二分类.

$error = -y \cdot \log\hat{y} - (1-y)\log(1-\hat{y})$.

或 $= \frac{1}{2}(y-\hat{y})^2$

error的选择关系到 w的更新函数.

1. $w_i \leftarrow w_i + 2(y-\hat{y})x_i$

即 $w \leftarrow w + 2(y-\hat{y})x$

2. $w \leftarrow w + 2(y-\hat{y})\cdot\hat{y}\cdot(1-\hat{y})$.

$y = sigmoid([w_1, w_2, \cdots w_n]\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix})$.

多分类.

$error = sum(-(y * \log(1-\hat{y}))$ 与上面一致. 误差函数决定下方的更新方式

$w \leftarrow w + 2(y-\hat{y})\cdot x$.

$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = softmax(\begin{vmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix})$.

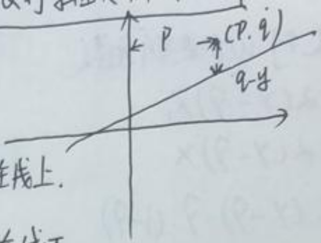
[ ] [ ] ⇒ [ ] [ ]

※ x = np.expand_dims(x, axis=1) x由[ ]变为[[]]

Linear Regression 线性回归

$y = w_1 x + w_2$

P为点Q到y轴距离即 |x|



$y = (w_1 + p)x + (w_2 + 1)$

⇓ α: learning Rate

$y = (w_1 + p\alpha)x + (w_2 + \alpha)$ 点在线上.

$y = (w_1 - p\alpha)x + (w_2 - \alpha)$ 点在线下.

⇓.

$y = (w_1 + p\alpha(q - y))x + (w_2 + \alpha(q - y))$. 不需绝对值 y为直线输出

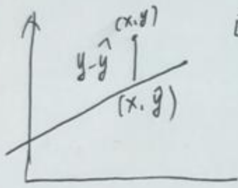
梯度下降更新法也可 $e = \frac{1}{2}(y - \hat{y})^2$ 求导更新.

两者一致

$w_i = w_i - \frac{\partial}{\partial w_i} Error.$

Error Functions

Mean Absolute Error.



$Error = \sum_{i=1}^{m} |y - \hat{y}| \cdot \frac{1}{m}$

$\frac{\partial}{\partial w_1} Error = \pm x$

$\frac{\partial}{\partial w_2} Error = \pm 1$

Mean Squared Error.

$Error = \frac{1}{2m} \sum_{i=1}^{m} (y - \hat{y})^2$

$Error = \frac{1}{2}(y - \hat{y})^2$

$\frac{\partial}{\partial w_1} Error = -(y - \hat{y})x.$

$\frac{\partial}{\partial w_2} Error = -(y - \hat{y})$

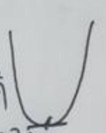
高维度 $\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_{n-1} x_{n-1} + w_n$

$E(w_1 w_2) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y} - y)^2.$

$\frac{\partial E}{\partial w_1} = \frac{\sum x_i^2}{m} w_1 + \frac{\sum x_i}{m} w_2 + \frac{\sum x_i y_i}{m}. = 0$

$\frac{\partial E}{\partial w_2} = \frac{\sum x_i}{m} w_1 + w_2 + \frac{\sum y_i}{m}. = 0$

n个则有n个方程 不易解



整理，得式子=0的目的是求 ⋃ 误差函数最小值即导数=0

二维：

$$E(w_1,w_2)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}-y)^2 \Rightarrow \frac{1}{2m}\sum_{i=1}^{m}(w_1x_i+w_2-y_i)$$

$$\frac{\partial E}{\partial w_1}=\sum_{i=1}^{m}(w_1x_i+w_2-y_i)x_i$$

$$=w_1\sum_{i=1}^{m}x_i^2+w_2\sum_{i=1}^{m}x_i-\sum_{i=1}^{m}x_iy_i=0$$

最低点导=0.

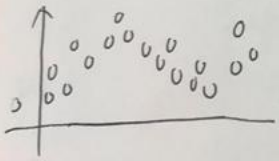
$$\frac{\partial E}{\partial w_2}=\sum_{i=1}^{m}(w_1x_i+w_2-y_i)$$

