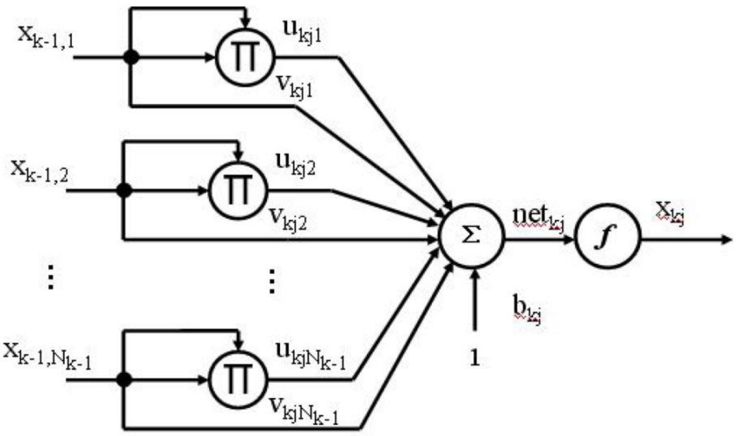
HW2 Report

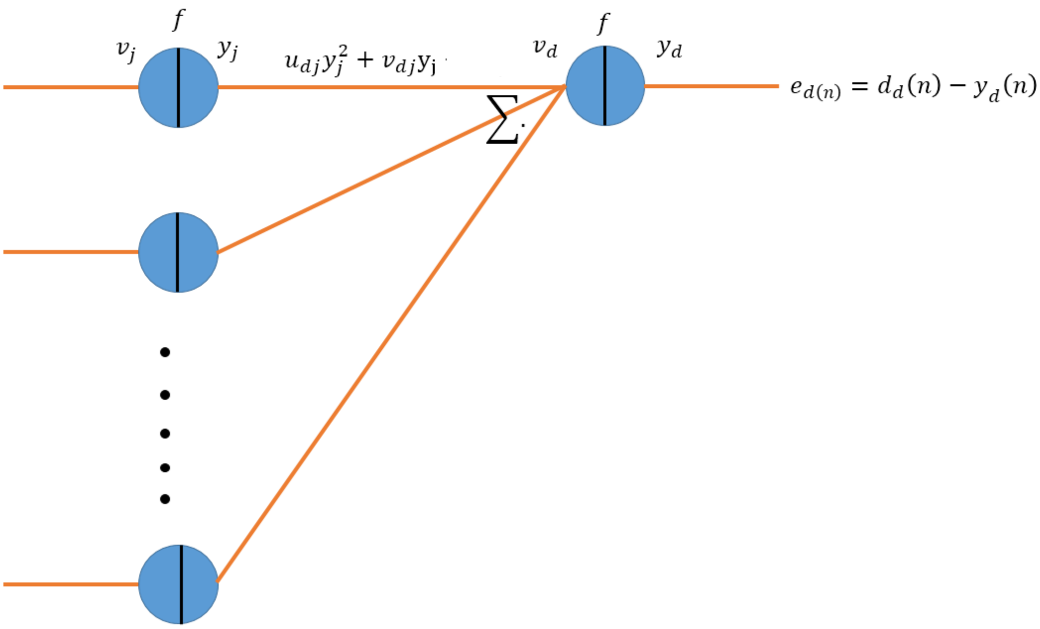
姓名：缪佳宇
学号：51502191092

1．试推导训练上述 MLQP 的下列两种 BP 算法：
a)批量学习； b）在线学习。



Solution：

先拿**在线学习**来说，MLQP 的 BP 算法和平常的 BP 的算法区别在于前向传播中神经元的输出 f 的作用域不是线性的而是二次的，但是推导是相近的。



输出层损失是$\varepsilon(\mathrm{n}) = \frac{1}{2}\sum_{k\epsilon C} e_k^2(n)$。

如果节点 j 是输出层的节点，那么损失函数在节点 j 上的梯度

$$\delta_j(n) = \frac{\partial\varepsilon(n)}{\partial v_j(n)} = \frac{\partial\varepsilon(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\frac{\partial\varepsilon(n)}{\partial e_j(n)} = e_{j(n)} = d_j(n) - y_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))(\varphi\text{是激活函数})$$

