中山大学计算机学院

人工智能

本科生实验报告

（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	系统结构信息计算方向	专业（方向）	信息与计算科学
学号	20337268	姓名	张文沁

一、 实验题目

> **感知机与逻辑回归模型 BP神经网络**

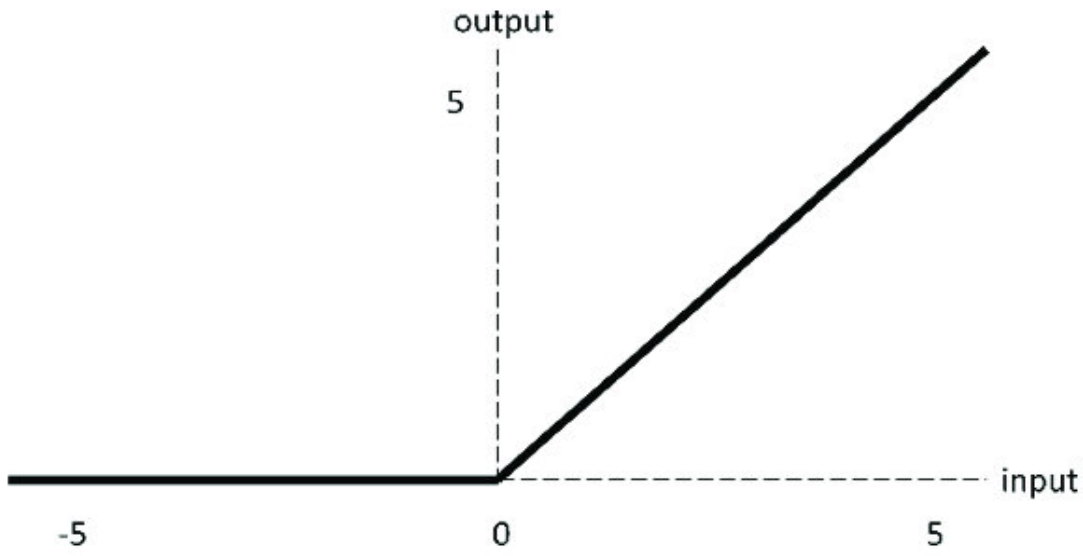
二、 实验内容

1. 算法原理

感知机（PLA）：

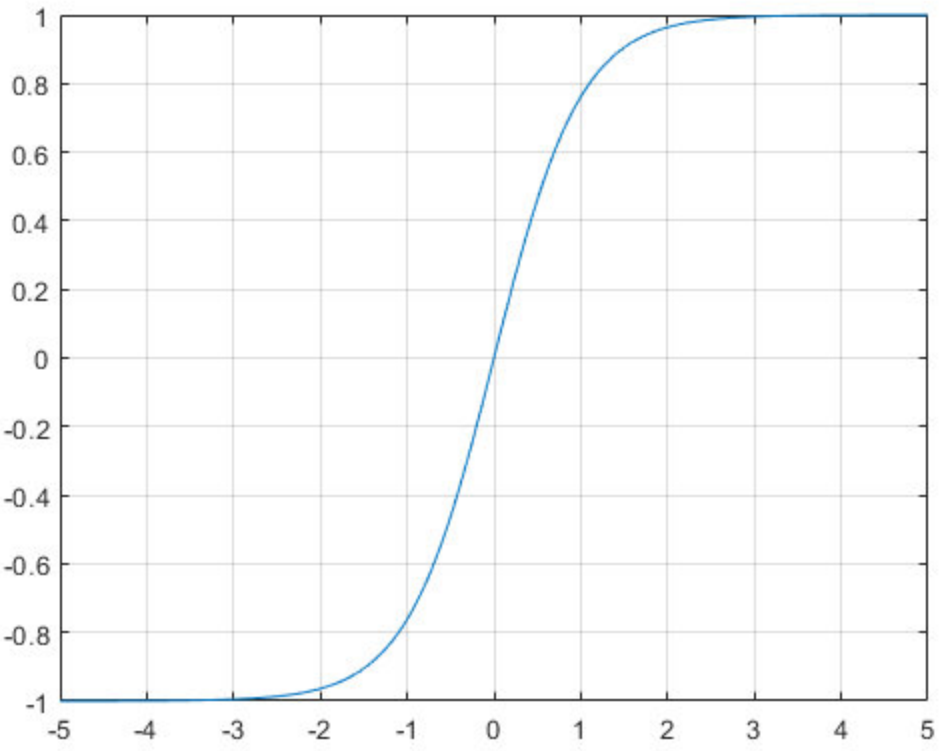
> 感知机可以看作是模拟神经元，有刺激的情况下会产生一定的反应，而我们要做的是模拟传入信息的处理过程。在实际神经元处理电信号时，会有一个阈值，只有超过这一阈值，才能有有意义的输出，本方法同样存在一个模拟这一过程的激活函数。
>
> 在神经网络当中常用的激活函数主要有三种，一种是我们之前就非常熟悉的`sigmoid`函数，一种是`relu`函数，另外一种是`tanh`函数。图像如下所示：