$$=w_1\sum_{i=1}^{m}x_i+w_2\sum_{i=1}^{m}1-\sum_{i=1}^{m}y_i=0$$

n维的情况是n个等于0的等式 则求解过程非常麻烦

所以一般是使用梯度下降的方式

多项式回归



$$\hat{y}=w_1x^3+w_2x^2+w_3x+w_4$$

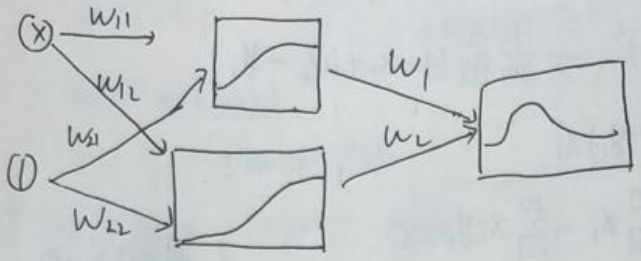
分类与回归均可

正则化：L_1 L_2 正则化可看作损失函数的惩罚项，惩罚是指对损失函数中的某些参数做一些限制。

L_1 正则化 w 中各元素的绝对值之和 $\|w\|_1$ 产生稀疏权值矩阵

L_2 正则化 w 中各元素的平方和再求平方根 $\|w\|_2$ 防止过拟合

Regression Neural Network. 回归神经网络



SSE. 对差值取平方再求和

$E = \frac{1}{2}\sum_u (y^u - \hat{y}^u)^2$. 误差项不同，则更新w的式不同

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}(y-\hat{y})^2$$

$$= \frac{\partial}{\partial w_i}\frac{1}{2}(y-\hat{y}(w_i))^2$$

$$\frac{\partial E}{\partial w_i} = (y-\hat{y})\frac{\partial}{\partial w_i}(y-\hat{y}) \quad \hat{y} = f(h)$$ 激活函数

$$h = \sum_i w_i x_i$$

$$\frac{\partial E}{\partial w_i} = -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_i} = -(y-\hat{y})f'(h)\frac{\partial}{\partial w_i}\sum w_i x_i$$

$$\frac{\partial E}{\partial w_i} = -(y-\hat{y})\cdot f'(h)\cdot x_i$$

$$\boxed{\Delta w_i = \eta(y-\hat{y})f(h)x_i}$$

$$\sigma'(x) = \sigma(x)\cdot(1-\sigma(x))$$

多层感知器.

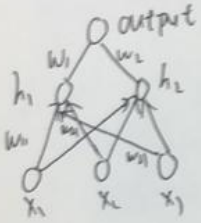
arr[:, None] 对一维行向量进行转置 features[:, None]

创建一个二维数组 [[ ]] 再进行转置T

x=np.expend_dims(x, axis=1) 扩充为二维数组此时x为4*1

y=np.expand_dims(y, axis=0) 扩充为二维数组此时y为1*3

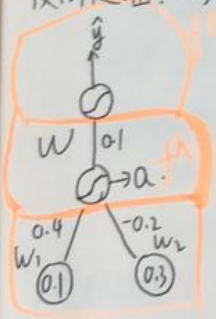
补充使用reshape( ) np.array(features, ndmin=2).T



$$\begin{matrix} h_1 & h_2 \\ \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \end{matrix} \quad [x_1\ x_2\ x_3] \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} = [h_1\ h_2]$$

$$\text{output} = [h_1, h_2] \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

反向传播.例.



$h = \sum w_i x_i = 0.1 \times 0.4 - 0.2 \times 0.3 = -0.02$

$a = f(h) = \text{sigmoid}(-0.02) = 0.495$

$\hat{y} = f(w \times a) = \text{sigmoid}(0.1 \times 0.495) = 0.512$

反向传播

$\delta^o = (y-\hat{y}) f'(w \cdot a) = (1-0.512) \cdot 0.512 \cdot (1-0.512) = 0.122$

$\delta^h = \eta \delta^o a = 0.5 \times 0.122 \times 0.495 = 0.0302$

$w = \eta \delta^o a = 0.5 \times 0.122$

$\delta^h = w \delta^o f'(h) = 0.1 \times 0.122 \times 0.495 \times (1-0.495) = 0.003$

$\Delta w = \eta \delta^o a = 0.5 \times 0.122 \times 0.495 = 0.0302$

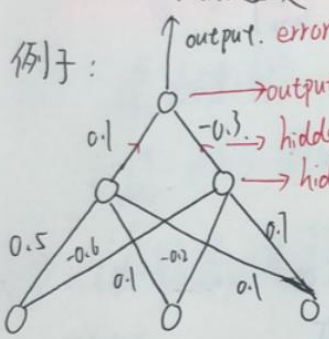
$\Delta w_i = \eta \delta^h x_i = (0.5 \times 0.003 \times 0.1, 0.5 \times 0.003 \times 0.3)$

1) $g^o = (y-\hat{y}) f'(h)$
第一个误差项.

