模式识别与机器学习

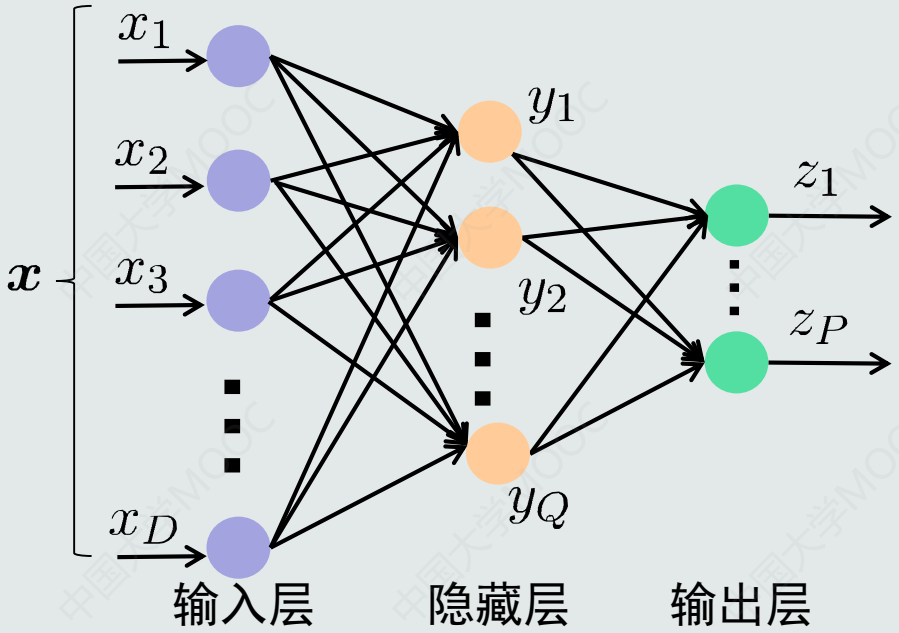
5.2、 BP学习算法

待学习的参数

学什么

- 给定训练样本，学习隐藏层和输出层每个神经元的权重：

$$\boldsymbol{\theta} = \{w_{dq}, w_{qp}\}$$



训练样本

训练样本

- 给定N个标定过的训练样本：

$$\mathcal{X}=\{(\boldsymbol{x}_1,\boldsymbol{t}_1),(\boldsymbol{x}_2,\boldsymbol{t}_2)\ldots,(\boldsymbol{x}_N,\boldsymbol{t}_N)\}$$

- 对于分类任务，输出真值$\boldsymbol{t}_n$采用one-hot方式。
  - ✓ 通常，K类分类任务对应K个输出神经元结点。
- 对于回归任务，输出真值是连续值。

目标函数

目标函数

- 由于神经网络早期较多用于回归任务，常使用针对训练样本的均值误差作目标函数：

$$\min J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{t}_n - \boldsymbol{z}_n\|_2^2$$

where $\boldsymbol{z}_n = [z_n^1, \ldots, z_n^p, \ldots, z_n^P]^T$

$$z_n^p = g(\boldsymbol{w}_p^T \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = [y_n^1, \ldots, y_n^Q]^T$$