$$relu函数：y = max(x, 0)$$



sigmoid()和tanh()函数图像相似，只是值域不同，sigmoid（）是(0,1)，tanh（）是(-1,1)。且tanh（）的敏感区间比较大，另外一点就是sigmoid（）输出的都是正数，在某些场景下可能会出现问题。

$$y = sigmoid()\&y = tanh()$$



这三个函数都可以作为神经网络中的激活函数，但是仍然有区别，在神经网络中最常用的激活函数是`relu`函数，因为它的收敛速度比其他两个快，而其他两个函数越靠近两边变化越小，导数结果越趋近于0，于是收敛速度也会变慢。

本方法经常被用于处理二分类的问题那么下面介绍二分类问题下的感知机模型：

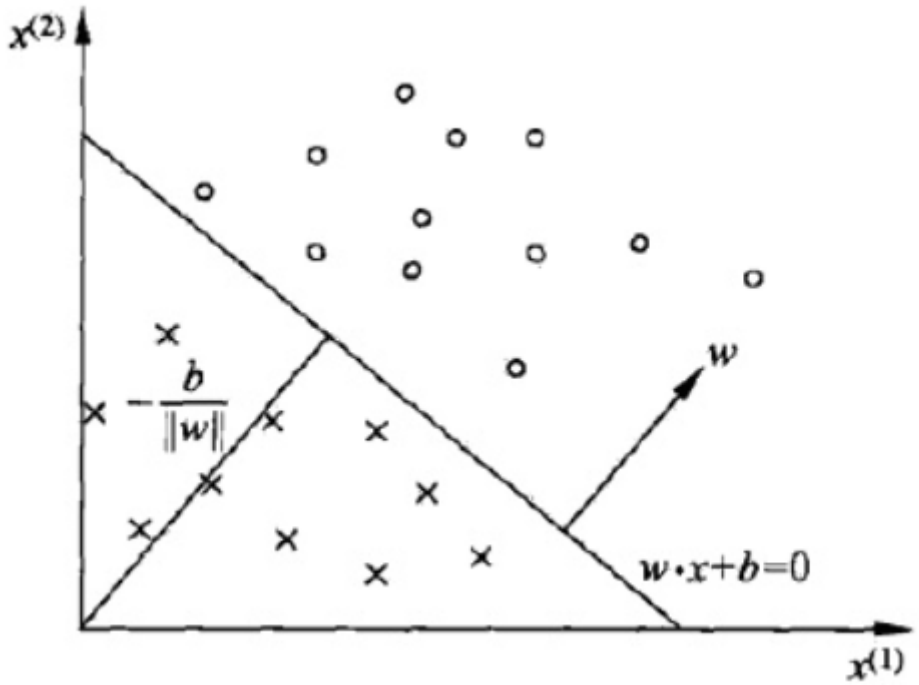
如果只有两个变量，可以写成下述形式：相当于一个线性的函数，如果数据是可二分类的，那么一定会有图片所示形式的直线，且直线方向向量为`（w1,w2,b）`，而方便起见可以通过sign的激活函数对平面上的点做出分类也就是打上标签。

我在本次实验中定义的模型如下：

$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 + b \leq 0 \\ 1, & w_1x_1 + w_2x_2 + b > 0 \end{cases}$$

继续解释算法：

$$w1x1 + w2x2 + b = 0$$



$$f(x) = sign(w \cdot x + b)$$

$$sign(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

同时为了估计我们找到的方向向量的拟合度，我们需要设立一个损失函数，来计算误分程度，而最显而易见的一个指标就是误分点到直线的距离之和。而我们所期望找到的直线也就是满足损失函数值最小的一个。

$$\frac{1}{\|w\|}|w \cdot x_0 + b|$$

这里，$\|w\|$ 是 w 的 L_2 范数.

其次，对于误分类的数据 (x_i, y_i) 来说，

$$-y_i(w \cdot x_i + b) > 0$$

成立. 因为当 $w \cdot x_i + b > 0$ 时， $y_i = -1$ ，而当 $w \cdot x_i + b < 0$ 时， $y_i = +1$. 因此，误分类点 x_i 到超平面 S 的距离是

$$-\frac{1}{\|w\|} y_i(w \cdot x_i + b)$$

这样，假设超平面 S 的误分类点集合为 M，那么所有误分类点到超平面 S 的总距离为

$$-\frac{1}{\|w\|} \sum_{x_i \in M} y_i(w \cdot x_i + b)$$

对损失函数进行极小化的方法大致有两个，一个是基于所有样本的梯度和的均值的梯度下降法（该方法行不通因为损失函数只有误分点样本的参与），另一个是随机梯度下降法。选取一个误分点进行更新。