$$\delta_j(n) = -e_{j(n)}\varphi_j'(v_j(n))$$

如果节点 j 是中间节点，与之相连的下一层节点集合为 **C**, 那么损失函数在节点 j 上的梯度

$$\delta_j(n) = \frac{\partial\varepsilon(n)}{\partial v_j(n)} = \sum_{(d\in C)} \frac{\partial\varepsilon(n)}{\partial v_d(n)} \cdot \frac{\partial v_d(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\frac{\partial\varepsilon(n)}{\partial v_d(n)}\text{即是}\delta_d(n)$$

$$\frac{\partial v_d(n)}{\partial y_j(n)} = 2u_{j,d}y_{j(n)} + v_{j,d}$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))(\varphi\text{是激活函数})$$

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_{d\in C}\delta_d(n)\,(2u_{j,d}y_{j(n)} + v_{j,d})$$

所以从最后一层节点反向传播误差，将各个节点的梯度都可以求出。

所以计算 $\Delta u_{d,j} = -\alpha\delta_d(n)y_j(n)^2$, $\Delta v_{d,j} = -\alpha\delta_d(n)y_j(n)$.

$\Delta b_d = -\alpha\delta_d(n)$ （a 是学习率）

批量学习同理亦然，不同的是损失函数是批量样本的损失和。

$$\varepsilon(n) = \frac{1}{2N}\sum_n\sum_{k\epsilon C} e_k^2(n)$$

最后的参数更改也要相应改变。

$\Delta u_{d,j} = -\frac{\alpha}{N}\delta_d(n)y_j(n)^2$, $\Delta v_{d,j} = -\frac{\alpha}{N}\delta_d y_j(n)$.（a 是学习率）

$\Delta b_d = -\frac{\alpha}{N}\delta_d(n)$

2.

编程实现问题 1）中的训练 MLQP 的**在线 BP** 算法（编程语言不限，可采用常用的程序设计语言或 Matlab）

Solution:

这里我使用的是 python 来实现的，为了和第三题同步，首先我实现了一个带有一层隐层的神经网络的类，然后将 BP 算法作为它的一个方法，主要就是先得到后两层的节点梯度，然

后得到各个参数值的梯度下降值。（同时注意是在线 BP 算法）

程序截图如下:

```python
    def BackPropagate(self, x, y, lr):
        '''
        the implementation of BP algorithms

        @param x, y: array, input and output of the data sample
        @param lr: float, the learning rate of gradient decent iteration
        '''

        # dependent packages
        import numpy as np

        # get current network output
        self.Pred(x)

        # calculate the gradient based on output
        o_grid = np.zeros(self.o_n)
        for j in range(self.o_n):
            o_grid[j] = (y[j] - self.o_v[j]) * self.afd(self.o_v[j])

        h_grid = np.zeros(self.h_n)
        for h in range(self.h_n):
            for j in range(self.o_n):
                h_grid[h] += (2*self.ho_u[h][j]*self.h_v[h]+self.ho_v[h][j]) * o_grid[j]
            h_grid[h] = h_grid[h] * self.afd(self.h_v[h])

            # updating the parameter
        for h in range(self.h_n):
            for j in range(self.o_n):
                self.ho_u[h][j] += lr * o_grid[j] * (self.h_v[h]**2)
                self.ho_v[h][j] += lr * o_grid[j] * self.h_v[h]

        for i in range(self.i_n):
            for h in range(self.h_n):
                self.ih_u[i][h] += lr * h_grid[h] * (self.i_v[i]**2)
                self.ih_v[i][h] += lr * h_grid[h] * self.i_v[i]

        for j in range(self.o_n):
            self.o_t[j] -= lr * o_grid[j]

        for h in range(self.h_n):
            self.h_t[h] -= lr * h_grid[h]
```

3.