$$y_n^q = g(\boldsymbol{w}_q^T \boldsymbol{x}_n)$$

链式结构

学习方式

学习方式

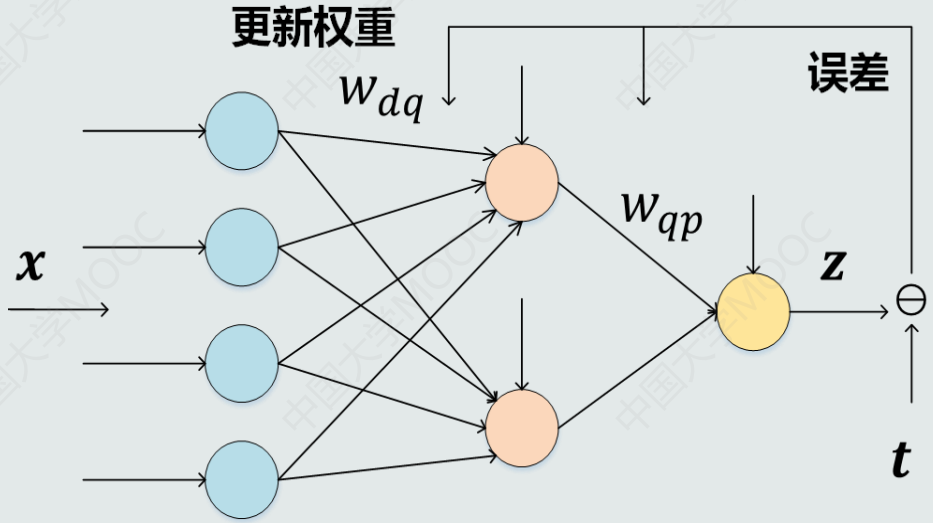
- 批量学习（batch mode）：
  - ✓ 把所有（或者部分）训练样本作为一个batch，计算输出误差，以此更新一次网络权重。
- 模式学习（pattern mode or sequential mode）：
  - ✓ 针对一个训练样本$(\boldsymbol{x}_n, \boldsymbol{t}_n)$，计算误差，更新一次网络权重。

$$J_n = \|\boldsymbol{t}_n - \boldsymbol{z}_n\|_2^2$$

目标优化：BP算法

基本思想

- 误差J只跟输出层权重$\{\boldsymbol{w}_p\}_{p=1,\ldots,P}$有直接的连接关系，与隐藏层权重$\{\boldsymbol{w}_q\}_{q=1,\ldots,Q}$只是间接连接。因此，不能直接求取误差$J$关于隐藏层权重的梯度。
- 先求取误差ε对输出权重的梯度。
- 再根据链式法则，将梯度从输出层反向传播、逐步求取关于隐藏层权重的梯度。由此设计反向传播算法，即BP（back-propagation）算法。



BP算法

多个样本误差梯度 vs 单个样本误差梯度

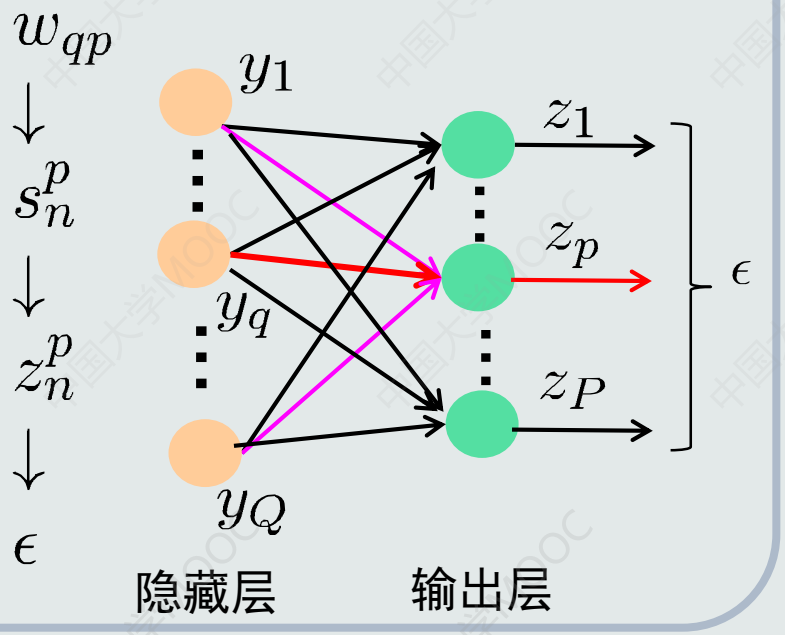
- 以对输出层权重w_{qp}的梯度（偏导）为例：

$$J = \frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{t}_n - \boldsymbol{z}_n\|_2^2 = \frac{1}{N}\sum_{n=1}^{N}\sum_{p=1}^{P}\|t_n^p - z_n^p\|_2^2$$

$$z_n^p = g(s_n^p) = \text{sigmoid}(s_n^p),\ \ s_n^p = \boldsymbol{w}_p^T\boldsymbol{y}_n = \sum_{q=1}^{Q} w_{qp}y_n^q$$

$$\frac{\partial J}{\partial w_{qp}} = \sum_{n=1}^{N}\frac{\partial J}{\partial z_n^p}\frac{\partial z_n^p}{\partial s_n^p}\frac{\partial s_n^p}{\partial w_{qp}}$$