更新参数的过程如下:

1. 设置一个学习率n，并随机初始化w,b
2. 选取一个误分点（x,y），对参数进行更新：

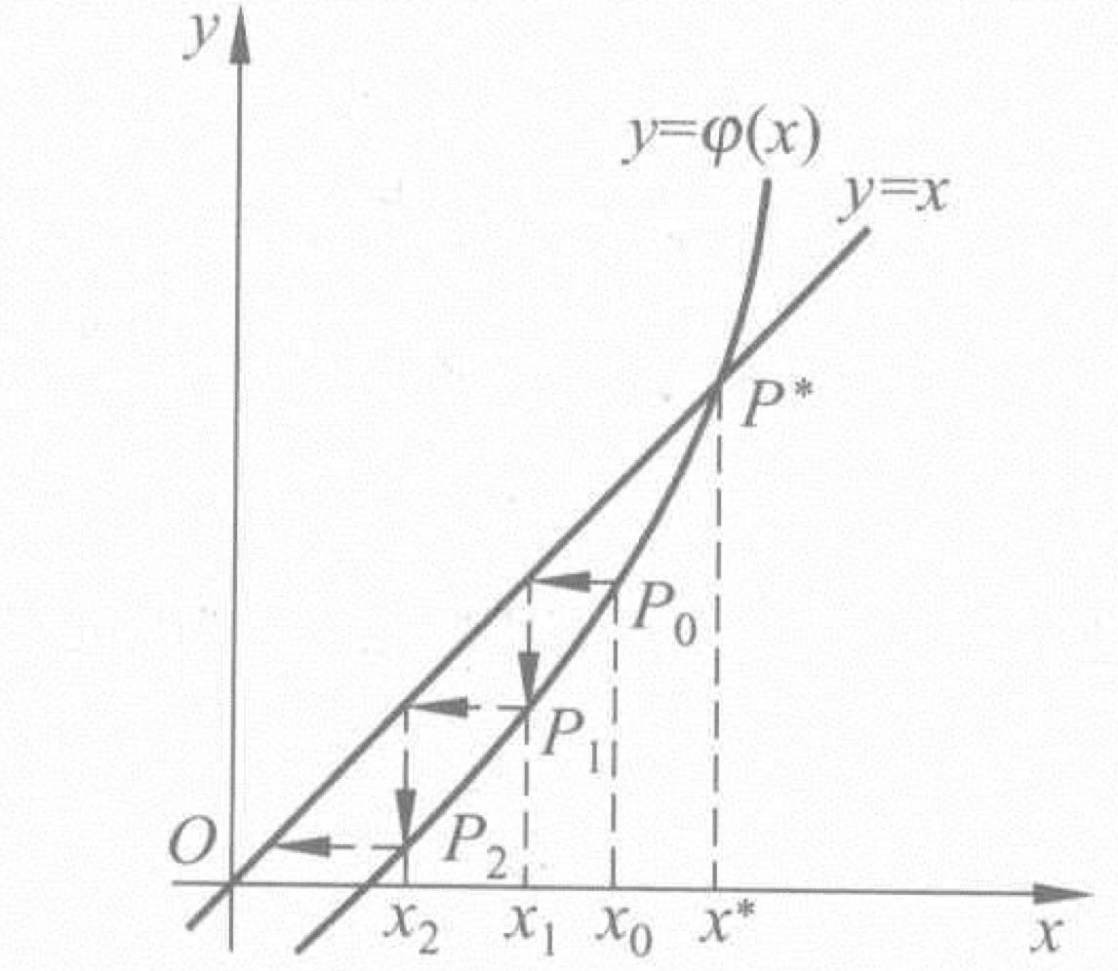
```
w = w + n * y * x

b = b + n * y
```

3. 重复上述流程，使得误分点尽量少

下欲通过数值分析的内容解释本课内容中的更新的参数部分：

在非线性方程与方程组的数值解法中存在不动点迭代的方法，原理为，转换非线性方程为 `y = x` 和 `y = f(x)` 的非线性方程组，通过不断迭代更新 `x` 的值来趋近精确解，图解如下：



而可以收敛到精确解的要求是

1. 任意 `x` 属于 `[a,b]` 都有 `f(x)` 属于 `[a,b]`
2. 存在L<1，使得任意x,y属于[a,b]都有

   `|f(x) - f(y)| <= L|x - y|`

可以看到一个必要的收敛条件是任意两点直线**导数值小于1**，也就是为了保证迭代过程不会出现向外围跳跃的过程。

而为了加速这个收敛过程，埃特金提出了新的加速收敛方法，然而实际原理是加速选取点的连线导数值趋于精确解处的导数值的过程：

$$\bar{x}_{k+1} = x_k - \frac{(x_{k+1} - x_k)^2}{x_k - 2x_{k+1} + x_{k+2}} = x_k - (\Delta x_k)^2 / \Delta^2 x_k, \quad k = 0, 1, \cdots$$