用含有一层中间层（中间层的神经元数目可设成 10）的两层 MLQP 学习双螺旋问题，比较在三种不同学习率下网络的训练时间和决策面。

Solution:

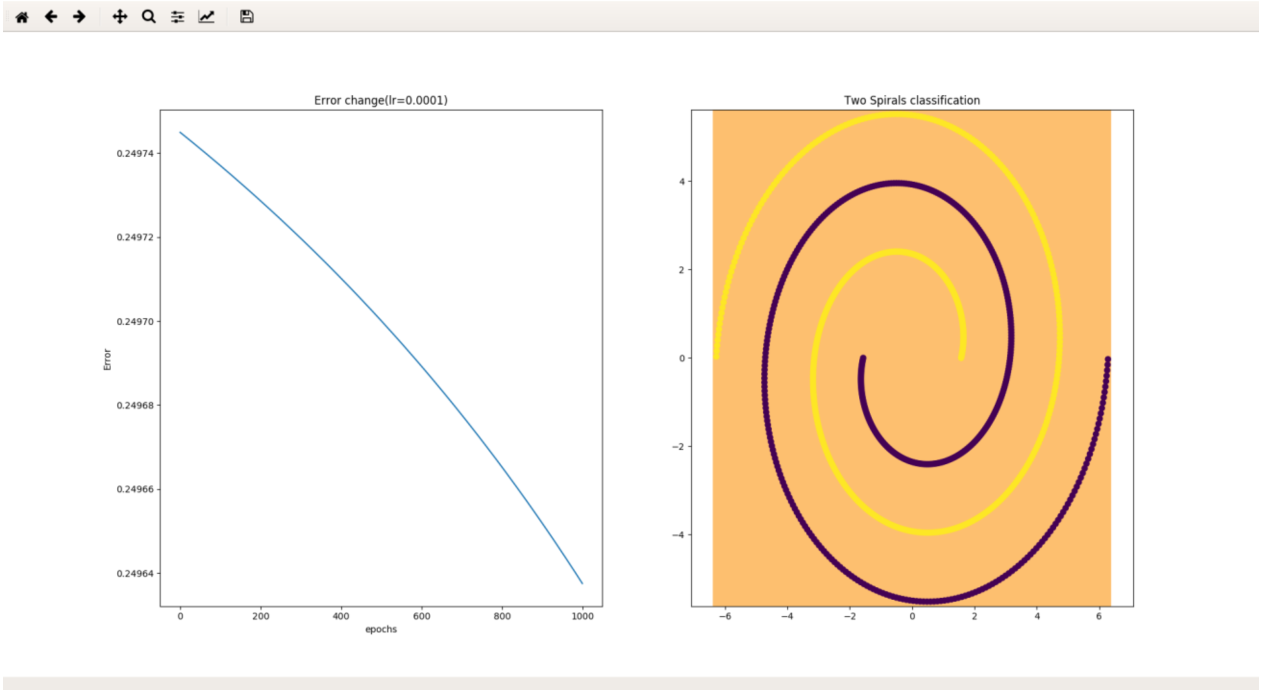
这里我从 csdn 一个聚类数据集里找到了一个双螺旋的数据集，当然也可以根据双螺旋的方程自己 generate 数据点。