- 可见，N个训练样本的误差对参数梯度等于每个训练样本的误差对参数梯度求和。
- 为了表述方便，后续推导只考虑单个训练样本的误差（简记做J）对参数的梯度。



BP算法

对输出层权重w_{qp}的梯度

$$J = \frac{1}{2}\|\boldsymbol{t}-\boldsymbol{z}\|_2^2 = \frac{1}{2}\sum_{p=1}^{P}\|t_p - z_p\|_2^2$$

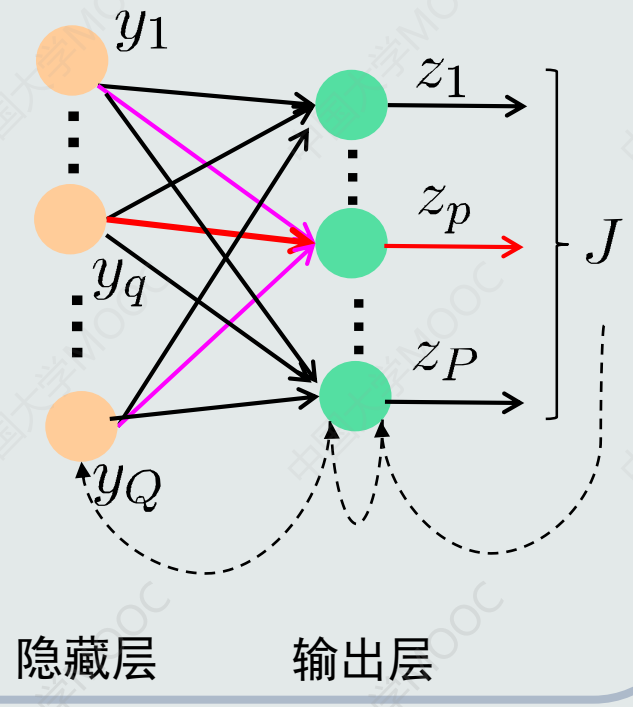
$$z_p = g(s_p) = \text{sigmoid}(s_p), \quad s_p = \boldsymbol{w}_p^T\boldsymbol{y} = \sum_{q=1}^{Q} w_{qp}y_q$$

$$\frac{\partial J}{\partial w_{qp}} = \frac{\partial J}{\partial z_p}\frac{\partial z_p}{\partial s_p}\frac{\partial s_p}{\partial w_{qp}}$$

$$\frac{\partial J}{\partial z_p} = -(t_p - z_p) = -e_p \qquad \frac{\partial s_p}{\partial w_{qp}} = y_q$$

$$\frac{\partial z_p}{\partial s_p} = g_p' = z_p(1-z_p)$$

$$\Rightarrow \frac{\partial J}{\partial w_{qp}} = -e_p g_p' y_q$$



BP算法

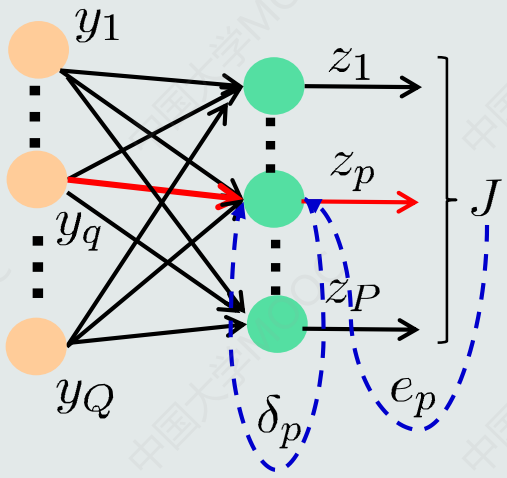
对输出层权重w_{qp}的梯度

$$\frac{\partial J}{\partial w_{qp}} = -e_p g_p' y_q$$

- 梯度反向传播：

$$\begin{cases} e_p = t_p - z_p \\ \delta_p = e_p g_p' \end{cases} \Rightarrow \frac{\partial J}{\partial w_{qp}} = -\delta_p y_q$$

