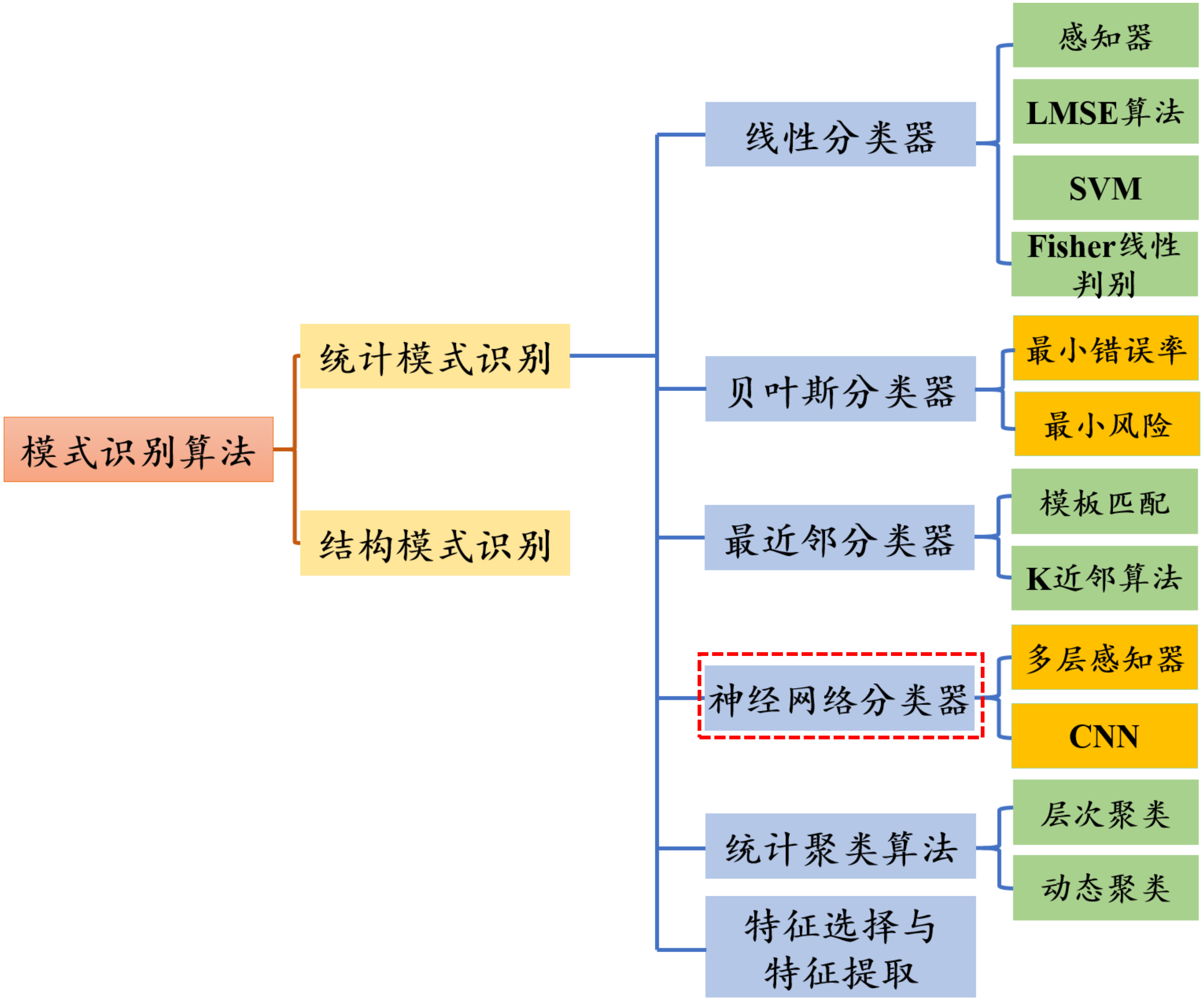
模式识别课程

神经网络分类器

张俊超

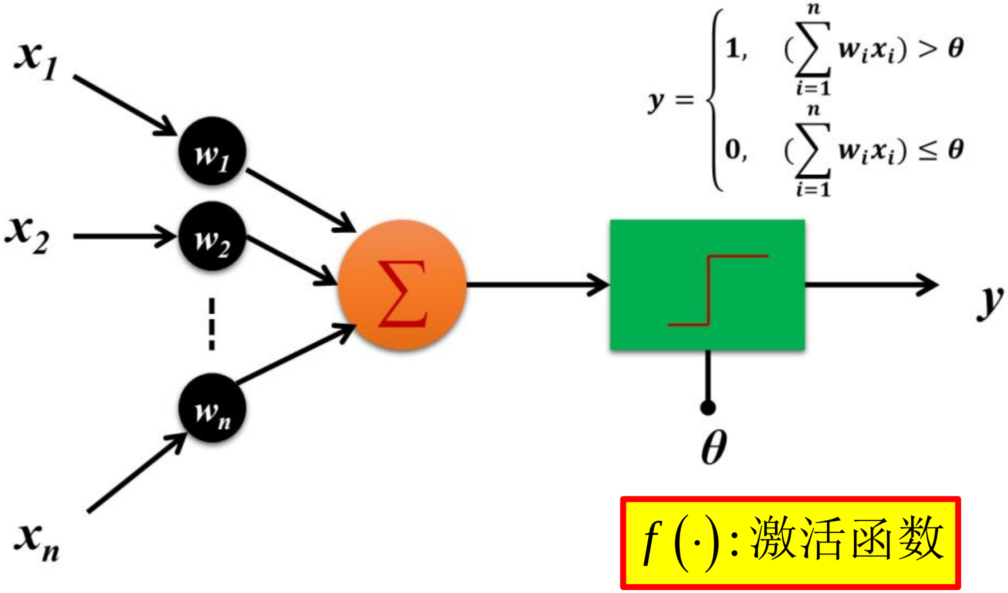
中南大学
航空航天学院

模式识别-神经网络分类器



模式识别-神经网络分类器

感知器： 1957 年罗森布拉特提出的感知器,激起了第一次人工神经网络的热潮。



$$y=\begin{cases}1, & (\sum_{i=1}^{n} w_i x_i) > \theta\\ 0, & (\sum_{i=1}^{n} w_i x_i) \le \theta\end{cases}$$

$f(\cdot)$:激活函数

$$G(\mathbf{x})=\mathbf{w}^T\mathbf{x}-\theta$$

$$y=\begin{cases}1, G(\mathbf{x})>0\\ 0, G(\mathbf{x})\le 0\end{cases}$$

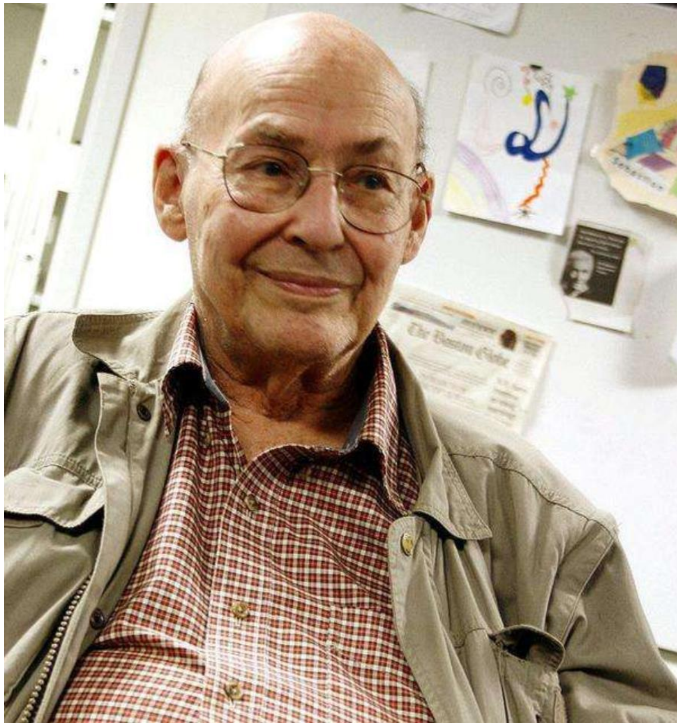
$$\Rightarrow$$

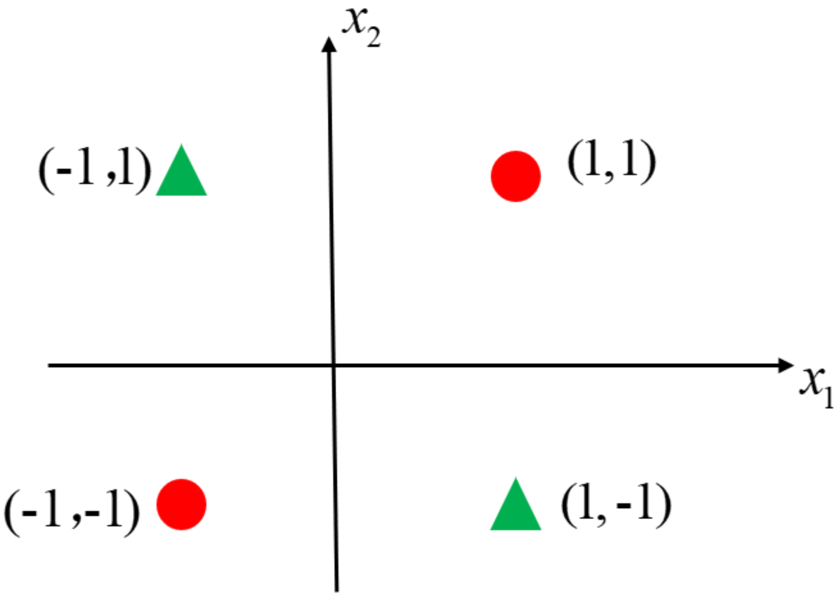
$$G(\mathbf{x})=\mathbf{w}^T\mathbf{x}+b$$

$$y=f\left(G(\mathbf{x})\right)=\begin{cases}1, G(\mathbf{x})>0\\ 0, G(\mathbf{x})\le 0\end{cases}$$

模式识别 -神经网络分类器

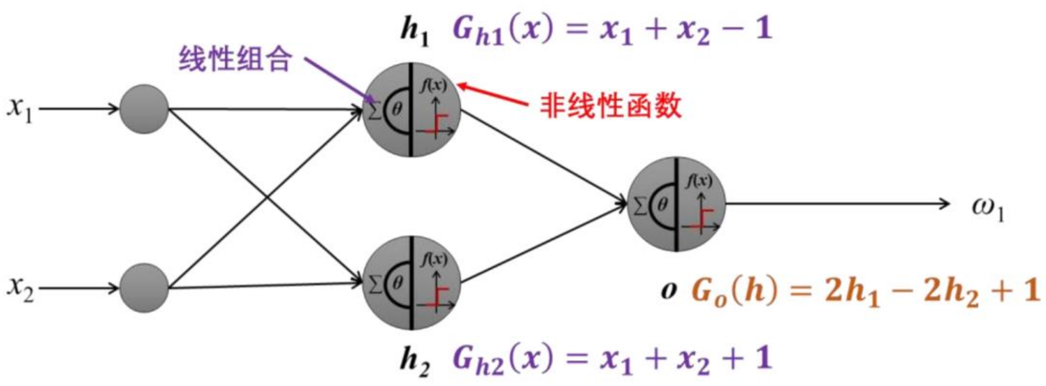
1969 年明斯基等人无情的批判，导致人工神经网络研究跌入第一个寒冬。

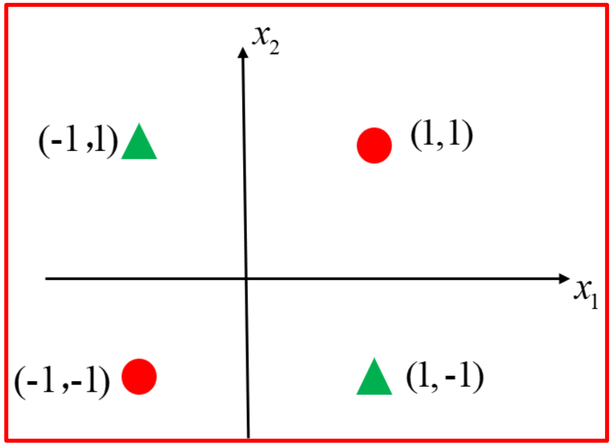




异或问题能够用感知器解决吗？

模式识别-神经网络分类器

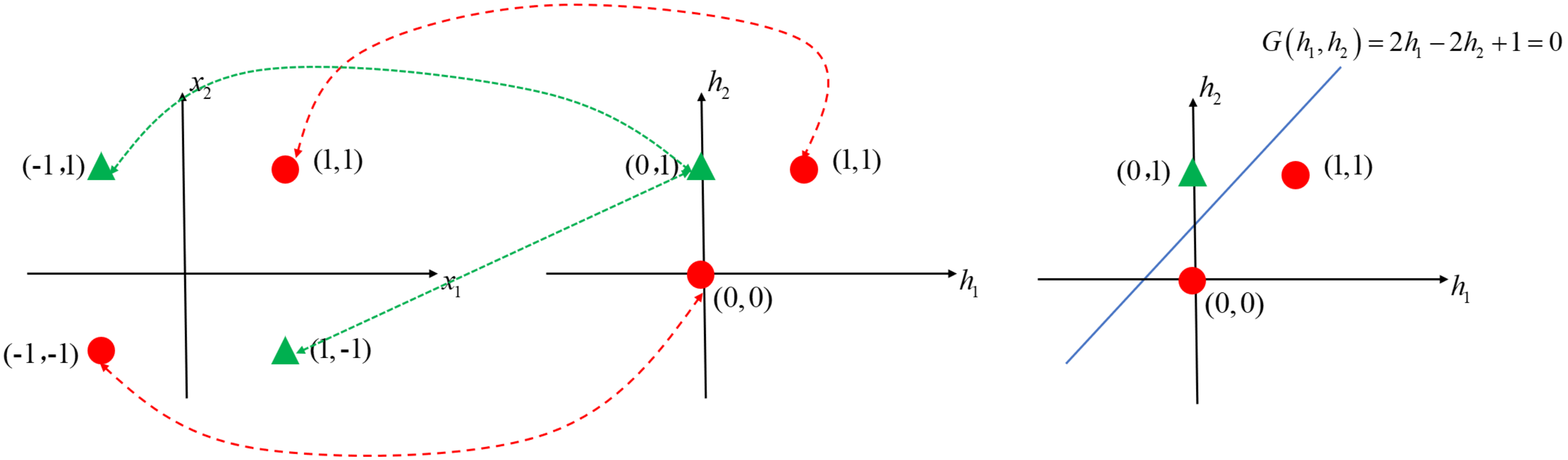




$$\left.\begin{aligned} G_{h_1}(1,1)&=1+1-1=1, y_{h_1}=1 \\ G_{h_2}(1,1)&=1+1+1=3, y_{h_2}=1 \end{aligned}\right\} \rightarrow G_o(1,1)=2*1-2*1+1=1, y_o=1$$

$$\left.\begin{aligned} G_{h_1}(1,\text{-}1)&=1-1-1=-1, y_{h_1}=0 \\ G_{h_2}(1,\text{-}1)&=1-1+1=1, y_{h_2}=1 \end{aligned}\right\} \rightarrow G_o(0,1)=2*0-2*1+1=\text{-}1, y_o=0$$

