深度学习引论

章 毅，张 蕾，郭 泉

四川大学·计算机学院·人工智能系

机器智能实验室

http://www.machineilab.org/

回顾作业

- 提供MATLAB模版
- 可以使用MATLAB或Python

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initialize each w_{ij}^l, and choose a learning rate α

Step 3. for each mini-batch sample $D_m \subseteq D$

$\nabla J_{ji} = 0$;

for each $(x, y^L) \in D_m$

$a^1 \leftarrow x \in D_m$;

for $l = 2{:}L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$J(x, y^L)$;

$\delta^L = \frac{\partial J(x, y^L)}{\partial z^L}$;

for $l = L - 1{:}2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

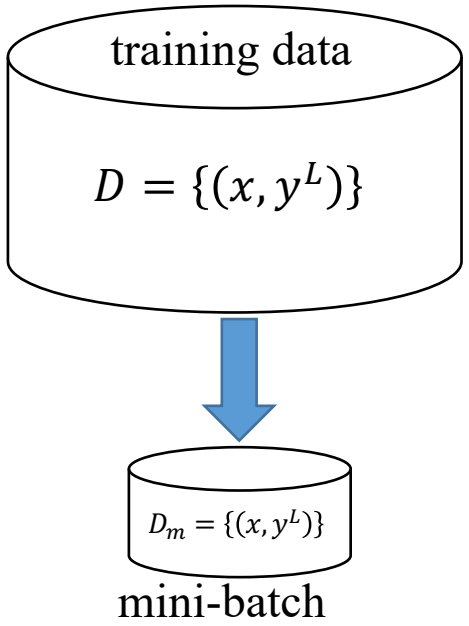
$\nabla J_{ji} \leftarrow \nabla J_{ji} + \delta_j^{l+1} \cdot a_i^l$;

end

$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \nabla J_{ji}$;

end

Step 4. Return to Step 3 until each w^l converge



function $fc(w^l, a^l)$

$for\ i = 1{:}n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

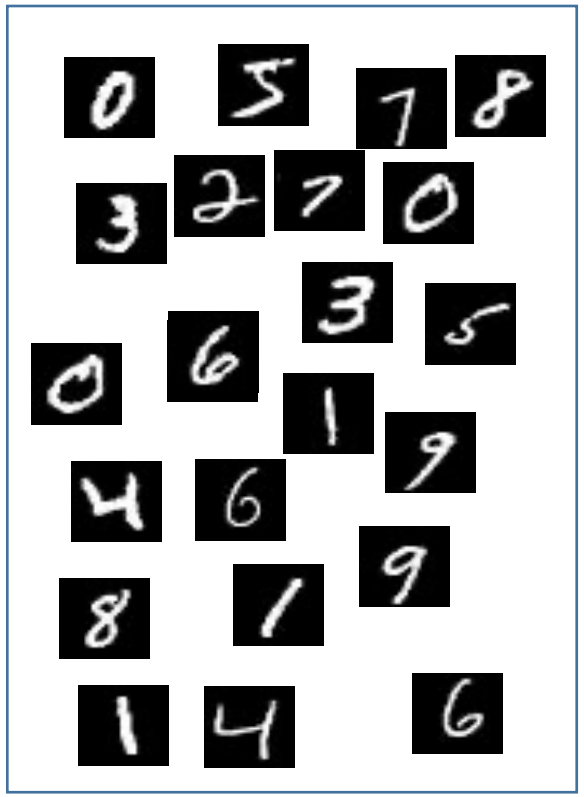
function $bc(w^l, \delta^{l+1})$

$for\ i = 1{:}n_l$

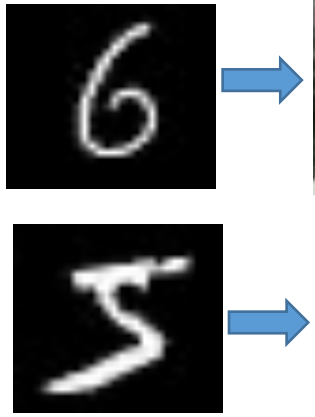
$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

回顾



训练集



训练

父亲知道正确答案

有监督学习

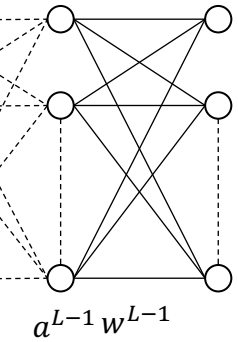
9

5

两个重要因素：

1. 需要有一个方法来度量正确答案和小孩的答案之间的误差——性能函数
2. 需要有一种机制来改变小孩的知识系统——学习算法

回顾



网络预测

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

目标

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

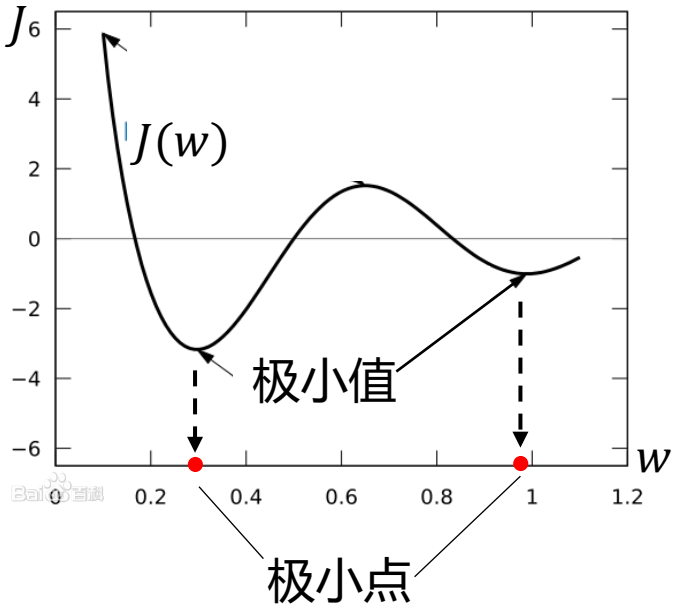
$J(a^L, y^L)$

性能函数或代价函数 $J(a^L, y^L)$ 用来刻画 a^L 和 y^L之间的距离，$J(a^L, y^L)$ 本质上是$(w^1, \cdots, w^L)$的函数，即

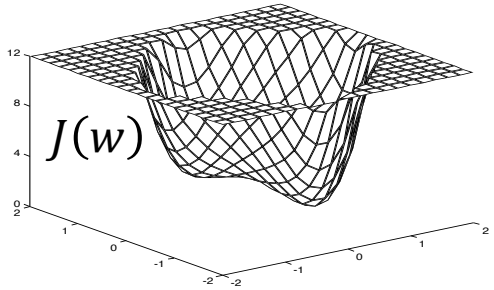
$$J = J(w^1, \cdots, w^L).$$

学习是使得网络输出 a^L 逐渐向 y^L 靠近的过程，也就是要使得性能函数 J 取极小值。性能函数 $J = J(w^1, \cdots, w^{L-1})$ 是变量 $w^l (l = 1, \cdots, L)$ 的函数，因此网络的学习就是寻找性能函数J的极小点$w^l (l = 1, \cdots, L)$的过程 。

问题: 如何寻找 J 的极小点?

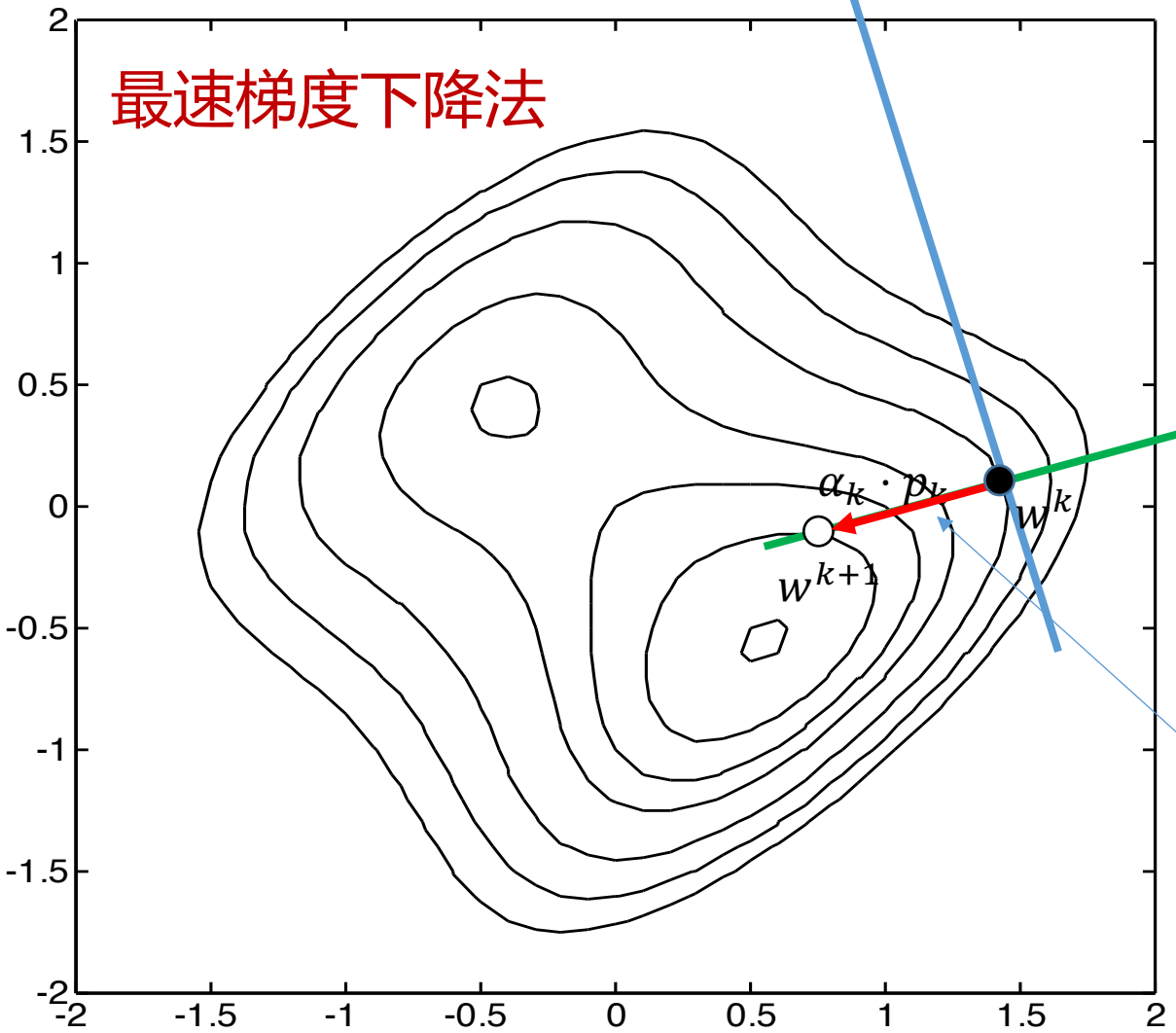


回顾



函数值变化最慢的方向

函数值上升最快的方向



梯度:

$$g_k = \nabla J(w)\Big|_{w(k)} = \frac{\partial J}{\partial w}\Big|_{w(k)} = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{pmatrix}\Big|_{w(k)}$$

最速下降算法:

$$p_k = -g_k$$

$$w(k+1) = w(k) - \alpha_k \cdot g_k$$

即

$$w(k+1) = w(k) - \alpha_k \cdot \frac{\partial J}{\partial w}\Big|_{w(k)}$$

函数值下降最快的方向

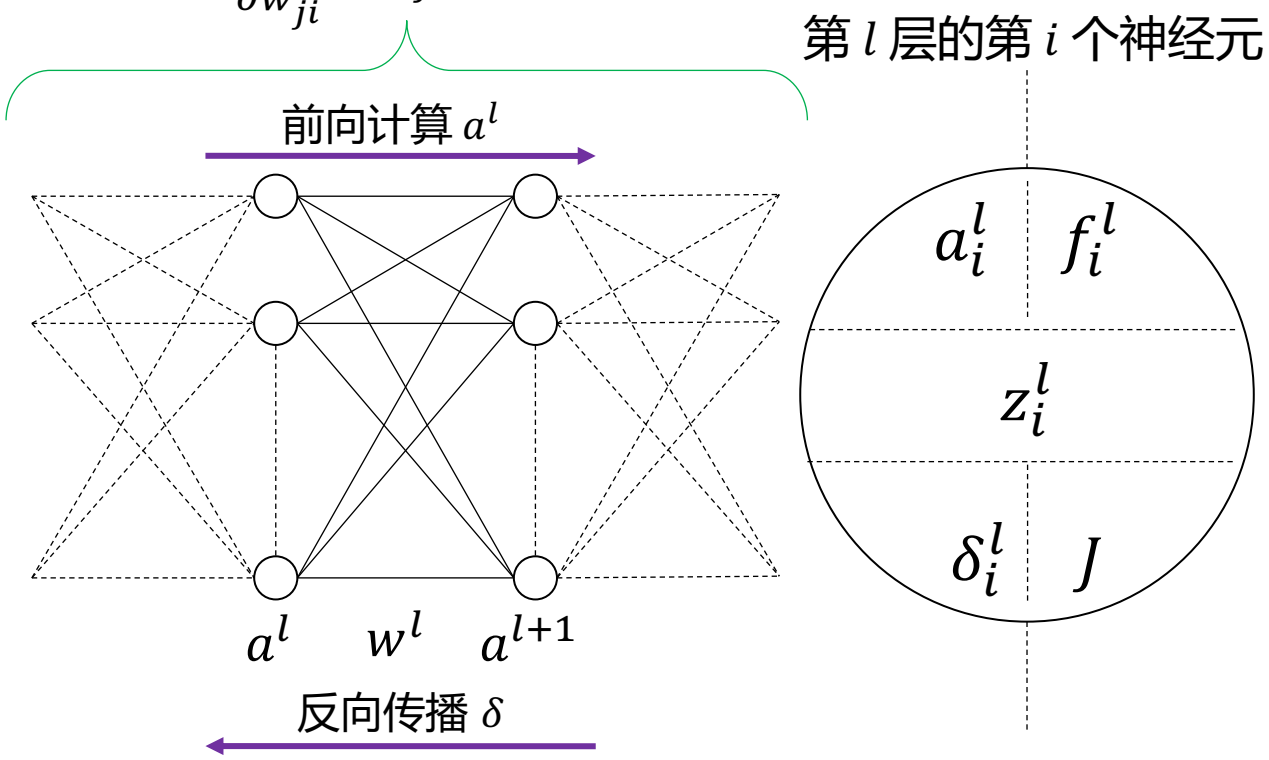
回顾

反向传播算法

性能函数：$J(w^1,\cdots,w^L)$

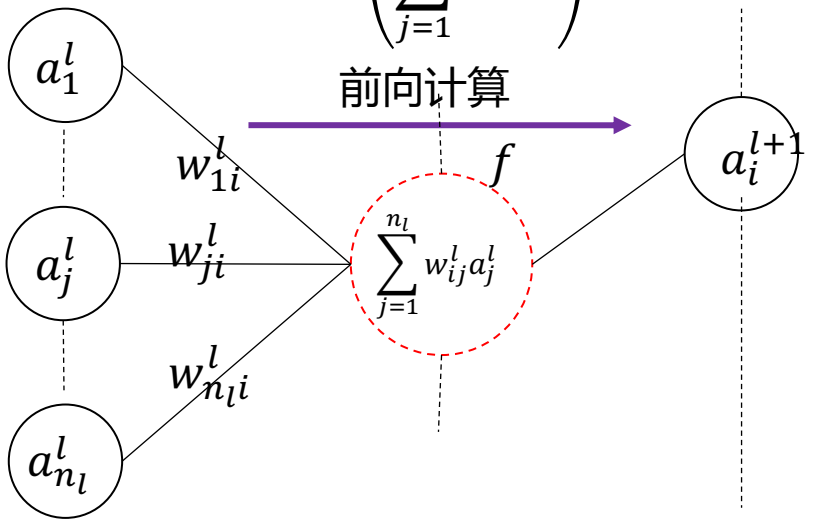
权值更新规则：$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

关系：$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

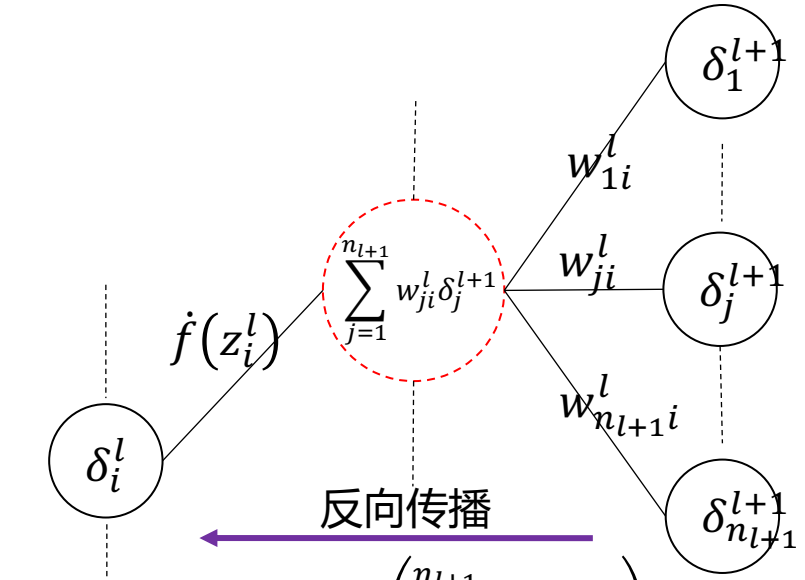


第 l 层的第 i 个神经元

$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$



第 l 层　　第$l+1$层



第 l 层　$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1}\right)$　第$l+1$层

反向传播算法

在线BP算法

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initialize each w_{ij}^l, and choose a learning rate α

Step 3. for each $(x, y^L) \in D$

$a^1 \leftarrow x \in D_m$;

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$J(x, y^L)$;

$\delta^L = \dfrac{\partial J(x, y^L)}{\partial z^L}$;

for $l = L - 1: 2$

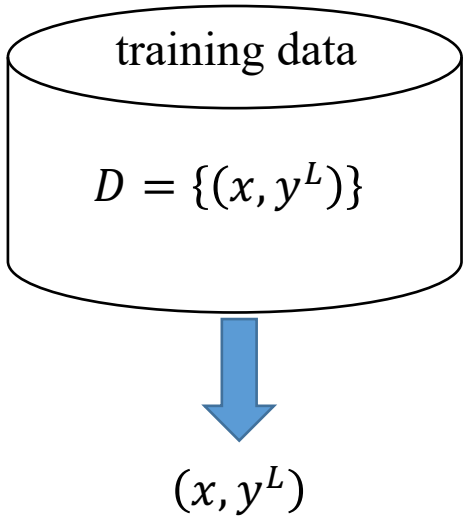
$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

$\nabla J_{ji} \leftarrow \delta_j^{l+1} \cdot a_i^l$;

$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \nabla J_{ji}$

Step 4. Return to Step 3 until each w^l converge



function $fc(w^l, a^l)$

$for\ i = 1: n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

function $bc(w^l, \delta^{l+1})$

$for\ i = 1: n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

反向传播算法

批处理BP算法

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initialize each w_{ij}^l, and choose a learning rate α

Step 3. for each mini-batch sample $D_m \subseteq D$

$\nabla J_{ji} = 0;$

for each $(x, y^L) \in D_m$

$a^1 \leftarrow x \in D_m;$

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l);$

end

$J(x, y^L);$

$\delta^L = \frac{\partial J(x, y^L)}{\partial z^L};$

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1});$

end

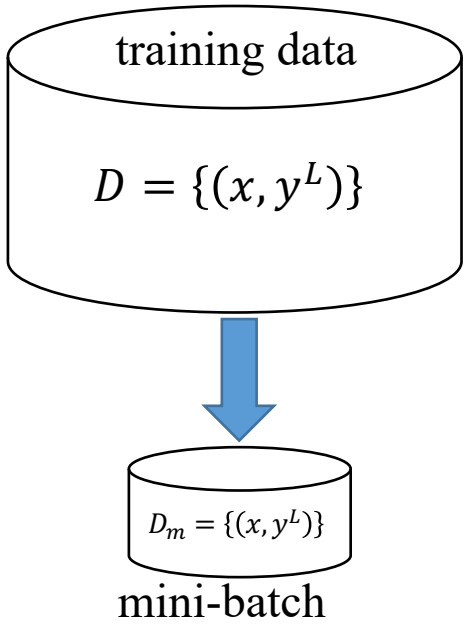
$\nabla J_{ji} \leftarrow \nabla J_{ji} + \delta_j^{l+1} \cdot a_i^l;$

end

$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \nabla J_{ji};$

end

Step 4. Return to Step 3 until each w^l converge



function $fc(w^l, a^l)$

$for\ i = 1: n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

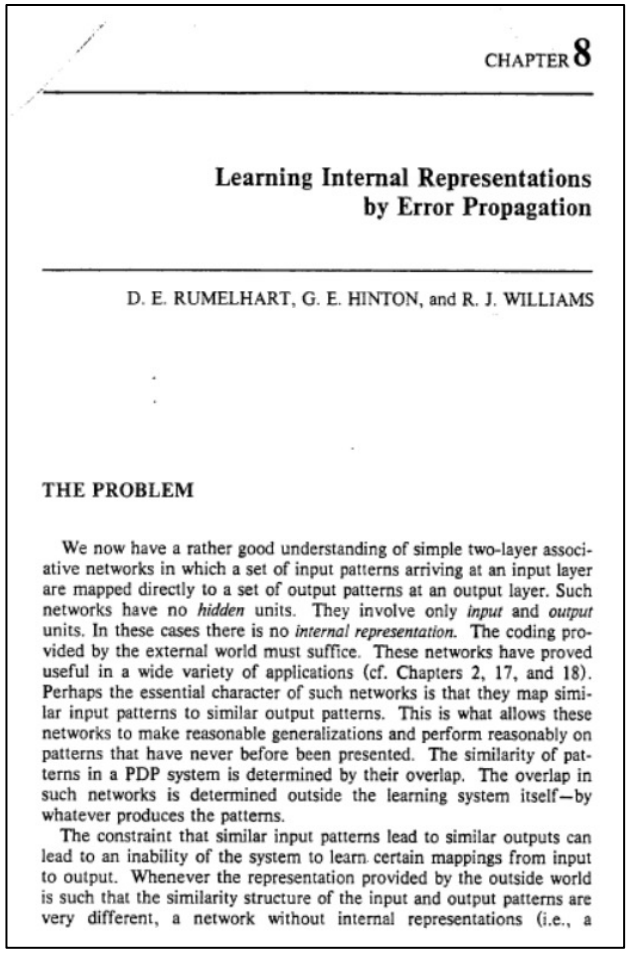
function $bc(w^l, \delta^{l+1})$

$for\ i = 1: n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

反向传播（BP）算法简史



CHAPTER 8

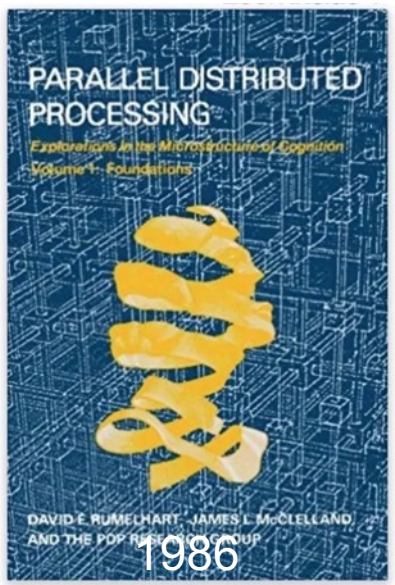
Learning Internal Representations by Error Propagation

D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

THE PROBLEM

We now have a rather good understanding of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. Such networks have no *hidden* units. They involve only *input* and *output* units. In these cases there is no *internal representation*. The coding provided by the external world must suffice. These networks have proved useful in a wide variety of applications (cf. Chapters 2, 17, and 18). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a PDP system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns.