- ✓ e_p：目标函数针对单个输出结点p的负梯度，也等于在单个输出结点p上的输出值与真值的误差。
- ✓ δ_p：梯度e_p通过激活函数通道反向传播到结点p内部活性处的偏导值，即乘以g_p的梯度。
- ✓ 乘以活性对权重w_{qp}的梯度，得到目标函数对权重w_{qp}的梯度。



BP算法

对隐藏层权重w_{dq}的梯度

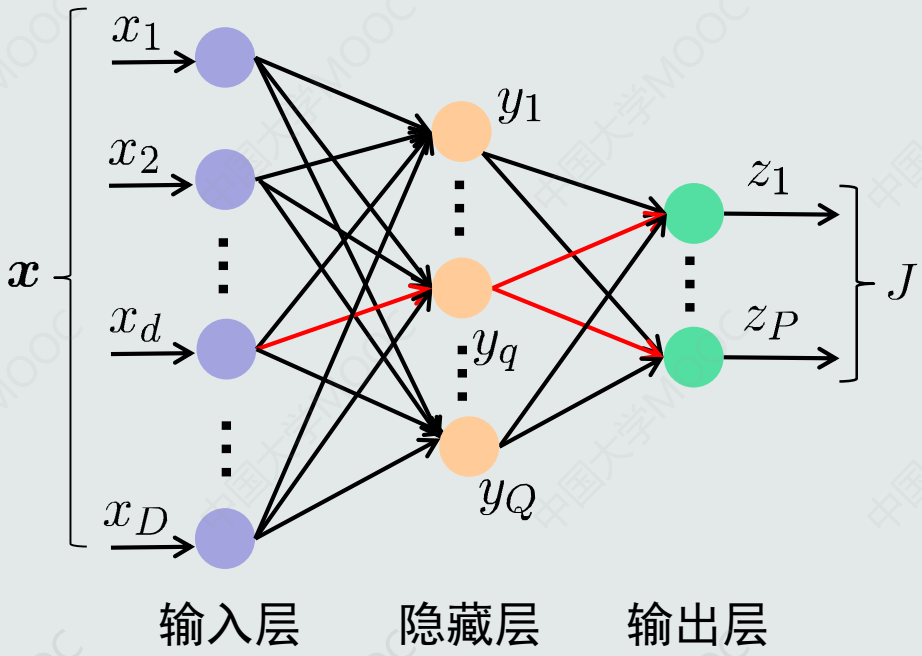
$$\frac{\partial J}{\partial w_{dq}} = \frac{\partial J}{\partial y_q}\frac{\partial y_q}{\partial s_q}\frac{\partial s_q}{\partial w_{dq}}$$

$$w_{dq} \to s_q \to y_q \to \{s_p, z_p\} \to J$$

- 由于隐藏层结点q与输出层所有结点都连接，所以J对输出信号y_q的梯度，由所有输出层结点通过对应权重$\{w_{qp}\}_{p=1,\ldots,P}$反向传播得到。

$$\frac{\partial J}{\partial y_q} = \sum_{p=1}^{P}\frac{\partial J}{\partial z_p}\frac{\partial z_p}{\partial s_p}\frac{\partial s_p}{\partial y_q}$$

$$= \sum_{p=1}^{P} -e_p g'_p w_{qp}$$



BP算法

对隐藏层权重w_{dq}的梯度

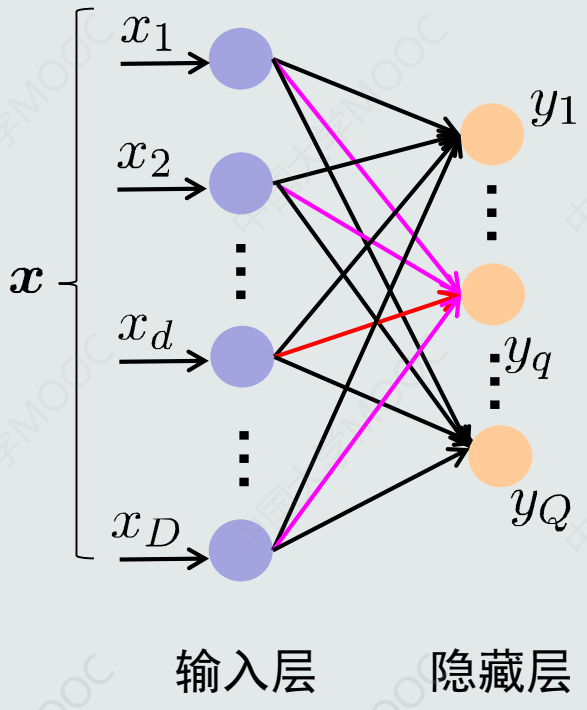
$$y_q = g(s_q) = \text{sigmoid}(s_q), \quad s_q = \boldsymbol{w}_q^T \boldsymbol{x} = \sum_{d=1}^{D} w_{dq} x_d$$