模式识别-神经网络分类器



两层感知器并联，再与一个串联，解决了异或问题。可是，多次感知器是如何训练的呢？

罗森布拉特：固定前面的权重，用感知器算法只更新最后一层的权重。

模式识别 -神经网络分类器

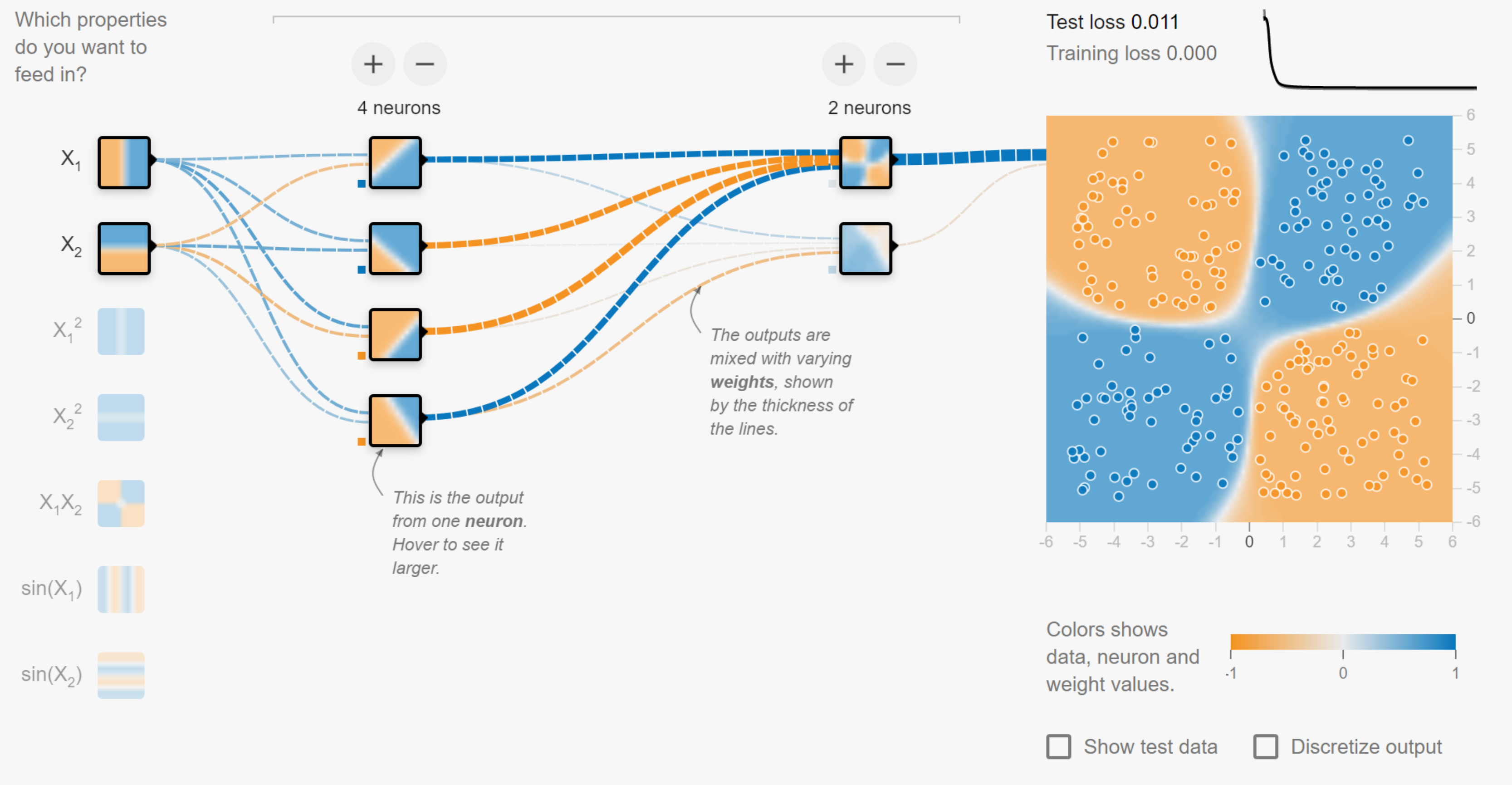
多层感知权重的更新： 固定前面的权重，用感知器算法只更新最后一层的权重

- 前面的权重需要事先人为设定，导致模型的性能很大程度上取决于事先选择的权重。
- 未能找到能够针对任意问题的前面权重赋值的方法。

1986年，Rumelhart 和Hinton等人提出误差反向传播算法(BP, back propagation)

模式识别 -神经网络分类器

http://playground.tensorflow.org



Which properties do you want to feed in?

Test loss 0.011

Training loss 0.000

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

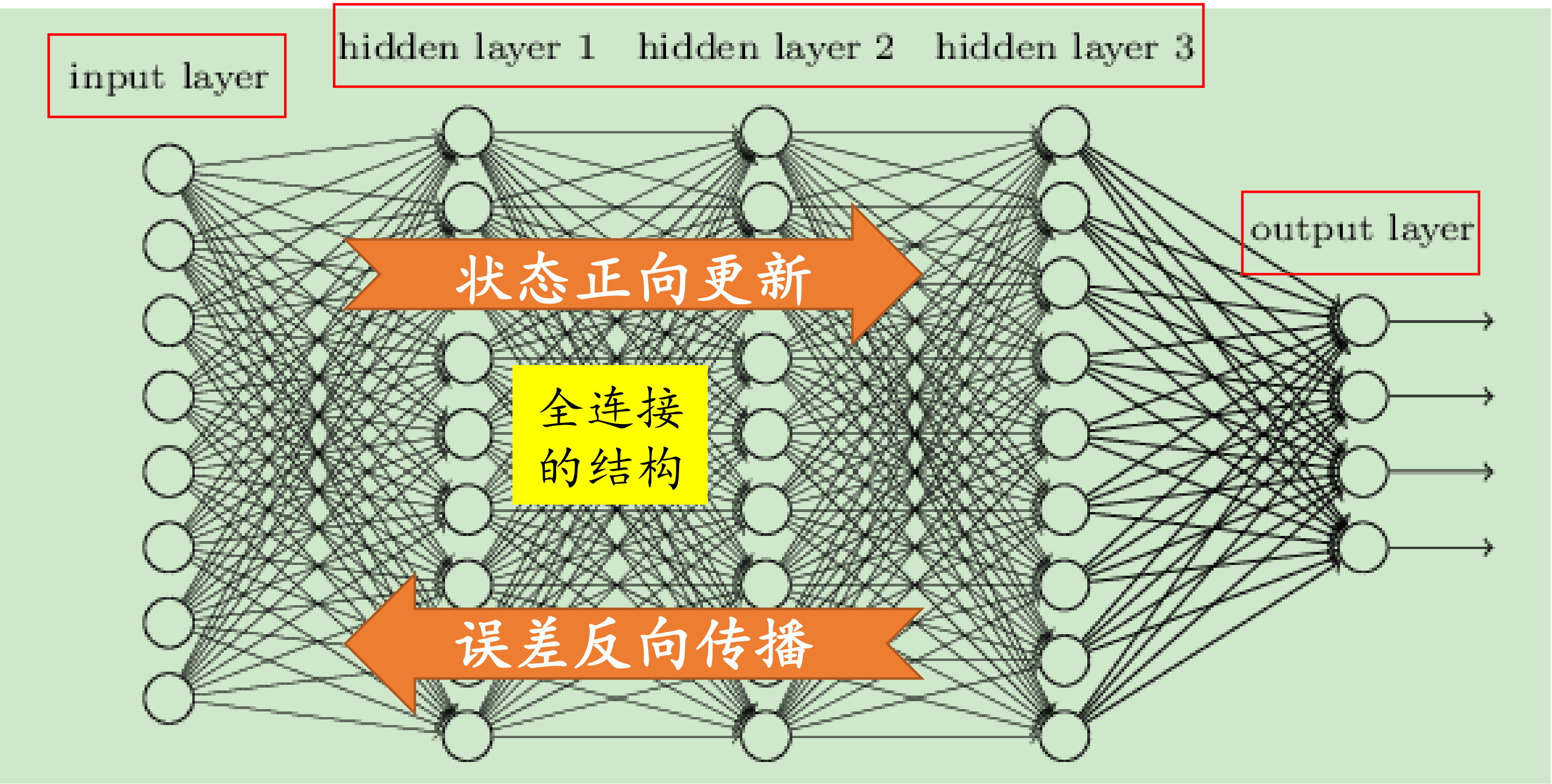
*This is the output from one **neuron**. Hover to see it larger.*

Colors shows data, neuron and weight values.

□ Show test data □ Discretize output

模式识别 -神经网络分类器

人工神经网络(ANN, Artificial Neural Network)