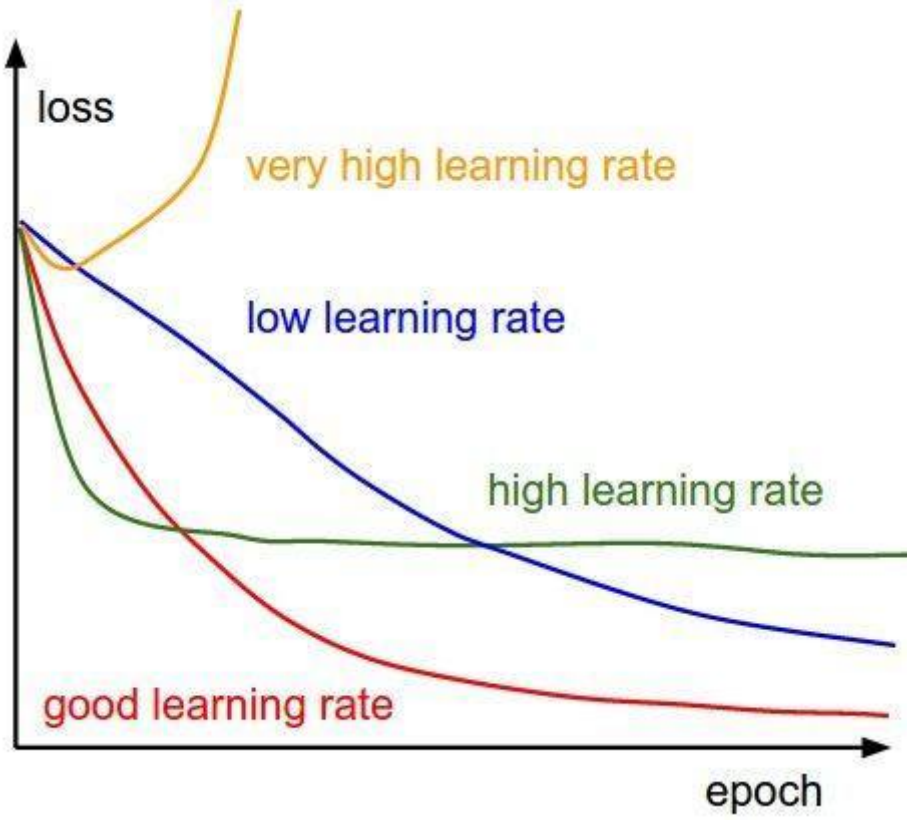
鉴于上述内容，PLA中的学习率理应可以被类比为导数值，决定着权重更迭的步长。为了保证收敛需要满足两个条件

1. 不管初始状态离优化状态多远，总可以收敛
2. 学习率要保证收敛的稳定性

而学习率过大会导致模型不收敛，过小会导致模型收敛过慢或无法学习。

阐述完学习率应该满足的条件，下面说明为了找到学习率应该怎么做。

最简单的一个方法是从小到大训练模型，记录损失的变化速度，结果表示随着学习率的增加，损失会慢慢变小，而后增加，而最佳学习率就可以从损失函数值的变化程度中找到。



逻辑回归模型（LR）：

逻辑回归模型针对二分类问题，输入是样本的特征向量，输出是属于某个样本的概率，从概率大小判断属于哪个类别。逻辑回归使用的方法是假设数据服从伯努利分布，通过极大化似然函数方法，运用梯度下降来求解参数，来达到将数据二分目的。

首先几个概念：

1. 对数几率函数：是一种Sigmoid函数，通过此函数来输出类别概率。其中y 代表的是样本视为正样本的可能性，则 1-y 为视为负样本的可能性。

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

2. 对数几率：定义为 如下，其中 y/(1-y) 称为比率。

$$\ln \frac{y}{1-y} = w^T x + b$$

3. 决策边界：作用在 n 维空间，将不同样本分开的平面或曲面，在逻辑回归中，决策边界对应 wx+b=0

在概率论中，为了估计某些参数的值，我们用到了对数似然函数，LR中亦然：

对于给定的数据集 $\left\{x^{(i)}, y^{(i)}\right\}$ 其中 i 从1到数据集大小m，来使得最大化对数似然。

首选，写出似然函数： $l=\prod_{i=1}^{m}\left[\pi\left(x^{(i)}\right)\right]^{y^{(i)}}\left[1-\pi\left(x^{(i)}\right)\right]^{1-y^{(i)}}$