$$\frac{\partial J}{\partial w_{dq}} = \frac{\partial J}{\partial y_q} \frac{\partial y_q}{\partial s_q} \frac{\partial s_q}{\partial w_{dq}}$$

$$\frac{\partial J}{\partial y_q} = \sum_{p=1}^{P} -e_p g'_p w_{qp}$$

$$\frac{\partial y_q}{\partial s_q} = g'_q = y_q(1 - y_q)$$

$$\frac{\partial s_q}{\partial w_{dq}} = x_d$$



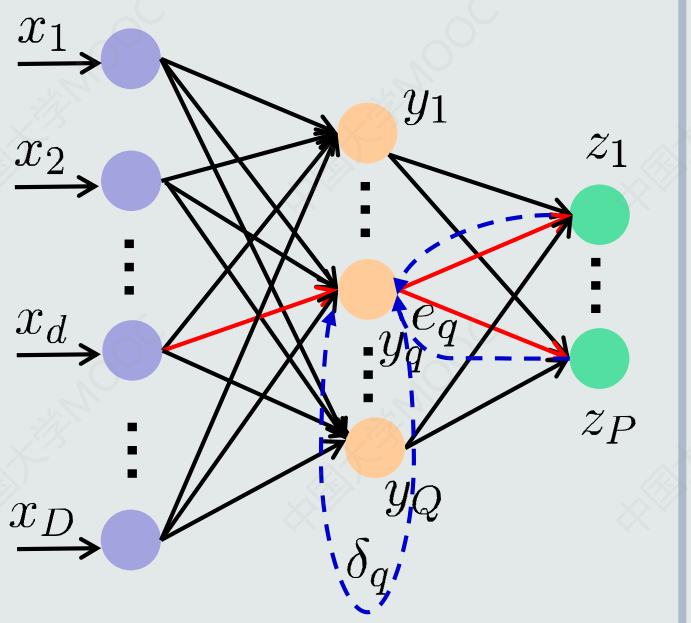
BP算法

对隐藏层权重w_{dq}的梯度

$$\frac{\partial J}{\partial w_{dq}} = \frac{\partial J}{\partial y_q}\frac{\partial y_q}{\partial s_q}\frac{\partial s_q}{\partial w_{dq}} = \sum_{p=1}^{P}\left\{-e_p g'_p w_{qp}\right\} g'_q x_d = -\sum_{p=1}^{P}\left\{\delta_p w_{qp}\right\} g'_q x_d$$

- 梯度反向传播：

$$\begin{cases} e_q = \sum_{p=1}^{P}\{\delta_p w_{qp}\} \\ \delta_q = e_q g'_q \end{cases} \Rightarrow \frac{\partial J}{\partial w_{dq}} = -\delta_q x_d$$



- ✓ e_q：所有输出结点收到的反向传播来的梯度$\{\delta_p\}$，通过各自对应的权重连接通道$\{w_{qp}\}$反向传播到q结点。
- ✓ δ_p：梯度e_{qp}通过激活函数通道（g_q的梯度）回传的值。
- ✓ 再乘以权重w_{dq}的梯度，得到目标函数对权重w_{dp}的梯度。

参数迭代更新

梯度下降法

- 采用梯度下降法逐层迭代更新参数：每次更新时刻构成一个epoch.
  - ✓ 设当前epoch索引为k，下一个epoch索引为$k+1$
  - ✓ η为更新步长。

$$w_{qp}^{k+1} = w_{qp}^{k} + \Delta w_{qp}^{k} \qquad w_{dq}^{k+1} = w_{dq}^{k} + \Delta w_{dq}^{k}$$

$$= w_{qp}^{k} + \eta\left(-\frac{\partial J^{k}}{\partial w_{qp}^{k}}\right) \qquad = w_{dq}^{k} + \eta\left(-\frac{\partial J^{k}}{\partial w_{dq}^{k}}\right)$$