模式识别-神经网络分类器

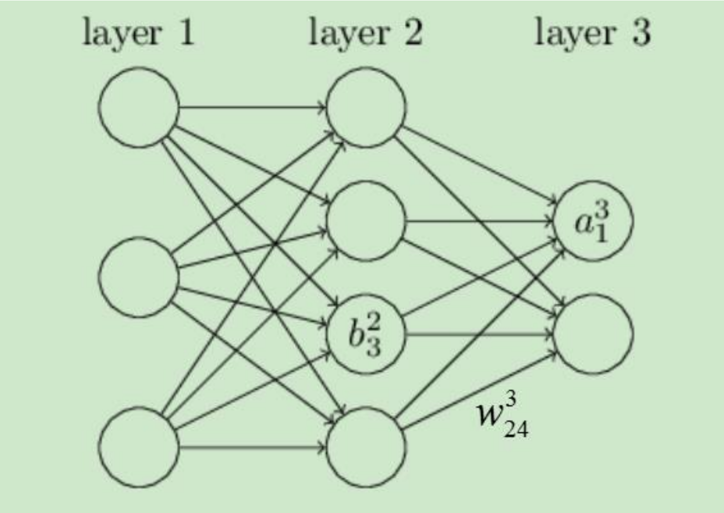
【记号】

w_{jk}^{l} :第$l-1$层的第k个神经元到第l层的第j个神经元的权重

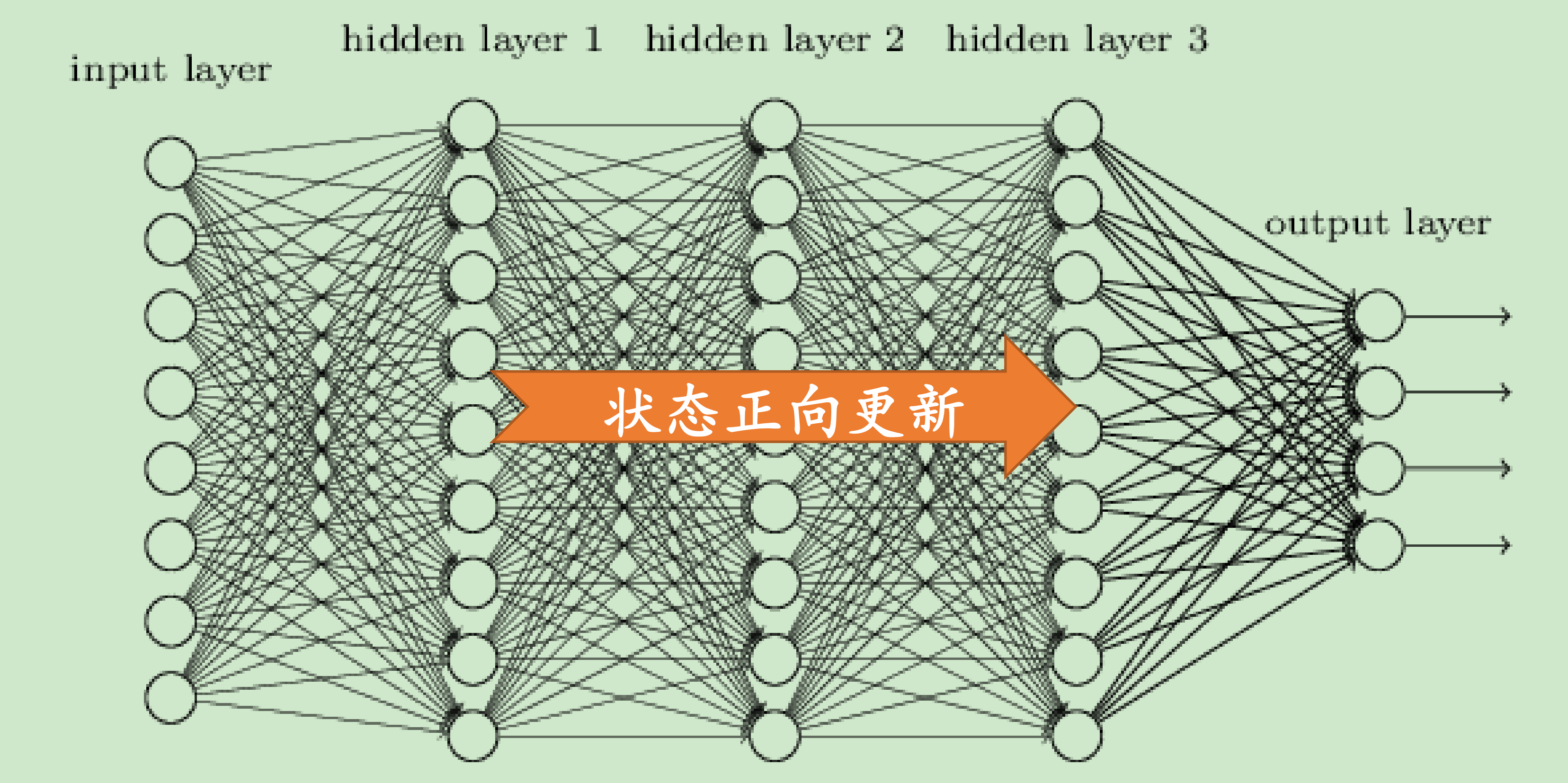
b_{j}^{l} :第l层的第j个神经元对应的偏置

需要考虑前一层

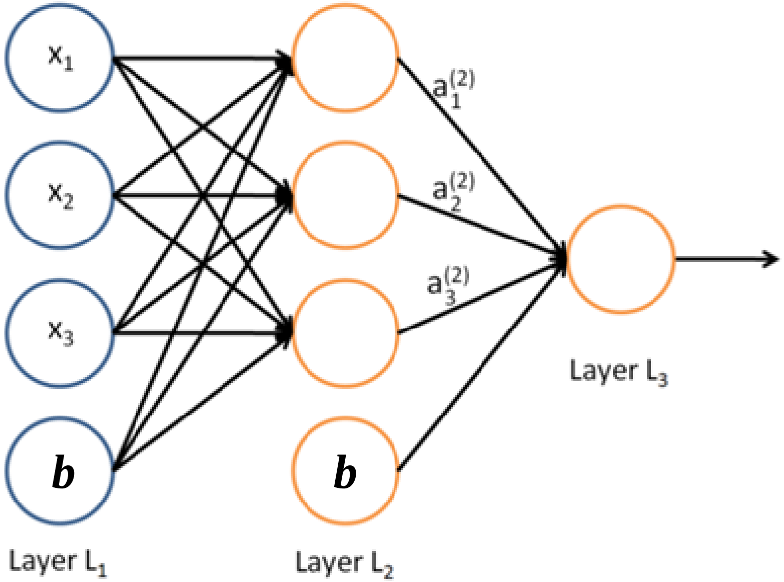
a_{j}^{l} :第l层的第j个神经元对应的输出



神经网络分类器-前向传播算法



神经网络分类器-前向传播算法



对于第二层的输出a_1^2, a_2^2, a_3^2，我们有：

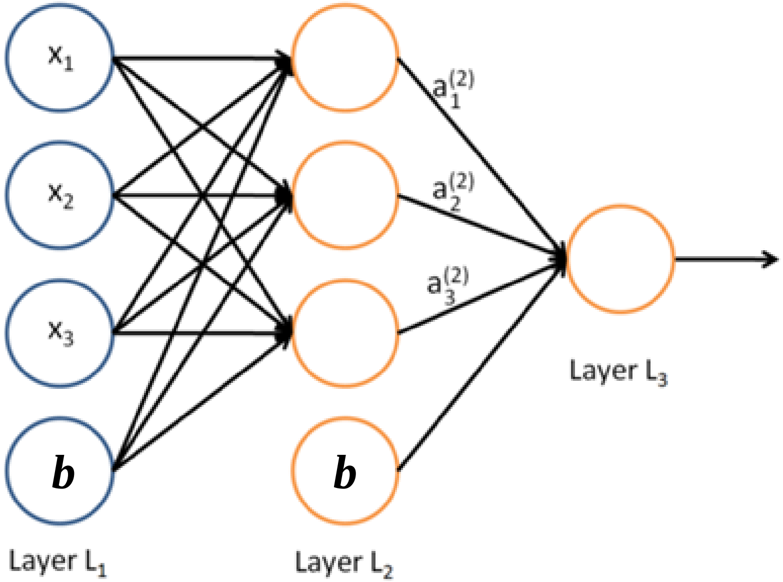
$$a_1^2 = \sigma(z_1^2) = \sigma(w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3 + b_1^2)$$

$$a_2^2 = \sigma(z_2^2) = \sigma(w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3 + b_2^2)$$

$$a_3^2 = \sigma(z_3^2) = \sigma(w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3 + b_3^2)$$

其中，$\sigma(\cdot)$是激活函数

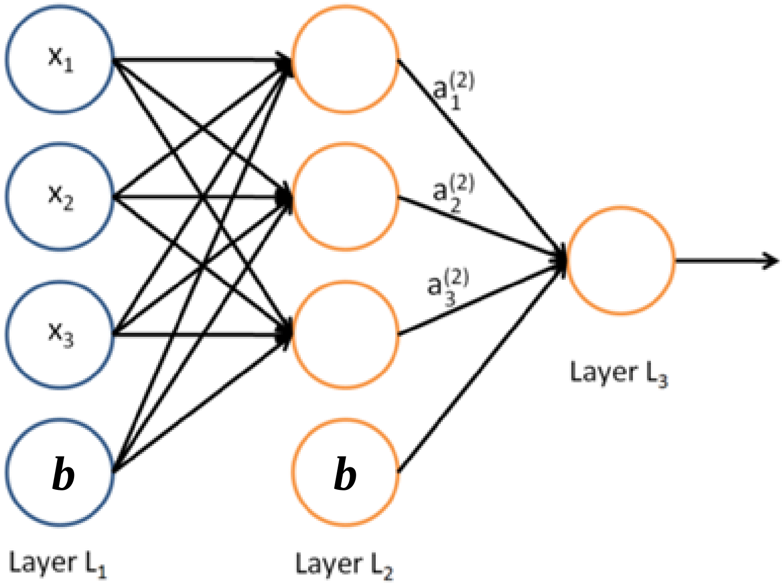
神经网络分类器-前向传播算法



对于第三层的输出a_1^3，我们有：

$$a_1^3 = \sigma(z_1^3) = \sigma(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + b_1^3)$$

神经网络分类器-前向传播算法



将上面的式子一般化，设$l-1$层有m个神经元，则对第l层的第j个神经元的输出a_j^l，我们有：

$$\boxed{a_j^l = \sigma(z_j^l) = \sigma(\sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l)}$$

神经网络分类器-前向传播算法

矩阵表示：

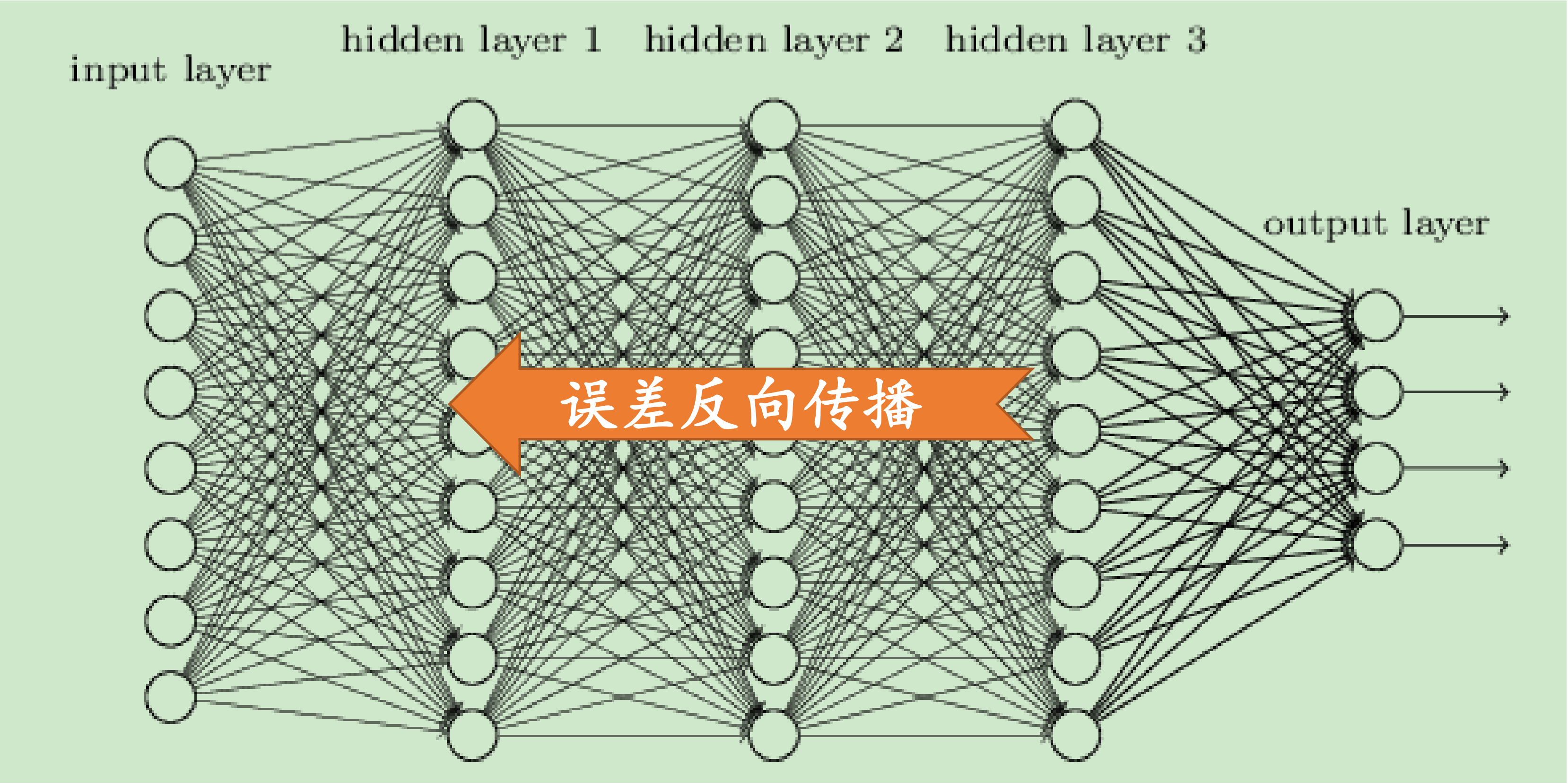
假设$l-1$层有m个神经元，l层有n个神经元，则l层的权重w就组成了一个$n\times m$的矩阵$\mathbf{W}^l$，偏置就组成了一个$n\times 1$的向量$\mathbf{b}^l$，第$l-1$的输出就组成了一个$m\times 1$的向量$\mathbf{a}^{l-1}$，第l层的未激活前的输出$\mathbf{z}^l$，是一个$n\times 1$的向量。第l的输出就组成了一个$n\times 1$的向量$\mathbf{a}^l$。

$$\mathbf{a}^l=\sigma(\mathbf{z}^l)=\sigma(\mathbf{W}^l\mathbf{a}^{l-1}+\mathbf{b}^l)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_n^l \end{bmatrix}=\sigma(\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1m}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2m}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^l & w_{n2}^l & \cdots & w_{nm}^l \end{bmatrix}\begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_m^{l-1} \end{bmatrix}+\boxed{\begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_n^l \end{bmatrix}})$$

神经网络分类器-反向传播算法

更新权重：



神经网络分类器-反向传播算法

一般步骤：

- 选择损失函数
- 最小化损失函数
- 梯度下降法更新权重

均方差损失函数: $\boxed{J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y}) = \frac{1}{2}\left\|\mathbf{a}^{L} - \mathbf{y}\right\|_2^2}$

$\mathbf{x}$: 输入

$\mathbf{y}$: 期望的输出

$\mathbf{W},\mathbf{b}$: 权重

L: 代表输出层

神经网络分类器-反向传播算法

矩阵求导过程参考链接：
https://zhuanlan.zhihu.com/p/24709748

首先是输出层：第 L 层，输出层满足如下关系：

$$\mathbf{a}^L=\sigma(\mathbf{z}^L)=\sigma(\mathbf{W}^L\mathbf{a}^{L-1}+\mathbf{b}^L)$$