对数似然函数就是 $L(w)=\sum_{i=1}^{m}[y^{(i)}\log\pi(x^{(i)})+(1-y^{(i)})\log(1-\pi(x^{(i)}))]$

可化简为

$$\sum_{i=1}^{m}\left[y^{(i)}\log\frac{\pi\left(x^{(i)}\right)}{1-\pi\left(x^{(i)}\right)}+\log\left(1-\pi\left(x^{(i)}\right)\right)\right]=\sum_{i=1}^{m}\left[y^{(i)}\left(w\cdot x^{(i)}\right)-\log\left(1+\exp\left(w\cdot x^{(i)}\right)\right)\right]$$

后面式子是带入 $\pi(x)$ 后化简得到。

之后就是对对数似然函数求极大值，即以对数似然函数为目标的最优化问题。 L(w) 是关于 w 的高阶连续可导凸函数，根据凸优化理论，可采用梯度下降法，牛顿下山法等优化方法求解。

根据极大似然估计，对其求极大值，可以得到w的估计值，这里采用梯度下降的方法解决问题：

$$-\nabla_w L(w)=-\sum_{i=1}^{N}\left[y_i-\pi\left(x_i\right)\right]x_i$$

$$w=w+\eta\sum_{i=1}^{N}\left[y_i-\pi\left(x_i\right)\right]x_i$$

同样的，为了标识参数的优劣，我们引入损失函数：

这里用到的是交叉熵损失函数，用于度量分布的差异性。

$$J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}logh_\theta(x)+(1-y^{(i)})log(1-h_\theta(x))]$$

也可以化简为：

$$
\begin{aligned}
L(w) &= \sum_{i=1}^{N} [y_i \log \pi(x_i) + (1 - y_i) \log(1 - \pi(x_i))] \\
&= \sum_{i=1}^{N} \left[y_i \log \frac{\pi(x_i)}{1 - \pi(x_i)} + \log(1 - \pi(x_i)) \right] \\
&= \sum_{i=1}^{N} [y_i (w \cdot x_i) - \log(1 + \exp(w \cdot x_i)]
\end{aligned}
$$

LR在实验过程中发现，很容易过拟合，且效果不好

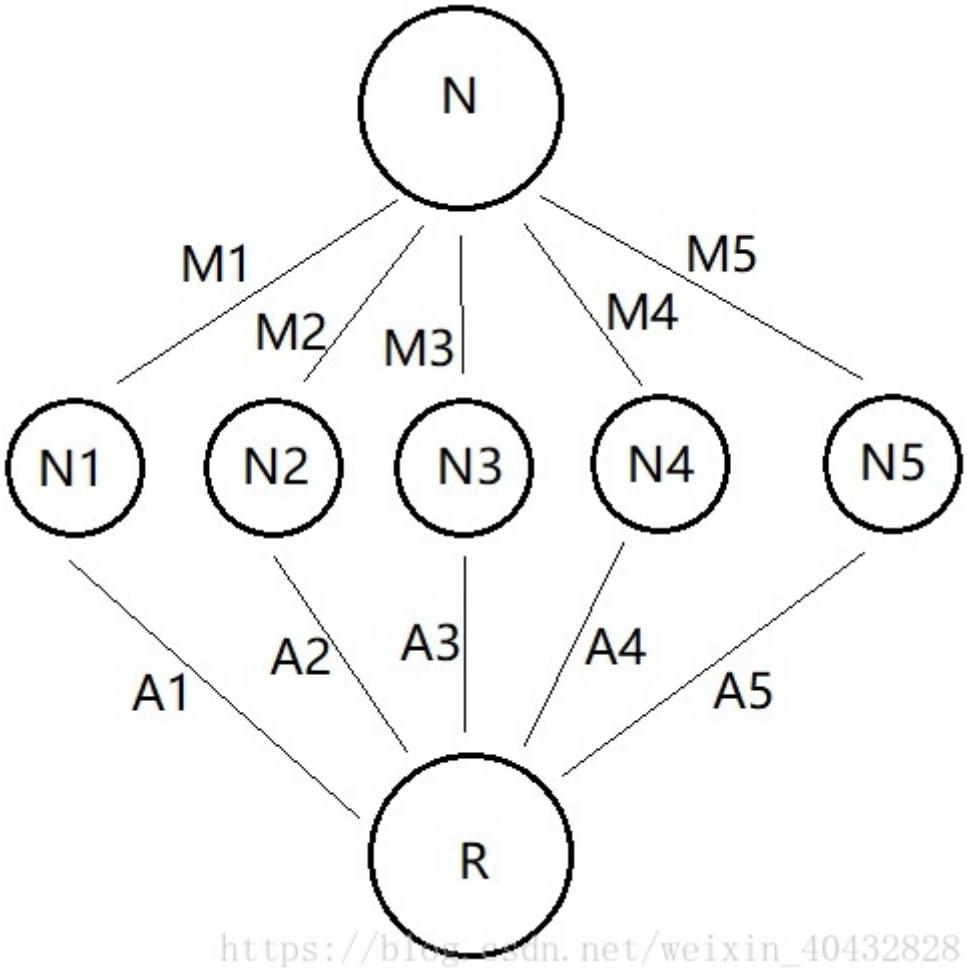
BP神经网络：

在前两种算法下， 只能解决线性二分问题，无法解决例如异或之类的分类问题，此时思考为了解决此类问题，要不然引入更多特征值，或者引入新的维度，例如同心圆类似的分类问题，可以将分类标准设置为