2) 后续误差项 对隐藏节点 j
$g_j^h = \sum w_{jk} g_k^o f'(h_j)$
→ 该层的权重
→ 上一层误差项

3) $\Delta w_{ij} = \eta g_j^h x_i$
↓ 该层误差项
↘ 该层的输入

例子:



output. error = $(Y-\hat{y})$.
→ output_error = error * $\hat{y}$ * $(1-\hat{y})$ 单值
→ hidden_error = hidden_output_weight * output_error * output
hidden_output * (1 - hidden_output). [双

ΔW_0 = 2 * output_error * hidd_output
ΔW_1 = 2 * hidden_error * X. 注意格式

实现反向传播 直接由数学公式逆为代码

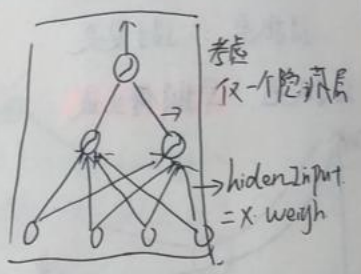
输出层的误差. 一般 $f = \sigma(x)$ $f' = \sigma(x)(1-\sigma(x))$

$\delta_k = (y_k - \hat{y}_k) f'(a_k)$

$\delta_k = (y_k - \hat{y}_k) f'(a_k)$

隐藏层误差:

$\delta_j = \sum [w_{jk} \delta_k] f'(h_j)$



1. 每一层权重更新初始步长为 0
   - 输入至隐藏层 $\Delta w_{ij} = 0$
   - 隐藏层到输出层 $\Delta w_j = 0$

2. 对于训练数据当中的每一个点.
   - 让它正向通过网络, 计算输出 $\hat{y}$
   - 计算输出节点的误差梯度 $\delta^o = (y - \hat{y}) f'(z)$. $z = \sum_j w_j a_j \Rightarrow \hat{y}$
   - 误差传播到隐藏层 $\delta_j^h = \delta^o w_j f'(h_j)$ h_j 该层的输出 → 该层的输出所需要的相关的权
   - 更新权重步长
     $\Delta w_j = \Delta w_j + \delta^o a_j$ → 输出层输入.
     $\Delta w_{ij} = \Delta w_{ij} + \delta_j^h a_i$ → 隐藏层输入. $x[:, None]$.

3. 更新权重.
   $w_j = w_j + \eta \Delta w_j / m$.
   $w_{ij} = w_{ij} + \eta \Delta w_{ij} / m$.

4. 重复这个过程 e 代

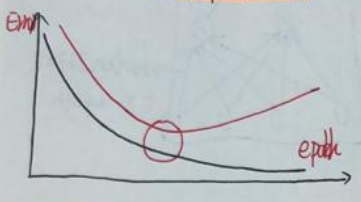
weight : size = n_features, n_hidden

训练神经网络

欠拟合：解决问题的方案太简单，无法完成工作

过拟合：过于复杂

过拟合：早期停止法

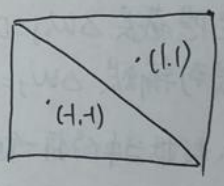


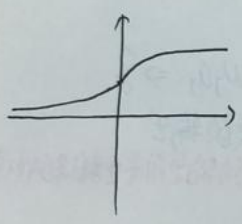
降低梯度，直到测试误差停止降低并开始

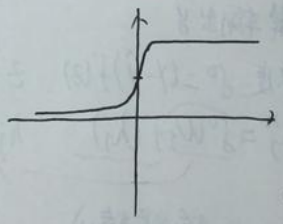
正则化

why 1. solution 1：x_1+x_2

solution 2：$10x_1+10x_2$







→ 利于梯度下降

惩罚高系数

L_1 error $= -\frac{1}{m}\sum_{i=1}^{m}\left((1-y_i)(1-\ln(\hat{y}_i))+y_i\ln(\hat{y}_i)\right)+\lambda(|w_1|+..+|w$

L_2 error $= -\frac{1}{m}\sum_{i=1}^{m}\left((1-y_i)(1-\ln(\hat{y}_i))+y_i\ln(\hat{y}_i)\right)+\lambda(w_1^2+..+w_n^2)$

L_1 稀疏矩阵，较小权重趋向于0

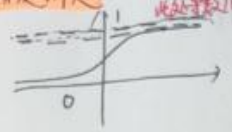
Dropout

TRAIN

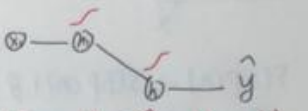
Don't

每个节点在特定 epoch 被放弃的概率

梯度消失



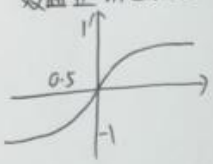
$\sigma(x)=\frac{1}{1+e^{-x}}$



$\frac{\partial E}{\partial w_{11}^{(1)}}=\frac{\partial E}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h}\cdot\frac{\partial h}{\partial h_1}\cdot\frac{\partial h_1}{\partial w_{11}}$

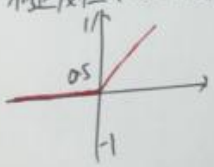
其他激活函数

双曲正切函数



$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ [-1,1]之间让导数更大

修正线性单元 Relu



$relu(x)=\begin{cases}x & \text{if } x\geq 0\\ 0 & \text{if } x<0\end{cases}$

model.add(Activation(' ')) sigmoid softmax relu tanh

tch 和随机梯度下降

将数据分批进行训练

model.fit(X_train, Y_train, epochs=1000, batch_size=100, verbose=0)

将数据分为100批，每批训练1000次