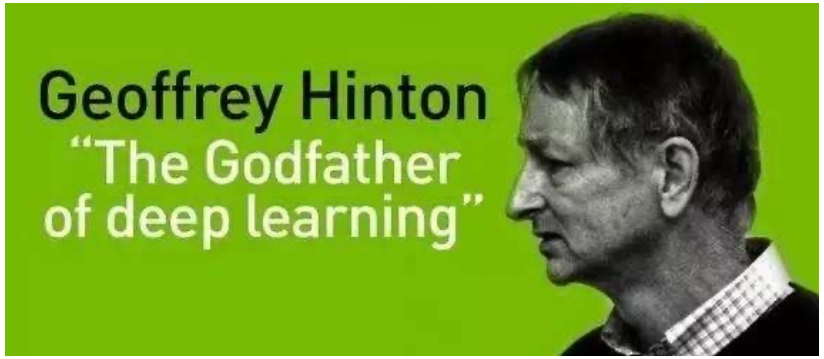
The constraint that similar input patterns lead to similar outputs can lead to an inability of the system to learn certain mappings from input to output. Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns are very different, a network without internal representations (i.e., a

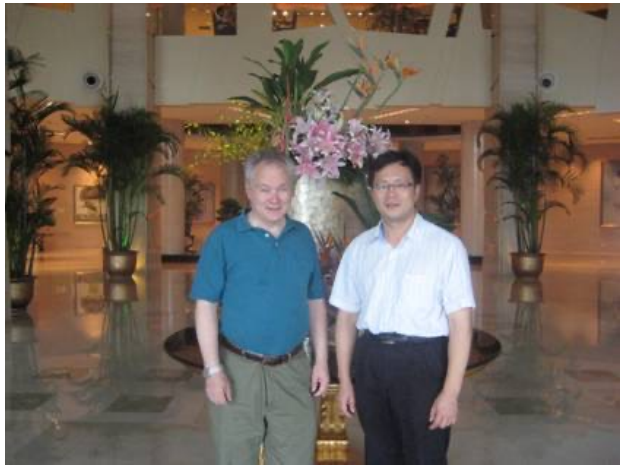


人工智能

深度神经网络

BP 学习算法





P. Werbos 教授

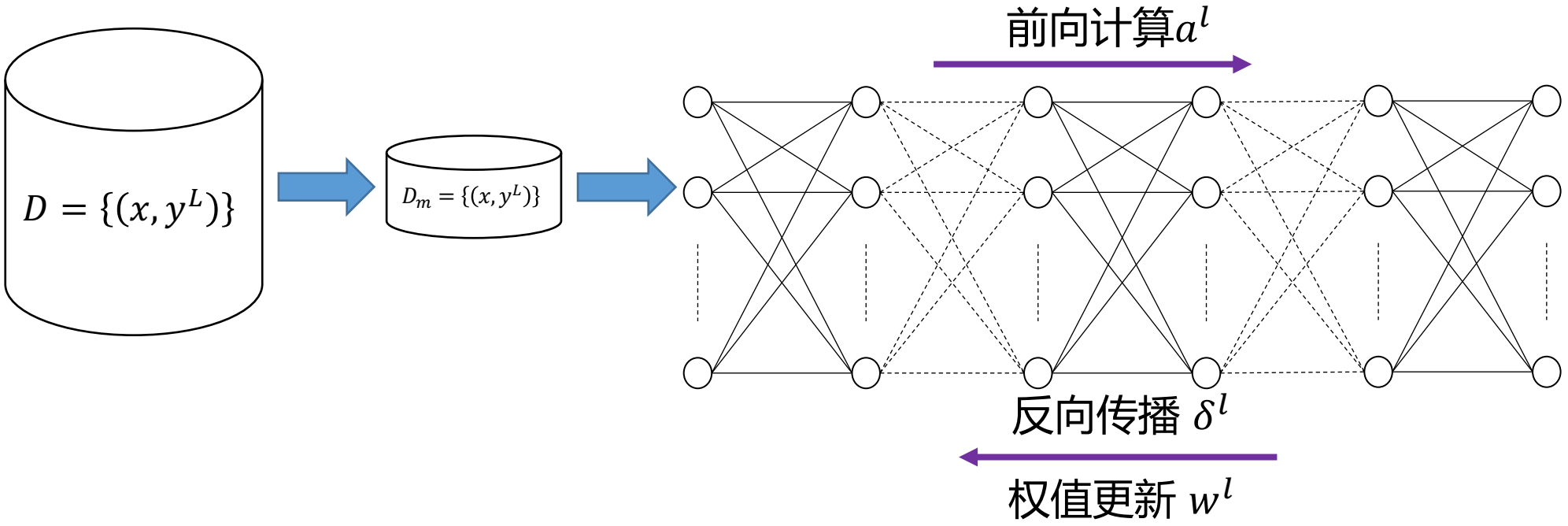
深度学习引论

第四章

神经网络训练

2022年 秋季

神经网络训练



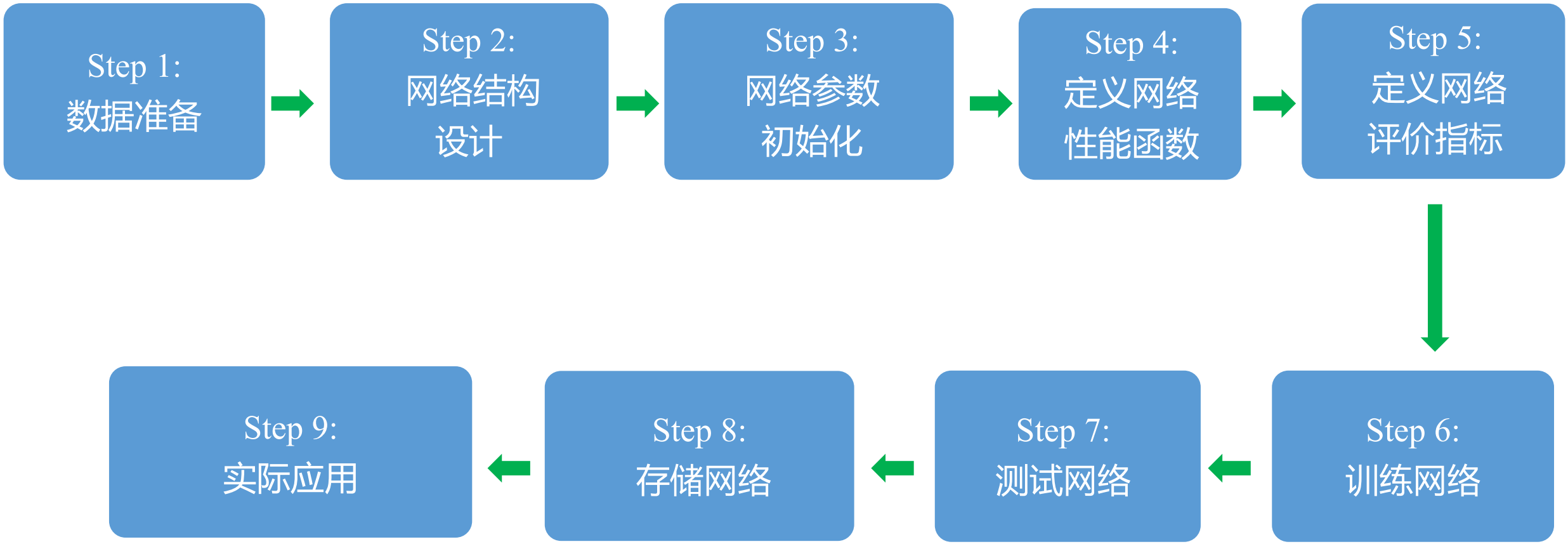
提纲

神经网络训练

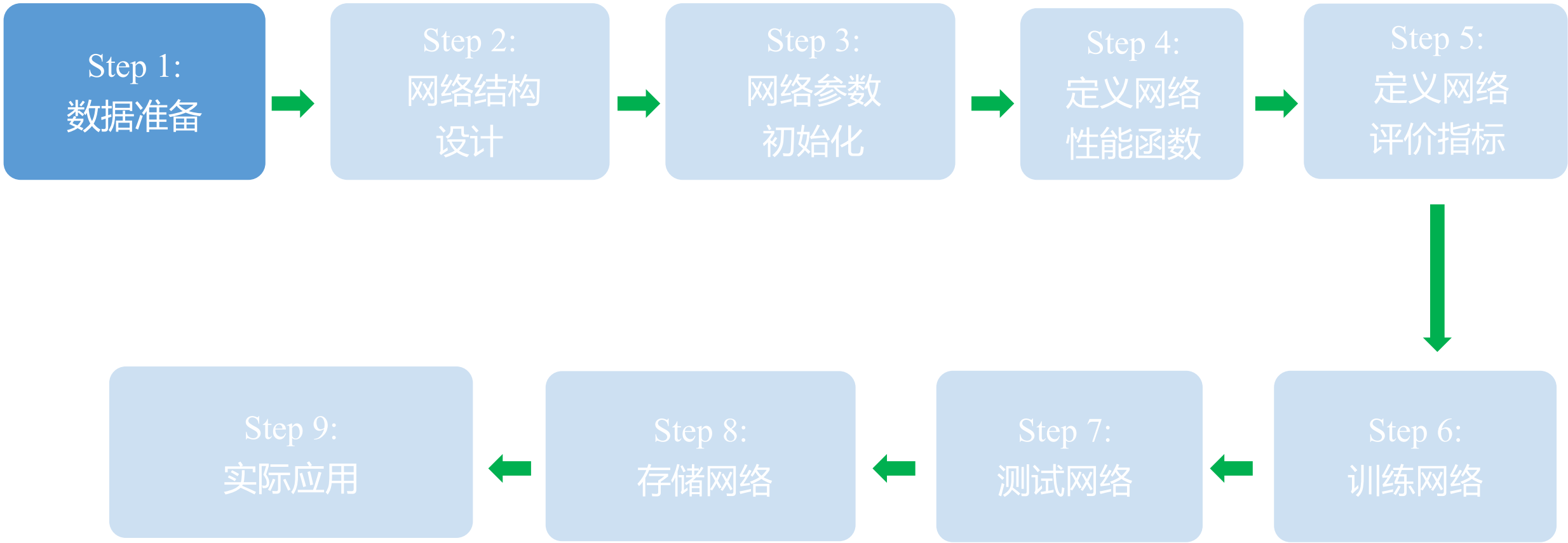
神经网络训练 - 实例

神经网络训练 - 问题

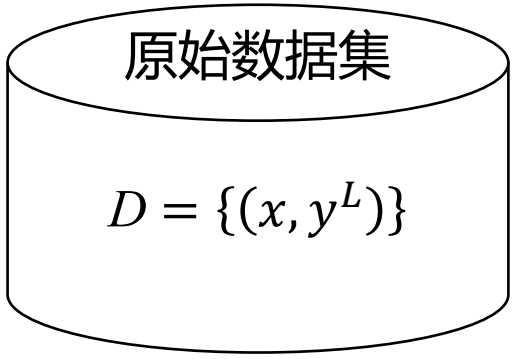
神经网络训练


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

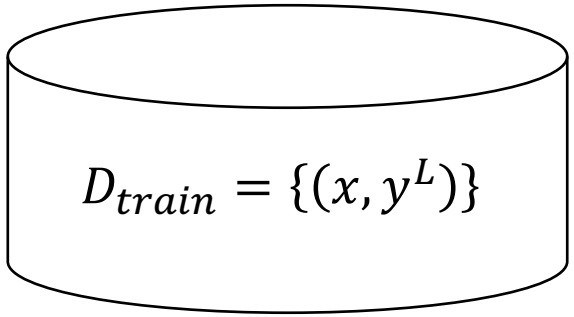
Step 1: 数据准备


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

Step 1: 数据准备

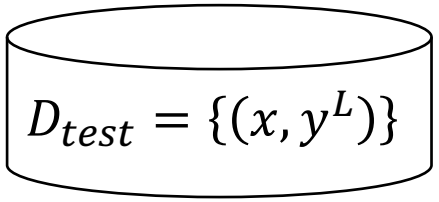


原图



训练数据集

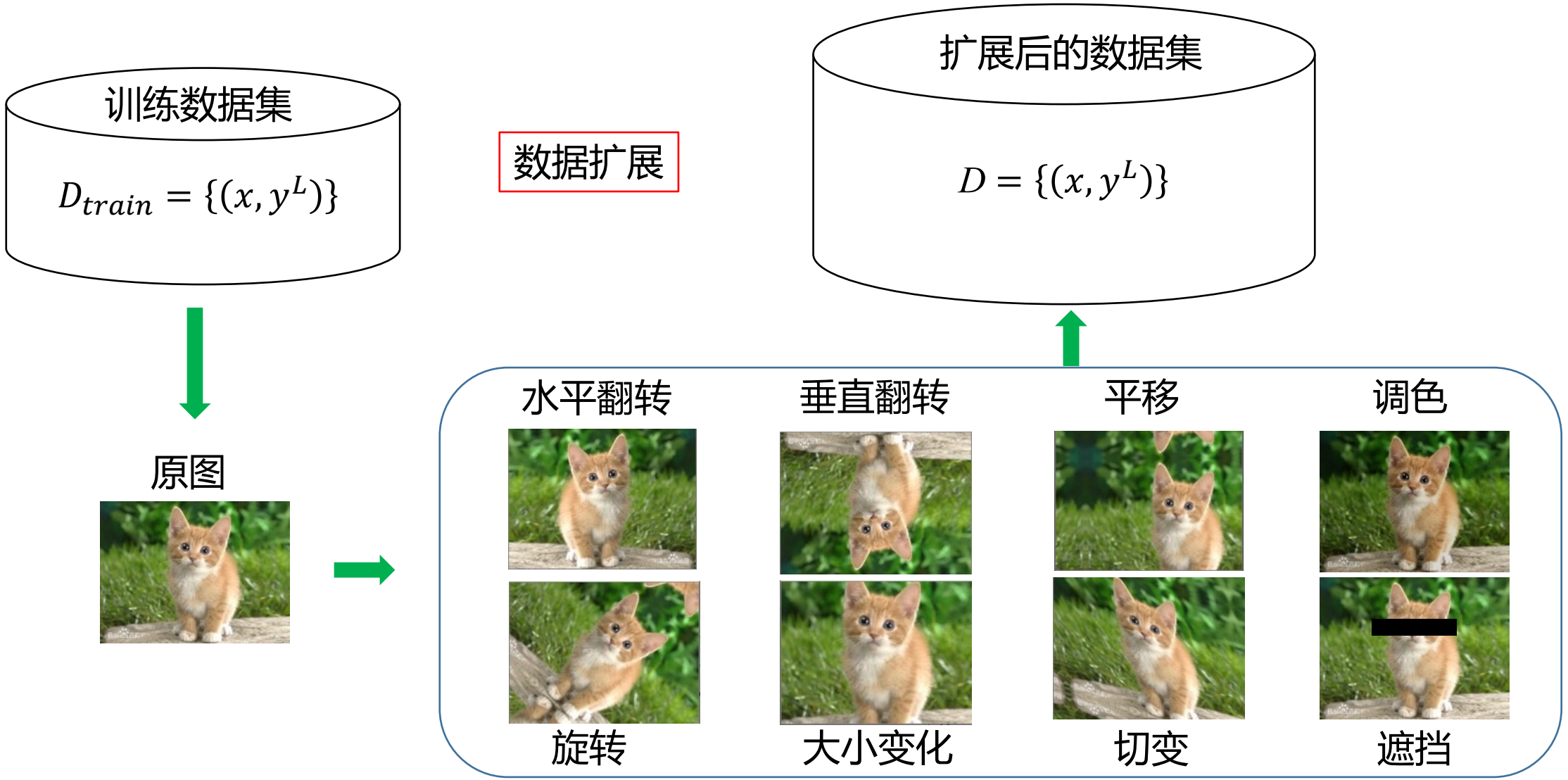
相对大一些
e.g. 80%



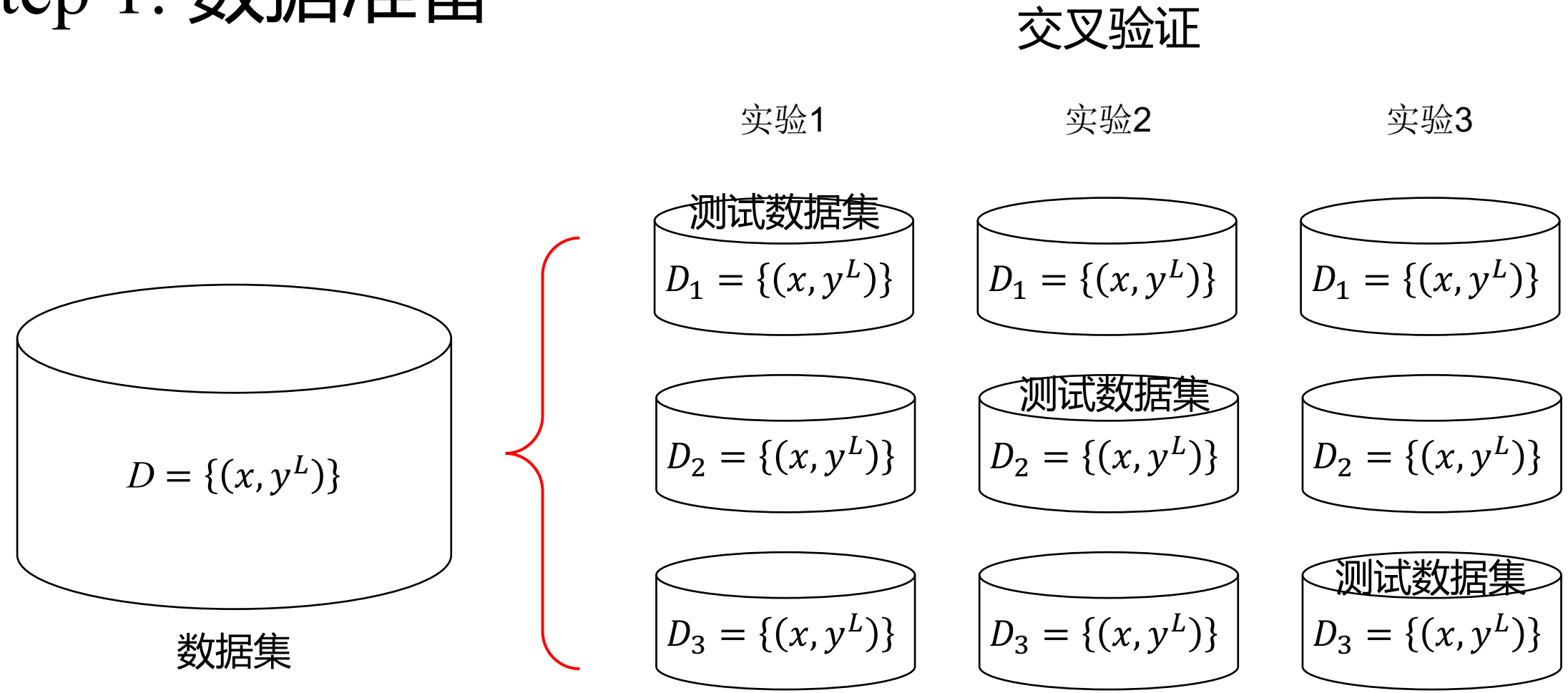
测试数据集

相对小一些
e.g. 20%

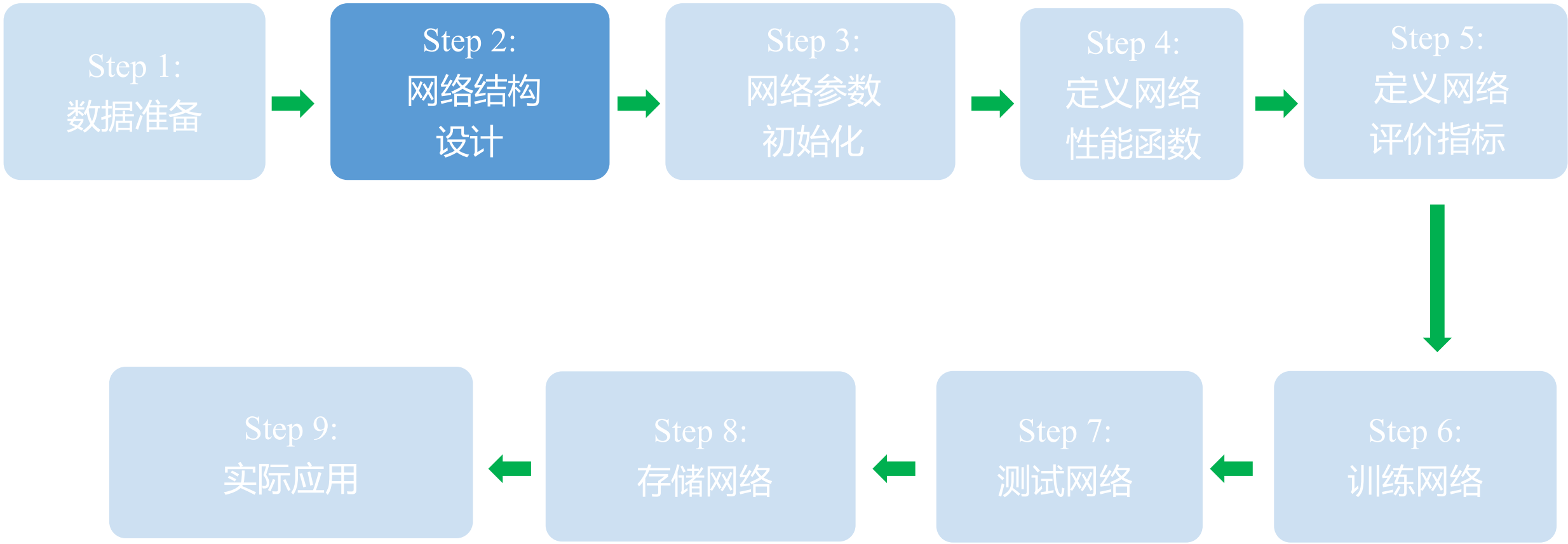
Step 1: 数据准备


训练数据集
$D_{train} = \{(x, y^L)\}$
数据扩展
扩展后的数据集
$D = \{(x, y^L)\}$
原图
水平翻转
垂直翻转
平移
调色
旋转
大小变化
切变
遮挡

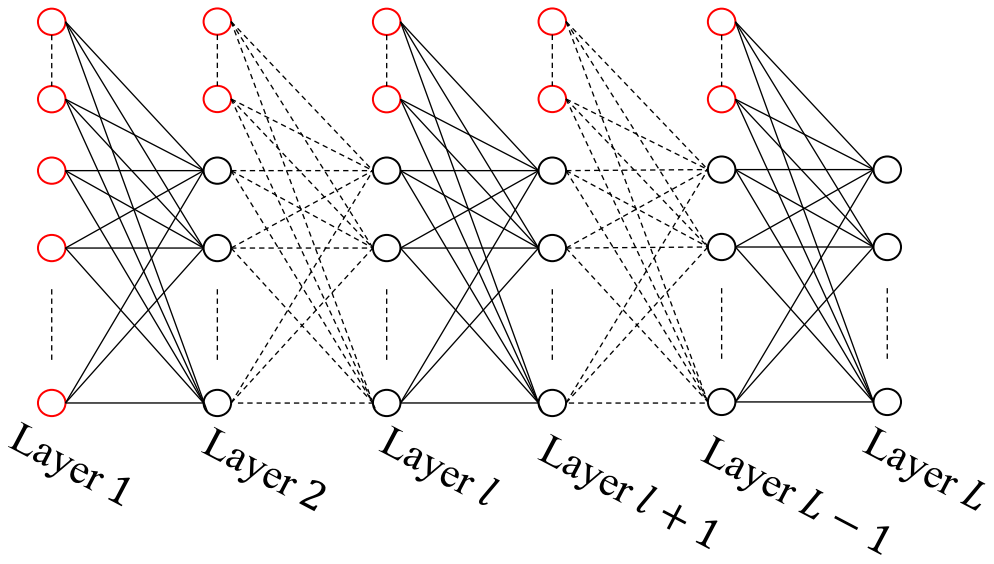
Step 1: 数据准备


交叉验证
实验1
实验2
实验3
$D = \{(x, y^L)\}$
数据集
测试数据集
$D_1 = \{(x, y^L)\}$
$D_2 = \{(x, y^L)\}$
$D_3 = \{(x, y^L)\}$
$D_1 = \{(x, y^L)\}$
测试数据集
$D_2 = \{(x, y^L)\}$
$D_3 = \{(x, y^L)\}$
$D_1 = \{(x, y^L)\}$
$D_2 = \{(x, y^L)\}$
测试数据集
$D_3 = \{(x, y^L)\}$

Step 2: 网络结构设计



Step 2: 网络结构设计



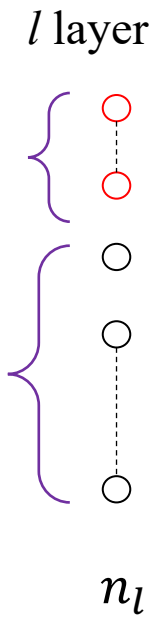
L=?

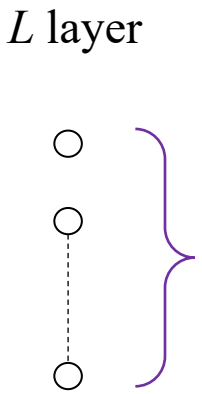
$(L \geq 3)$

- 设计网络层数
- 设计每层神经元数目
  - 输入神经元数量
  - 输出神经元数量
  - 表达神经元数量
- 设计激活函数

Step 2: 网络结构设计

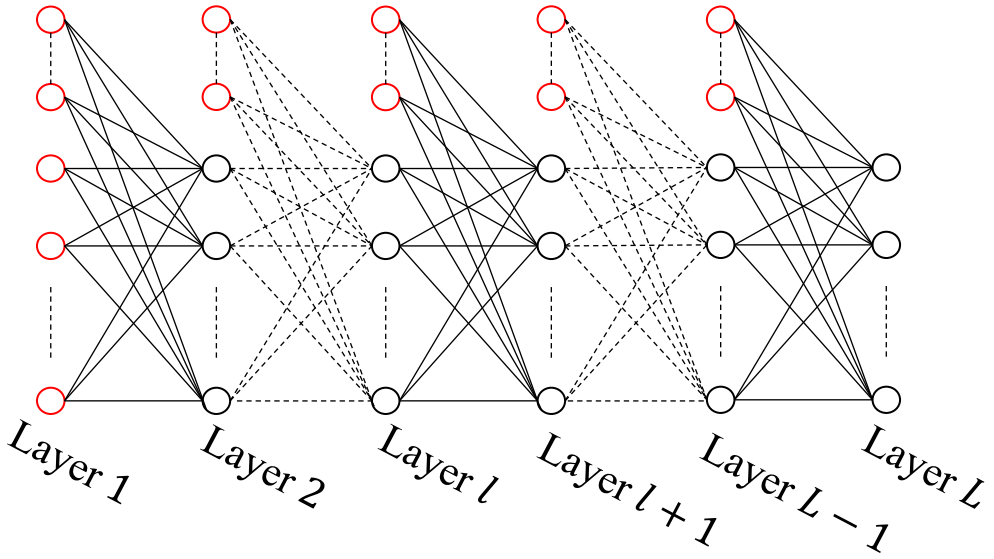
- 设计 l 层中外部输入神经元数量
- 设计 l 层中内部表达神经元数量





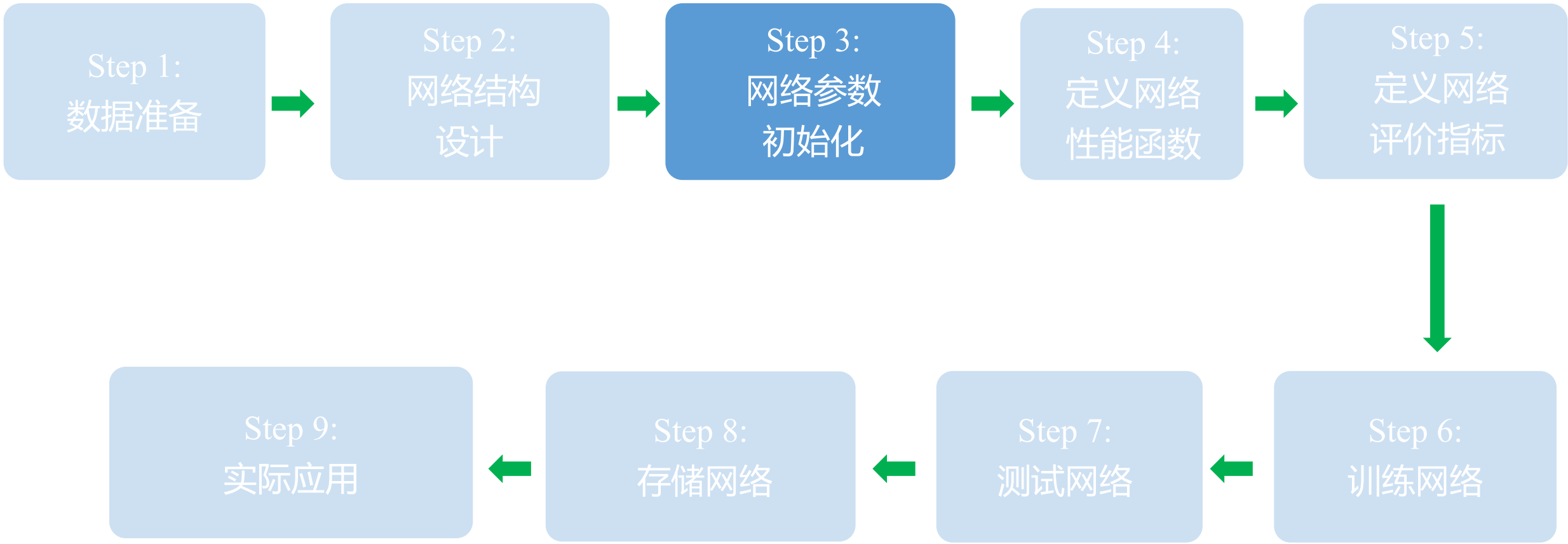
- 设计最后一层神经元数量

$n_l \geq n_L$

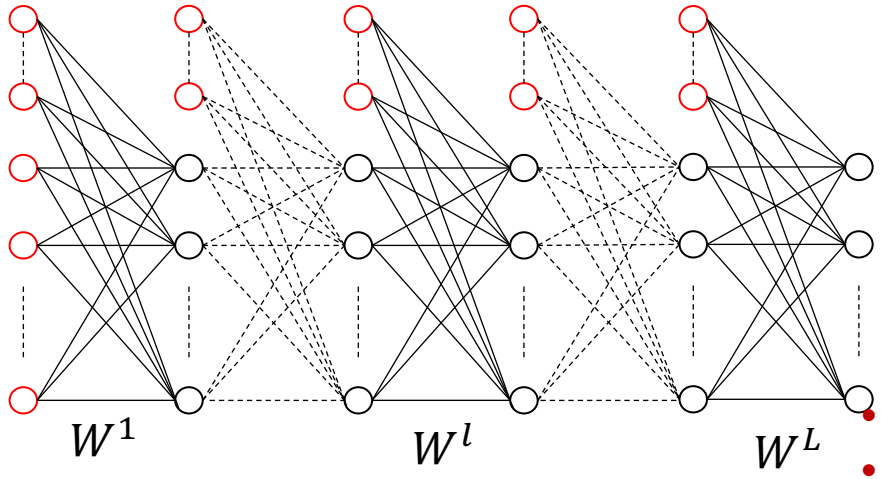


- 设计每层神经元数目
  - 输入神经元数量
  - 输出神经元数量
  - 表达神经元数量

Step 3: 初始化网络参数



Step 3: 初始化网络参数



- 连接权矩阵
- 初始外部输入
- 学习率

■ 连接权矩阵

$$W^l = \begin{bmatrix} w_{11}^l & \cdots & w_{1n_{l+1}}^l \\ \cdots & w_{ij}^l & \cdots \\ w_{n_l 1}^l & \cdots & w_{n_l \times n_{l+1}}^l \end{bmatrix}_{n_l \times n_{l+1}}$$