$$x^2 + y^2$$

此时就解决了无法线性划分的问题。所以神经网络相当于在前两种算法的基础上加入针对该数据集的特征以更好地分类数据。则此时问题聚焦在如何加入新的特征上。

解释BP，实际上是Back Propagation的简写，意思是反向传播。顾名思义，我们是要通过输出值来反推参数设置。大致表达如下：



把 N 作为输入层 ，R作为输出层，N1到N5则整体作为隐藏层，共三层。而M1到M5则可以理解为输入层到隐藏层的权重，A1到A5为隐藏层到输出层的权重。

解释一下四个概念：

输入层：信息的输入端，上图中 输入层 只有 1 个节点实际的网络中可能有很多个

隐藏层：信息的处理端，用于模拟一个计算的过程，上图中，隐藏层只有一层，节点数为 5 个。

输出层：信息的输出端，也就是我们要的结果，上图中，R 就是输出层的唯一一个节点，实际上可能有很多个输出节点。

权重：连接每层信息之间的参数，上图中只是通过乘机的方式来体现。

而最能体现BP特征的是正向传播和反向传播， 正向传播就是让信息从输入层进入网络，依次经过每一层的计算，得到最终输出层结果的过程。和前两种算法类似。而反向传播的信息是误差，也就是 输出层（output ）的结果与输入信息 x 对应的真实结果之间的差距。也就是BP算法的损失函数。（实际中会有更多的损失函数计算方法，例如上面提到的交叉熵损失）

解释完之后，我们要计算参数，最基本的思想和之前一样，降低损失，此处用到了梯度下降法，和上面不同，此处的梯度下降常常使用的是矩阵乘法。

2. 关键代码展示

感知机模型:

PLA模型在测试中性能高于LR模型，且有时正确率可以达到100%

判断是否需要更新：

```
# 如果是真，则需要更新
def identify(w,i):
    index = [0 for i in range(40)]
    for j in range(40):
        index[j] += w[j]*x[i][j]
    sum_index = sum(index) + b
    sum_index = sum_index * answer[i]

    if (sum_index != 0 and answer[i] == 0) or (sum_index <= 0 and answer[i] ==
1):
        return 1
    elif (sum_index == 0 and answer[i] == 0) or (sum_index > 0 and answer[i] ==
1):
        return 0
```

更新指示函数：

```
def update():
    for i in range(LEN):
        if identify(w,i):
            need[i] = 1
        else:
            need[i] = 0
```

更新w,b：

```
def __PLA__():
    update()
    sum = 0
    while 1 in need:
        sum == 1
        if sum > 100:
            break
        i = need.index(1)
        for j in range(40):
            w[j] += n*x[i][j]*answer[i]
        global b
        b += n*answer[i]
        update()
        print(w,b)
```

逻辑回归模型：

> LR模型正确率随迭代次数改变为变化较大，正确率在75%附近波动。

pi()函数

```
def pi(j):
    sum = 0
    for i in range(41):
        sum += w[i]*x[j][i]

    P = 1.0/(1+np.exp(-sum))
    return P
```

更新w

```
def update_w():
    update = [0 for i in range(41)]
    rate = 0
    for i in range(7000):
        rate = answer[i] - pi(i)
        for j in range(41):
            update[j] += rate * x[i][j]
    for i in range(41):
        w[i] += n * update[i]
```

损失函数:

```
def L():
    L = 0
    for i in range(7000):
        sum = 0
        for j in range(41):
            sum += x[i][j]*w[j]
        L += (answer[i]*sum - math.log(1 + np.exp(sum)))
    return L
```

BP神经网络

实验结果不理想，多层神经网络正确率不如单层

激励函数:

```
def AF(p,kind):   # 激励函数
    t = []
    if kind == 1:   # sigmoid
        pass
    elif kind == 2:   # tanh
        pass
    elif kind == 3:    # ReLU

        return where(p<0,0,p)
    else:
        pass
```

激励函数导数:

```
def dAF(p,kind):   # 激励函数导数
    t = []
    if kind == 1:   # sigmoid
        pass
    elif kind == 2:   # tanh
        pass
    elif kind == 3:    # ReLU
        return where(p<0,0,1)
    else:
        pass
```