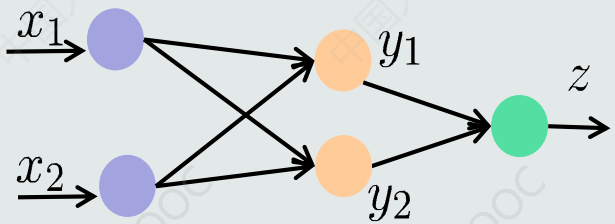
$$= w_{qp}^{k} + \eta\delta_{p}^{k} y_{q}^{k} \qquad = w_{dq}^{k} + \eta\delta_{q}^{k} x_{d}^{k}$$

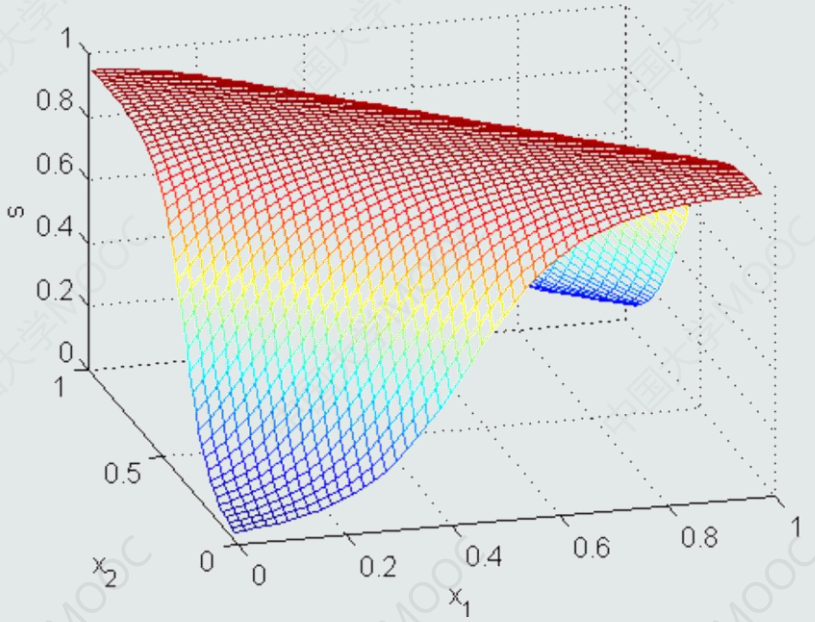
例子1：分类

XOR分类问题

- 二维输入空间的XOR问题，如何用神经网络实现异或分类？
  - ✓ 设计一个2-2-1的单隐层神经网络。
  - ✓ 由于使用了sigmoid函数，逻辑0和1用实数值0.05和0.95代替。