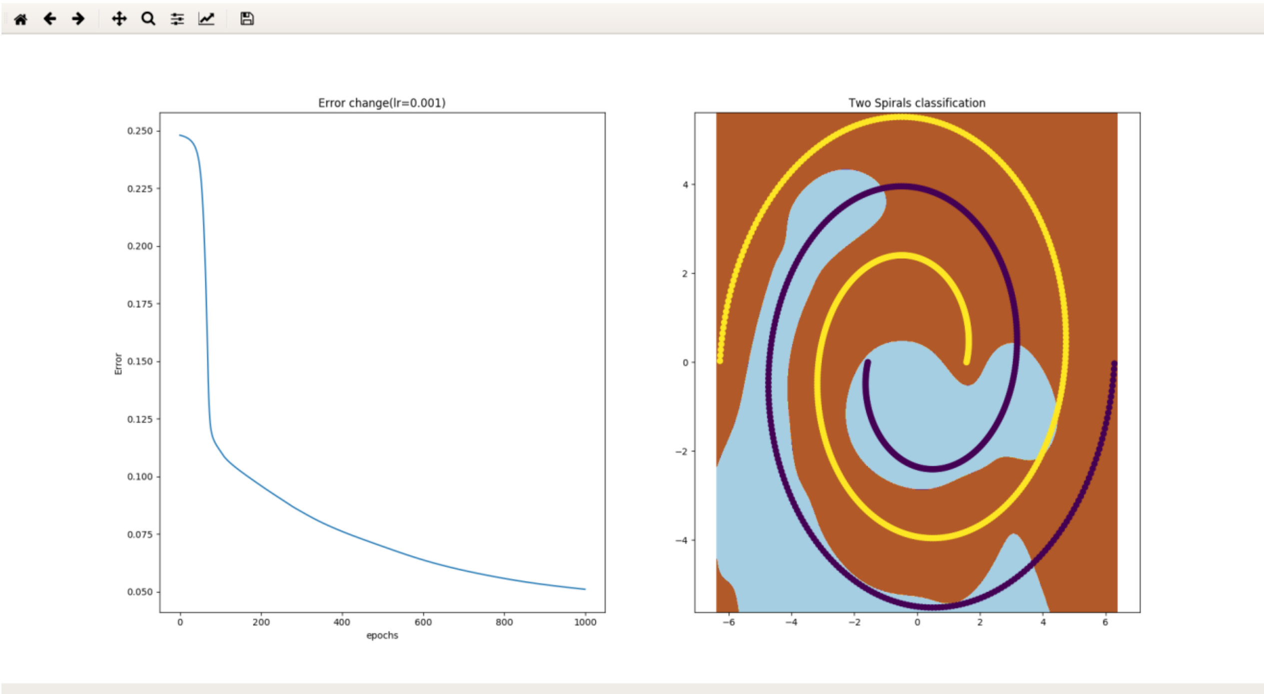
然后构造了神经网络(in:2 hidden:10 out:1)，采用不同的学习率进行训练和测试(learning rate:=0.0001，0.001，0.01，0.1，0.8)，这里为了更好的比较，我选了 5 个学习率，训练的 epoch 是 1000。

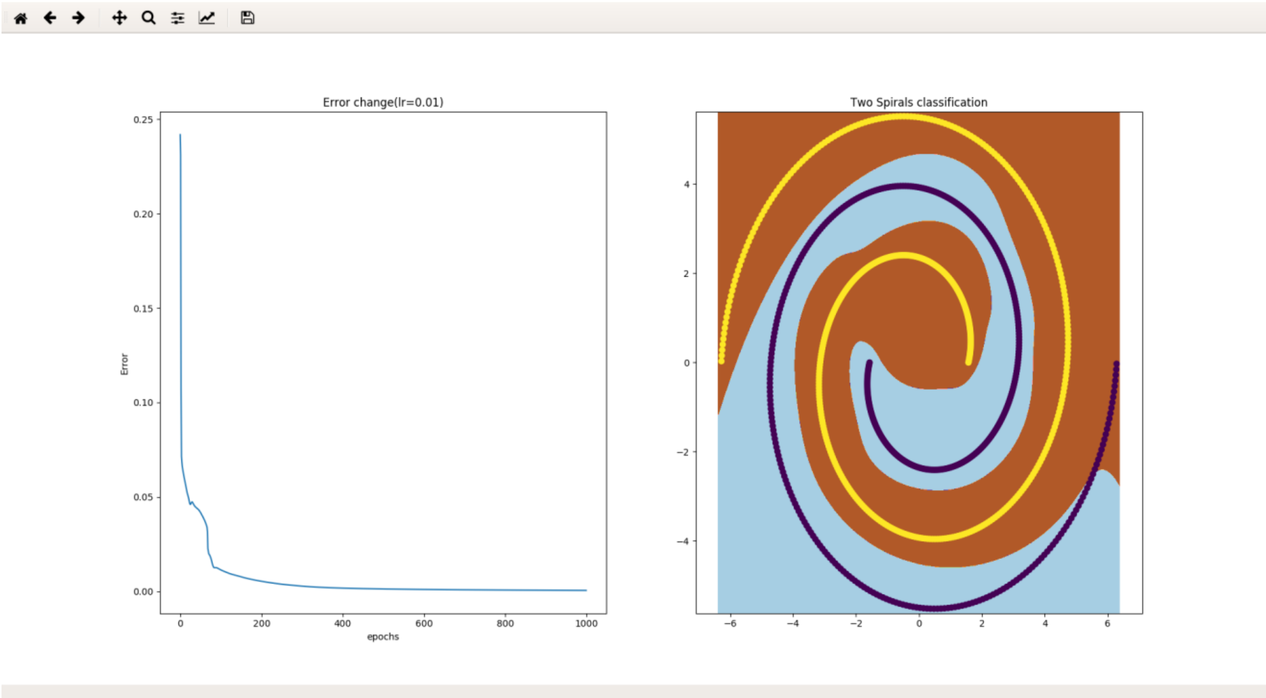
程序主要部分截图如下:

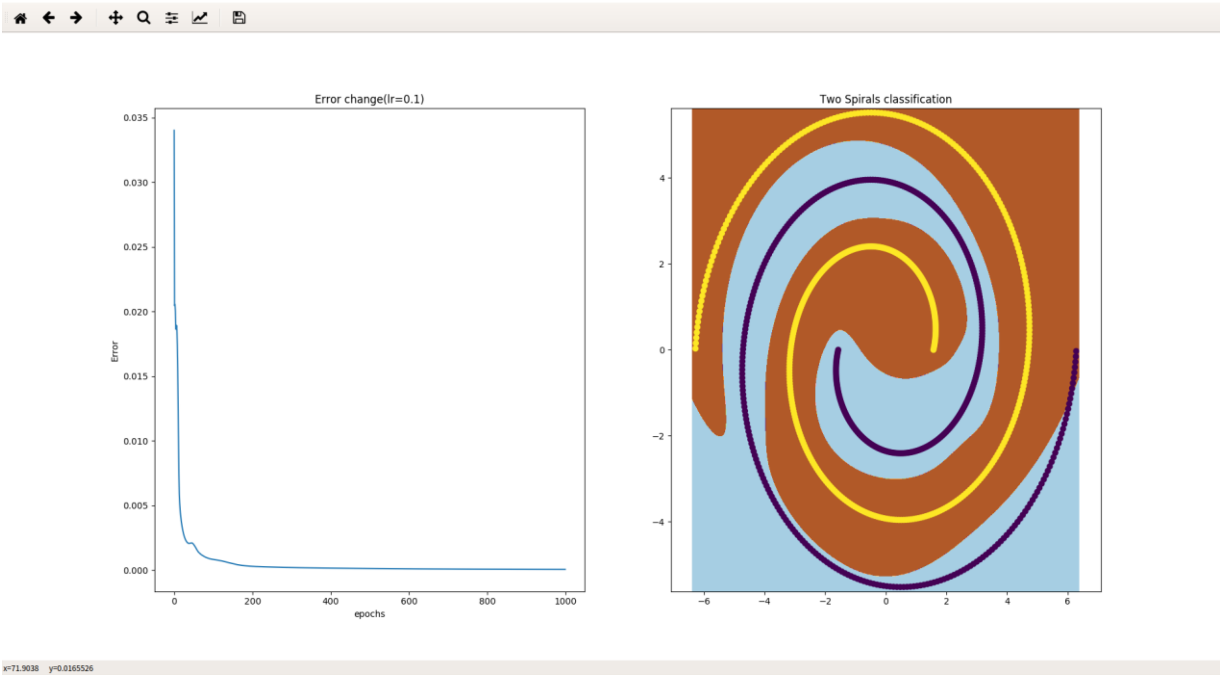
```python
nn = BP_network()  # build a BP network class
nn.CreateNN(2, 10, 1, 'Sigmoid')  # build the network

e = []
for i in range(1000):
    print(i)
    err, err_k = nn.TrainStandard(X, Y.reshape(len(Y), 1), lr=0.01)
    e.append(err)

axes[0].set_xlabel("epochs")
axes[0].set_ylabel("Error")
axes[0].set_title("Error change(lr=0.01)")
axes[0].plot(e)
```

