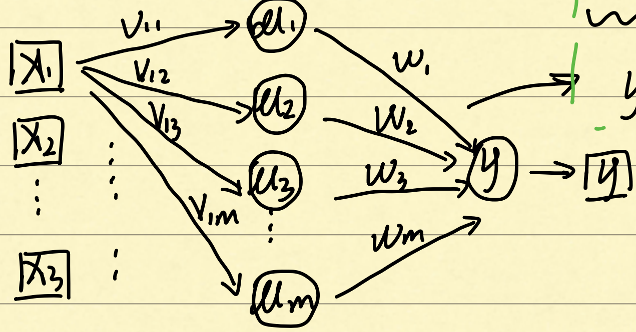
# BP 神经网络




$$u_j = f\left(\sum x_i v_{ij} + \theta_j^u\right)$$ 阈值

$$y = f\left(\sum u_j w_j + \theta^y\right)$$

前向传播公式

input hidden output

## 激活函数

- sigmod $f(x) = \frac{1}{1+e^{-x}}$
- tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ReLU $f(x) = \begin{cases} x, & \text{if } x>0 \\ 0, & \text{if } x<0 \end{cases}$
- Purelin $f(x) = x$

## 以 sigmod 为例

$$u_i = \frac{1}{1+e^{-(\sum x_i v_{ij} + \theta_j^u)}} \qquad y = \frac{1}{1+e^{-(\sum u_j w_j + \theta^y)}}$$

## 损失函数

以 $L = \sum (y-\hat{y})^2$ 以例

## 梯度下降法

优化参数 $\hat{w} = w - \mu \frac{dy}{dw}$

$dy$ → 目标函数

$dw$ → 需要优化的参数

$\mu$ → 学习率

$$\begin{cases} \hat{v}_{ij} = v_{ij} - \mu \frac{\partial L}{\partial v_{ij}} \\ \hat{w}_{ij} = w_{ij} - \mu \frac{\partial L}{\partial w_{ij}} \end{cases} \quad \text{and} \quad \begin{cases} \hat{\theta}_j^u = \theta_j^u - \mu \frac{\partial L}{\partial \theta_j^u} \\ \hat{\theta}^y = \theta^y - \mu \frac{\partial L}{\partial \theta^y} \end{cases}$$

① 求 $w_j$ 的梯度 $\frac{\partial L^{(k)}}{\partial w_j}$

$$L^{(k)} = (y^{(k)} - \hat{y}^{(k)})^2$$

$$\Rightarrow \frac{\partial L^{(k)}}{\partial w_j} = 2(y^{(k)} - \hat{y}^{(k)}) \frac{\partial y^{(k)}}{\partial w_j}$$

$$y = f(\sum u_j w_j + \theta^y)$$

令 $G = \sum u_j w_j + \theta^y$

$$\Rightarrow y = f(G)$$

$$\Rightarrow \frac{\partial y^{(k)}}{\partial w_j} = \frac{\partial y^{(k)}}{\partial G} \frac{\partial G}{\partial w_j} \qquad \frac{\partial G}{\partial w_j} = u_j$$

$$\frac{\partial y^{(k)}}{\partial G} = \left(\frac{1}{1+e^{-G}}\right)' = \frac{e^{-G}}{(1+e^{-G})^2} = \frac{1}{(1+e^{-G})}\left(1 - \frac{1}{1+e^{-G}}\right)$$

$$= y^{(k)}(1 - y^{(k)})$$

∴ 整理得 $\frac{\partial L^{(k)}}{\partial w_j} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1-y^{(k)})\, u_j$

$\Downarrow$ $G = \sum w_j u_j + \theta^y$. 同理可得

$$\frac{\partial L^{(k)}}{\partial u_j} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1-y^{(k)})\, w_j$$

② 求 $\theta^y$ 的梯度 $\frac{\partial L^{(k)}}{\partial \theta^y}$

$$\frac{\partial G}{\partial \theta^y} = 1$$

$$\frac{\partial L^{(k)}}{\partial \theta^y} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1-y^{(k)})$$

③ 求 $v_{ij}$ 的梯度 $\frac{\partial L^{(k)}}{\partial v_{ij}} = \frac{\partial L^{(k)}}{\partial u_j} \cdot \frac{\partial u_j}{\partial v_{ij}}$

$$u_j = f(\sum x_i v_{ij} + \theta_j^u)$$

$G$

$$\frac{\partial L^{(k)}}{\partial v_{ij}} = \frac{\partial L^{(k)}}{\partial u_j} \cdot \frac{\partial u_j}{\partial v_{ij}} = \frac{\partial L^{(k)}}{\partial u_j} \cdot \frac{\partial u_j}{\partial G} \cdot \frac{\partial G}{\partial v_{ij}}$$

$$= 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)})\, w_j u_j (1 - u_j)\, x_i^{(k)}$$