对于**输出层**，我们的损失函数变为：

$$J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})=\frac{1}{2}\left\|\mathbf{a}^L-\mathbf{y}\right\|_2^2=\frac{1}{2}\left\|\sigma(\mathbf{W}^L\mathbf{a}^{L-1}+\mathbf{b}^L)-\mathbf{y}\right\|_2^2$$

这样求解 **W**,**b** 的梯度为：

$$t=f\left(\mathbf{s}\right),\mathbf{s}=\mathbf{Ax}+\mathbf{v}\Rightarrow\frac{\partial t}{\partial\mathbf{A}}=\frac{\partial t}{\partial\mathbf{s}}\mathbf{x}^T$$

标量对矩阵求导

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{W}^L}=(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)(\mathbf{a}^{L-1})^T$$

标量对向量求导

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{b}^L}=\left(\frac{\partial\mathbf{z}^L}{\partial\mathbf{b}^L}\right)^T\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{z}^L}=(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)$$

从上式我们可以看到，存在公共部分，记为：

$$\boldsymbol{\delta}^L=\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{z}^L}=(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)$$

神经网络分类器-反向传播算法

矩阵微分的性质

(1)微分加减法：$d(\mathbf{X}+\mathbf{Y})=d\mathbf{X}+d\mathbf{Y}, d(\mathbf{X}-\mathbf{Y})=d\mathbf{X}-d\mathbf{Y}$

(2)微分乘法：$d(\mathbf{X}\mathbf{Y})=(d\mathbf{X})\mathbf{Y}+\mathbf{X}(d\mathbf{Y})$

(3)微分转置：$d(\mathbf{X}^T)=(d\mathbf{X})^T$

(4)微分的迹：$dtr(\mathbf{X})=tr(d\mathbf{X})$

(5)微分哈达玛乘积：$d(\mathbf{X}\odot\mathbf{Y})=\mathbf{X}\odot d\mathbf{Y}+d\mathbf{X}\odot\mathbf{Y}$

(6)逐元素求导：$d\sigma(\mathbf{X})=\sigma'(\mathbf{X})\odot d\mathbf{X}$

(7)逆矩阵微分：$d\mathbf{X}^{-1}=-\mathbf{X}^{-1}d\mathbf{X}\mathbf{X}^{-1}$

(8)行列式微分：$d|\mathbf{X}|=|\mathbf{X}|tr(\mathbf{X}^{-1}d\mathbf{X})$

$$df=tr((\frac{\partial f}{\partial \mathbf{X}})^T d\mathbf{X})$$

$$df=tr((\frac{\partial f}{\partial \mathbf{x}})^T d\mathbf{x})$$

$$tr((\mathbf{A}\odot\mathbf{B})^T\mathbf{C})=tr(\mathbf{A}^T(\mathbf{B}\odot\mathbf{C}))$$

神经网络分类器-反向传播算法

$$\mathbf{z}^L=\mathbf{W}^L\mathbf{a}^{L-1}+\mathbf{b}^L$$

$$(1)\Rightarrow\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{W}^L}=\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{z}^L}(\mathbf{a}^{L-1})^T$$

$$J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})=\frac{1}{2}\left\|\mathbf{a}^L-\mathbf{y}\right\|_2^2=\frac{1}{2}\left\|\sigma(\mathbf{W}^L\mathbf{a}^{L-1}+\mathbf{b}^L)-\mathbf{y}\right\|_2^2$$

$$dJ=\left(\sigma'\left(\mathbf{z}^L\right)\odot d\mathbf{z}^L\right)^T\left(\mathbf{a}^L-\mathbf{y}\right)$$

链式法则求导:

向量: x, y, z, $\frac{\partial z}{\partial x}=\frac{\partial z}{\partial y}$、$\frac{\partial y}{\partial x}$

标: a, 向量 x, y. $\frac{\partial a}{\partial x}=\left(\frac{\partial y}{\partial x}\right)^T\cdot\frac{\partial a}{\partial y}$

$$tr\left(dJ\right)=tr\left[\left(\sigma'\left(\mathbf{z}^L\right)\odot d\mathbf{z}^L\right)^T\left(\mathbf{a}^L-\mathbf{y}\right)\right]$$

$$=tr\left[\left(\mathbf{a}^L-\mathbf{y}\right)^T\left(\sigma'\left(\mathbf{z}^L\right)\odot d\mathbf{z}^L\right)\right]$$

$$=tr\left[\left[\left(\mathbf{a}^L-\mathbf{y}\right)\odot\sigma'\left(\mathbf{z}^L\right)\right]^T d\mathbf{z}^L\right]$$

$$\frac{\partial J}{\partial\mathbf{z}^L}=\left(\mathbf{a}^L-\mathbf{y}\right)\odot\sigma'\left(\mathbf{z}^L\right)$$

$$\boxed{\begin{aligned}&\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{W}^L}=\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{z}^L}(\mathbf{a}^{L-1})^T\\&=\left(\mathbf{a}^L-\mathbf{y}\right)\odot\sigma'\left(\mathbf{z}^L\right)(\mathbf{a}^{L-1})^T\end{aligned}}$$

公式: $t=f(s)$, $s=Ax+V$. $\Rightarrow \frac{\partial t}{\partial A}=\frac{\partial t}{\partial s}\cdot x^T$

$$Z^L=W^La^{L-1}+b^L$$

$$J=\frac{1}{2}\|a^L-y\|_2^2=\frac{1}{2}\|\sigma(W^La^{L-1}+b^L)-y\|_2^2=\frac{1}{2}\|\sigma(z^L)-y\|_2^2$$

$$=f(z^L)$$

$$\frac{\partial J}{\partial W^L}=\frac{\partial J}{\partial Z^L}\cdot(a^{L-1})^T$$

(2)微分乘法: $d(\mathbf{XY})=(d\mathbf{X})\mathbf{Y}+\mathbf{X}(d\mathbf{Y})$

(3)微分转置: $d(\mathbf{X}^T)=(d\mathbf{X})^T$

$$J=\frac{1}{2}\|\sigma(z^L)-y\|_2^2=\frac{1}{2}[\sigma(z^L)]^T[\sigma(z^L)]-[\sigma(z^L)]^Ty+\frac{1}{2}y^Ty$$

$$dJ=\frac{1}{2}d\{[\sigma(z^L)]^T\}[\sigma(z^L)]+\frac{1}{2}[\sigma(z^L)]^Td\{[\sigma(z^L)]\}-d\{[\sigma(z^L)]^T\}y$$

$$-[\sigma(z^L)]^Td\{y\}+\frac{1}{2}d\{y^T\}y+\frac{1}{2}y^Td\{y\}$$

$$dJ=\frac{1}{2}[d\{\sigma(z^L)\}]^T[\sigma(z^L)]+\frac{1}{2}[\sigma(z^L)]^Td\{\sigma(z^L)\}-[d\{\sigma(z^L)\}]^Ty+0$$

$$=[d\{\sigma(z^L)\}]^T[\sigma(z^L)]-[d\{\sigma(z^L)\}]^Ty$$

$$=[d\{\sigma(z^L)\}]^T(\sigma(z^L)-y)$$

x $$=[d\{\sigma(z^L)\}]^T(a^L-y)$$

$$=[\sigma'(z^L)\odot d\{z^L\}]^T(a^L-y)$$

$$tr(dJ)=tr\left[\left(\sigma'(\mathbf{z}^L)\odot d\mathbf{z}^L\right)^T(\mathbf{a}^L-\mathbf{y})\right]$$

$$=tr\left[(\mathbf{a}^L-\mathbf{y})^T\left(\sigma'(\mathbf{z}^L)\odot d\mathbf{z}^L\right)\right]$$

$$=tr\left[\left[(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)\right]^Td\mathbf{z}^L\right]$$

$$\frac{\partial J}{\partial\mathbf{z}^L}=(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)$$

$$d\{\sigma(x)\}$$
$$=\sigma'(x)\odot dx$$

$$tr((\mathbf{A}\odot\mathbf{B})^T\mathbf{C})=tr(\mathbf{A}^T(\mathbf{B}\odot\mathbf{C}))$$

标量对向量求导

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{b}^L}=\left(\frac{\partial\mathbf{z}^L}{\partial\mathbf{b}^L}\right)^T\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial\mathbf{z}^L}=(\mathbf{a}^L-\mathbf{y})\odot\sigma'(\mathbf{z}^L)$$

神经网络分类器-反向传播算法

其它层的计算：

我们注意到第l层的未激活输出为$\mathbf{z}^l$，它的梯度可以表示为：

$$\boldsymbol{\delta}^l = \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^l} = \left(\frac{\partial \mathbf{z}^L}{\partial \mathbf{z}^{L-1}} \cdot \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{z}^{L-2}} \cdots \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^L}$$

如果我们依次获得$\boldsymbol{\delta}^l$，根据$\mathbf{z}^l = \mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l$，则：

$$t = f(\mathbf{s}), \mathbf{s} = \mathbf{A}\mathbf{x} + \mathbf{v} \Rightarrow \frac{\partial t}{\partial \mathbf{A}} = \frac{\partial t}{\partial \mathbf{s}}\mathbf{x}^T$$

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l(\mathbf{a}^{l-1})^T$$

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l$$

那如何求$\boldsymbol{\delta}^l$？

用数学归纳法，可得$\boldsymbol{\delta}^l = (\mathbf{W}^{l+1})^T\boldsymbol{\delta}^{l+1} \odot \sigma'(\mathbf{z}^l)$

神经网络分类器-反向传播算法

$$\boldsymbol{\delta}^l = \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^l} = \left(\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^{l+1}} = \left(\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \boldsymbol{\delta}^{l+1}$$

可见，用归纳法递推 $\boldsymbol{\delta}^{l+1}$ 和 $\boldsymbol{\delta}^l$ 的关键在于求解 $\boxed{\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}}$。

而 $\mathbf{z}^{l+1}$ 和 $\mathbf{z}^l$ 的关系是：

$$\mathbf{z}^{l+1} = \mathbf{W}^{l+1}\mathbf{a}^l + \mathbf{b}^{l+1} = \mathbf{W}^{l+1}\sigma(\mathbf{z}^l) + \mathbf{b}^{l+1}$$

很容易得到：

$$\boxed{\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l} = \mathbf{W}^{l+1} diag\left(\sigma'(\mathbf{z}^l)\right)}$$

将上式带入 $\boldsymbol{\delta}^l$ 中，可得：

$$\boldsymbol{\delta}^l = \left(\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \boldsymbol{\delta}^{l+1} = (\mathbf{W}^{l+1})^T \boldsymbol{\delta}^{l+1} \odot \sigma'(\mathbf{z}^l)$$

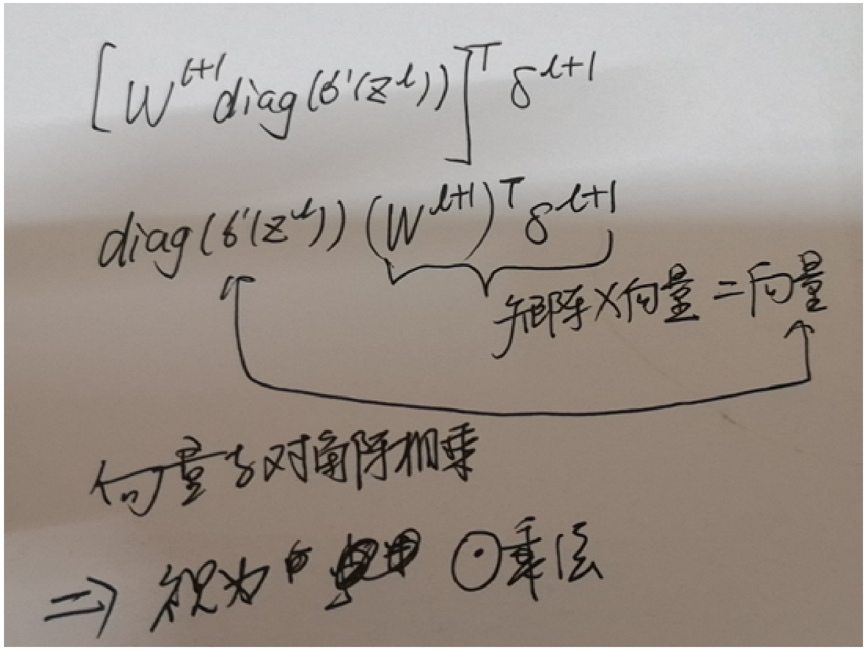
$$Z^{l+1} = W^{l+1}\sigma(Z^l) + b^{l+1}$$

$$\frac{\partial Z^{l+1}}{\partial Z^l} = \frac{\partial Z^{l+1}}{\partial\{\sigma(Z^l)\}} \cdot \frac{\partial\{\sigma(Z^l)\}}{\partial Z^l}$$

$$\sigma(Z^l) = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \qquad Z^l = \begin{Bmatrix} z_1 \\ \vdots \\ z_n \end{Bmatrix} \qquad \begin{aligned} a_1 &= \sigma(z_1) \\ \frac{da_1}{dz_1} &= \sigma'(z_1) \end{aligned}$$

$$\boldsymbol{\delta}^l = \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^l} = \left(\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{z}^{l+1}} = \left(\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}\right)^T \boldsymbol{\delta}^{l+1}$$

$$\boxed{\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l} = \mathbf{W}^{l+1} diag\left(\sigma'(\mathbf{z}^l)\right)}$$



$$\left[W^{l+1} diag(\sigma'(z^l))\right]^T \delta^{l+1}$$

$$diag(\sigma'(z^l)) \underbrace{(W^{l+1})^T \delta^{l+1}}_{\text{矩阵×向量=向量}}$$

向量与对角阵相乘

$\Rightarrow$ 视为 ⊙乘法

神经网络分类器-反向传播算法

矩阵求导辅助学习资料

- https://www.cnblogs.com/pinard/p/10750718.html
- https://www.cnblogs.com/pinard/p/10773942.html
- https://www.cnblogs.com/pinard/p/10791506.html
- https://www.cnblogs.com/pinard/p/10825264.html
- https://www.cnblogs.com/pinard/p/10930902.html