- 不能为0
- 不能相同
- 随机初始化
  - $W^l \sim N(0, \sigma^2)$
  - $W^l \sim U(-a, +a)$

Xavier初始化

$$a = \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

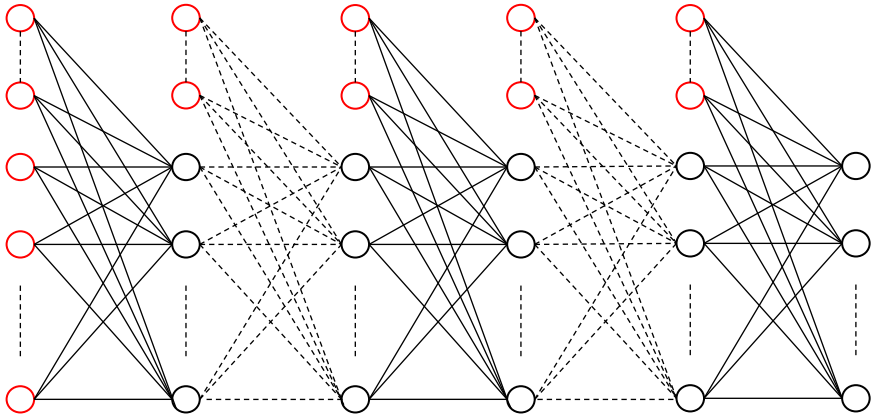
- 特殊初始化
  - Identical初始化
  - ……

$$W^l = \begin{bmatrix} 0.7267, 0.0485, -0.8780 \\ 0.4068, -0.2969, 0.5010 \\ 0.8697, -0.4613, -0.4553 \end{bmatrix}_{3X3}$$

(e.g. 随机初始化)

Step 3: 初始化网络参数

- 初始外部输入

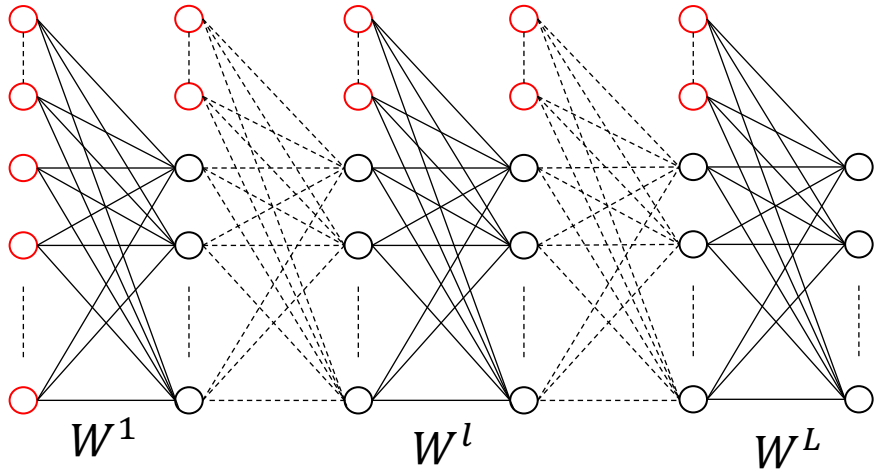


例如：

- 置为0
- 置为同样的数
- ……

- 连接权矩阵
- 初始外部输入
- 学习率

Step 3: 初始化网络参数

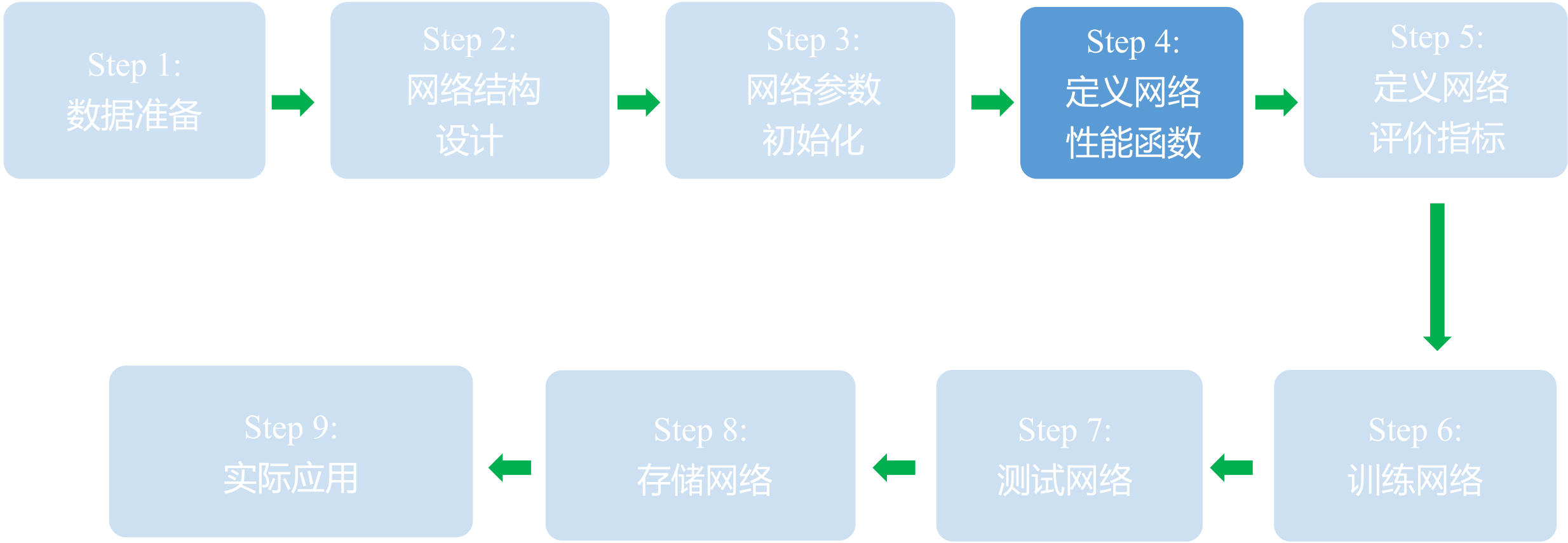


- 连接权矩阵
- 初始外部输入
- 学习率

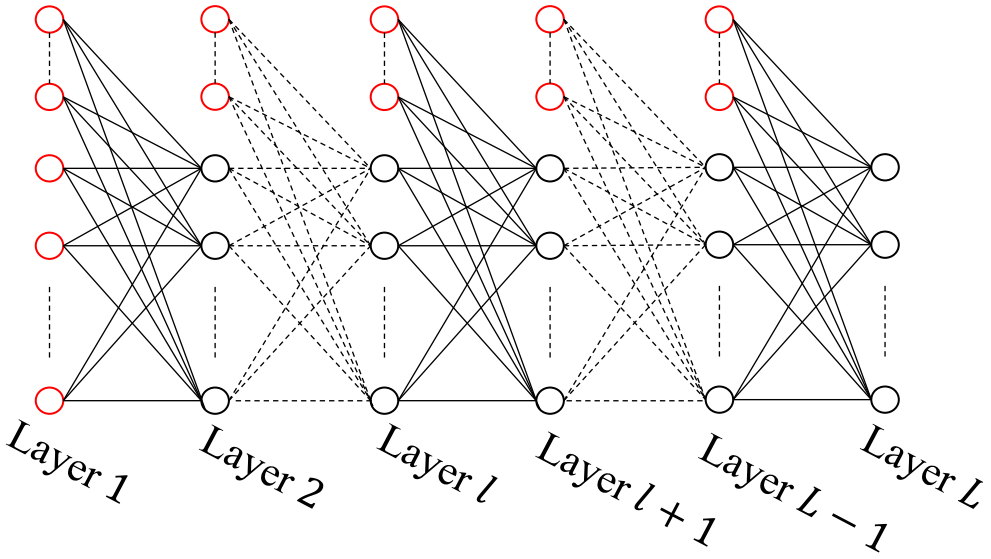
$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$$

- 学习率
  - 正实数
  - 小学习率：收敛较慢
  - 大学习率：可能在震荡
  - 可变学习率

Step 4: 定义性能函数



Step 4: 定义性能函数

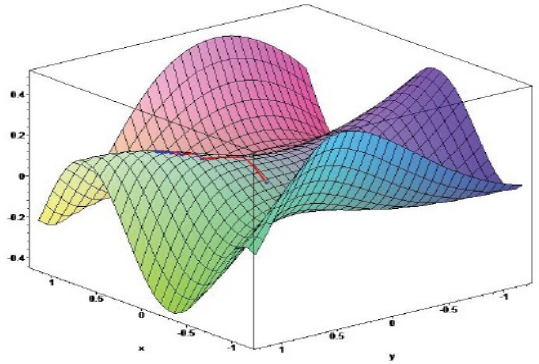


网络预测

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

目标输出

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

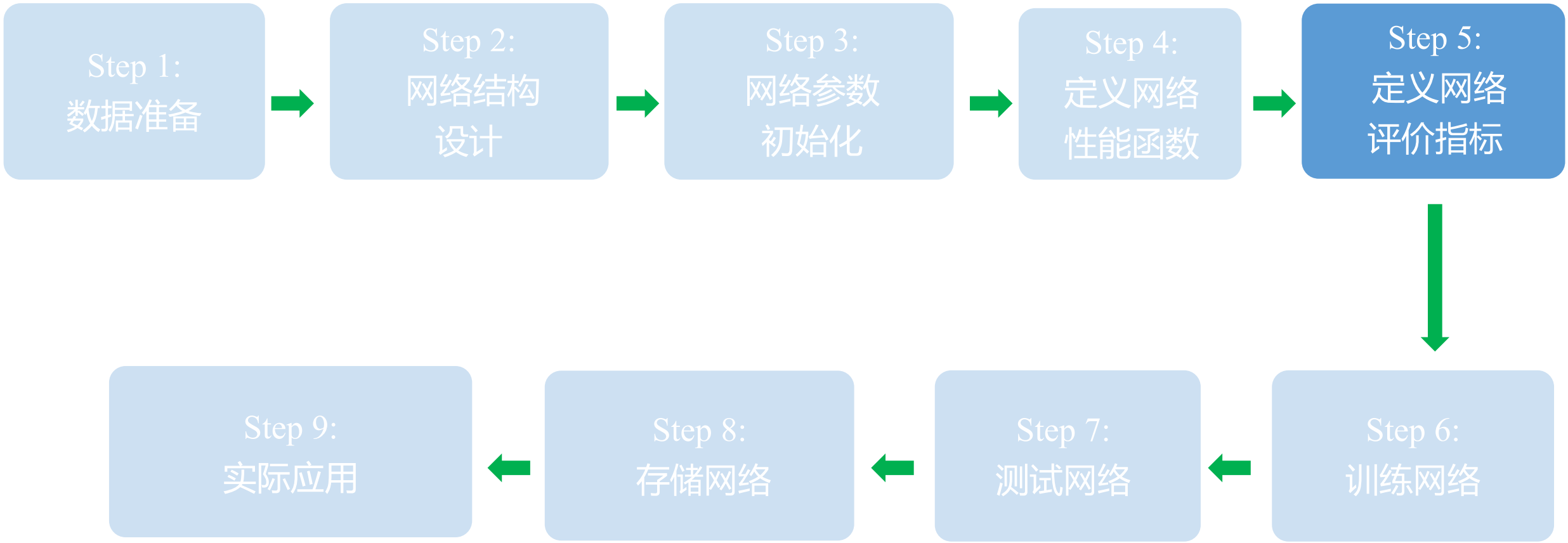


定义性能函数的方式是多种多样的。一种常见的性能函数如下：

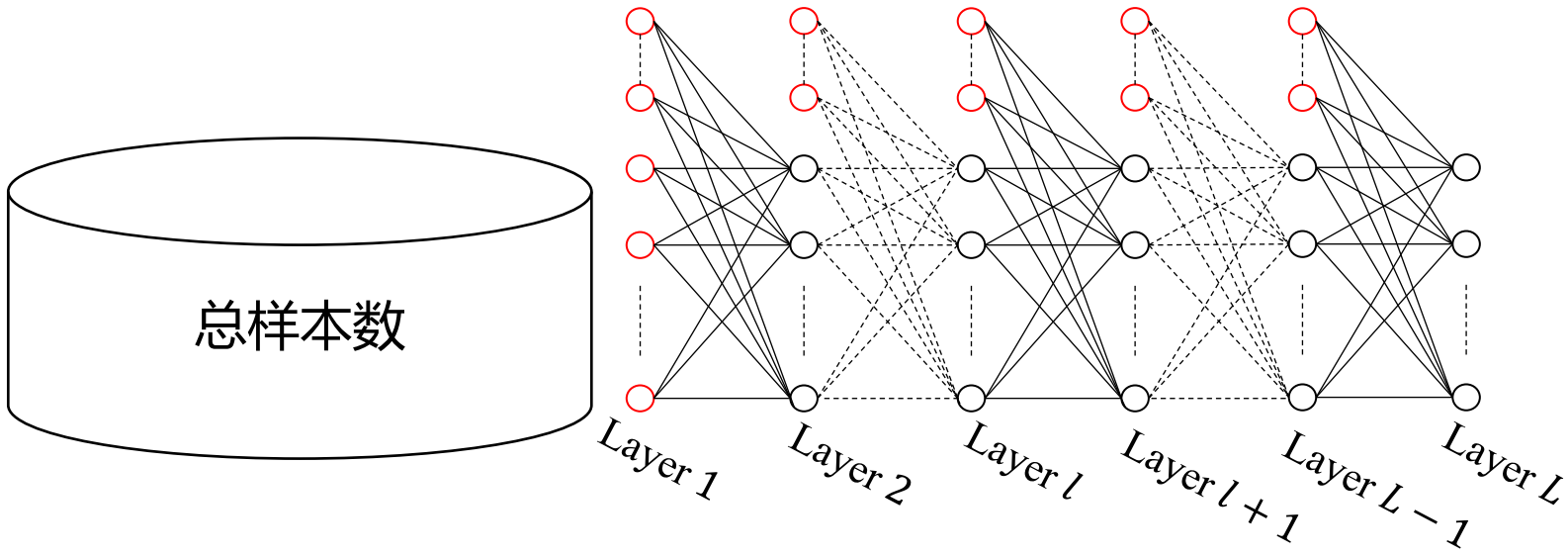
$$e_j = a_j^L - y_j^L$$

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2 = J(w^1, \cdots, w^L)$$

Step 5: 定义网络评价指标



Step 5: 定义网络评价指标



网络预测　　目标输出

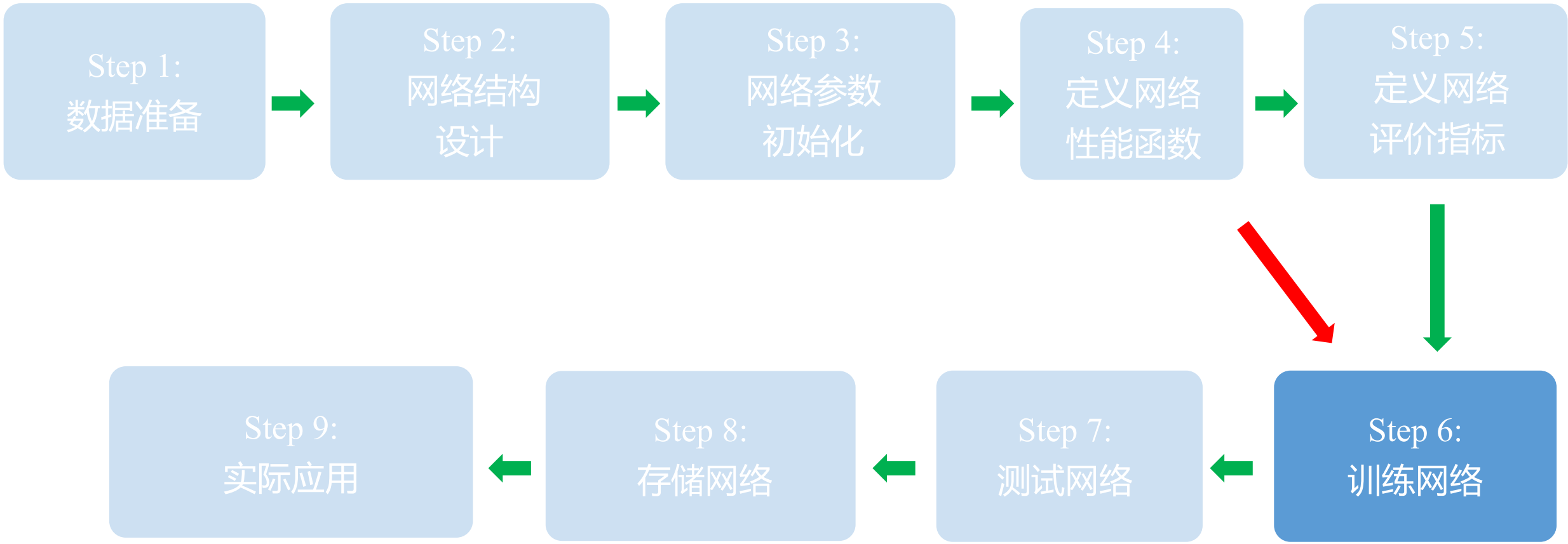
$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix} \quad y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

模型预测正确的样本数

$$\text{准确率} = \frac{\text{预测正确的样本数}}{\text{总样本数}}$$

- 敏感性、特异性、F1分数、……
- mAP、FROC、……
- BLEU、GLEU、……

Step 6：训练网络


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

BP算法

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initialize each w_{ij}^l, and choose a learning rate α

Step 3. for each $D_m \subseteq D$

$\nabla J_{ji} = 0;$

for each $(x, y^L) \in D_m$

$a^1 \leftarrow x \in D_m;$

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l);$

end

$J(x, y^L);$

$\delta^L = \frac{\partial J(x, y^L)}{\partial z^L};$

for $l = L - 1{:}1$

$\delta^l \leftarrow bc(w^l, \delta^{l+1});$

end

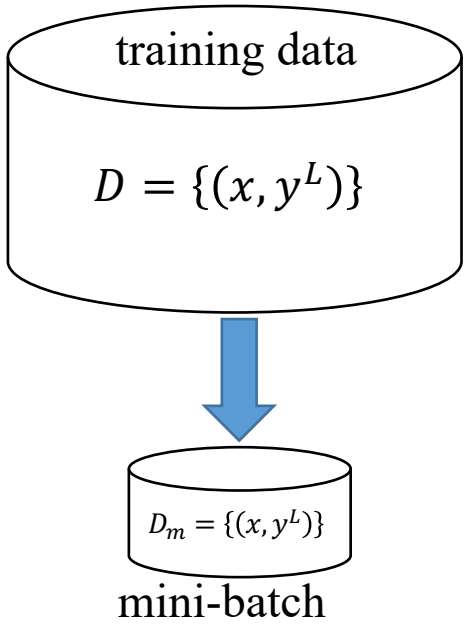
$\nabla J_{ji} \leftarrow \nabla J_{ji} + \delta_j^{l+1} \cdot a_i^l;$

end

$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \nabla J_{ji};$

end

Step 4. Return to Step 3 until each w_{ji}^l converge



function $fc(w^l, a^l)$

$for\ i = 1{:}n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

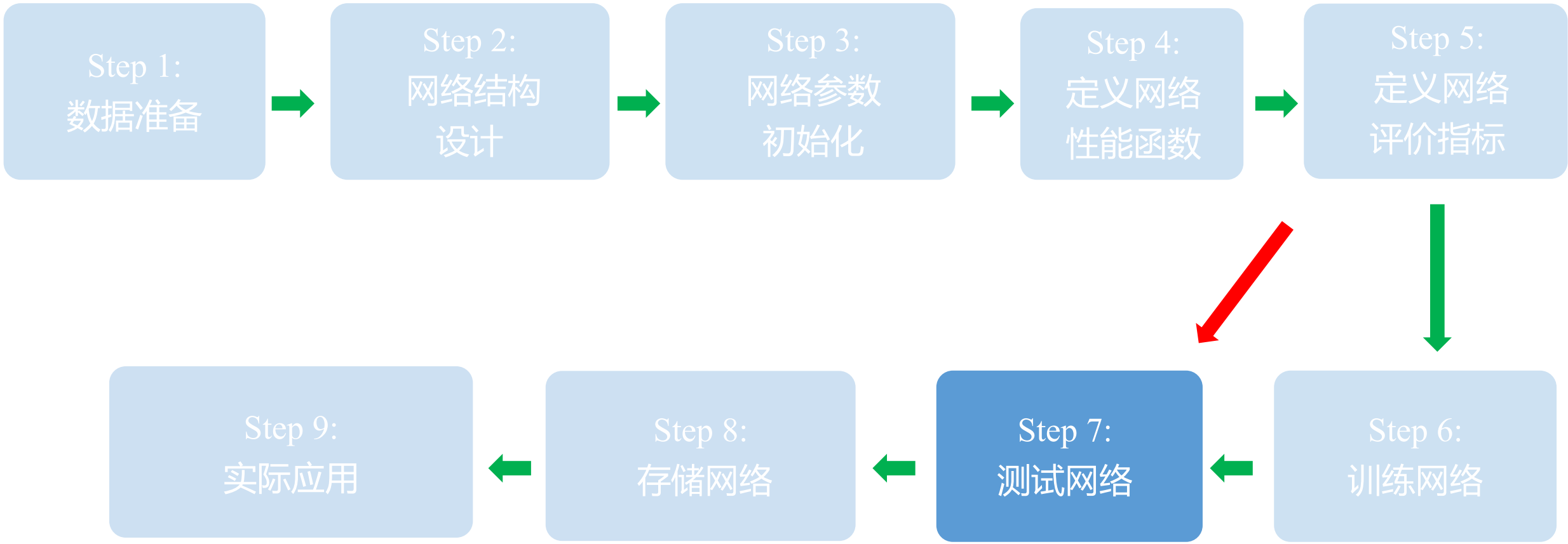
function $bc(w^l, \delta^{l+1})$

$for\ i = 1{:}n_l$

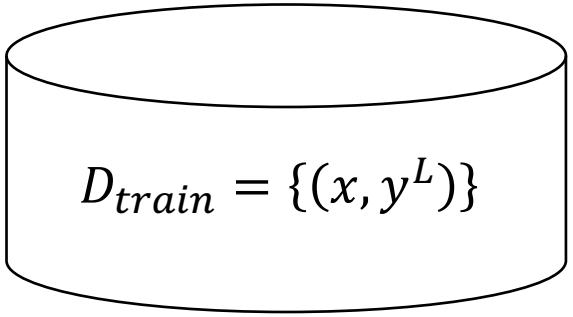
$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