x_1	x_2	$f_{\oplus}$
0.05	0.05	0.05
0.05	0.95	0.95
0.95	0.05	0.95
0.95	0.95	0.05

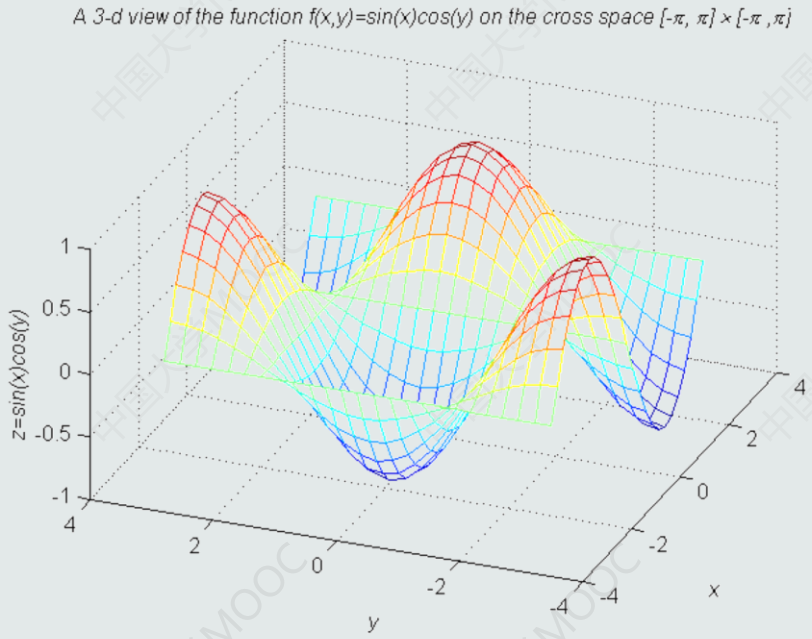




例子2：回归

函数拟合

- 拟合如下二维函数：$f(x_1, x_2) = \sin(x_1)\cos(x_2)$
  - ✓ 每个输入维度的取值空间 $[-\pi, \pi]$，在空间均匀采样625个样本。
  - ✓ 设计一个2-10-10-1的双隐层神经网络。