④ 求 $\theta_j^u$ 的梯度 $\frac{\partial L^{(k)}}{\partial \theta_j^u}$

$$\frac{\partial L^{(k)}}{\partial \theta_j^u} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)})\, w_j u_j (1 - u_j)$$

综上.

$$\begin{cases} \frac{\partial L^{(k)}}{\partial w_j} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)})\, u_j \\ \frac{\partial L^{(k)}}{\partial \theta^y} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)}) \\ \frac{\partial L^{(k)}}{\partial v_{ij}} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)})\, w_j u_j (1 - u_j)\, x_i^{(k)} \\ \frac{\partial L^{(k)}}{\partial \theta_j^u} = 2(y^{(k)} - \hat{y}^{(k)})\, y^{(k)}(1 - y^{(k)})\, w_j u_j (1 - u_j) \end{cases}$$

MATLAB 代码实现

Step 1° 导入数据. 80% 作为训练集

Step 2° 初始化参数

```
train_num = round(0.8*m)
test_num = m - train_num
Neurous_num = 6    % 隐藏神经元个数
input_num = n-1
outDut_num = 1
```

[x_train_std, x_train_mu, x_train_sigma] = zscore(x_train)

[y_train_std, y_train_mu, y_train_sigma] = zscore(y_train)

↓ Z标准化，使得 $\mu=0, \sigma=1$

Min-Max

MaxAbs

RobustScaler

↓ 用训练集参数标准化测试集

Z-score $z=\frac{x-\mu}{\sigma}$

x_test_std = (x_test - repmat(x_train_mu, test_num, 1))./
repmat(x_train_sigma, test_num, 1)

↳ repmat(A, m, n)

↳ ans = $\begin{bmatrix} A & A & A \\ A & A & A \end{bmatrix}_{m\times n}$

## step 3°. 网络参数

vij = rand(Neurons_num, input_num)      % 6×8

theta_u = rand(Neurons_num, 1)      % 阈值

Wj = rand(output_num, Neurons_num)

theta_y = rand(output_num, 1)

learn_rate = 0.0001      % 学习率

Epochs_max = 50000      % 最大迭代次数

error_rate = 0.1      % 目标误差

Obj_save = zeros(1, Epochs_max)      % 损失函数

## step 4°. 训练网络

### step 4.1° 误差分析

epoch_num = 0

```
while    epoch_num < Epochs_num
            ++
    y_pre_std_u = vij * x_train_std + repmat(theta_u,1,train_num)
    y_pre_std_u1 = logsig(y_pre_std_u)

    y_pre_std_y = wj * y_pre_std_u1 + repmat(theta_y,1,...)
    y_pre_std_y1 = logsig(y_pre_std_y)

    obj = y_pre_std_y1 - y_train_std
    Ems = sumsqr(obj)                    % Σ(y-ŷ)^2
    Obj_save(epoch_num) = Ems.

    if Ems < error_rate
        break;
    end
```

step 4.2° 梯度下降

```
    c_wj =
    c_theta_y =
    c_vij =
    c_theta_u =
    wj = wj - learn_rate * c_wj
    theta_y = ..
     ⋮
end
```

step 5° 使用模型

```
test-put = logsig(Vij * x_test-std + repmat(theta-u,1,test_num)
test-out-std = logsig(Wj * test-put + repmat(theta-y,1,test_num)
```

step 6° 反归一化

```
test_pre_out = test-out_std * y_train-sigma + y_train_mu
errors_nn = sum(abs(test-pre_out' - y_test)./y-test))/length( )
                                        % 误差统计
```

step 7° 画图

```
figure(1)
plot(Obj-save,'b-','LineWidth',1.5)
title('损失函数')
xlabel('epoch')
ylabel('errors')
figure(2)
color = [111,168,86;128,199,252;112,138,248;184,84,240]/255
plot(y-test,'color',color(2,:),'Linewidth',1)
hold on
plot(test-pre-out,'*','Color',color(1,:))
hold on
titlestr = ['公式推导BP神经网络','误差为:',num2str(errors_nn)]
titile(titlestr)
```