归一化系数:

```
def Calcu_KtoOne(p,t):                          # 确定归一化系数
    p_max = p.max()
    t_max = t.max()
    return max(p_max,t_max)
```

更新参数:

```
#计算各层隐藏层输入输出 Hi Ho
        for k in range(0,HCNum,1):
            if k == 0:
                Hi[k] = dot(To,u)
                Ho[k] = AF(add(Hi[k],Hb[k]),AFKind)
            else:
                Hi[k] = dot(Ho[k-1],w[k-1])
                Ho[k] = AF(add(Hi[k],Hb[k]),AFKind)
        #计算输出层输入输出 Oi Oo
        Oi = dot(Ho[HCNum-1],v)
        Oo = AF(add(Oi,Ob),AFKind)
        #反向 nnbp

        #反向更新 v
        Oe = subtract(Teacher,Oo)
        Oe = multiply(Oe,dAF(add(Oi,Ob),AFKind))

        e += sum(multiply(Oe,Oe))
        #v梯度
        dv = dot(array([Oe]),array([Ho[HCNum-1]])).transpose()            # v 的
梯度
        v = add(v,dv*LearnRate)
        #反向更新 w
        He = zeros((HCNum,HNum))

        for c in range(HCNum-2,-1,-1):
            if c == HCNum-2:
                He[c+1] = dot(v,Oe)
                He[c+1] = multiply(He[c+1],dAF(add(Hi[c+1],Hb[c+1]),AFKind))
                dw[c] = dot(array([Ho[c]]).transpose(),array([He[c+1]]))
                w[c] = add(w[c],LearnRate*dw[c])
            else:
                He[c+1] = dot(w[c+1],He[c+2])
                He[c+1] = multiply(He[c+1],dAF(add(Hi[c+1],Hb[c+1]),AFKind))
                dw[c] = dot(array([Ho[c]]).transpose(),array([He[c+1]]))
                w[c] = add(w[c],LearnRate*dw[c])
```

```
        #反向更新u
        He[0] = dot(w[0],He[1])
        He[0] = multiply(He[0],dAF(add(Hi[0],Hb[0]),AFKind))
        du = dot(array([To]).transpose(),array([He[0]]))
        u = add(u,du)
        #更新阈值 b
        Ob = Ob + Oe*LearnRate
        Hb = Hb + He*LearnRate
    e = sqrt(e)
```