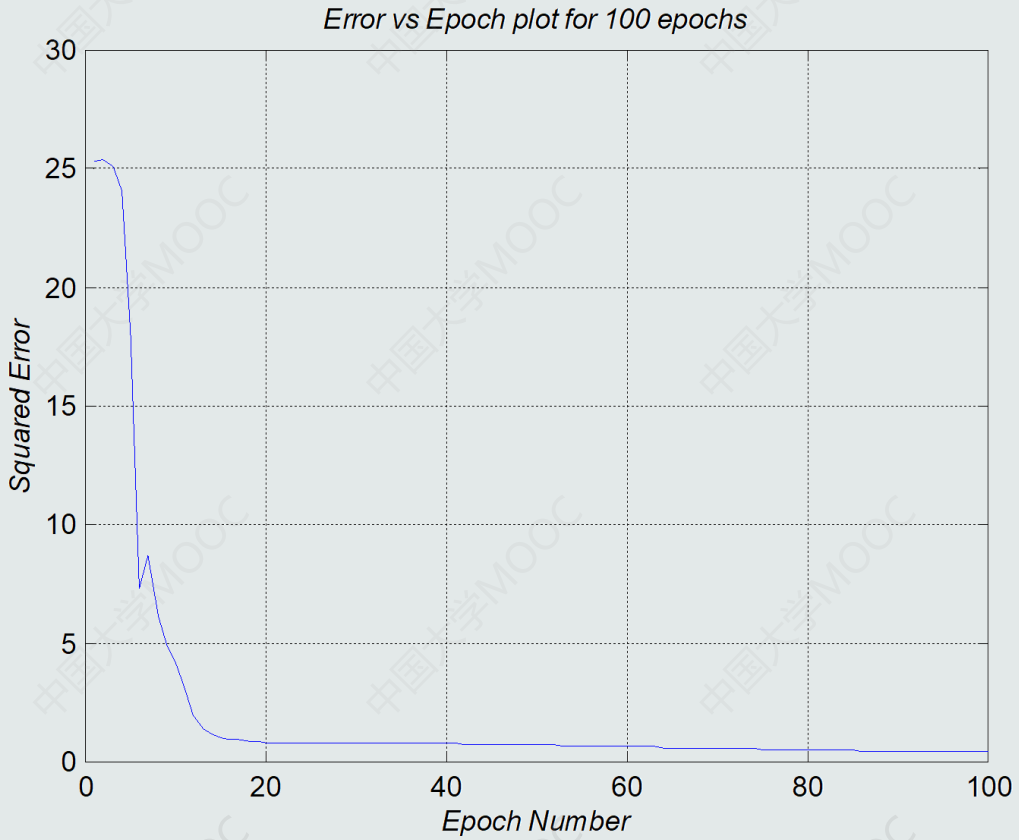


超参数	取值
步长η	0.9
动量系数α	0.6
误差阈值τ	0.05

例子2：回归

收敛过程：训练误差

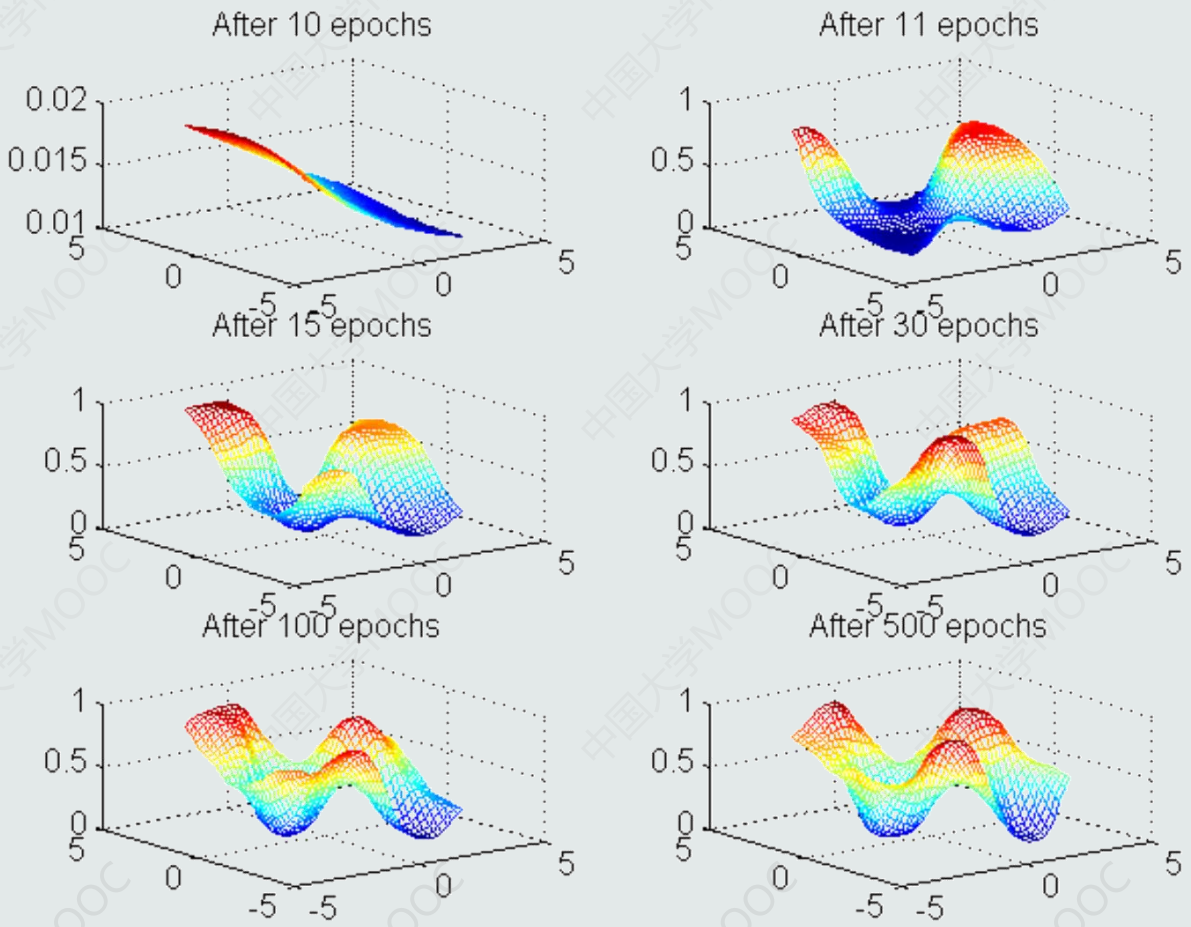
- 随着迭代次数的增加，训练误差（均方差）快速降低，随后逐渐收敛到一个较小的范围。
- 因此，可以使用一个误差阈值来作为迭代过程的停止条件。



例子2：回归

收敛过程：输出信号

- 随着迭代次数的增加，输出信号逐渐逼近于真实信号。



总结&讨论

神经网络：宽度vs深度

- 数学理论已经证明：只要隐层结点数目足够，单隐层神经网络就可以无限逼近任意非线性函数。
- 但由于训练算法的局限，针对高维输入数据和高度非线性数据，即使隐层结点个数足够多（即网络宽度足够宽），单隐层或浅层神经网络很难实现很好的分类和拟合。
- 如果增加网络层数（即网络深度），效果会不会更好呢？如果网络层数过深，势必导致参数数目增加，如何有效训练呢？