神经网络分类器-反向传播算法

梯度下降法更新权重：

$$\mathbf{W}^{l}(t+1)=\mathbf{W}^{l}(t)-\eta\cdot\nabla_{\mathbf{W}^{l}}=\mathbf{W}^{l}(t)-\eta\boldsymbol{\delta}^{l}(t)\left(\mathbf{a}^{l-1}(t)\right)^{T}$$

$$\mathbf{b}^{l}(t+1)=\mathbf{b}^{l}(t)-\eta\cdot\nabla_{\mathbf{b}^{l}}=\mathbf{b}^{l}(t)-\eta\boldsymbol{\delta}^{l}(t)$$

$Mini-batch\quad SGD$(随机梯度下降法)

m个样本

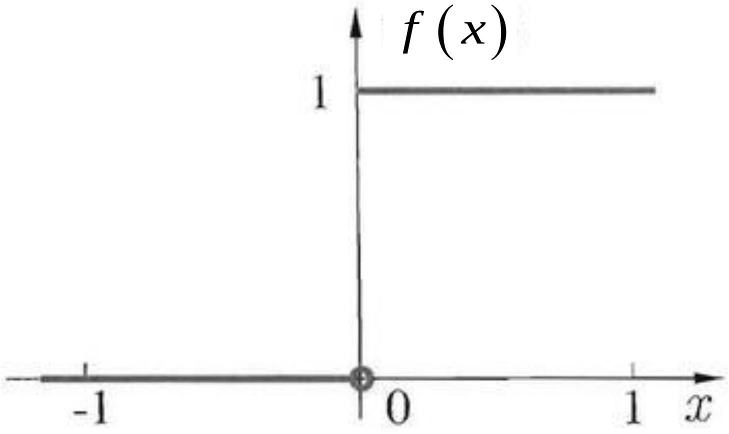
$$\mathbf{W}^{l}(t+1)=\mathbf{W}^{l}(t)-\eta\cdot\nabla_{\mathbf{W}^{l}}=\mathbf{W}^{l}(t)-\eta\sum_{i=1}^{m}\boldsymbol{\delta}^{i,l}(t)\left(\mathbf{a}^{i,l-1}(t)\right)^{T}$$

$$\mathbf{b}^{l}(t+1)=\mathbf{b}^{l}(t)-\eta\cdot\nabla_{\mathbf{b}^{l}}=\mathbf{b}^{l}(t)-\eta\sum_{i=1}^{m}\boldsymbol{\delta}^{i,l}(t)$$

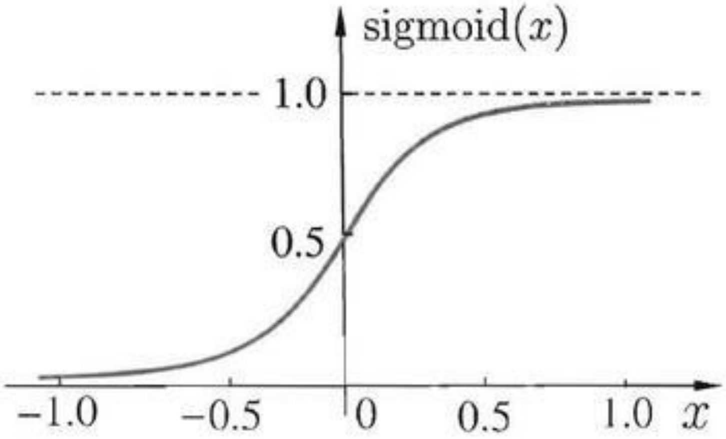
神经网络分类器-反向传播算法

$$\mathbf{a}^l=\sigma(\mathbf{z}^l)=\sigma(\mathbf{W}^l\mathbf{a}^{l-1}+\mathbf{b}^l)$$

激活函数如何选择？



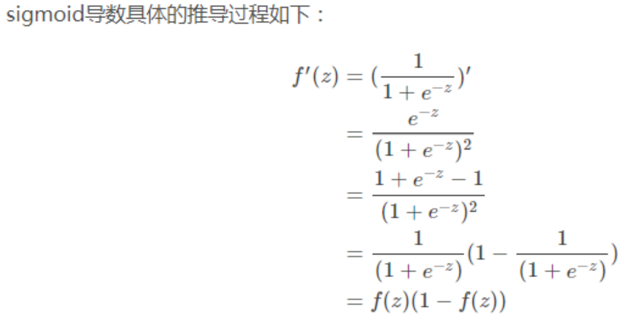
$$f(x)=\begin{cases}1, & x\geqslant 0;\\ 0, & x<0.\end{cases}$$



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

神经网络分类器-反向传播算法

➢均方差损失函数+Sigmoid 激活函数



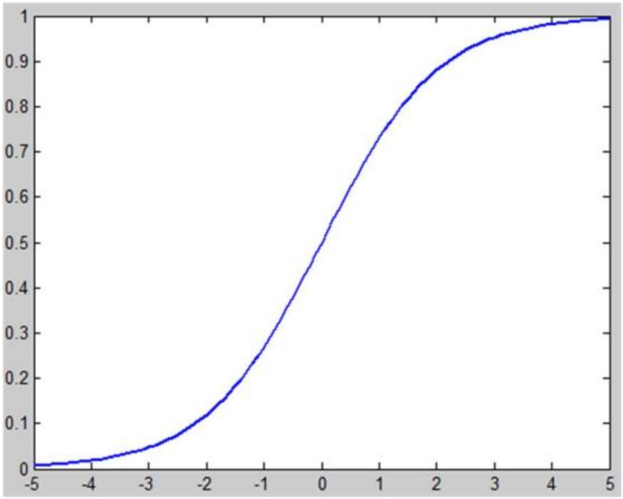
sigmoid导数具体的推导过程如下：

$$\begin{aligned} f'(z) &= (\frac{1}{1+e^{-z}})' \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\ &= \frac{1}{(1+e^{-z})}(1-\frac{1}{(1+e^{-z})}) \\ &= f(z)(1-f(z)) \end{aligned}$$

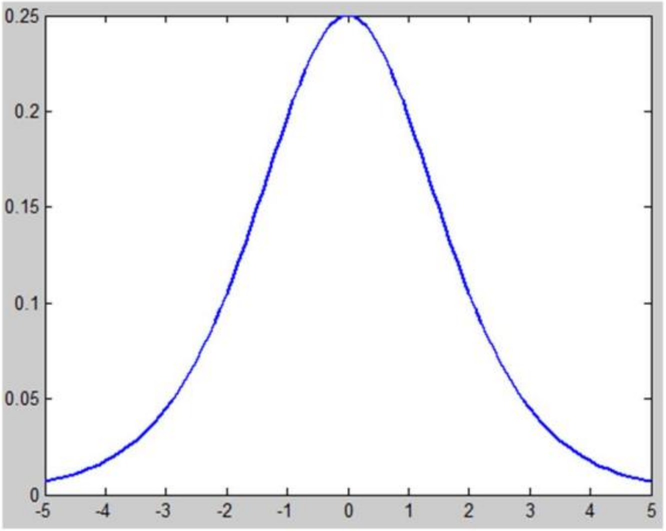
均方差损失函数：$J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y}) = \frac{1}{2}\left\|\mathbf{a}^L - \mathbf{y}\right\|_2^2$

$$\mathbf{a}^l = \sigma(\mathbf{z}^l) = \sigma(\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l) = \frac{1}{1+e^{-(\mathbf{W}^l\mathbf{a}^{l-1}+\mathbf{b}^l)}}$$

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



$$\sigma^{'}(z) = \sigma(z)(1-\sigma(z))$$



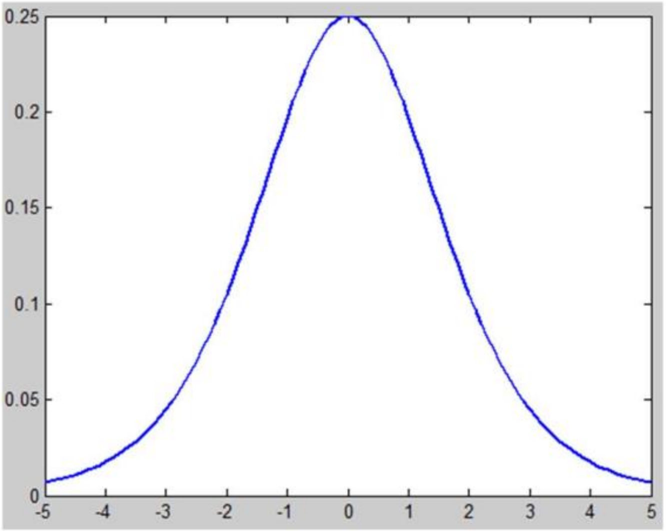
神经网络分类器-反向传播算法

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l (\mathbf{a}^{l-1})^T$$

$$\frac{\partial J(\mathbf{W},\mathbf{b},\mathbf{x},\mathbf{y})}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l$$

$$\boldsymbol{\delta}^l = (\mathbf{W}^{l+1})^T \boldsymbol{\delta}^{l+1} \odot \sigma^{'}(\mathbf{z}^l)$$

$$\sigma^{'}(z) = \sigma(z)(1-\sigma(z))$$

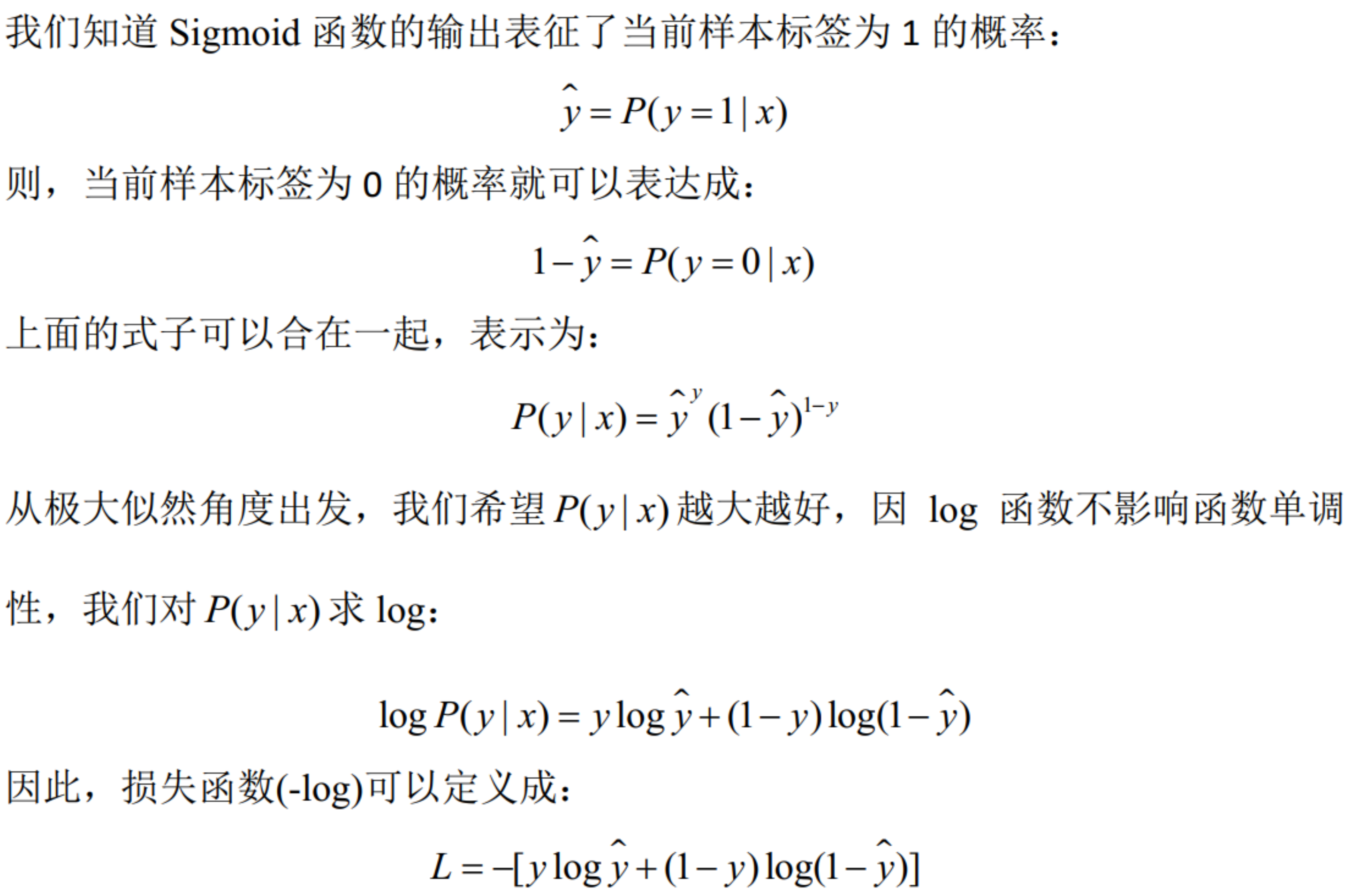


从上面的图像可以看出，$g^{'}(z)$在$z=0$时取得最大值 0.25，且是小于 1 的。

从 BP 算法中，我们知道，每一层向前递推时都要乘以$\sigma^{'}(z)$，得到梯度变化值。若采用 Sigmoid 函数作为激活函数，将导致权重更新较慢，网络收敛较慢。