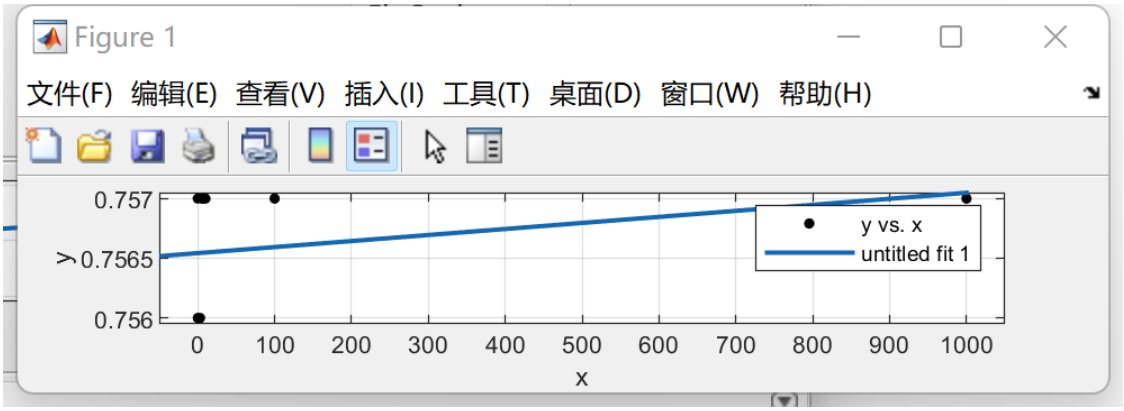
3. 创新点&优化

三、 实验结果及分析

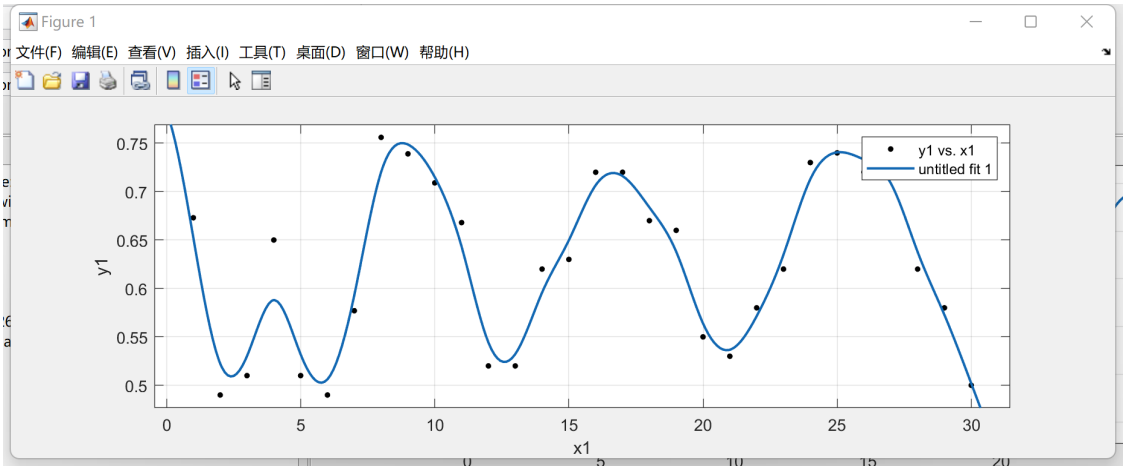
四、思考题

不同的学习率对模型收敛有何影响？从收敛速度和是否收敛两方面来回答（可以结合实验结果）。

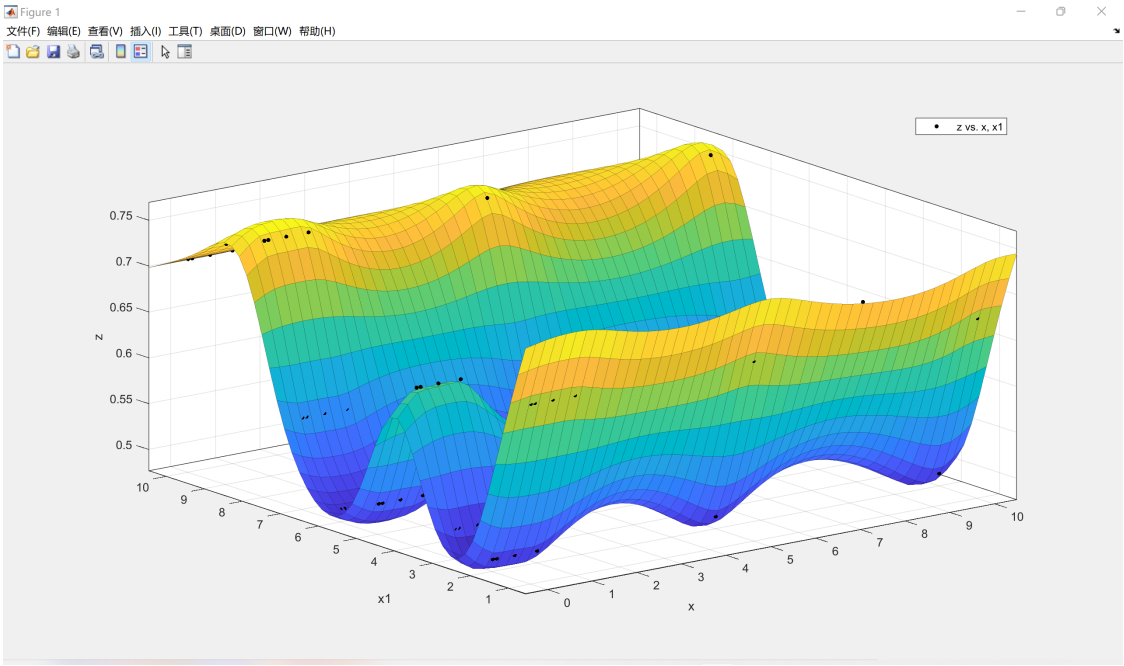
> **学习率可以近似认为是模型向最低点靠近的速率，在学习率小的时候收敛速度慢但是会收敛于最低点，而当学习率较大时，可能会出现跳跃的现象使得模型不收敛或者是过早达到拟合点，出现过拟合现象。过小会导致模型收敛的特别慢或者无法学习。**
>
> **在实际实验过程中并没有感觉到太大的学习率带来的影响？实验结果表明对正确率有影响的因素主要是迭代次数**
>
> 
>
> 根据实验结果绘图，在迭代次数不变的情况下：



在学习率不变的条件下：



三者共同作用大致图像：



理想情况应该为一个马鞍面

使用梯度的模长是否为零作为梯度下降的收敛终止条件是否合适，为什么？一般如何判断模型收敛？

不合适，梯度为0只是函数取极值的必要条件而不是充分条件，即梯度为0的点可能不是极值点。

模型收敛：

1. 误差小于某个预先设定的较小的值
2. 两次迭代之间的权值变化已经很小，可设定一个阈值，当小于这个阈值后，就停止训练。

> 3. 设定最大迭代次数，当迭代超过最大次数就停止训练。

五、参考资料

(1条消息) 机器学习（一）：BP神经网络（含代码及注释）_Karthus冲冲冲的博客-CSDN博客_bp神经网络代码