end

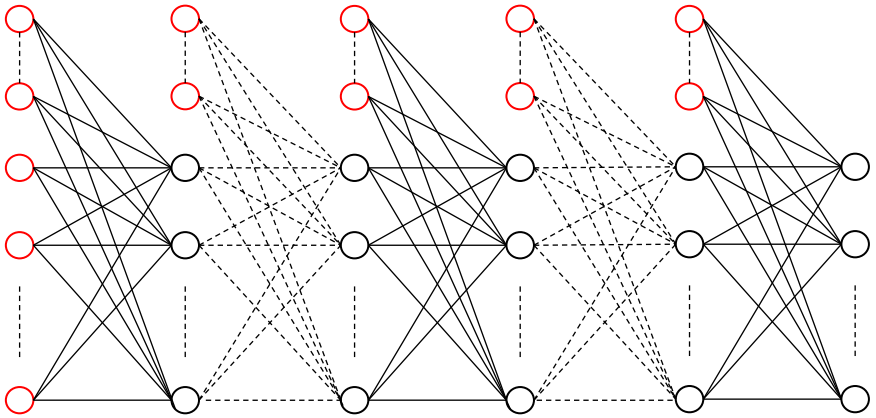
Step 7: 测试网络


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

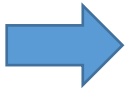
Step 7: 测试网络



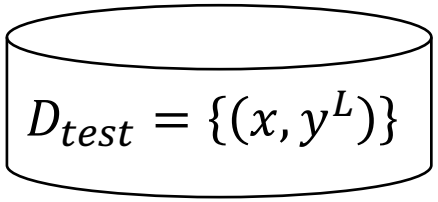
训练集



测试神经网络



准确率 $=$ $\frac{\text{预测正确的样本数}}{\text{总样本数}}$

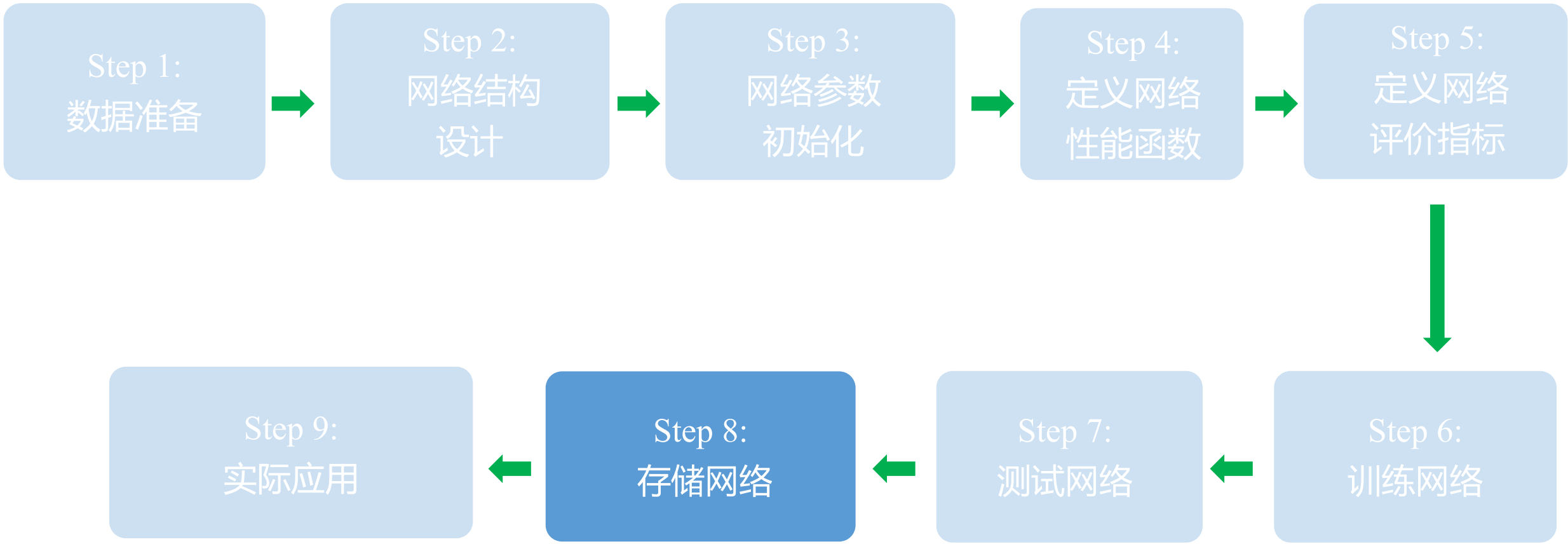


测试集

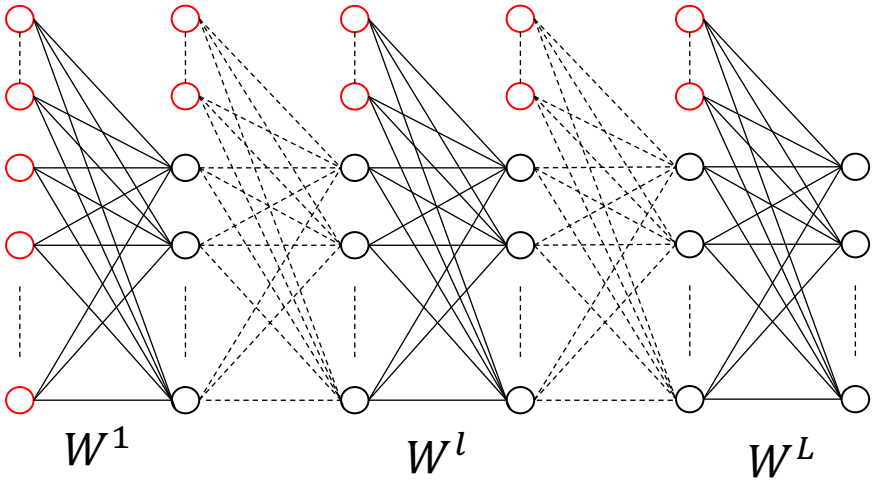
- 在训练集上测试
- 在测试集上测试

欠拟合：在训练集上表现不佳
过拟合：在训练数据上表现很好，但在测试数据上表现不佳
表现良好：在训练集和测试集均表现良好

Step 8: 存储网络


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

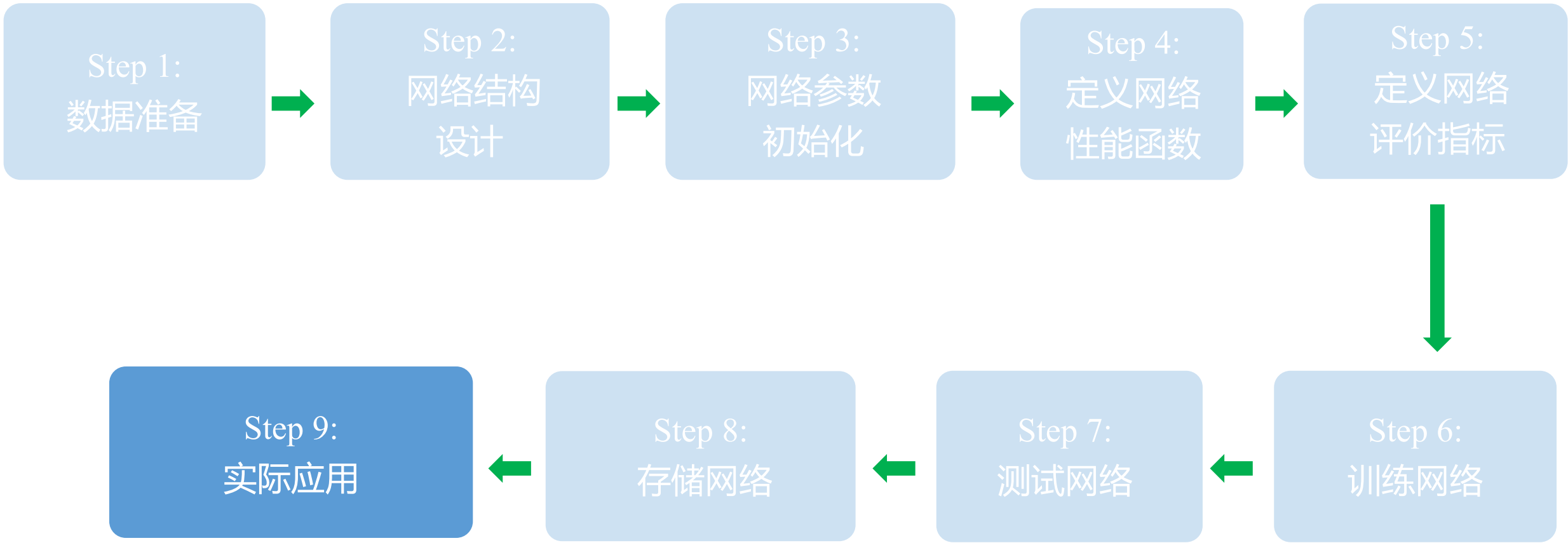
Step 8: 存储网络



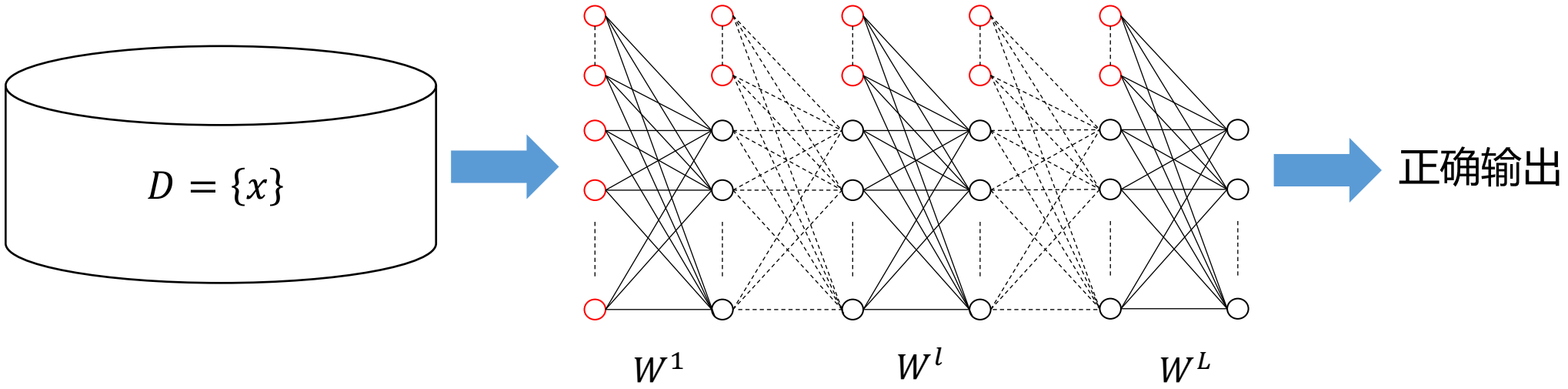
$$W^l = \begin{bmatrix} w_{11}^l & \dots & w_{1n_{l+1}}^l \\ \dots & w_{ij}^l & \dots \\ w_{n_l 1}^l & \dots & w_{n_l \times n_{l+1}}^l \end{bmatrix}_{n_l \times n_{l+1}}$$

- 保存网络的所有参数
  - 模型层数
  - 每层的神经元数目
  - 学习率
  - ……
  - 每一个连接权矩阵

Step 9: 实际应用


Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

Step 9: 实际应用



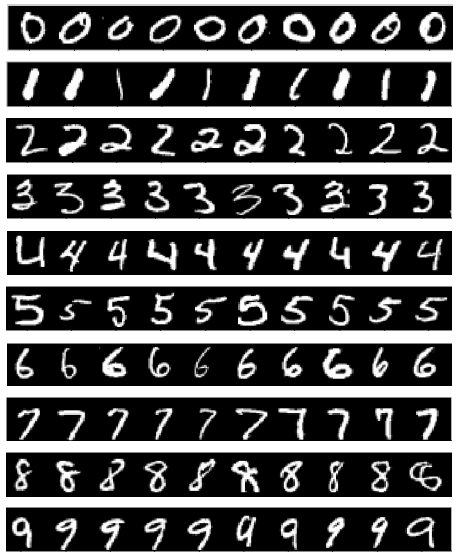
提纲

神经网络训练

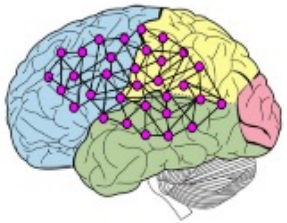
神经网络训练 - 实例

神经网络训练 - 问题

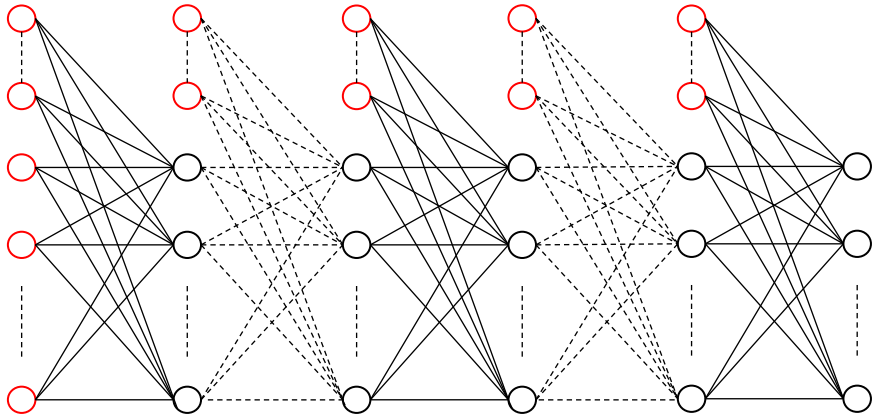
手写体数字识别问题



数字化



≈

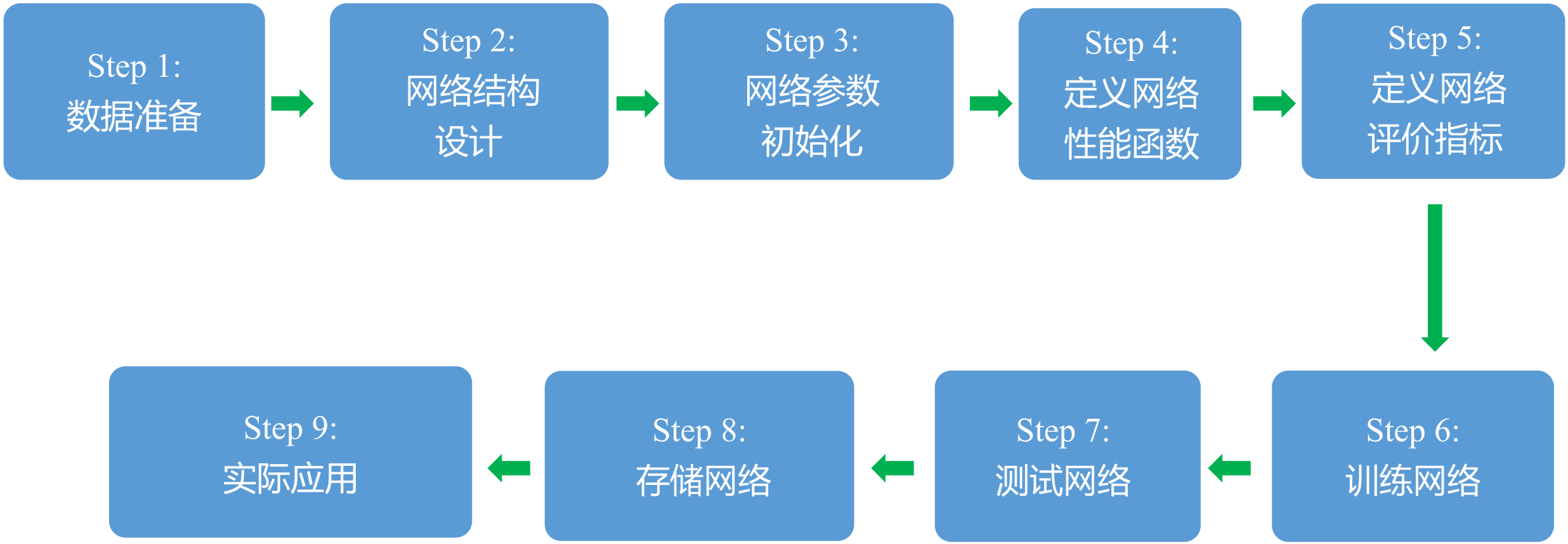


表征

0
1
2
3
4
5
6
7
8
9

任务：
使用BP算法训练神经网络来识别手写数字

神经网络训练

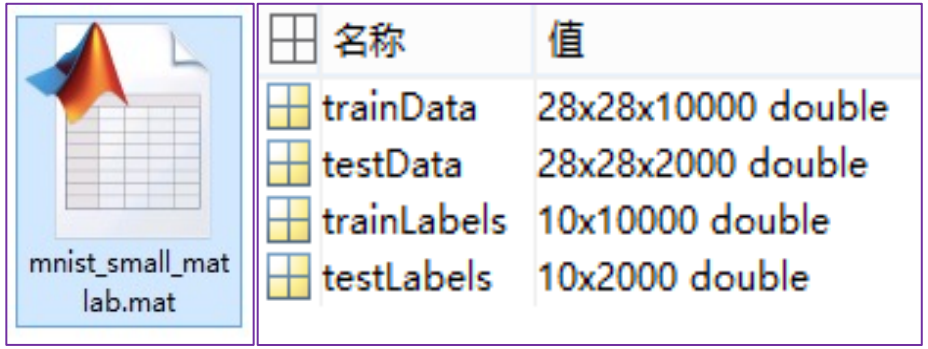

Step 1:
数据准备
Step 2:
网络结构
设计
Step 3:
网络参数
初始化
Step 4:
定义网络
性能函数
Step 5:
定义网络
评价指标
Step 6:
训练网络
Step 7:
测试网络
Step 8:
存储网络
Step 9:
实际应用

Step 1: 数据准备

数据集: **MNIST_small**

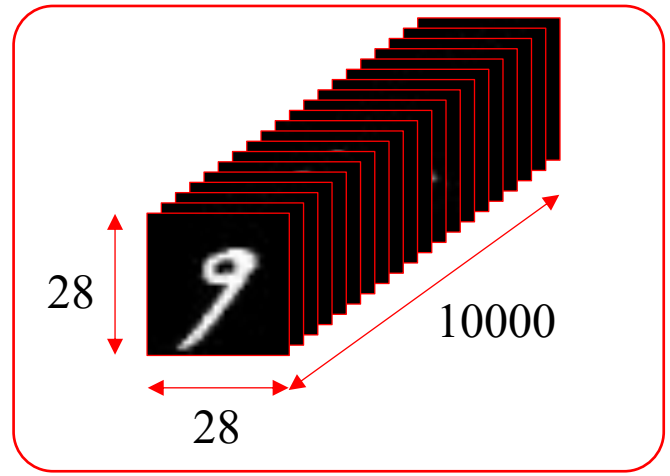
MNIST 是一个手写数字数据库，通过“重新混合”来自 MNIST 原始数据集的样本创建。 它包含由美国人口普查局的高中生和雇员书写的数字。这些数字已经过大小归一化并以 28×28 图像为中心。

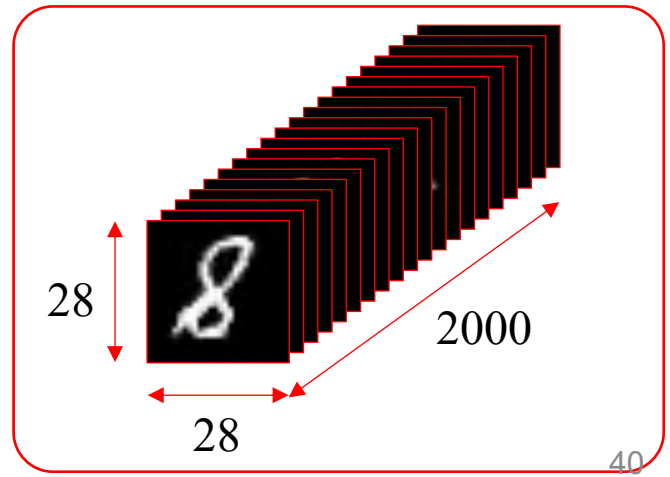
MNIST_small 数据集是 MNIST 的子集，包含 10000个训练样本和2000个测试样本。



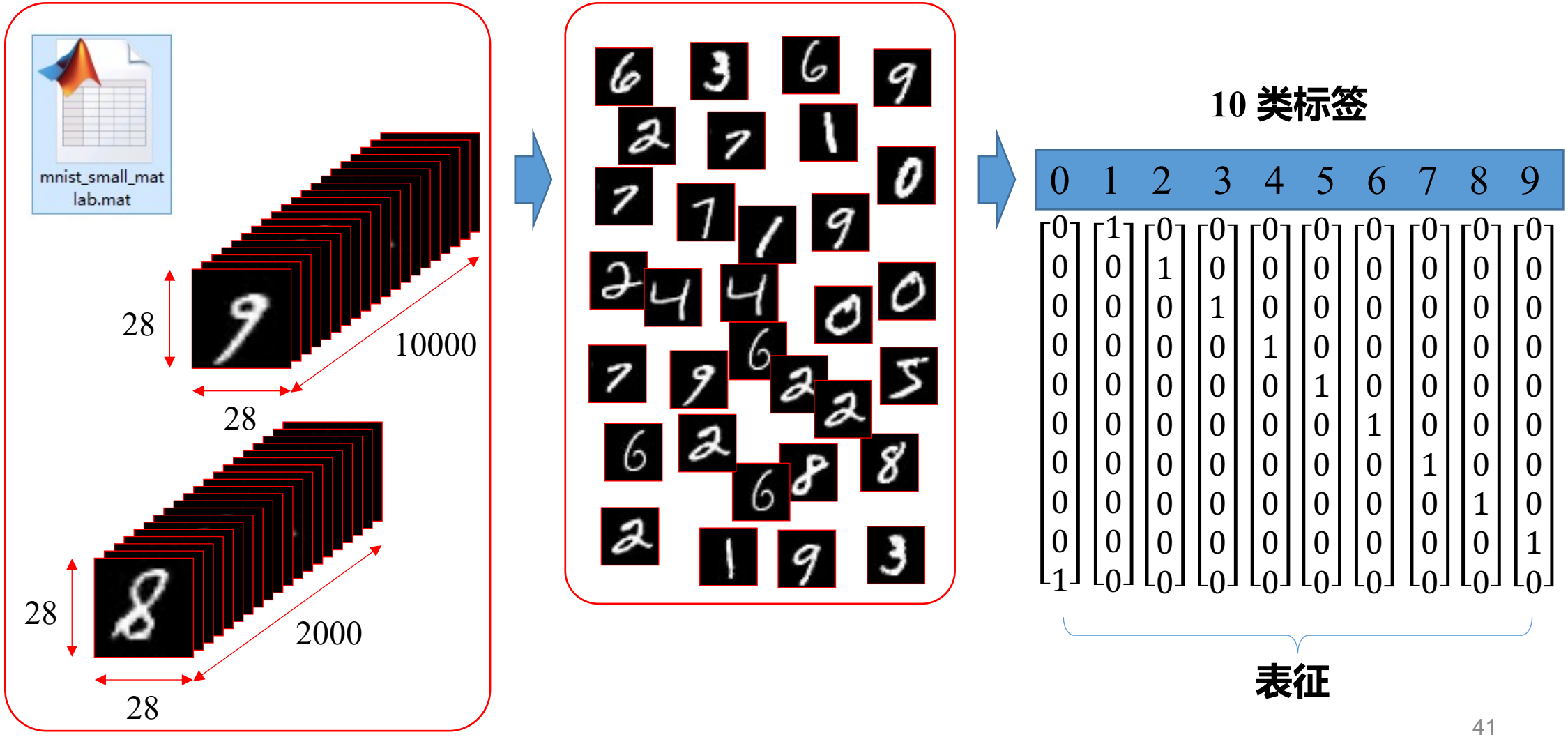
下载链接:
MNIST http://yann.lecun.com/exdb/mnist/
MNIST_small: https://github.com/kswersky/nnet/blob/master/mnist_small.mat

数据





Step 1: 数据准备


mnist_small_mat
lab.mat
28
28
10000
28
28
2000
10 类标签
0 1 2 3 4 5 6 7 8 9
表征

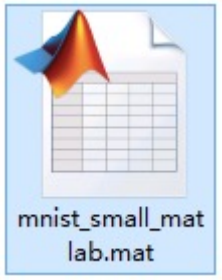
Step 1: 数据准备

28×28

数字化

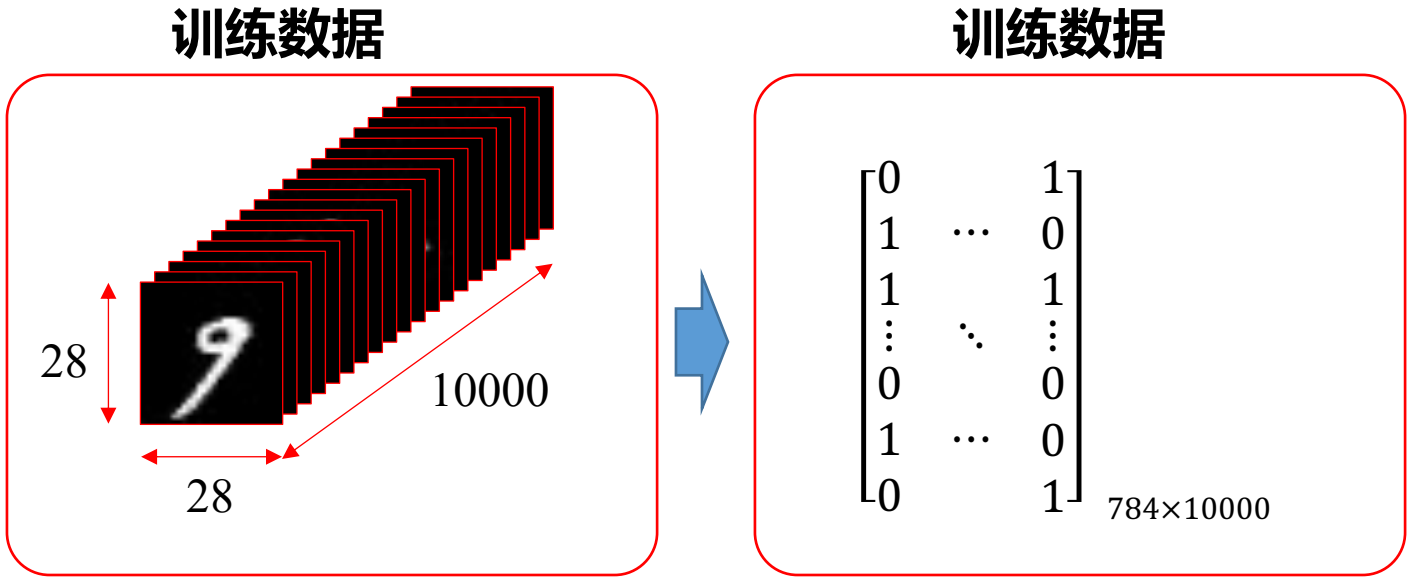
$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{784}$$

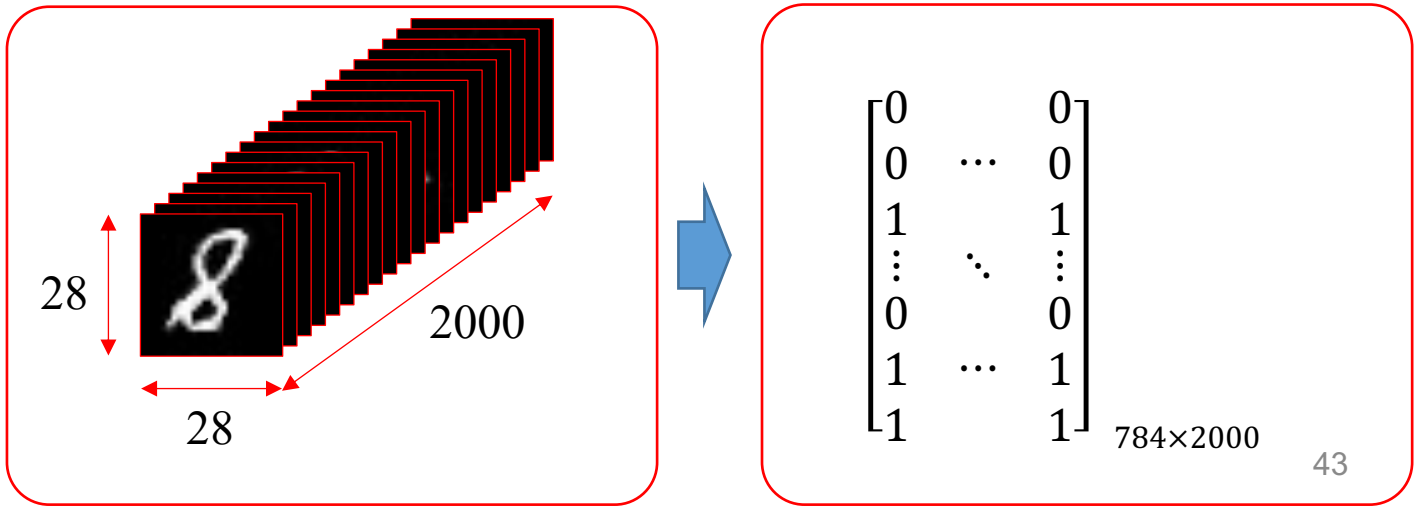
Step 1: 数据准备



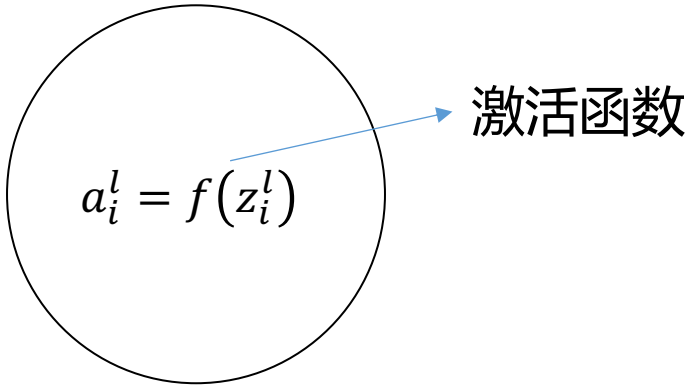
训练集
- 用于网络训练
- 10000 个样本

测试集
- 用于评估网络的性能
- 2000 个样本



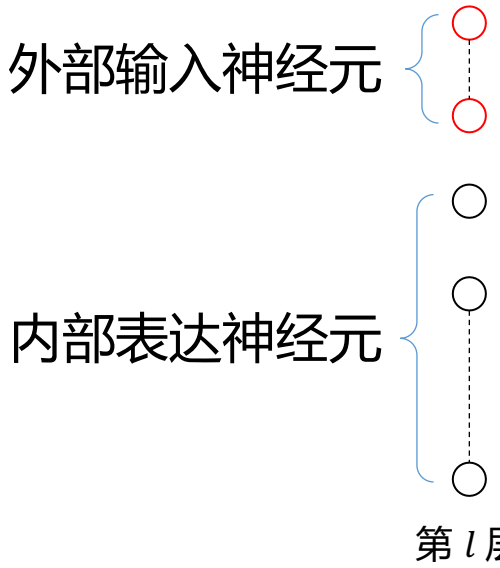


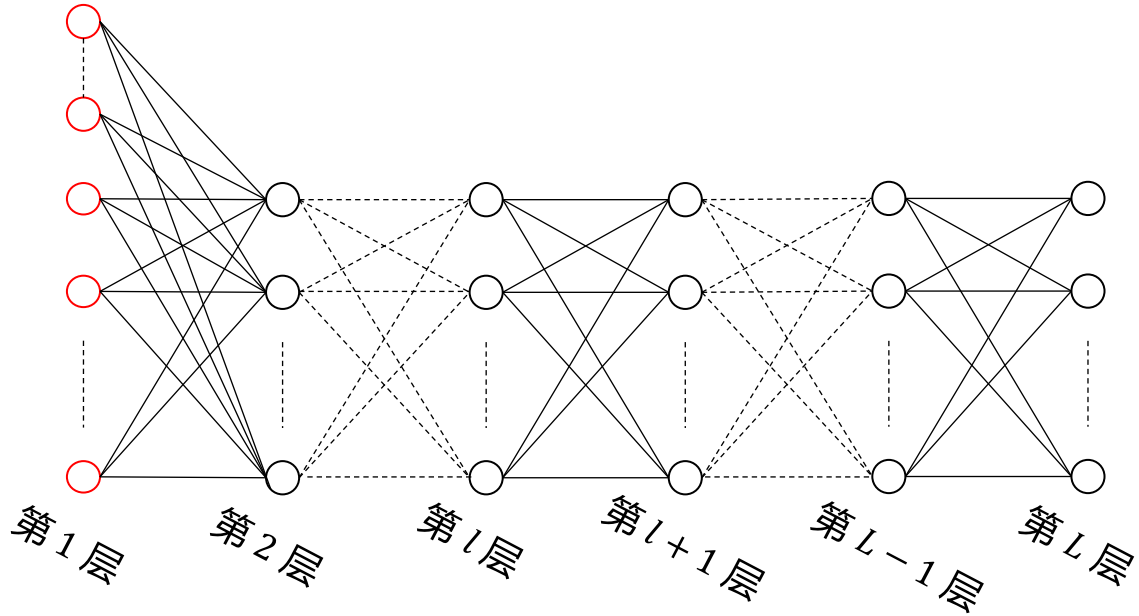
Step 2: 网络结构设计



网络结构设计:

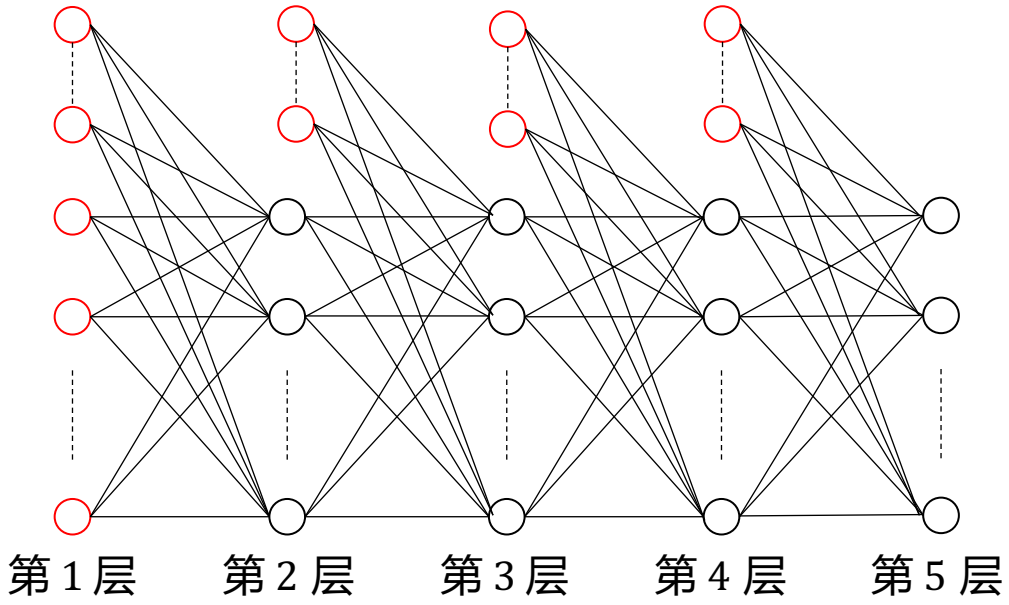
1. 网络层数
2. 每层神经元数量（外部神经元和内部神经元）
3. 激活函数





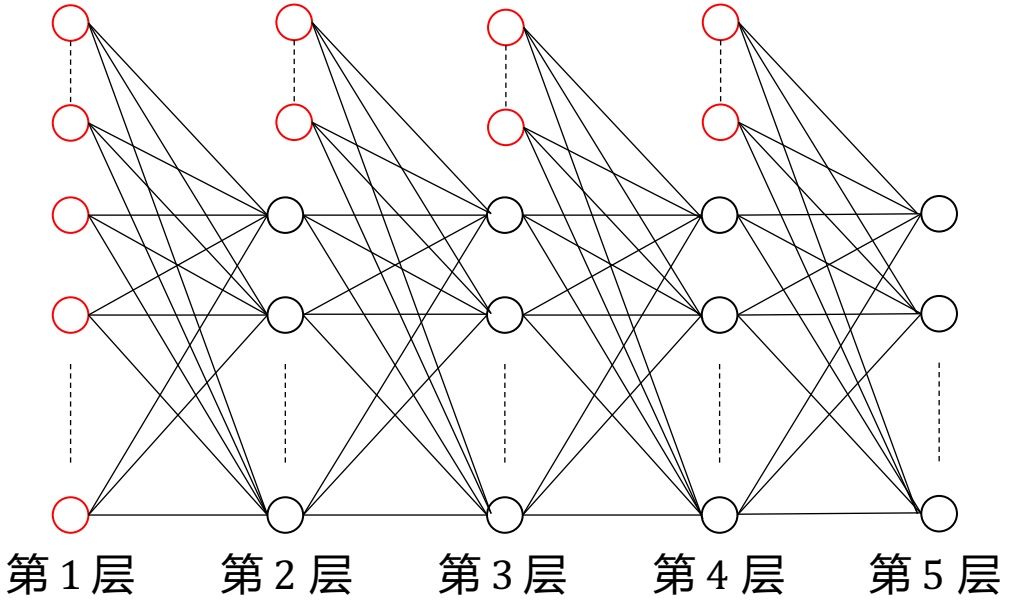
Step 2: 网络结构设计

定义网络的层数 $L=5$

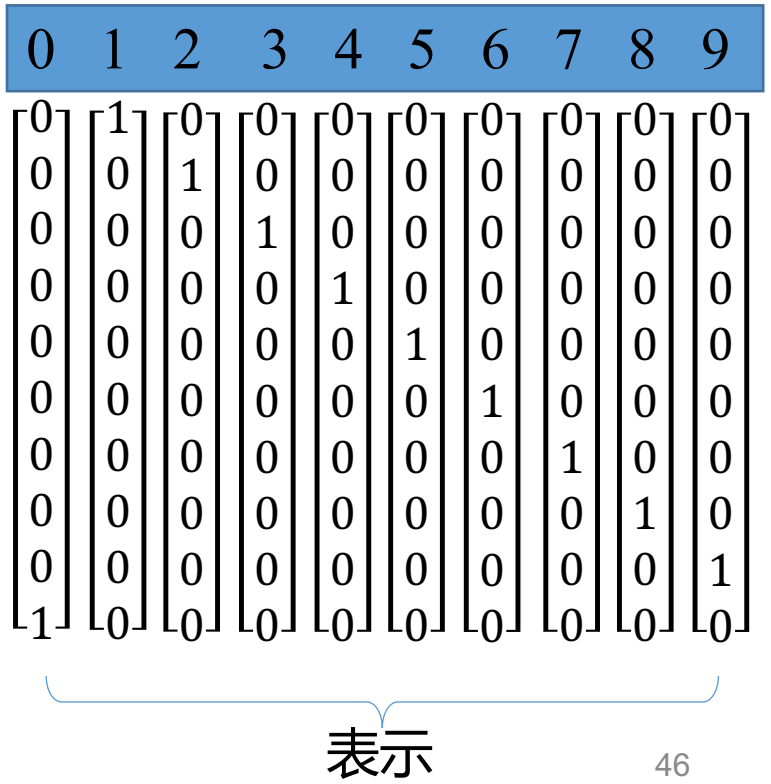


Step 2: 网络结构设计

定义最后一层神经元数

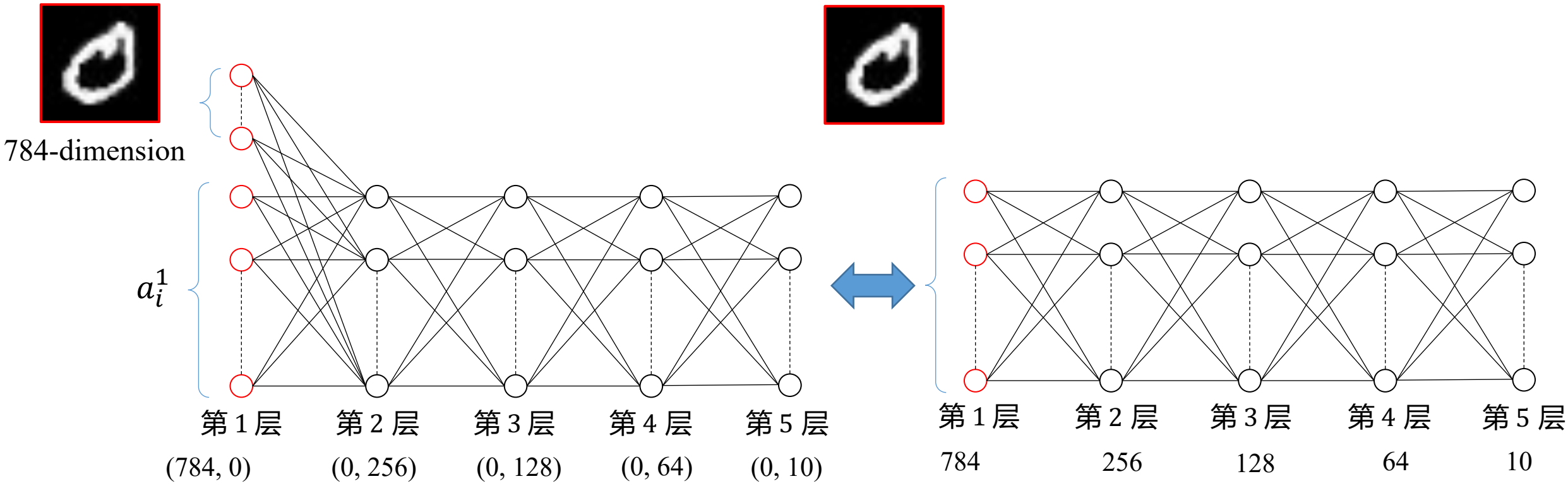


10 类标签



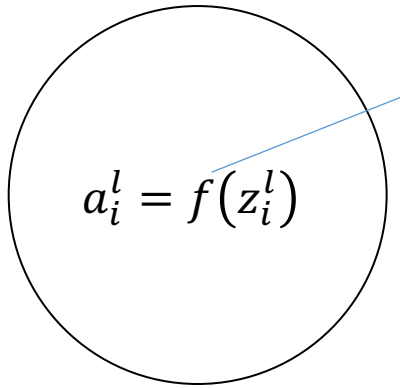
Step 2: 网络结构设计

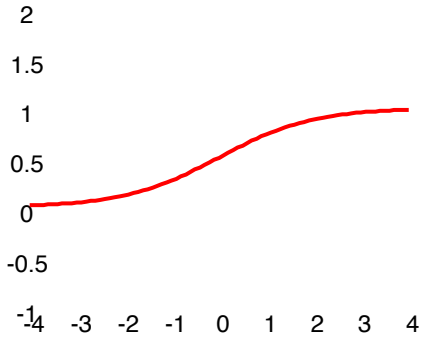
定义每层神经元的数量



Step 2: 网络结构设计

定义激活函数

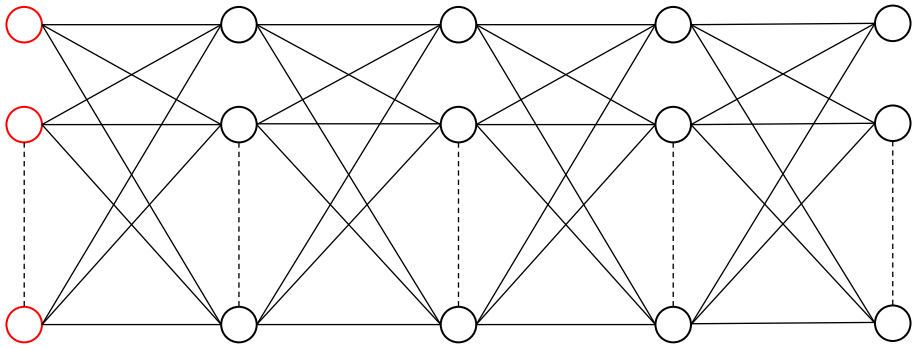




$$f(z) = \frac{1}{1+e^{-z}}$$

Sigmoid 函数

784-dimension



第 1 层 第 2 层 第 3 层 第 4 层 第 5 层

784 256 128 64 10

w^1 w^2 w^3 w^4

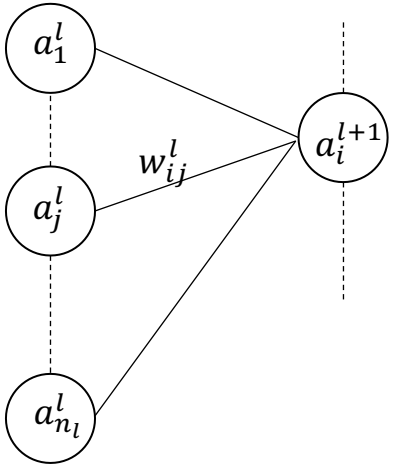
256×784 128×256 64×128 10×64

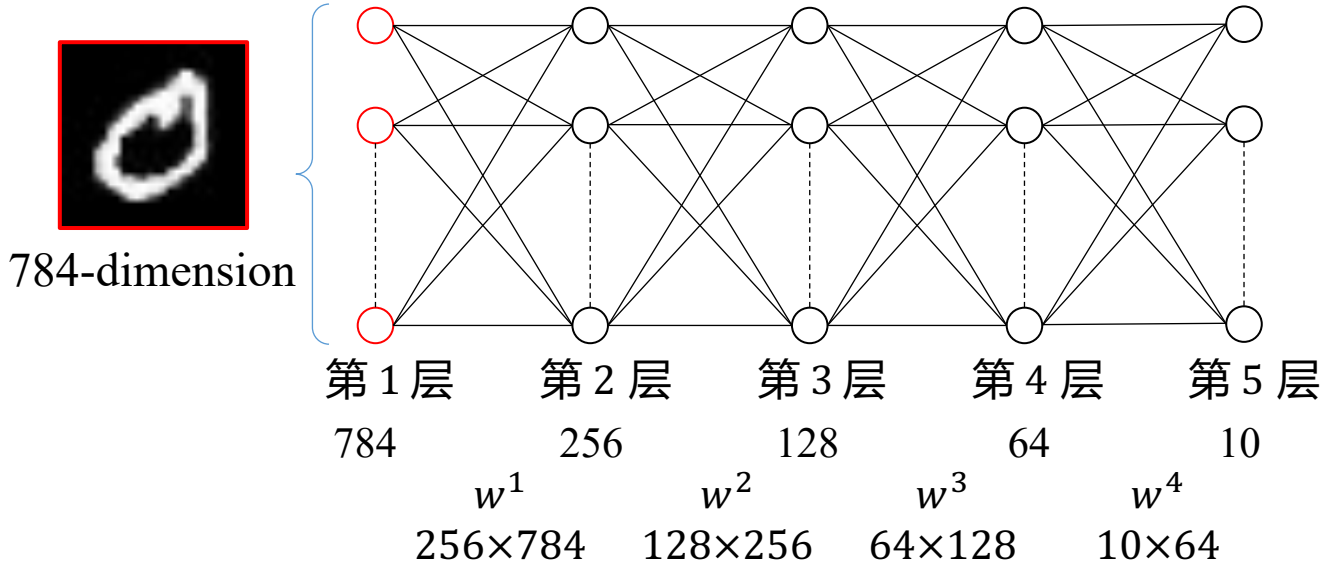
0	1	2	3	4	5	6	7	8	9
$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}$

Step 3: 初始化网络参数

初始化连接权

随机初始化：
高斯分布: $w_{ij}^{l} \sim N(0,1)$



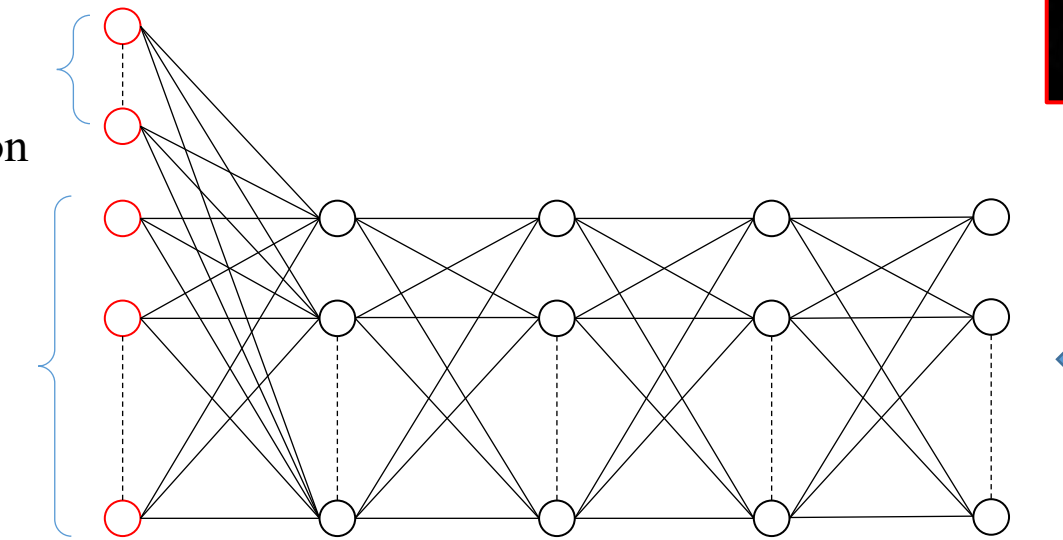


Step 3: 初始化网络参数

初始化初始外部输入神经元值

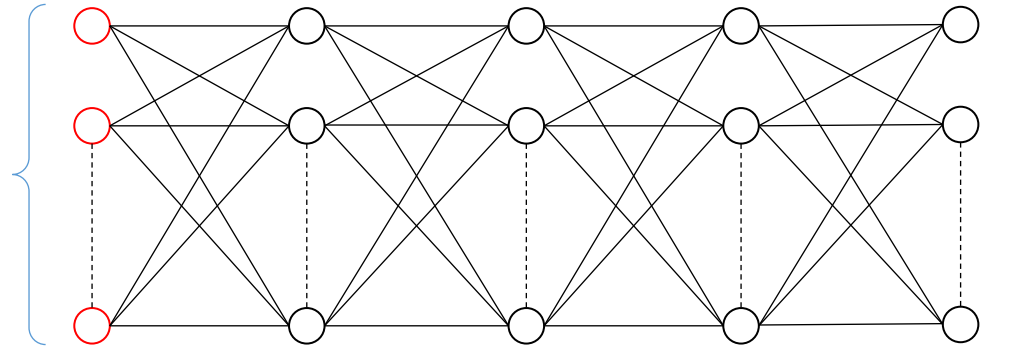
784-dimension

$a_i^1 = 0$



第 1 层 第 2 层 第 3 层 第 4 层 第 5 层

(784, 0) (0, 256) (0, 128) (0, 64) (0, 10)



第 1 层 第 2 层 第 3 层 第 4 层 第 5 层

784 256 128 64 10

w^1 w^2 w^3 w^4

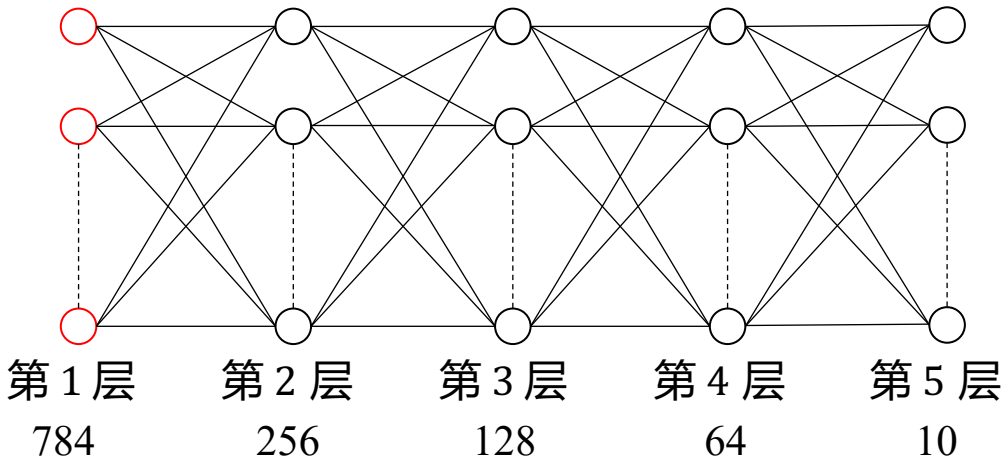
256×784 128×256 64×128 10×64

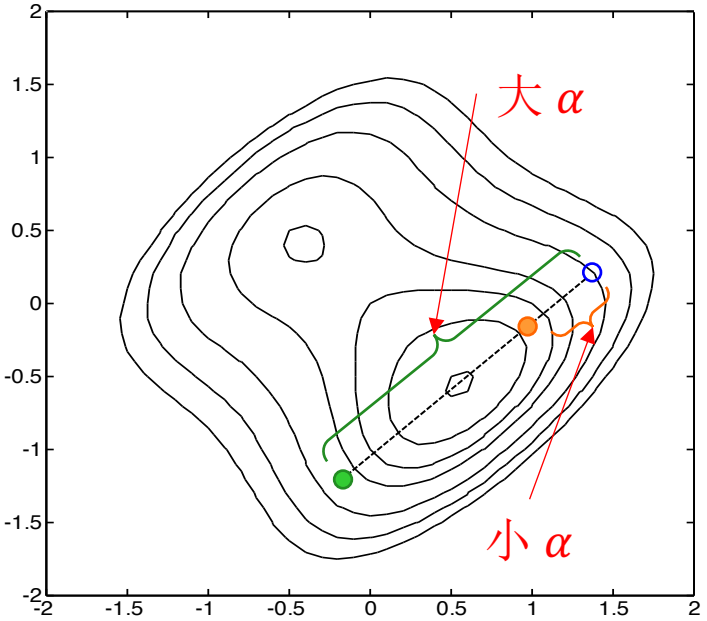
Step 3: 初始化网络参数

学习率：

- 小：学习慢
- 大：学习快，可能不会收敛到最小值

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}} \qquad \alpha = \cdots, 0.005, 0.01, 0.02, 0.04, \cdots$$



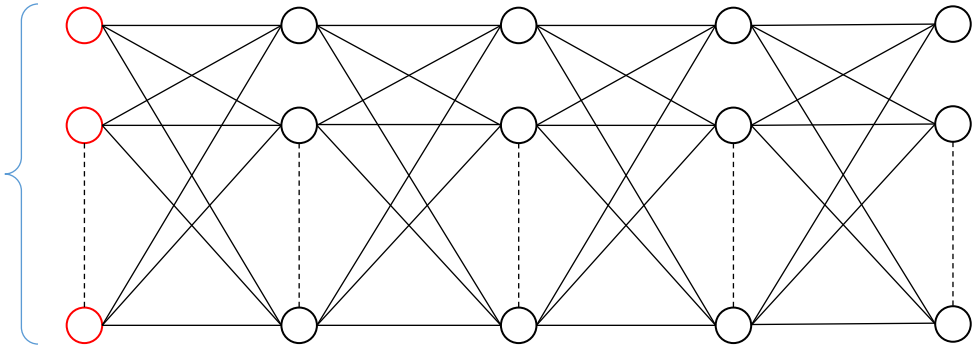


Step 4: 定义性能函数

性能函数

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2$$

网络预测



$$e_j = a_j^L - y_j^L$$

目标输出

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

标签

0

表征

Step 5: 定义网络评价指标

$$\text{Acc}=\frac{\text{正确预测样本数}}{\text{样本总数}}$$

一个样例

测试数据 

预测 7 9 0 4 8 8 8 3 1

正确预测 7

错误预测 2

$$\text{Accuracy}=\frac{7}{9}=77.78\%$$

在训练数据测试：

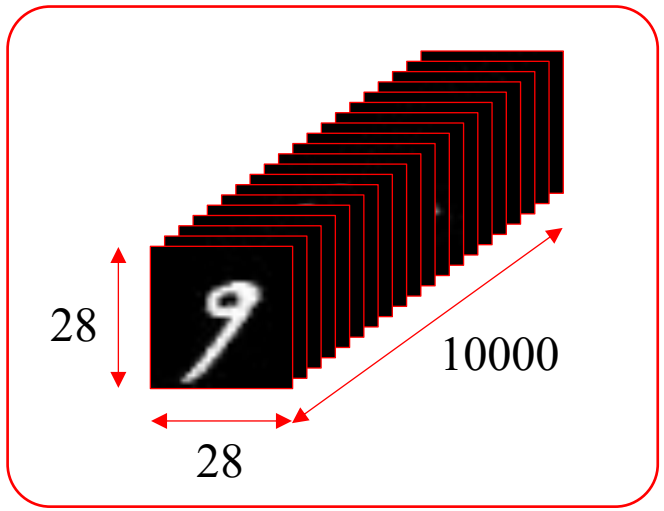
- 评估模型拟合给定数据的能力。

在测试数据测试：

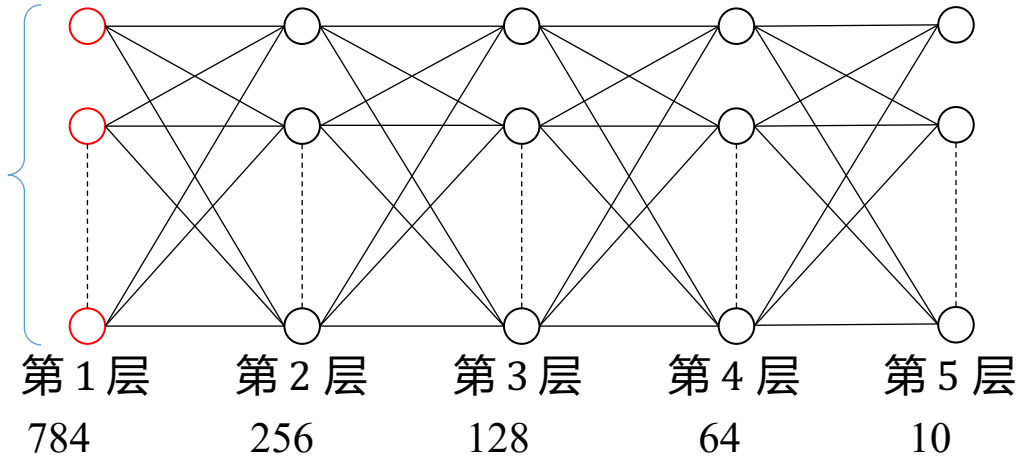
- 评估模型泛化知识的能力。

Step 6：训练网络

训练数据



前向计算a^l



反向传播 δ^l

更新连接权

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$$

BP算法

Step 1. Input the training data set $D = \{(x, y^L)\}$

Step 2. Initialize each w_{ij}^l, and choose a learning rate α

Step 3. for each mini-batch sample $D_m \subseteq D$

$\nabla J_{ji} = 0;$

for each $(x, y^L) \in D_m$

$a^1 \leftarrow x \in D_m;$

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l);$

end

$J(x, y^L);$

$\delta^L = \frac{\partial J(x, y^L)}{\partial z^L};$

for $l = L - 1: 1$

$\delta^l \leftarrow bc(w^l, \delta^{l+1});$

end

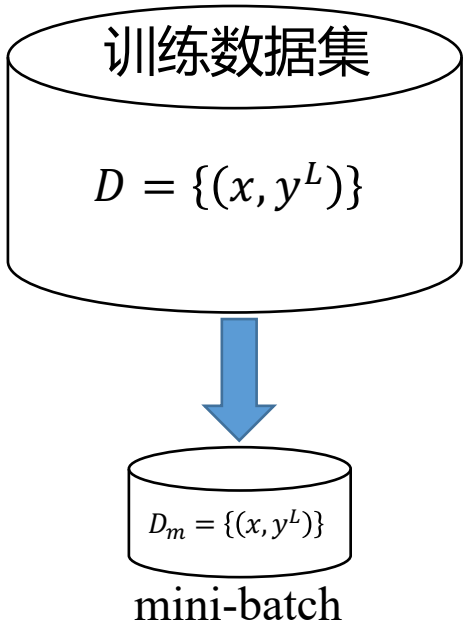
$\nabla J_{ji} \leftarrow \nabla J_{ji} + \delta_j^{l+1} \cdot a_i^l;$

end

$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \nabla J_{ji};$

end

Step 4. Return to Step 3 until each w_{ji}^l converge



function $fc(w^l, a^l)$

$for\ i = 1: n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$$

function $bc(w^l, \delta^{l+1})$

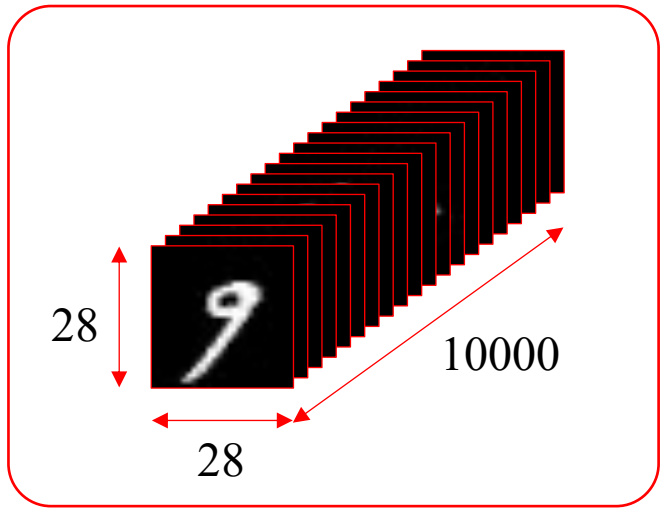
$for\ i = 1: n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