神经网络分类器-反向传播算法

➢ 交叉熵损失函数+Sigmoid 激活函数



我们知道 Sigmoid 函数的输出表征了当前样本标签为 1 的概率：

$$\hat{y} = P(y = 1 | x)$$

则，当前样本标签为 0 的概率就可以表达成：

$$1 - \hat{y} = P(y = 0 | x)$$

上面的式子可以合在一起，表示为：

$$P(y | x) = \hat{y}^{y} (1 - \hat{y})^{1-y}$$

从极大似然角度出发，我们希望 $P(y | x)$ 越大越好，因 log 函数不影响函数单调性，我们对 $P(y | x)$ 求 log：

$$\log P(y | x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

因此，损失函数(-log)可以定义成：

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

神经网络分类器-反向传播算法

我们计算输出层δ^L：

$$\begin{aligned}\delta^L &= \frac{\partial J(W,b,a^L,y)}{\partial z^L}\\ &= -y\frac{1}{a^L}(a^L)(1-a^L)+(1-y)\frac{1}{1-a^L}(a^L)(1-a^L)\\ &= -y(1-a^L)+(1-y)(a^L)\\ &= a^L-y\end{aligned}$$

可见此时，δ^L表达式中已经没有了$\sigma'(z)$，因此避免了反向传播收敛速度慢的问题。

通常情况下，若使用了 Sigmoid 函数做激活函数，交叉熵损失函数肯定比均方差损失函数好用。

神经网络分类器-反向传播算法

多分类问题的交叉熵损失函数怎么定义呢？

Softmax 函数：

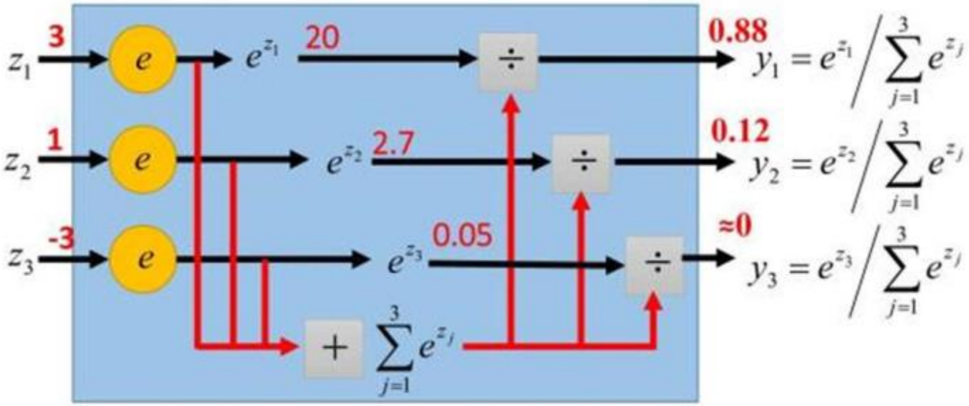
$$a_i^L = \frac{e^{z_i^L}}{\sum_{j=1}^{n_L} e^{z_j^L}}$$

- Softmax layer as the output layer

Probability:

- ■ $1 > y_i > 0$
- ■ $\sum_i y_i = 1$

Softmax Layer



z_1
3
e
e^{z_1}
20
÷
0.88
$y_1 = e^{z_1} \Big/ \sum_{j=1}^{3} e^{z_j}$
z_2
1
e
e^{z_2}
2.7
÷
0.12
$y_2 = e^{z_2} \Big/ \sum_{j=1}^{3} e^{z_j}$
z_3
-3
e
e^{z_3}
0.05
÷
≈0
$y_3 = e^{z_3} \Big/ \sum_{j=1}^{3} e^{z_j}$
+
$\sum_{j=1}^{3} e^{z_j}$

神经网络分类器-反向传播算法

对应的损失函数(对数似然函数)：

$$J(\mathbf{W},\mathbf{b},\mathbf{a}^L,\mathbf{y})=-\sum_k y_k \ln a_k^L$$

其中 y_k 的取值为 0 或者 1。如果某一类样本的输出为第 i 类，则 $y_i=1$，其余的 $j\neq i$ 都有 $y_j=0$。由于每个样本属于一个类别，所以这个似然函数可以简写为：

$$J(\mathbf{W},\mathbf{b},\mathbf{a}^L,\mathbf{y})=-\ln a_i^L$$

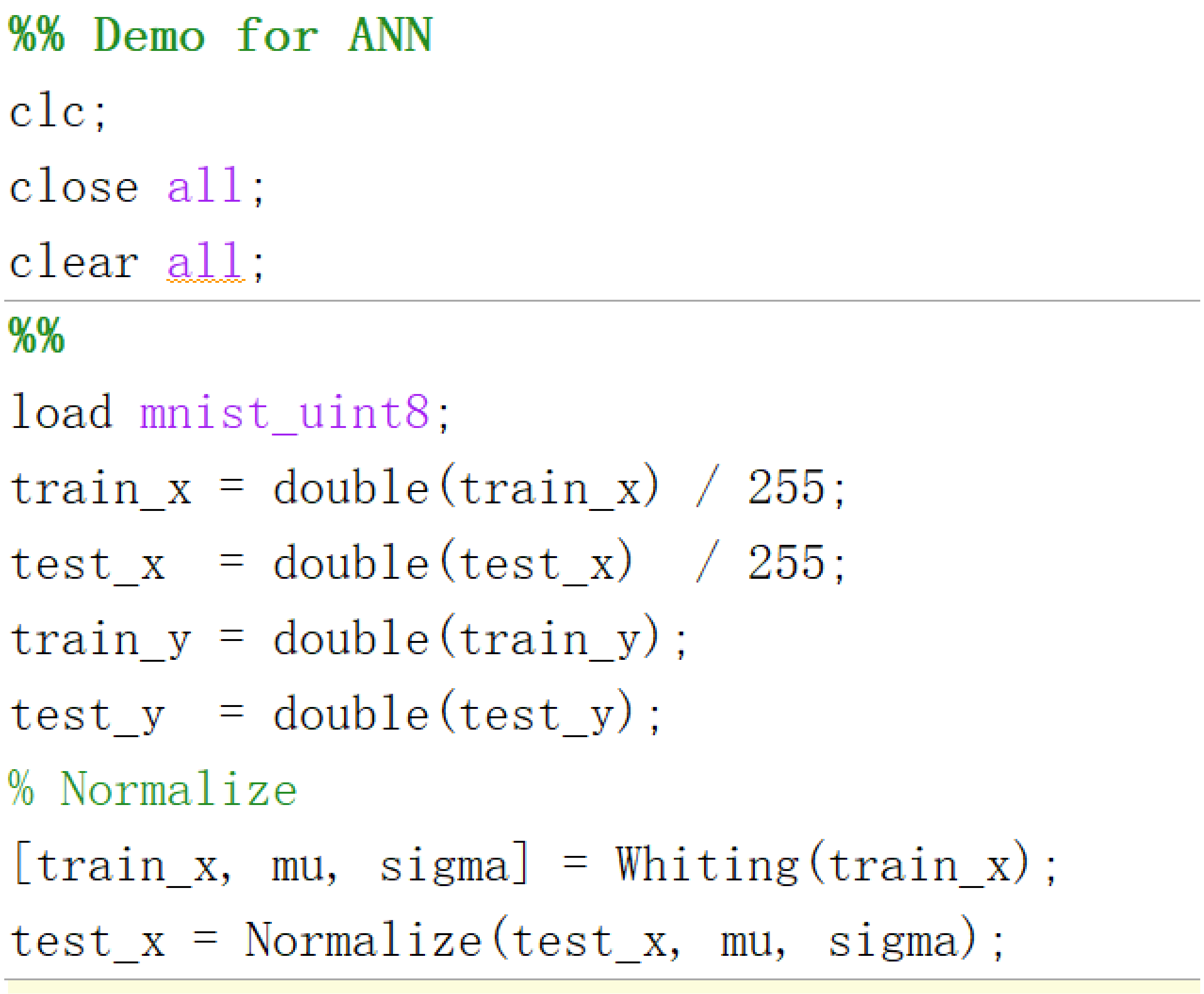
模式识别 -神经网络分类器

Matlab神经网络工具：

https://blog.csdn.net/weixin_42296976/article/details/81123843

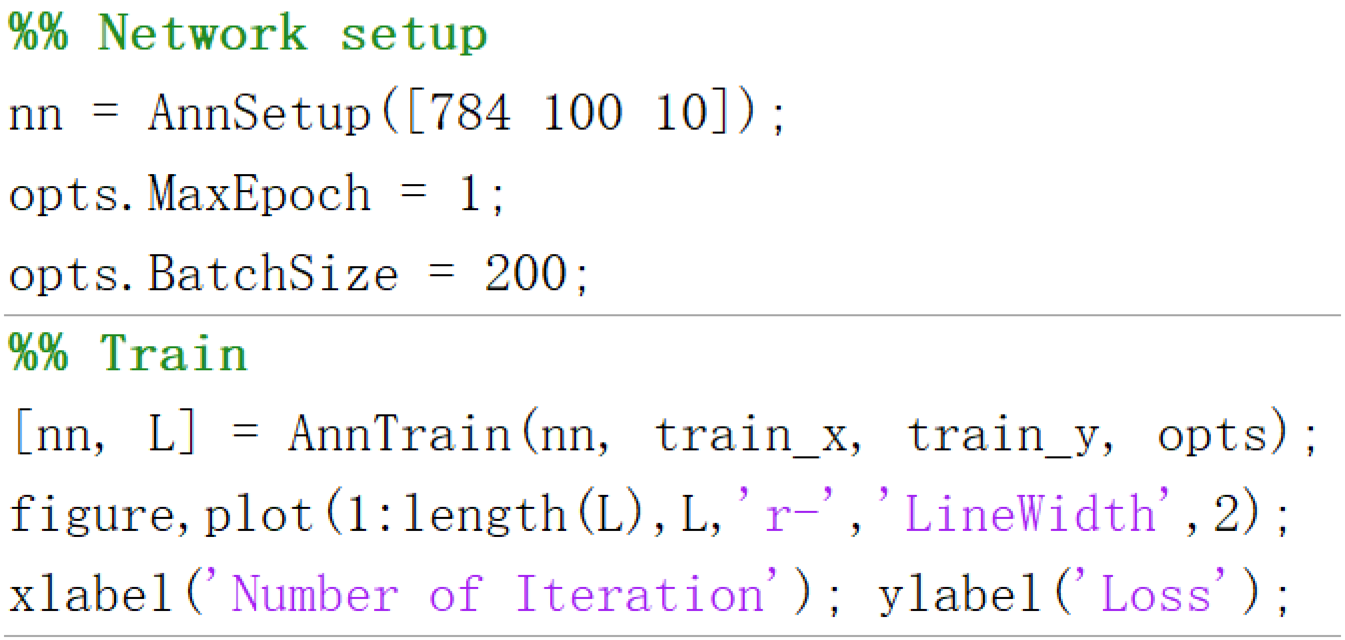
Talk is cheap, Show me the code.

模式识别-神经网络分类器



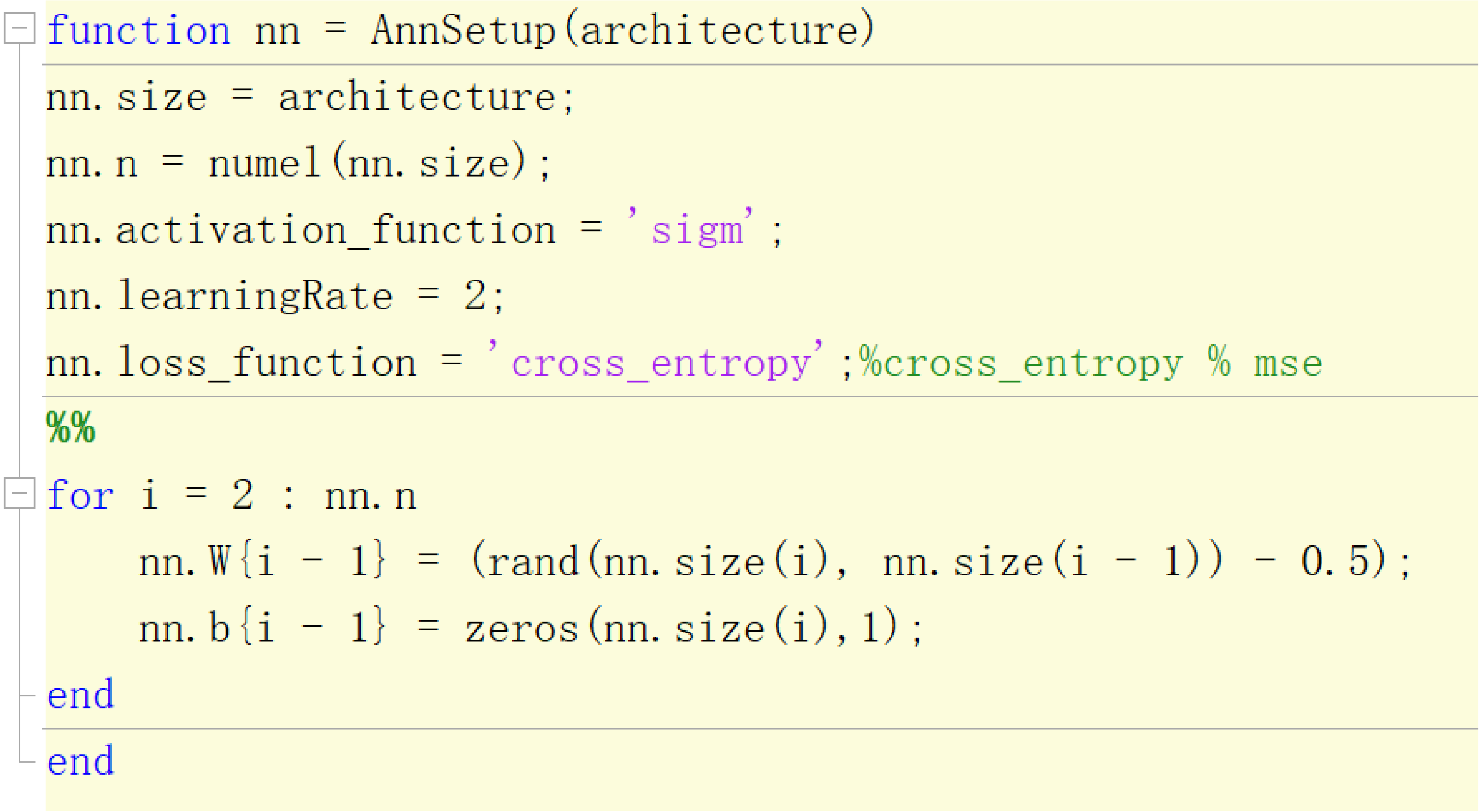
```
%% Demo for ANN
clc;
close all;
clear all;
%%
load mnist_uint8;
train_x = double(train_x) / 255;
test_x  = double(test_x)  / 255;
train_y = double(train_y);
test_y  = double(test_y);
% Normalize
[train_x, mu, sigma] = Whiting(train_x);
test_x = Normalize(test_x, mu, sigma);
```