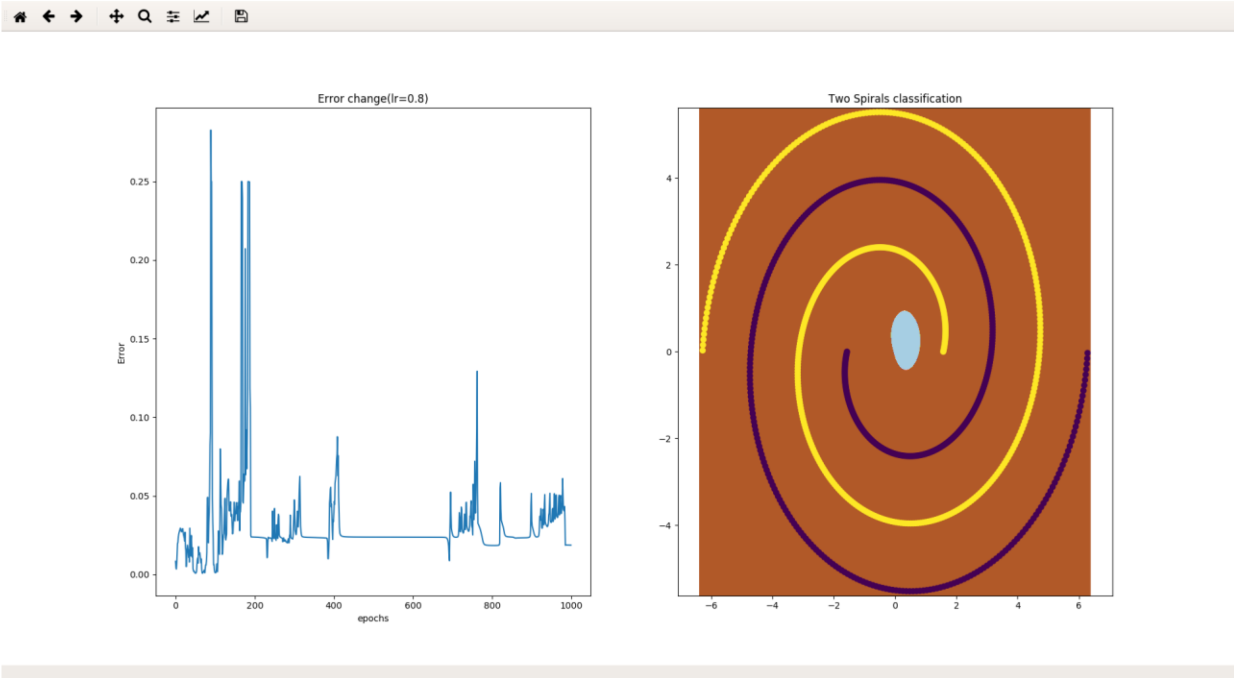
训练结果如下（学习率在每张图的左图上方标题标出）：






Error change(lr=0.01)
Error
epochs
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000
Two Spirals classification
4
2
0
−2
−4
−6
−4
−2
0
2
4
6


Error change(lr=0.1)
Error
epochs
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000
Two Spirals classification
4
2
0
−2
−4
−6
−4
−2
0
2
4
6
x=71.9038 y=0.0165526



分析:
首先从训练的时间上来看，lr=0.0001 在 1000epoches 后还没有收敛，lr=0.001-0.1 收敛时间越来越快，lr=0.8 的时候误差抖动很大没有达到收敛。

从决策面的效果来看，lr=0.0001 的决策面还没有训练出来，训练十分缓慢，lr=0.001 决策面稍微有点贴合整体结构了，lr=0.01 基本符合，lr=0.1 训练出来的就很贴合了，然后 lr=0.8 的时候决策面又偏离了数据集的结构特征。

综上 lr=0.1 左右的时候训练效果是比较好的。