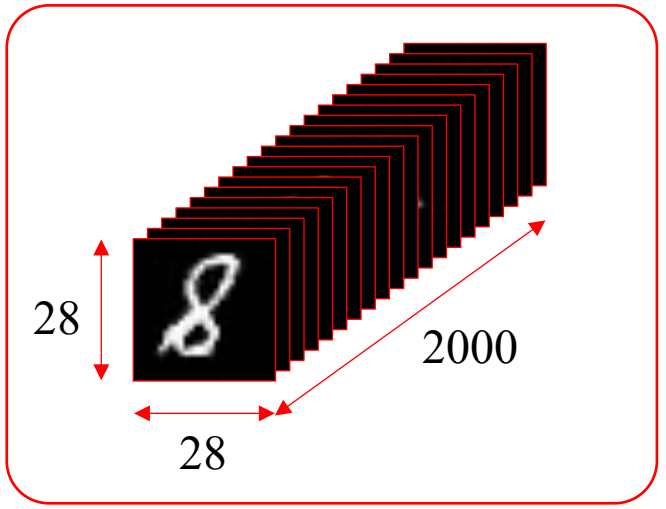
end

Step 7: 测试网络

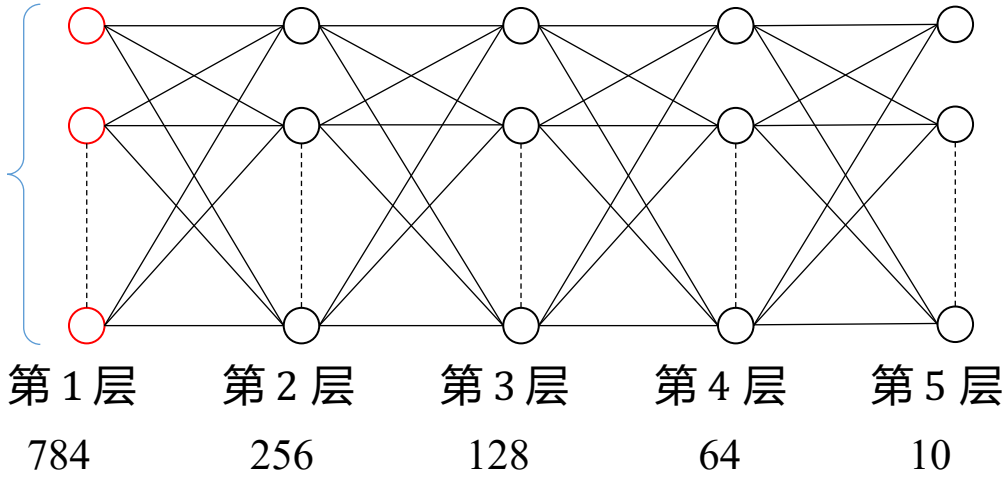
训练数据



测试数据



前向计算 a^l



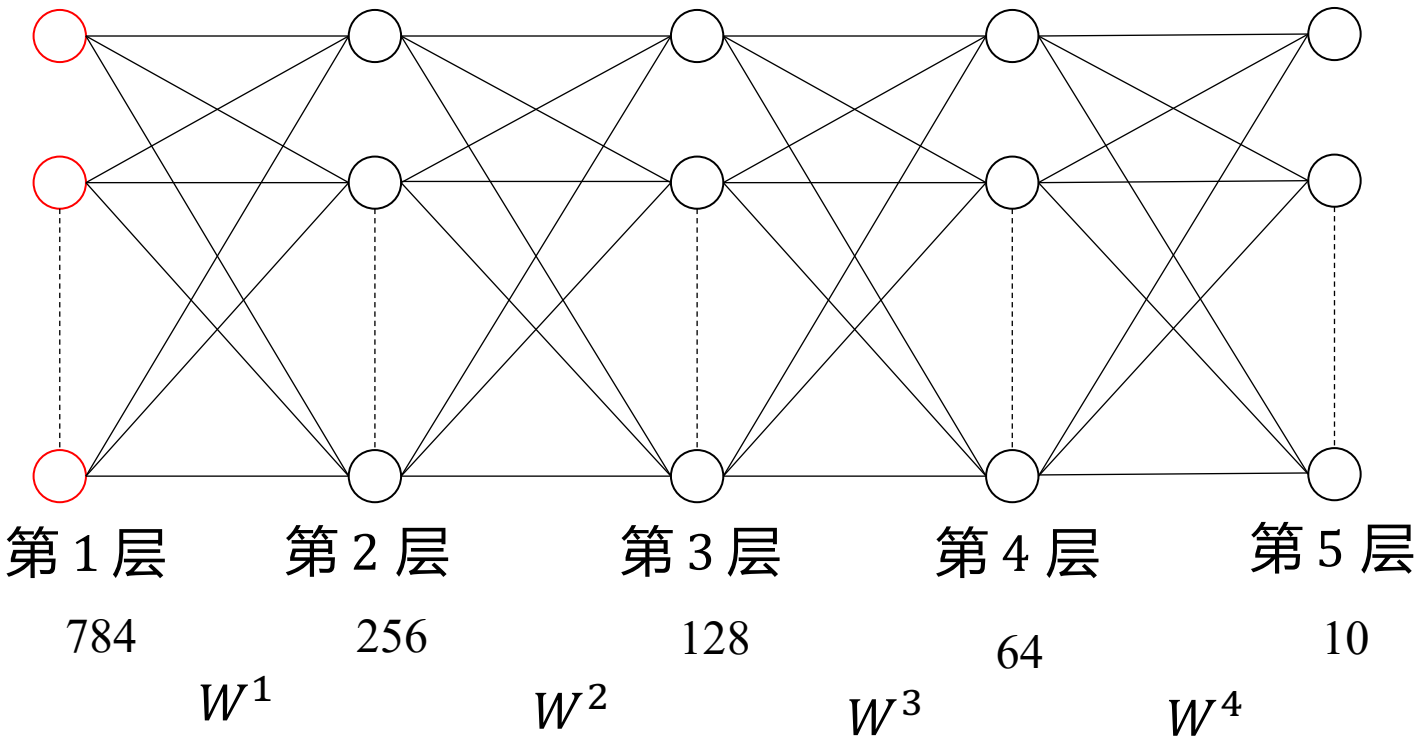
计算在训练集的评估指数

$$\text{Acc} = \frac{\text{正确预测样本数}}{10000}$$

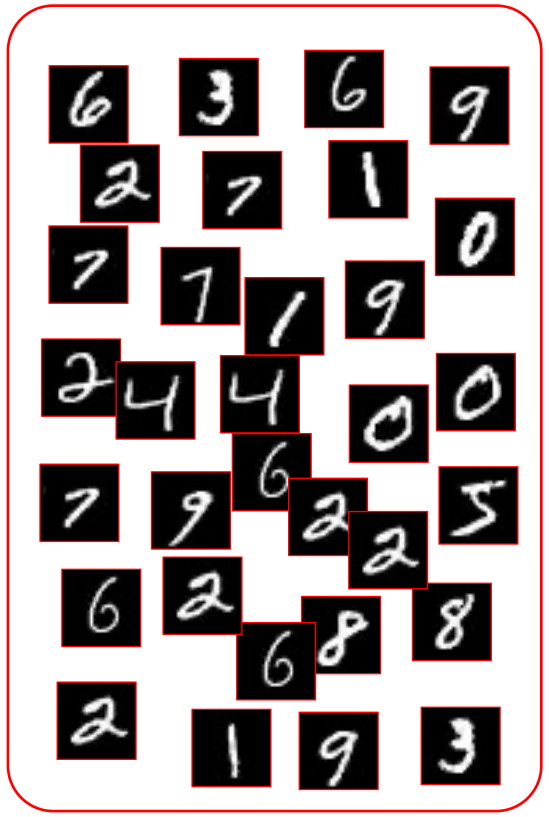
计算在测试集的评估指数

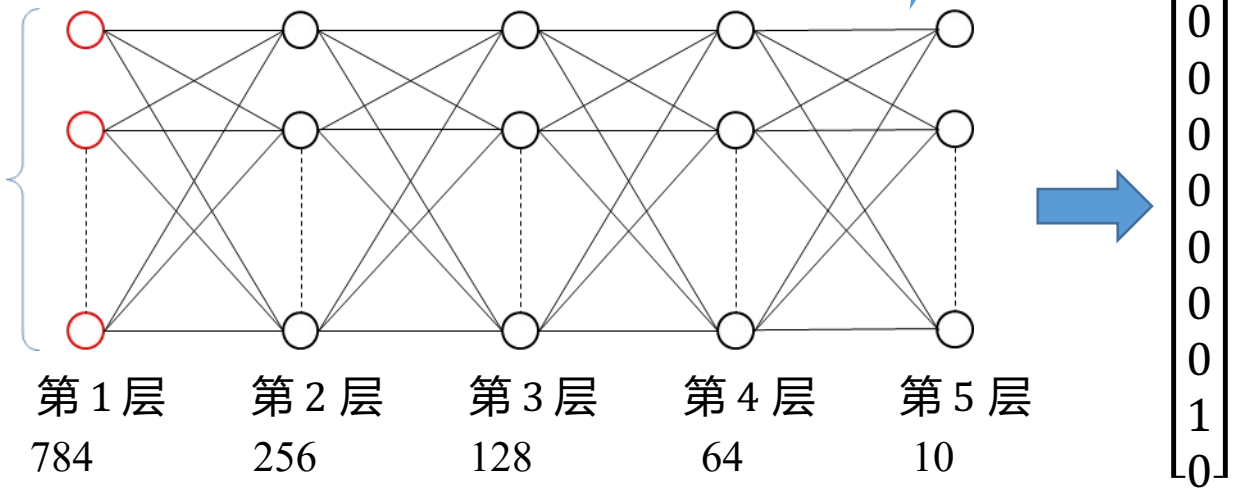
$$\text{Acc} = \frac{\text{正确预测样本数}}{2000}$$

Step 8: 存储网络



Step 9: 实际应用





提纲

神经网络训练

神经网络训练 - 实例

神经网络训练 - 问题

提纲

神经网络训练 - 问题

I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

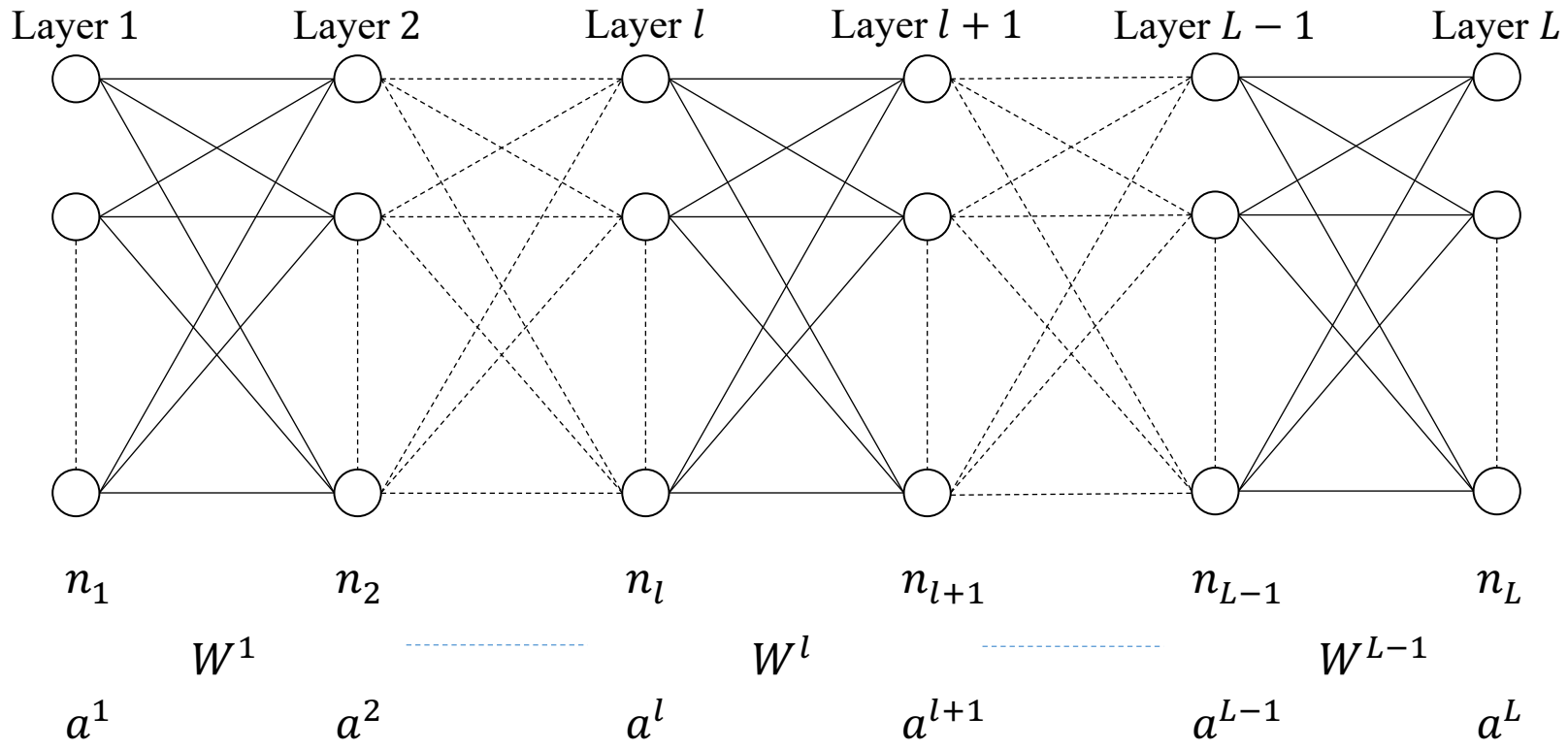
II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

网络结构问题

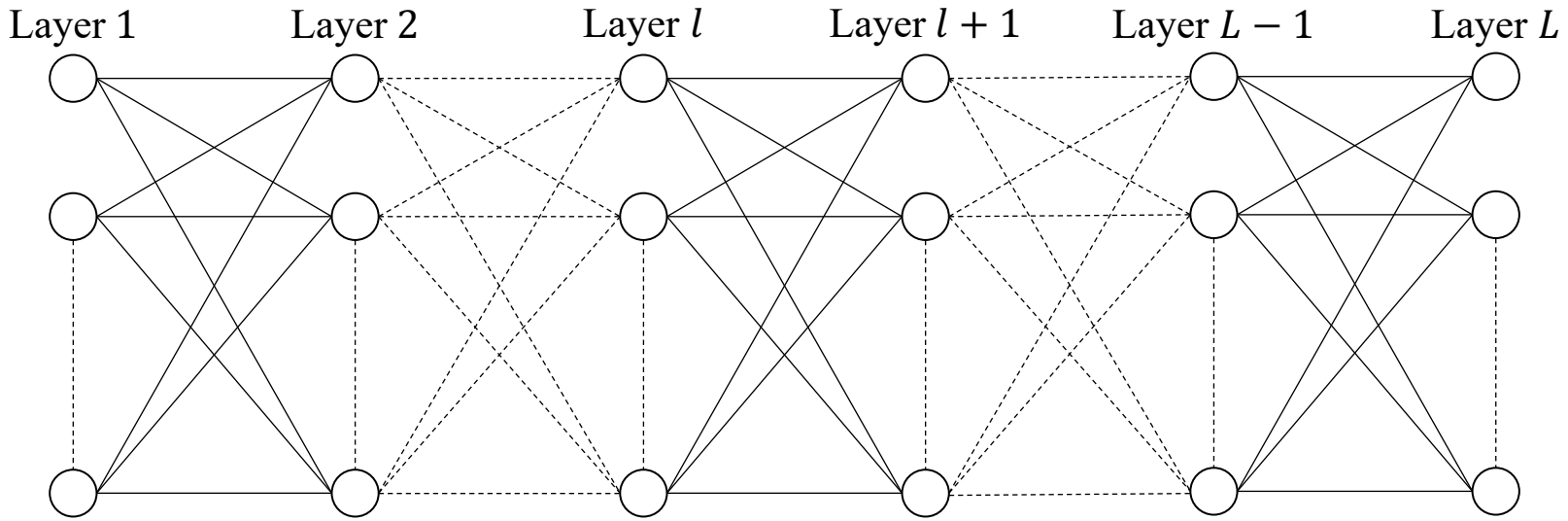
两大重要特征：

- 同层神经元间没有连接
- 跨层神经元间没有连接



网络结构问题

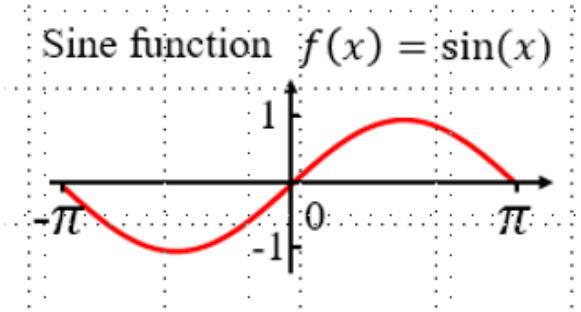
- 回复神经网络：
  - 每层神经元数量相同
  - 连接权矩阵共享



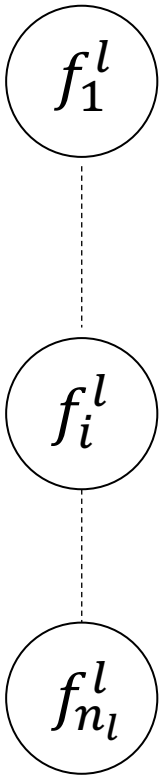
$$a^{l+1} = f(Wa^l)$$

网络结构问题

网络中每个神经元激活函数可以是不相同

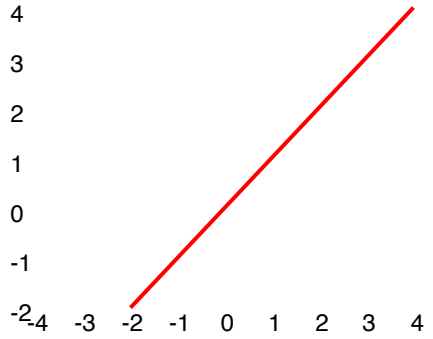


Layer l



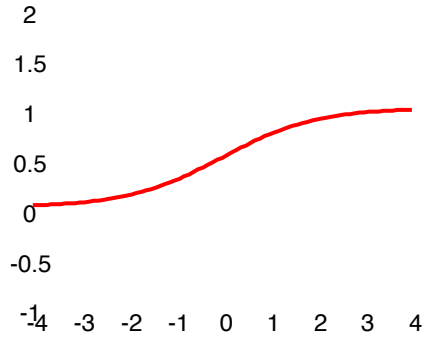
Linear function

$$f(z) = z$$



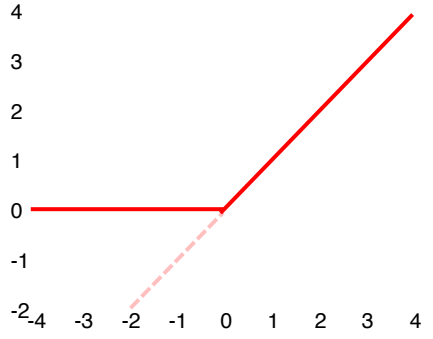
Sigmoid function

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



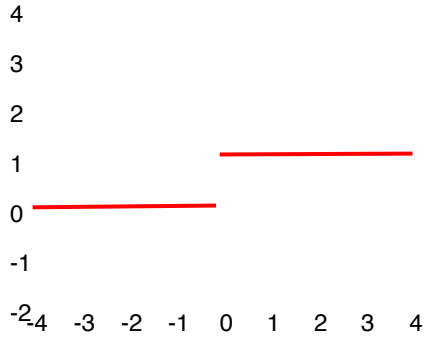
Rectifier function

$$f(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$



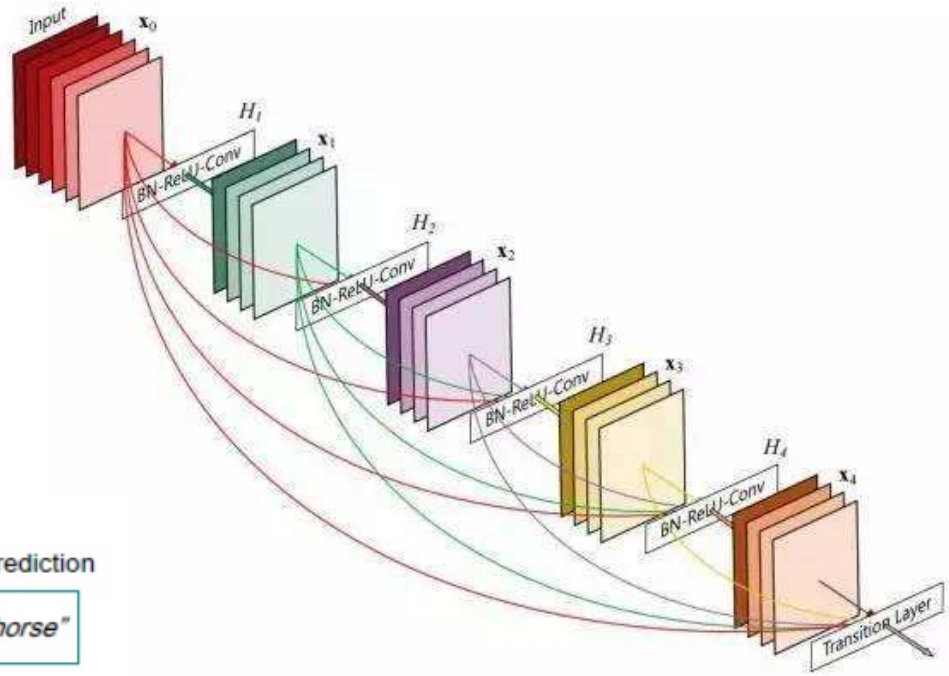
Hard-limit function

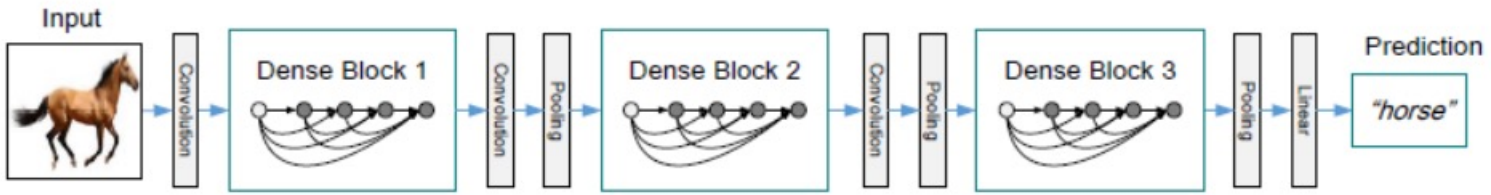
$$f(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