模式识别-神经网络分类器



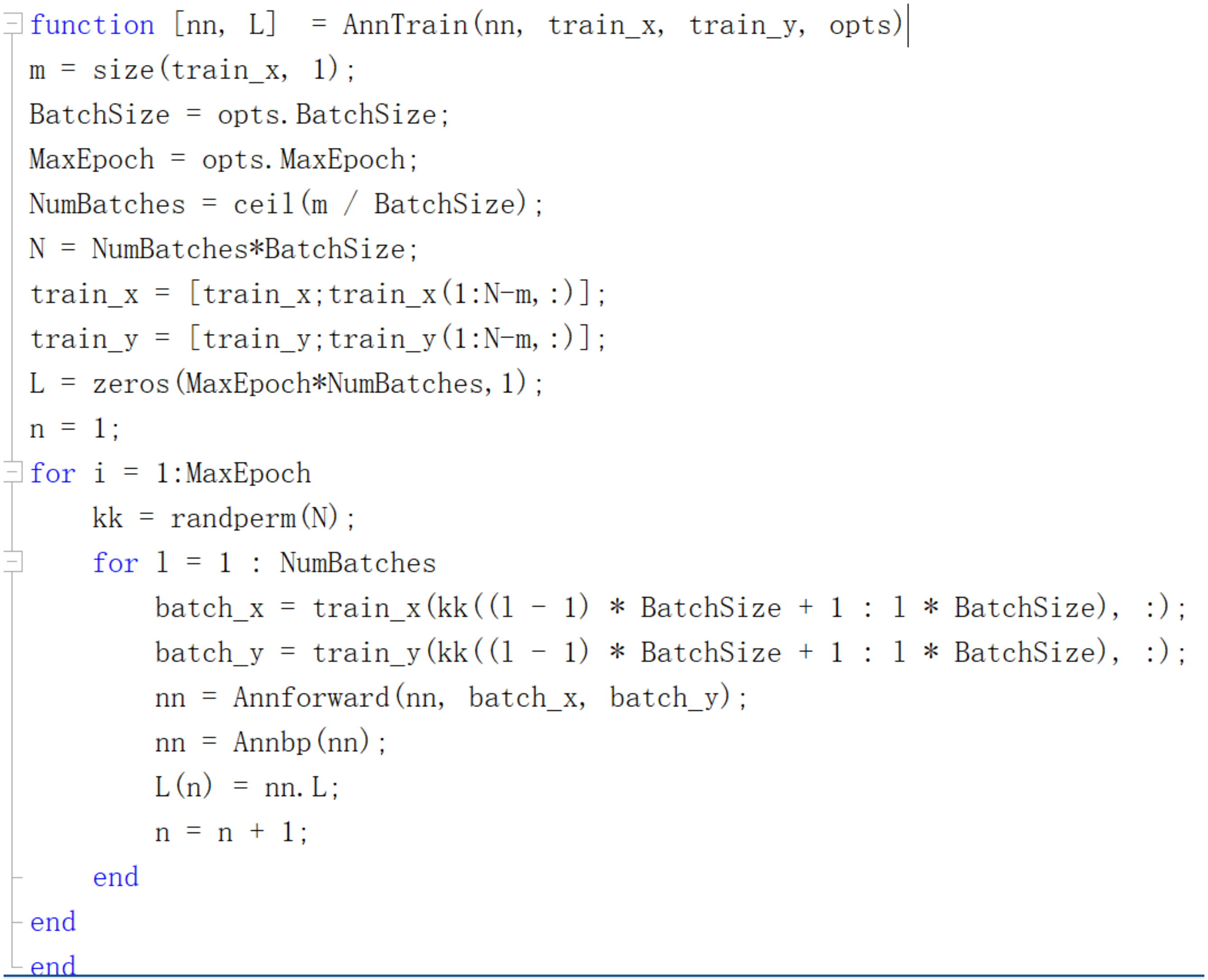
```
%% Network setup
nn = AnnSetup([784 100 10]);
opts.MaxEpoch = 1;
opts.BatchSize = 200;
%% Train
[nn, L] = AnnTrain(nn, train_x, train_y, opts);
figure,plot(1:length(L),L,'r-','LineWidth',2);
xlabel('Number of Iteration'); ylabel('Loss');
```

模式识别 -神经网络分类器



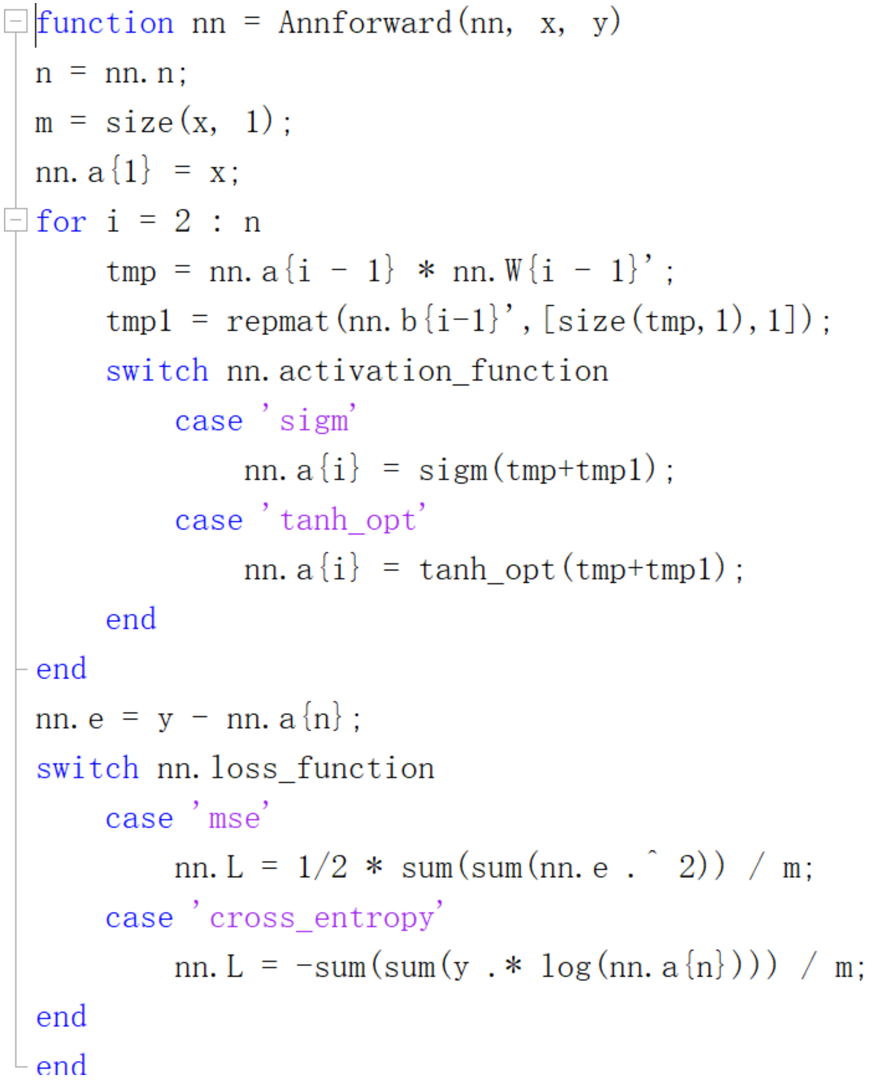
```
function nn = AnnSetup(architecture)
nn.size = architecture;
nn.n = numel(nn.size);
nn.activation_function = 'sigm';
nn.learningRate = 2;
nn.loss_function = 'cross_entropy';%cross_entropy % mse
%%
for i = 2 : nn.n
    nn.W{i - 1} = (rand(nn.size(i), nn.size(i - 1)) - 0.5);
    nn.b{i - 1} = zeros(nn.size(i),1);
end
end
```

模式识别-神经网络分类器



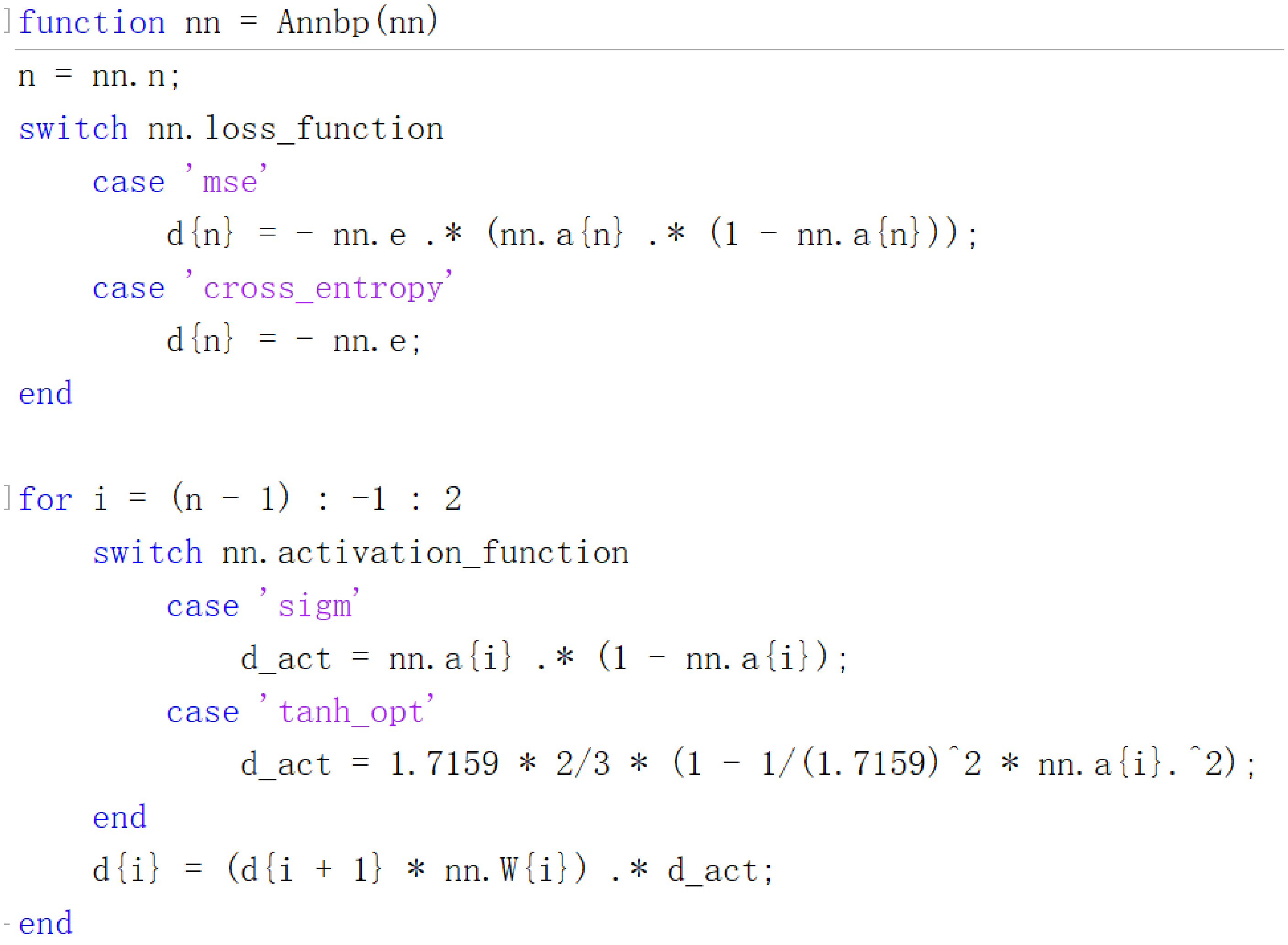
```
function [nn, L]  = AnnTrain(nn, train_x, train_y, opts)
m = size(train_x, 1);
BatchSize = opts.BatchSize;
MaxEpoch = opts.MaxEpoch;
NumBatches = ceil(m / BatchSize);
N = NumBatches*BatchSize;
train_x = [train_x;train_x(1:N-m,:)];
train_y = [train_y;train_y(1:N-m,:)];
L = zeros(MaxEpoch*NumBatches,1);
n = 1;
for i = 1:MaxEpoch
    kk = randperm(N);
    for l = 1 : NumBatches
        batch_x = train_x(kk((l - 1) * BatchSize + 1 : l * BatchSize), :);
        batch_y = train_y(kk((l - 1) * BatchSize + 1 : l * BatchSize), :);
        nn = Annforward(nn, batch_x, batch_y);
        nn = Annbp(nn);
        L(n) = nn.L;
        n = n + 1;
    end
end
end
```

