
  (7.0*p.w[0]+p.b)/p.w[1]]) # 预测分割线
  plt.grid(True)
  ```

- ```
  预设分割线：y = -5 x + 1
  预测分割线：  10.016109076953963 * x +  1.94598155830155 * y = 2.500000000000002
  ```



- 最后进行测试，和生成样本数据一样，生成一定数量的测试数据，输入到Perceptron中得到预测结果，与真实结果比较，计算得到准确率，然后进行可视化

- ```
  # 测试部分
  import numpy as np
  import matplotlib.pyplot as plt

  # 测试数据个数
  ```

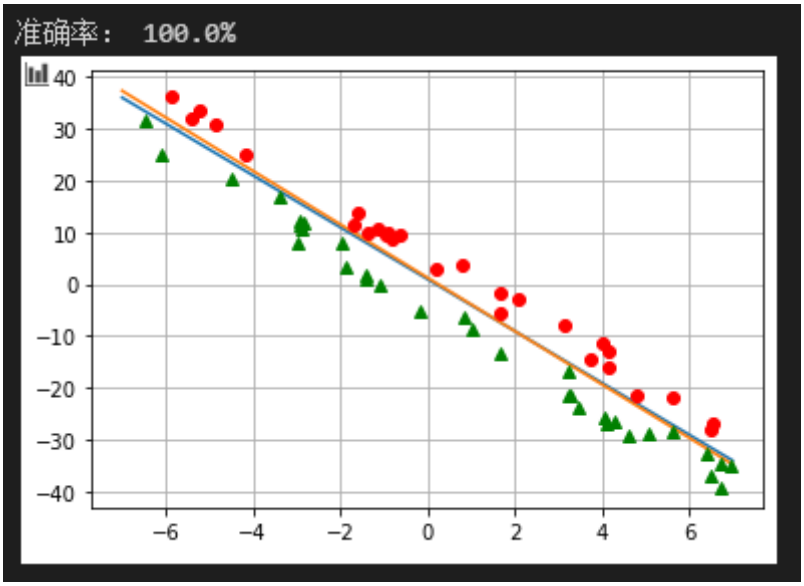
```
test_N = 60
test_data = []

plt.plot([-7,7],[-7*w+b,7*w+b]) # 预设分割线
plt.plot([-7,7],[-(-7.0*p.w[0]+p.b)/p.w[1],-
(7.0*p.w[0]+p.b)/p.w[1]]) # 预测分割线

# 生成测试数据
i = 0
while i < test_N:
    tmp_x = np.random.uniform(-7,7)
    tmp_y = tmp_x * w + b + np.random.uniform(-8,8)
    if abs(tmp_y - (tmp_x * w + b)) <= 1:
        continue
    # 正样本
    if tmp_y > tmp_x * w + b:
        # plt.plot(tmp_x,tmp_y,'ro')    # 绘制点
        test_data.append([[tmp_x,tmp_y],1])
    # 负样本
    elif tmp_y < tmp_x * w + b:
        # plt.plot(tmp_x,tmp_y,'g^')    # 绘制点
        test_data.append([[tmp_x,tmp_y],0])
    i += 1

total_error = 0
for data in test_data:
    r = p.response(data[0])
    if r != data[1]:
        total_error += 1
    if r == 1:
        plt.plot(data[0][0],data[0][1],'ro')    # 绘制点
    else:
        plt.plot(data[0][0],data[0][1],'g^')    # 绘制点

print("准确率: ",'%s%%'%((1 - total_error/test_N)*100))
plt.grid(True)  # 绘制
```

- 

Conclusion

- 在实验中，遇到了几个小问题，首先就是刚开始对要求的错误理解，以为是去预测该条直线的两个参数，所以对于网路的输出也是搞错了，后来才重新修正过来。
- 在数据生成上，将其分布限制在预设直线两侧，所以在学习过程中也能够更快的学习到分界线
- 通过本次实验，也是加强了自己的代码能力和对于神经网络底层实现的认识