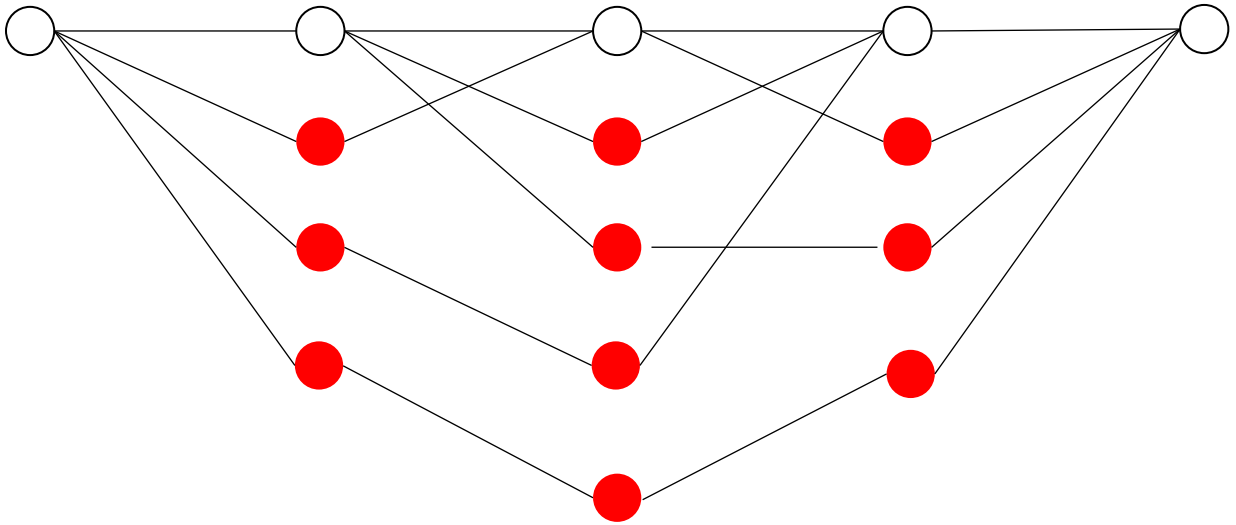


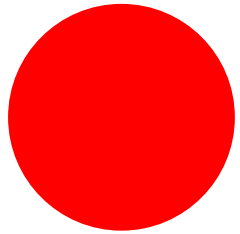
网络结构问题

DenseNets







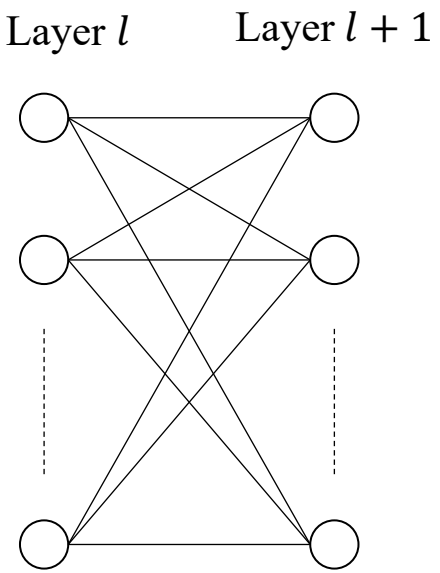


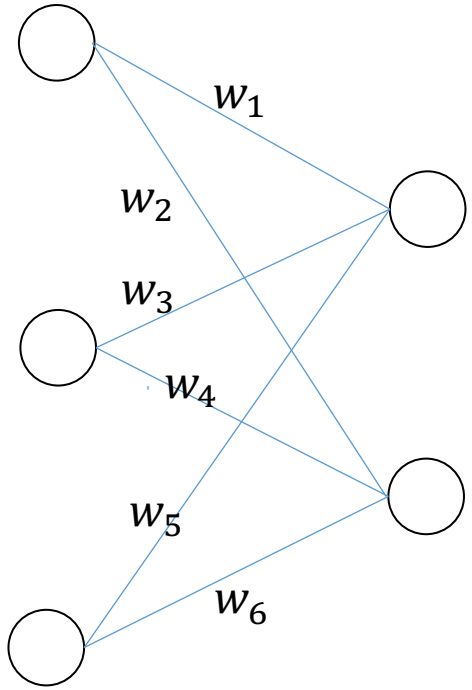
Linear neuron
$f(s) = s$

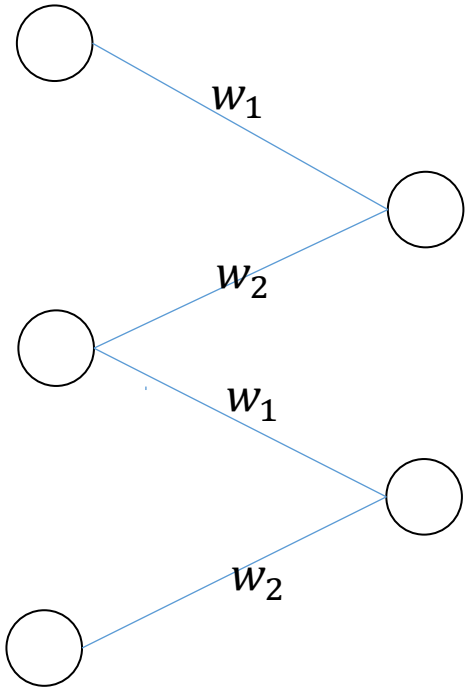
网络结构问题

卷积神经网络

相邻层之间的神经元共享部分连接权





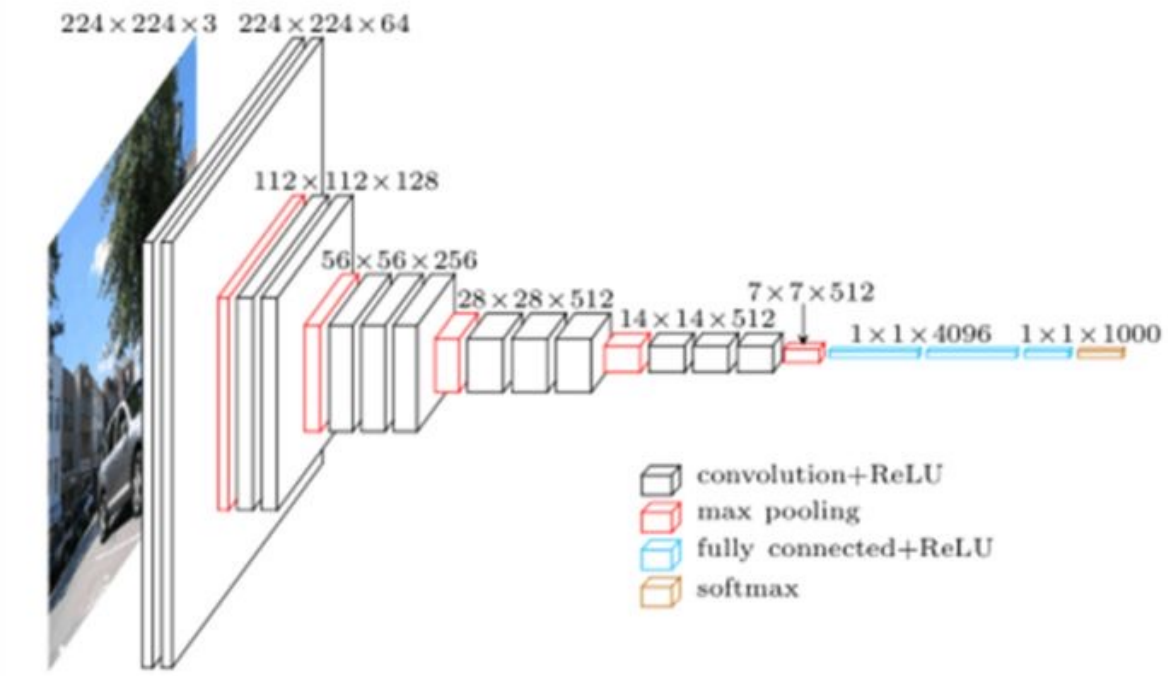


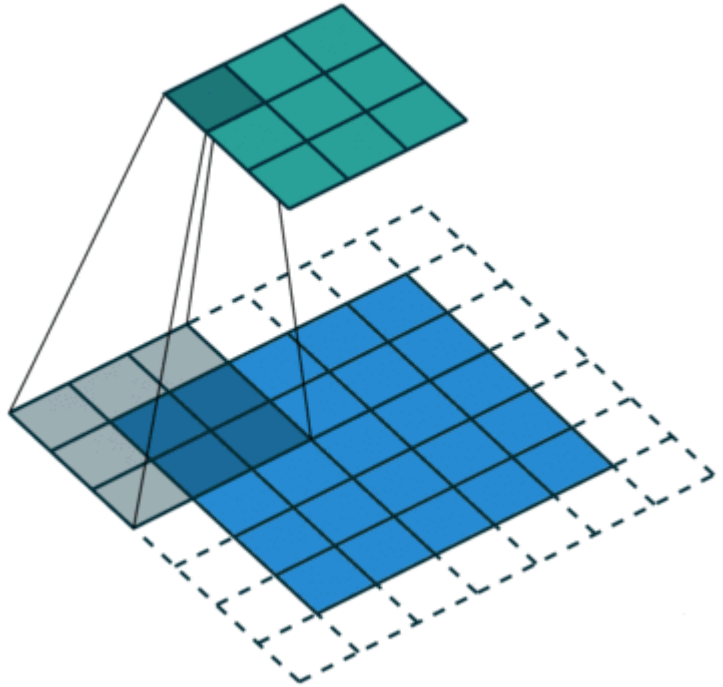
$$W^l = \left(w_{ij}^l\right)_{n_{l+1}\times n_l}$$

网络结构问题

卷积神经网络

相邻层之间的神经元共享部分连接权





提纲

神经网络训练 - 问题

I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

学习算法问题

梯度下降算法

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$$

$$\frac{\partial J}{\partial w_{ji}^{l}} = \delta_j^{l+1} \cdot a_i^{l}$$

BP算法

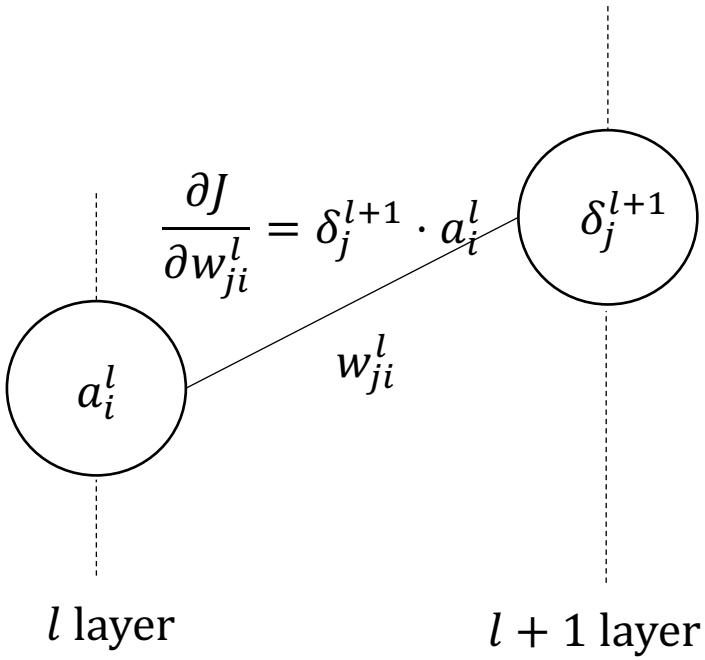
$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \left(\delta_j^{l+1} \cdot a_i^{l}\right)$$

$$\delta_j^{l+1} = \frac{\partial J}{\partial z_j^{l+1}}$$

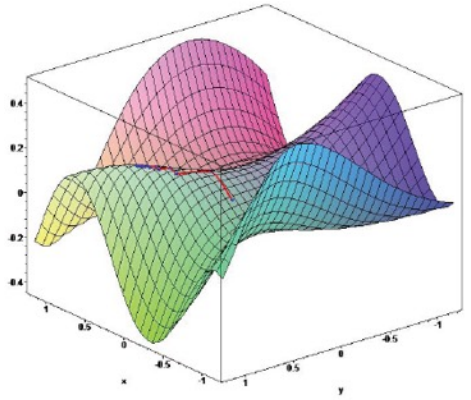
$$a_i^{l} = f\left(z_i^{l}\right)$$

BP算法

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \left(\frac{\partial J}{\partial z_j^{l+1}}\right) \cdot f\left(z_i^{l}\right)$$



$$J = J\left(\cdots, w_{ji}^{l}, \cdots\right)$$



问题：

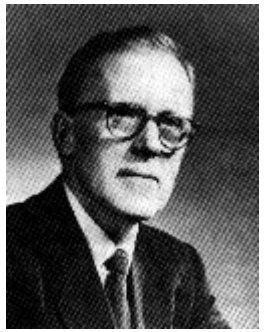
BP算法有没有神经科学依据？

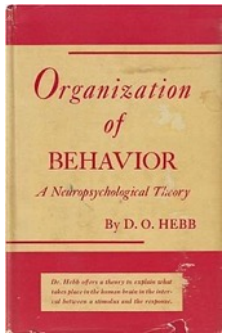
学习算法问题

Hebb假说：

当神经元A的轴突足够接近到能够激发神经元B，且反复或持续地刺激神经元B，那么A或B中的一个或两个神经元将会产生某种增长过程或代谢变化，从而增强神经元A对神经元B的刺激效果。

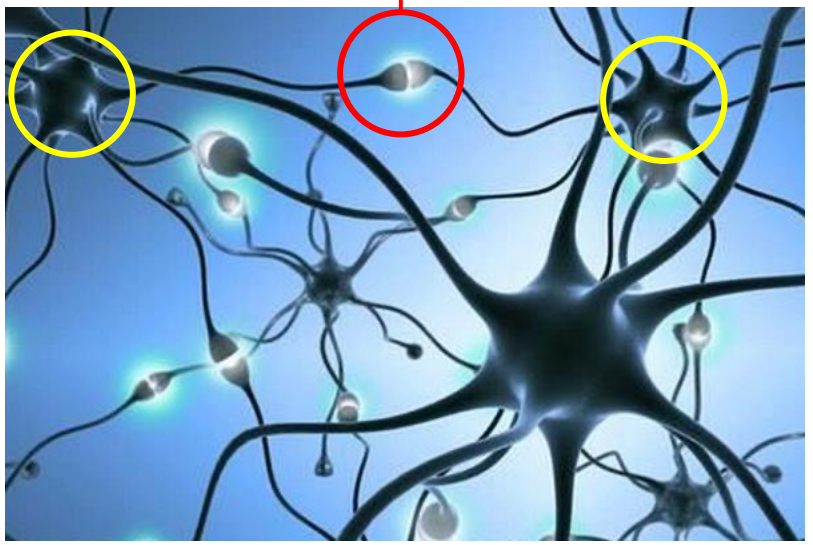
——D.O Hebb，1949





D. O. Hebb
认知心理生物学之父
1904-1985

突触



学习算法问题

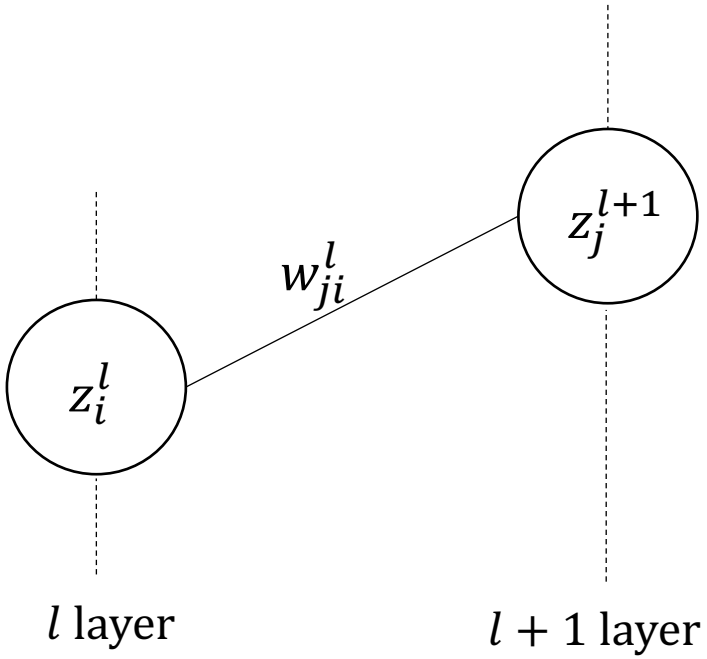
当神经元A的轴突足够接近到能够激发神经元B，且反复或持续地刺激神经元B，那么A或B中的一个或两个神经元将会产生某种增长过程或代谢变化，从而增强神经元A对神经元B的刺激效果。

如果连接突触两端的两个神经元同时被激活，这个突触的连接强度将会增强。

数学抽象

Hebbian 学习规则

$$w_{ji}^{l} \leftarrow w_{ji}^{l} + F\left(z_j^{l+1}, z_i^{l}\right)$$



学习算法问题

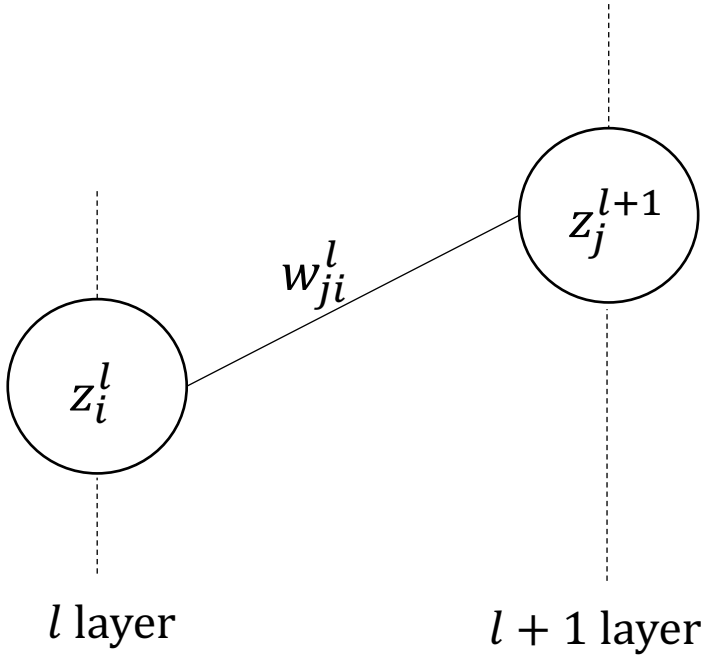
如果连接突触两端的两个神经元同时被激活，这个突触的连接强度将会增强。

Hebbian 学习规则

$$w_{ji}^{l} \leftarrow w_{ji}^{l} + F\left(z_j^{l+1}, z_i^{l}\right)$$

$$w_{ji}^{l} \leftarrow w_{ji}^{l} + \alpha \cdot f_j^{l+1}\left(z_j^{l+1}\right) \cdot f_i^{l}\left(z_i^{l}\right)$$

$$w_{ji}^{l} \leftarrow w_{ji}^{l} + \alpha \cdot a_j^{l+1} \cdot a_i^{l}$$



学习算法问题

如果连接突触两端的两个神经元同时被激活，这个突触的连接强度将会增强。

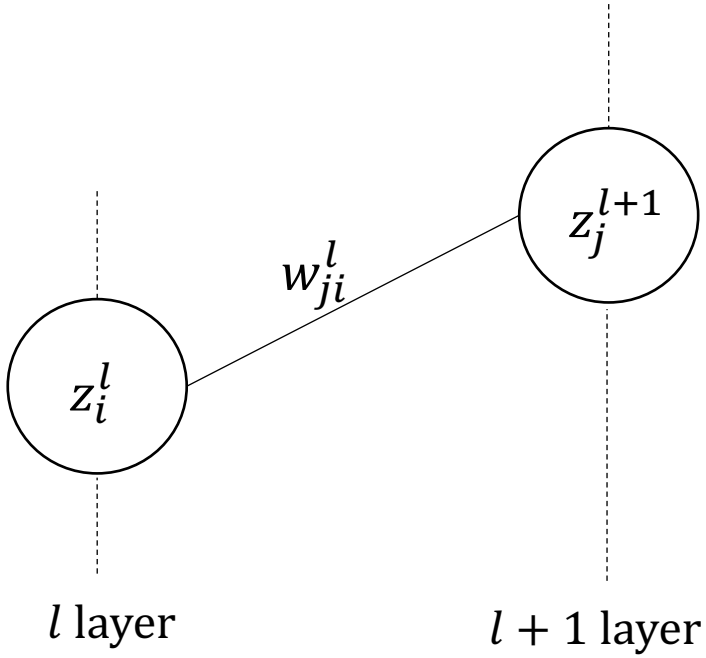
Hebbian 学习算法

$$w_{ji}^{l} \leftarrow w_{ji}^{l} + F\left(z_j^{l+1}, z_i^{l}\right)$$

BP 学习算法

$$F\left(z_j^{l+1}, z_i^{l}\right) = -\alpha \cdot \left(\frac{\partial J}{\partial z_j^{l+1}}\right) \cdot f\left(z_i^{l}\right)$$

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \left(\frac{\partial J}{\partial z_j^{l+1}}\right) \cdot f\left(z_i^{l}\right)$$



结论：BP算法是一种Hebb学习算法

提纲

神经网络训练 - 问题

I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

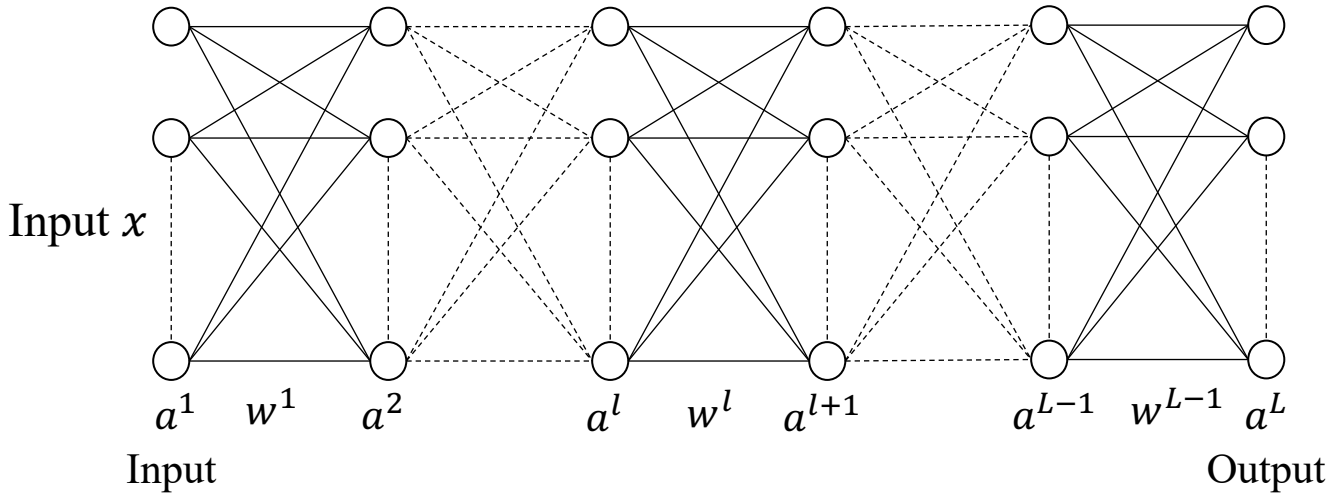
II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

目标输出问题

问题：如何定义目标输出？

原则上，目标输出的定义必须符合具体任务的要求。因此目标输出是从具体任务 / 应用中产生的。此外，目标输出必须和输入相对应。



定义在最后一层的目标输出

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix} \Longleftrightarrow \text{Input } x$$

一个训练样本 (x, y^L)

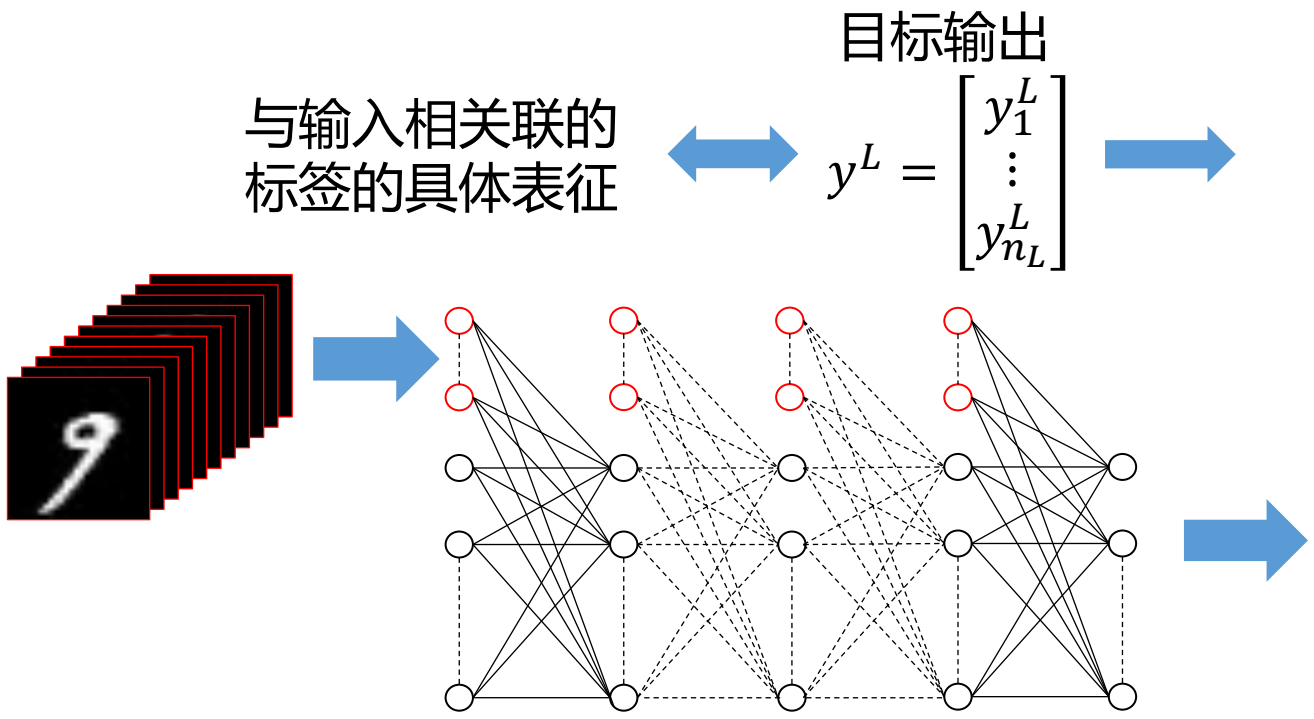
$$\dim(a^L) = \dim(y^L)$$

目标输出问题

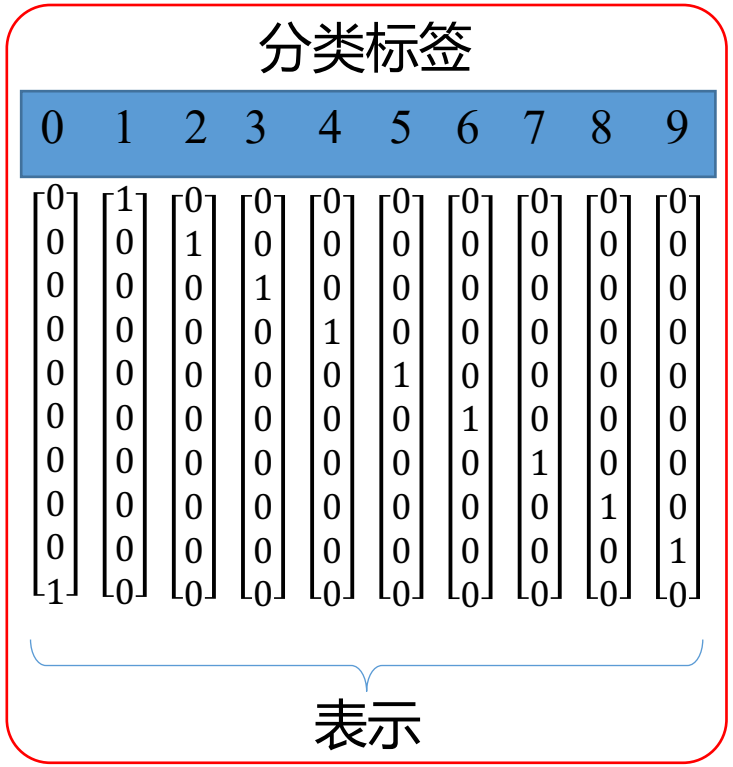
分类问题：

目标是将每个输入数据映射到它对应的分类标签，

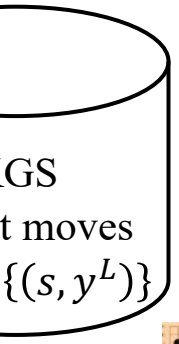
因此，目标输出定义为标签的表示。



Tip:
输出层神经元的数量等于类别数量。

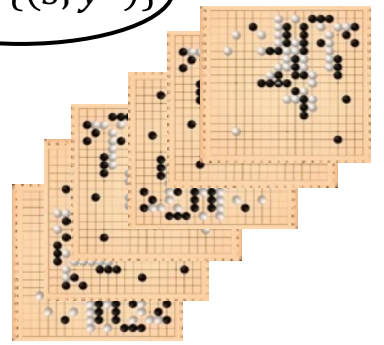


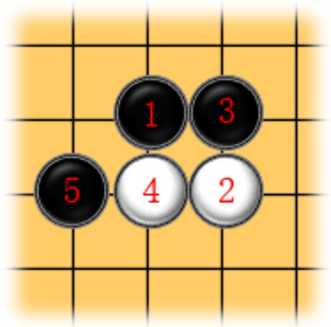
目标输出问题

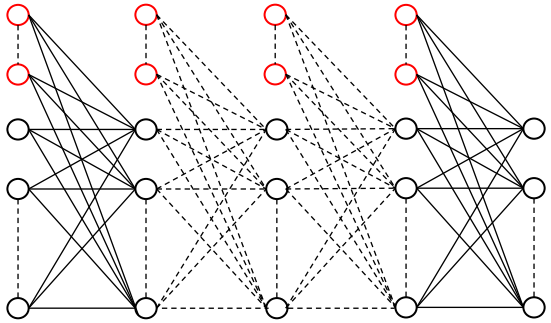


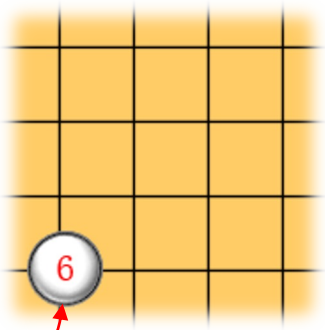
目标输出定义为专家落子向量

$$y^L = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$









当前状态 S

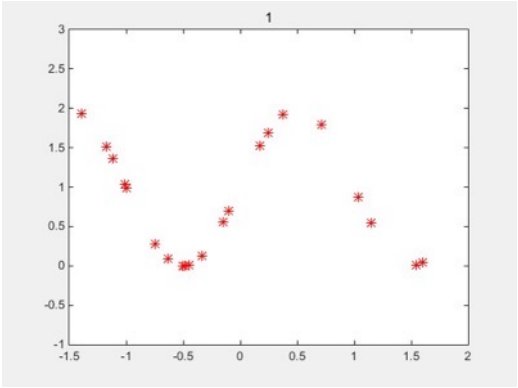
专家落子

后继状态

目标输出问题

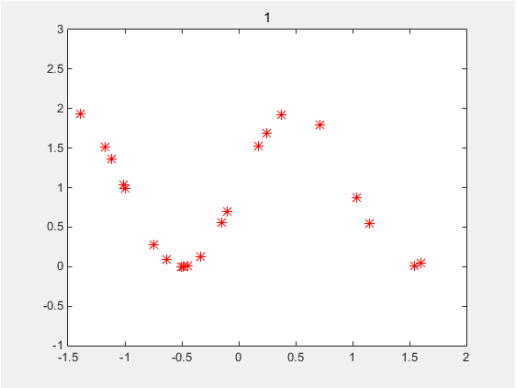
曲线拟合问题：

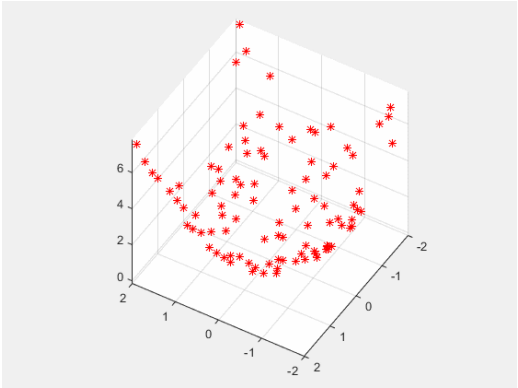
给定一组样本数据，来估计通过这些点的曲线。



采样数据

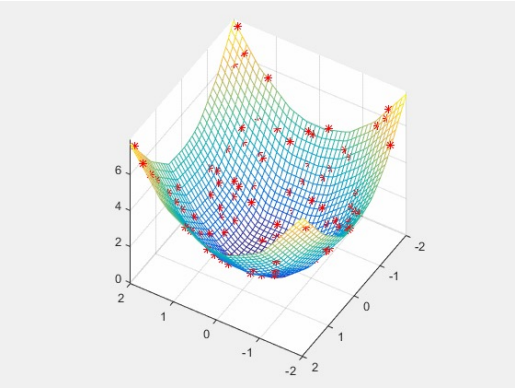
	1	2	3	4	5	6
x	-0.5000	0.1740	0.7100	-0.9980	-0.6340	1.0400
y	0	1.5198	1.7902	0.9937	0.0873	0.8747