模式识别-神经网络分类器



```
function nn = Annforward(nn, x, y)
n = nn.n;
m = size(x, 1);
nn.a{1} = x;
for i = 2 : n
    tmp = nn.a{i - 1} * nn.W{i - 1}';
    tmp1 = repmat(nn.b{i-1}',[size(tmp,1),1]);
    switch nn.activation_function
        case 'sigm'
            nn.a{i} = sigm(tmp+tmp1);
        case 'tanh_opt'
            nn.a{i} = tanh_opt(tmp+tmp1);
    end
end
nn.e = y - nn.a{n};
switch nn.loss_function
    case 'mse'
        nn.L = 1/2 * sum(sum(nn.e .^ 2)) / m;
    case 'cross_entropy'
        nn.L = -sum(sum(y .* log(nn.a{n}))) / m;
end
end
```

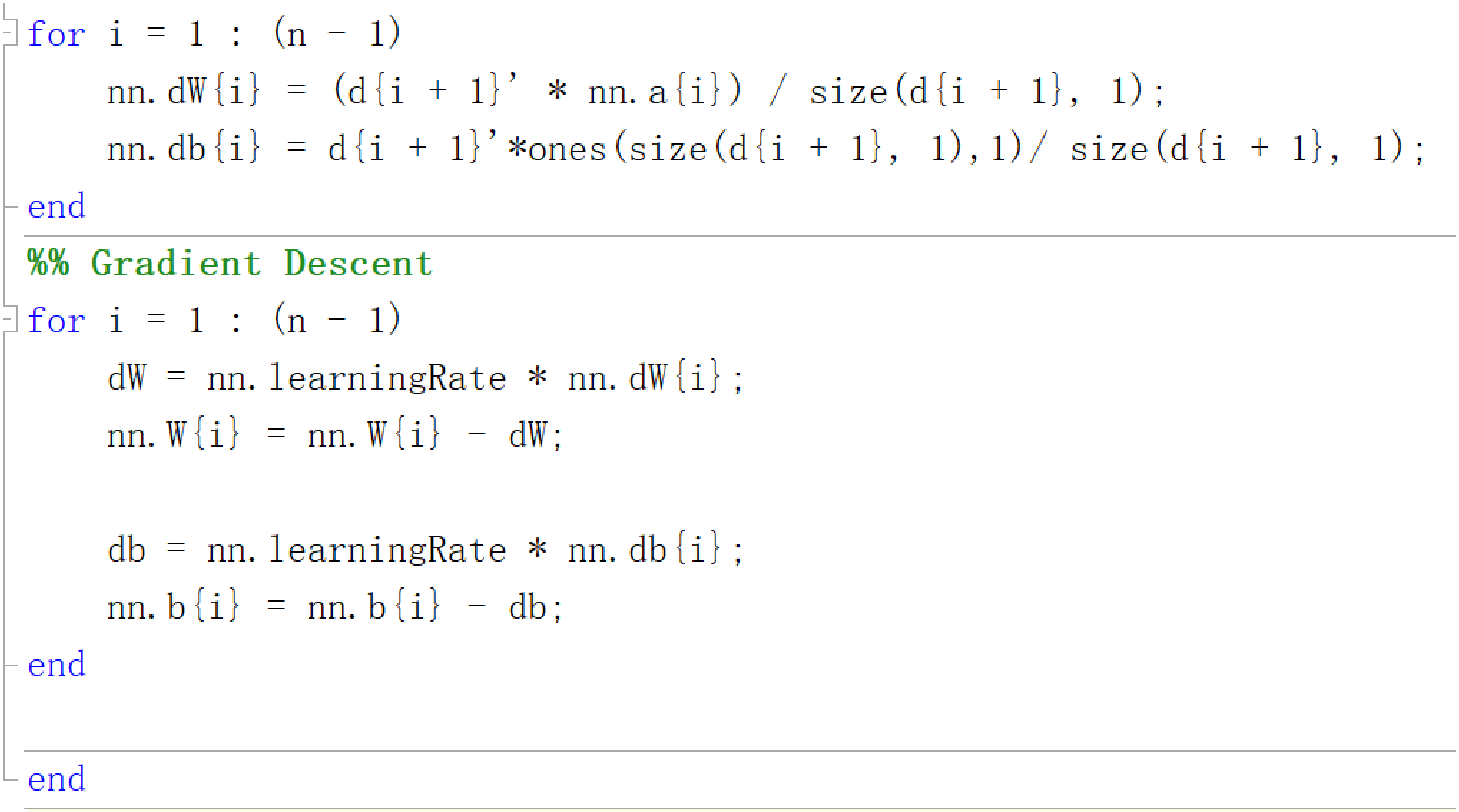
模式识别-神经网络分类器



```
function nn = Annbp(nn)
n = nn.n;
switch nn.loss_function
    case 'mse'
        d{n} = - nn.e .* (nn.a{n} .* (1 - nn.a{n}));
    case 'cross_entropy'
        d{n} = - nn.e;
end

for i = (n - 1) : -1 : 2
    switch nn.activation_function
        case 'sigm'
            d_act = nn.a{i} .* (1 - nn.a{i});
        case 'tanh_opt'
            d_act = 1.7159 * 2/3 * (1 - 1/(1.7159)^2 * nn.a{i}.^2);
    end
    d{i} = (d{i + 1} * nn.W{i}) .* d_act;
end
```

模式识别 -神经网络分类器



```
for i = 1 : (n - 1)
    nn.dW{i} = (d{i + 1}' * nn.a{i}) / size(d{i + 1}, 1);
    nn.db{i} = d{i + 1}'*ones(size(d{i + 1}, 1),1)/ size(d{i + 1}, 1);
end
%% Gradient Descent
for i = 1 : (n - 1)
    dW = nn.learningRate * nn.dW{i};
    nn.W{i} = nn.W{i} - dW;

    db = nn.learningRate * nn.db{i};
    nn.b{i} = nn.b{i} - db;
end

end
```

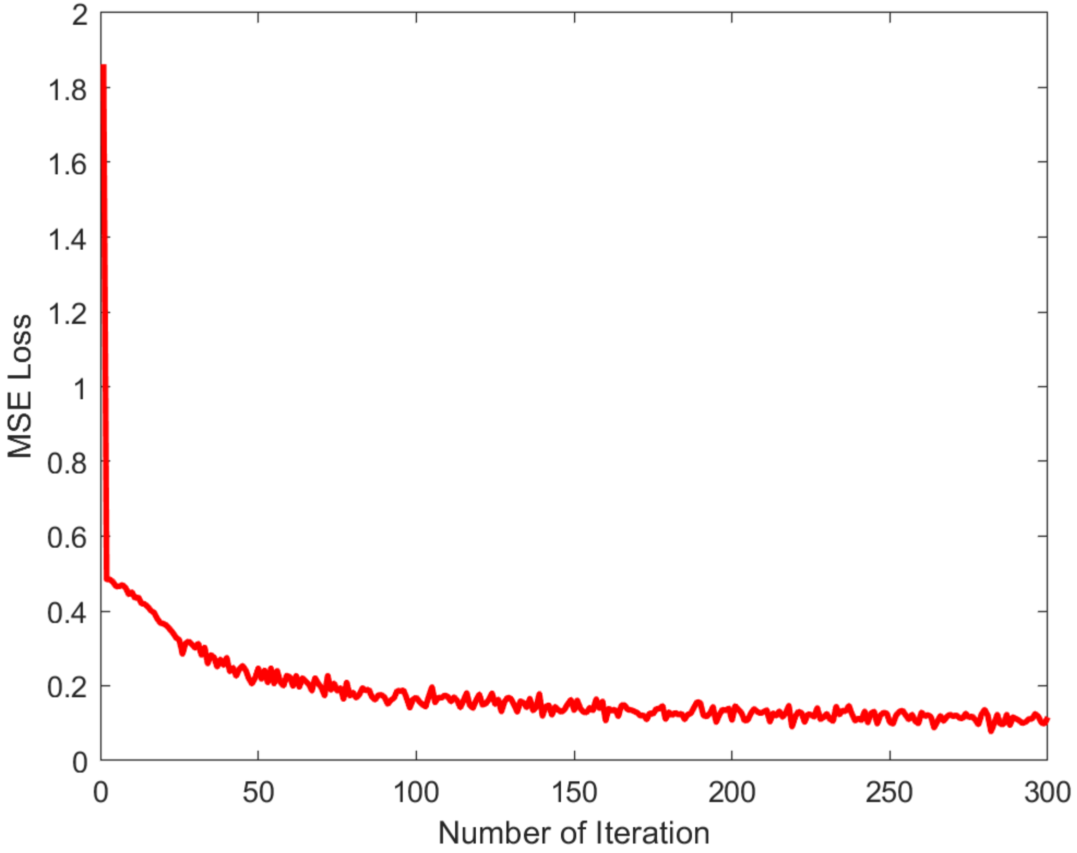
模式识别-神经网络分类器

```
%% Test
nn = Annforward(nn, test_x, test_y);
[~, labels] = max(nn.a{end},[],2);
[~, expected] = max(test_y,[],2);
bad = find(labels ~= expected);
er = numel(bad) / size(test_x, 1);
acr = 1-er;
fprintf('Accuracy ratio is %f.\n', acr);
```

模式识别-神经网络分类器

MSE loss

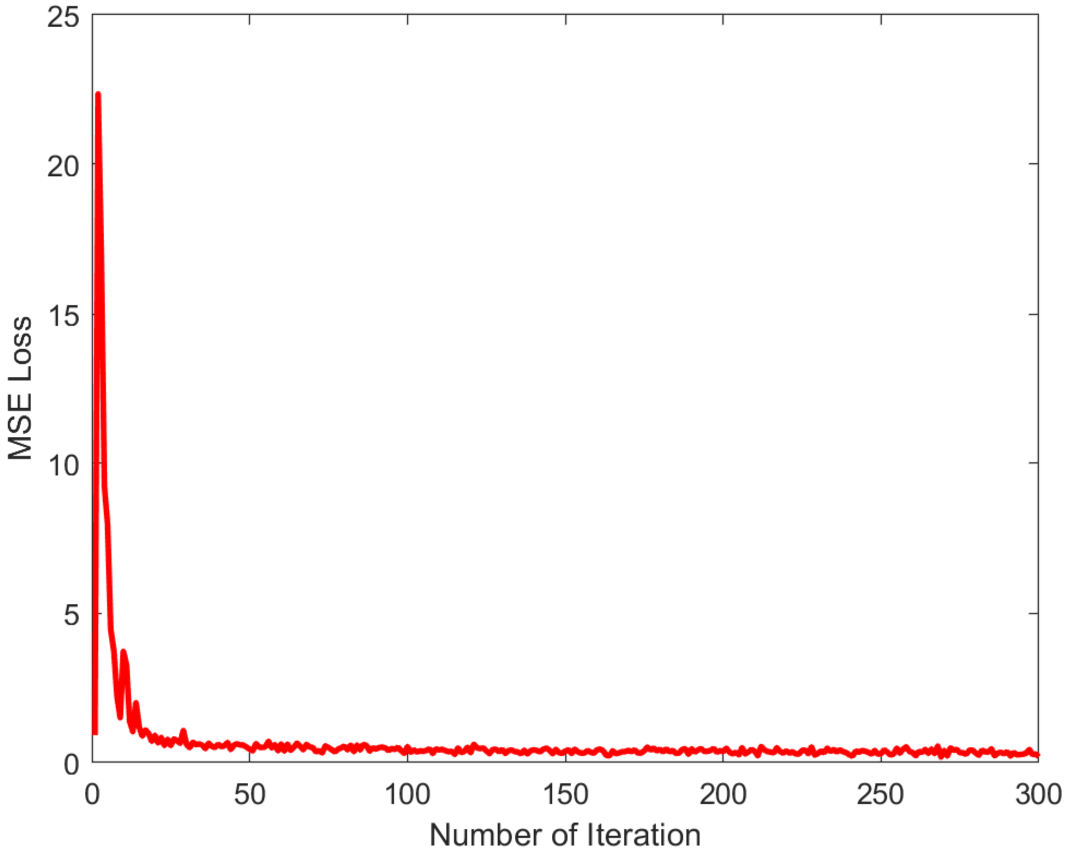
准确率：0.8798



模式识别-神经网络分类器

Cross_entropy loss

准确率：0.9277



模式识别-神经网络分类器

思考与讨论：

- 激活函数的作用是什么？没有激活函数可以吗？
- Sigmoid激活函数有什么缺点？
- 全连接网络有什么不足？

模式识别-神经网络分类器

若干素材取自网络，特此致谢！

谢谢聆听！