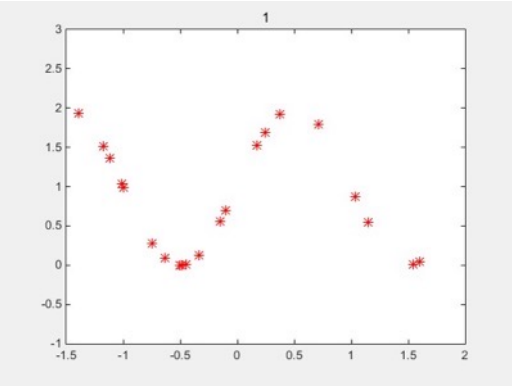


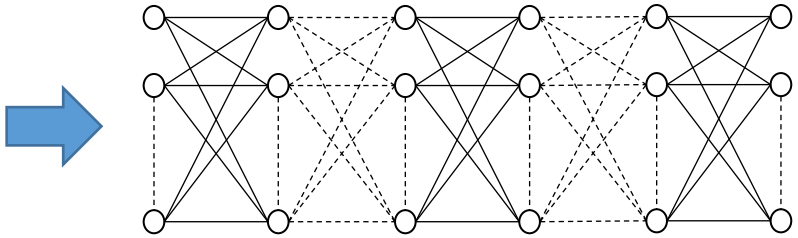
采样数据

	1	2	3	4	5	6
x	-0.2000	-1.9000	1.9000	0.4000	-1.9000	0.8000
	1.4000	-1.9000	-1.5000	-0.5000	0.3000	-0.1000
y	2.0000	7.2200	5.8600	0.4100	3.7000	0.6500

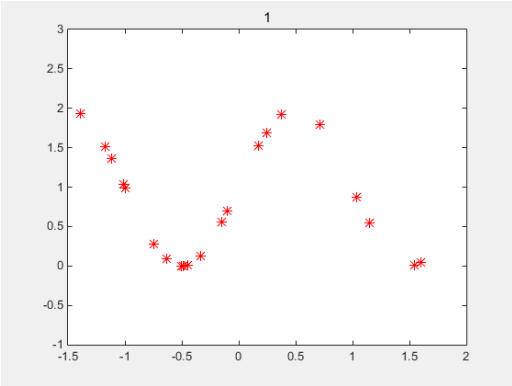


目标输出问题

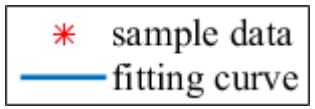


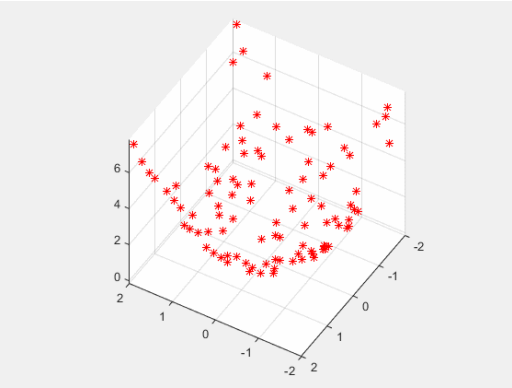


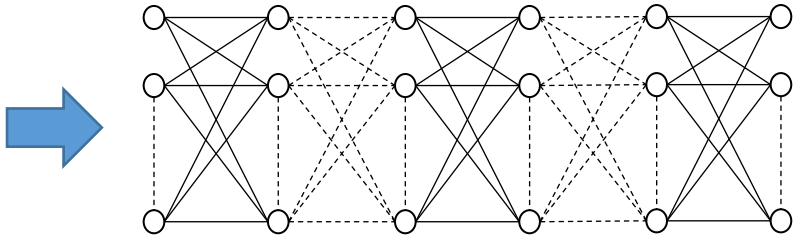
训练样本 (x, y)

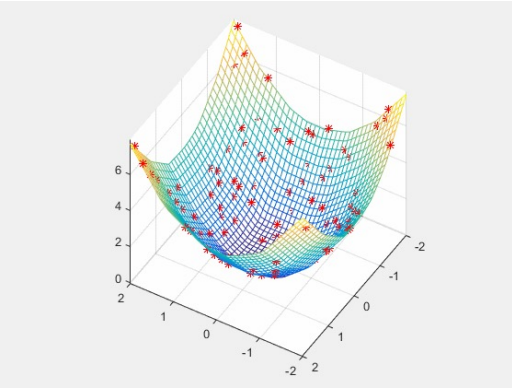


目标输出定义为每个样本x对应的y值，即， $y^L = y$





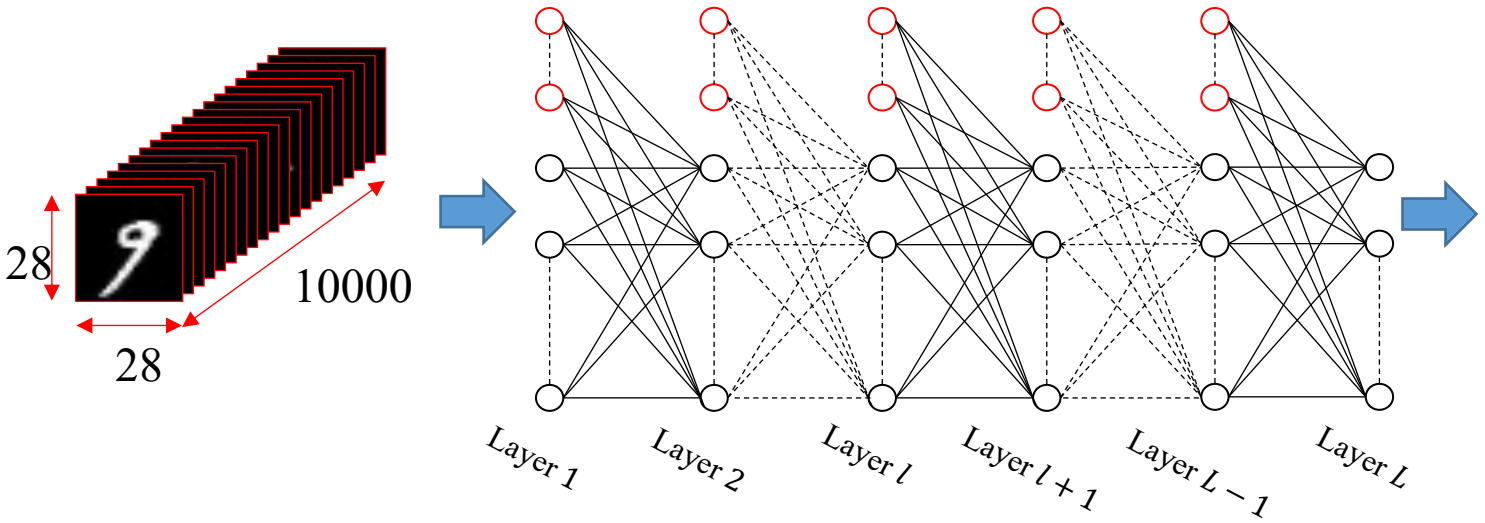


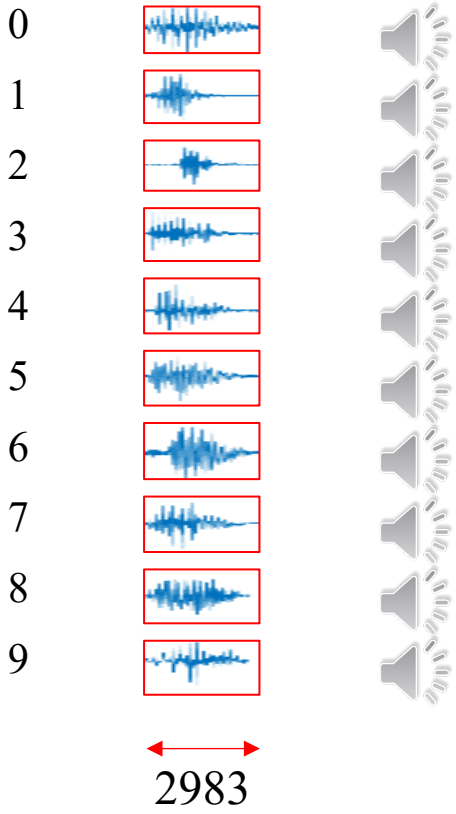


目标输出问题

目标输出定义为每个数字的语音向量

$$y^L = \begin{bmatrix} y_1^L \\ y_2^L \\ \vdots \\ y_{2983}^L \end{bmatrix}$$





提纲

神经网络训练 - 问题

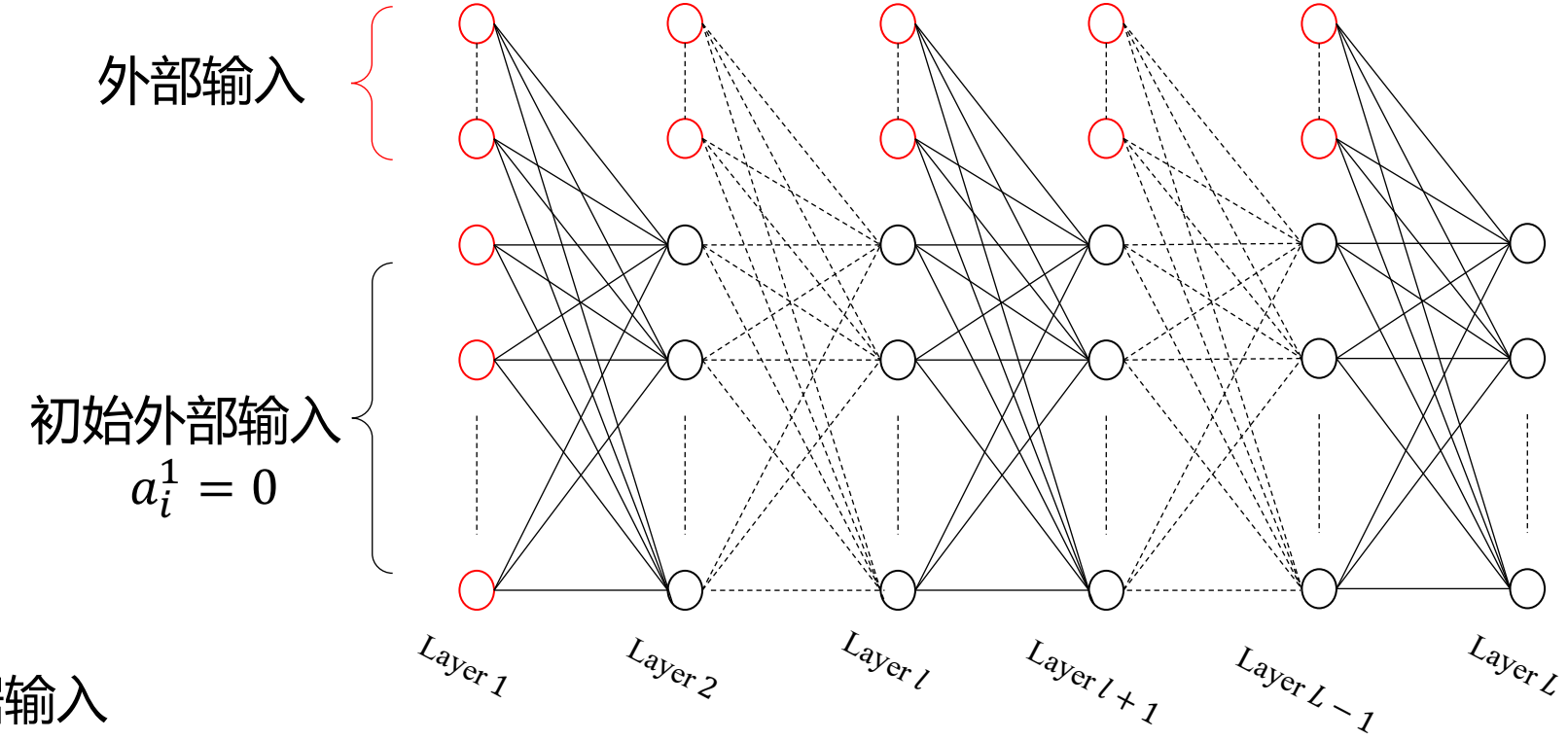
I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

网络输入问题

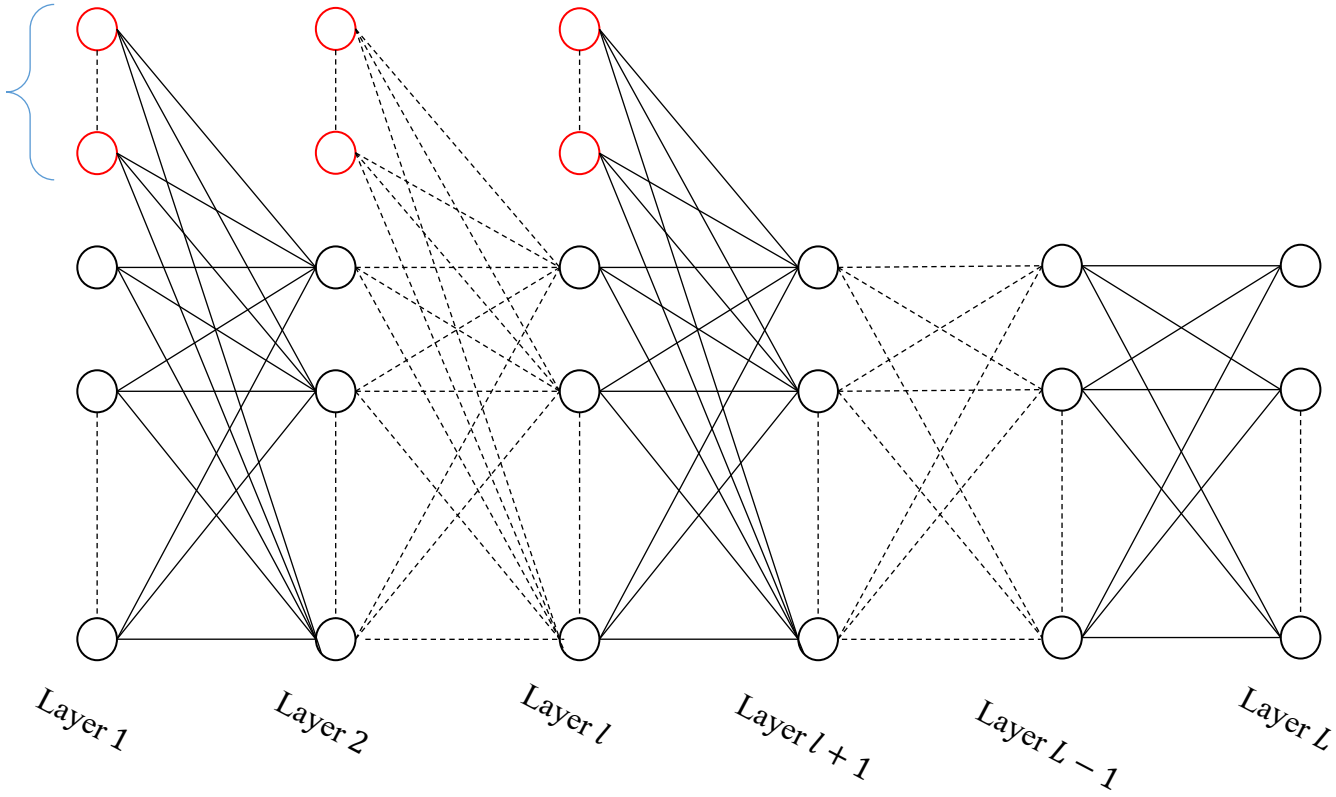


外部输入
- 直接从数据输入

初始外部输入
- 根据应用情况设置
- 许多情况下可以直接取0

网络输入问题

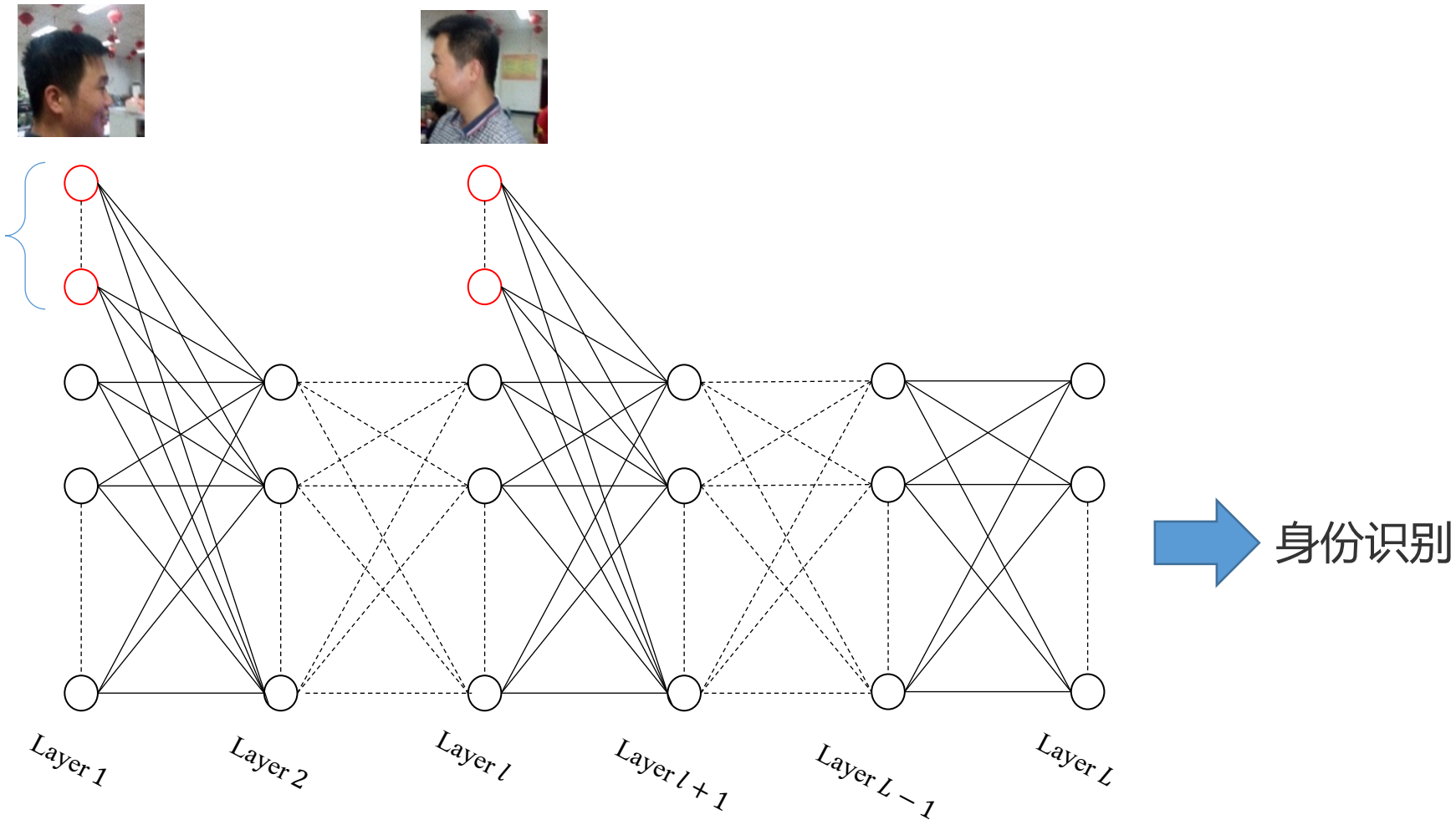
序列输入



身份识别

网络输入问题

序列输入

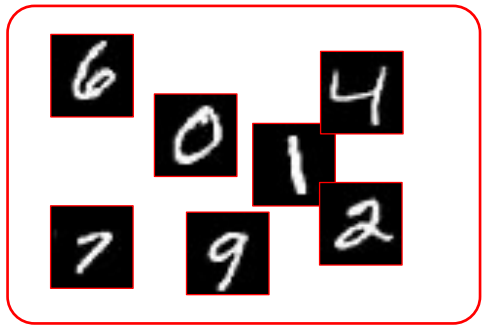


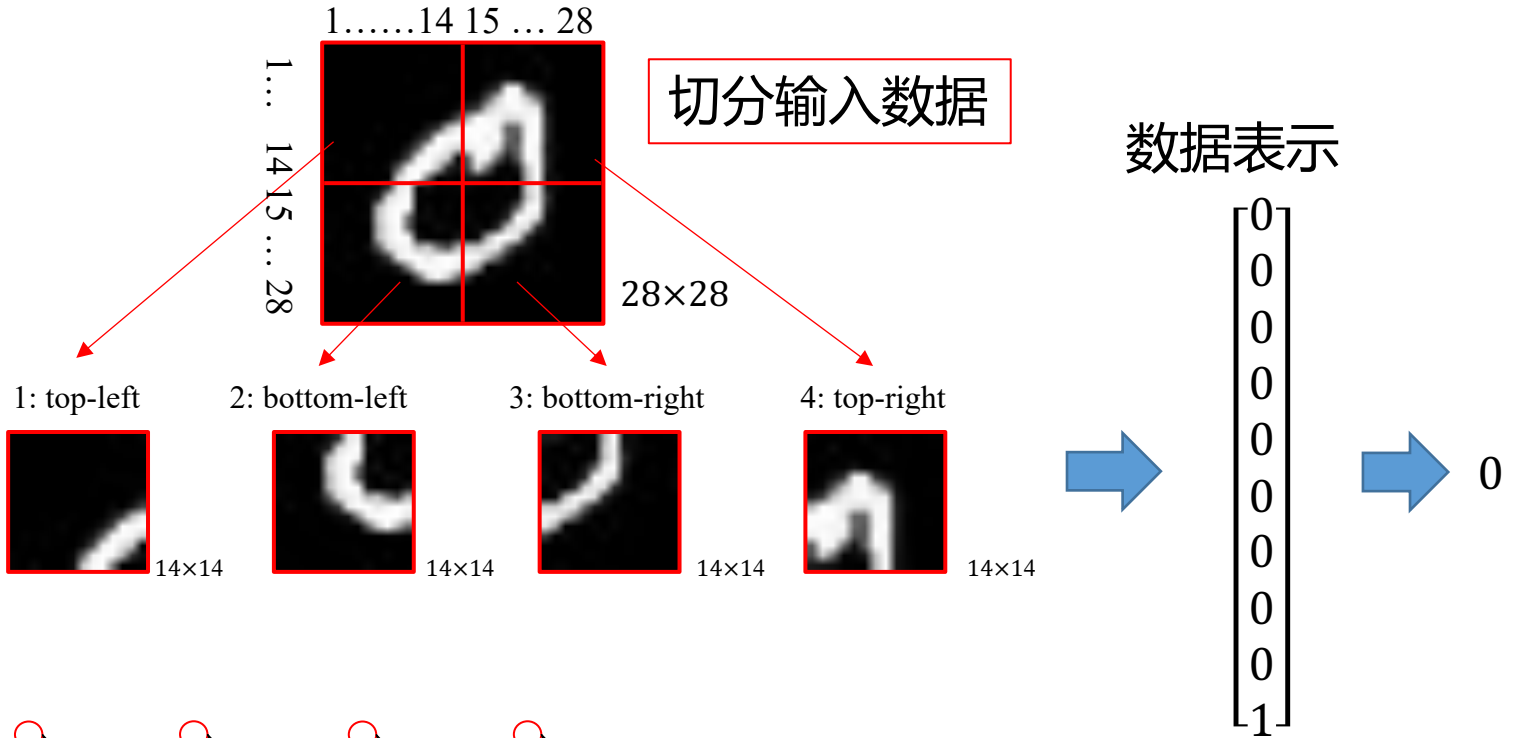
身份识别

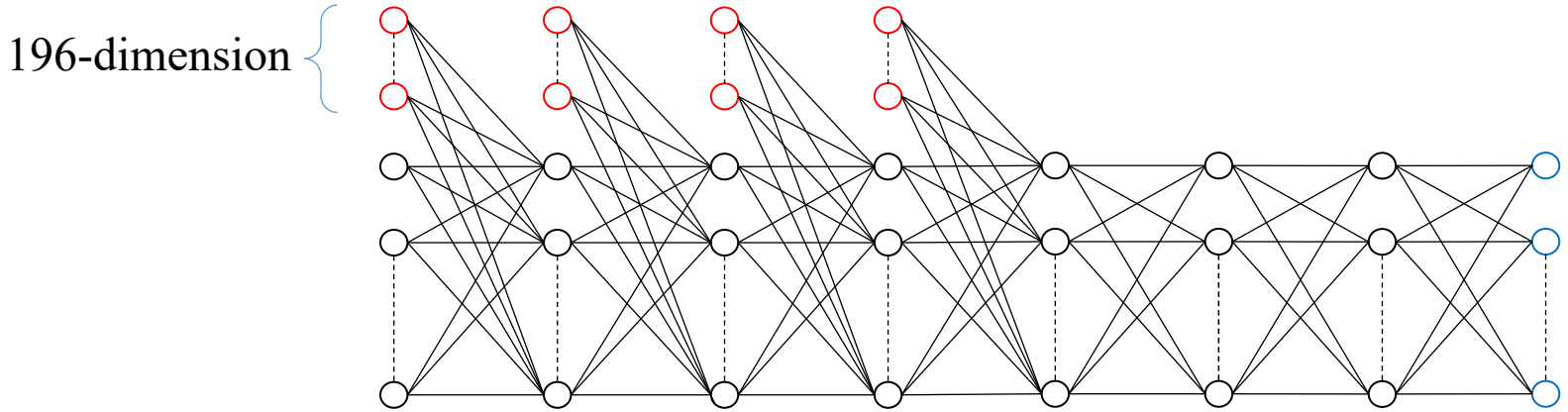
网络输入问题

分块输入

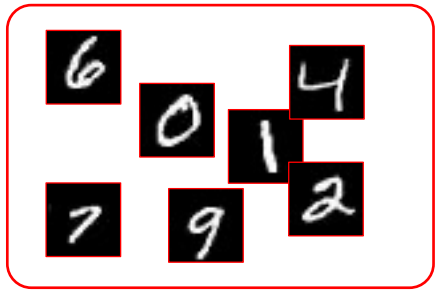
如果输入数据的维度太大，
可以将其分切分成小的维度



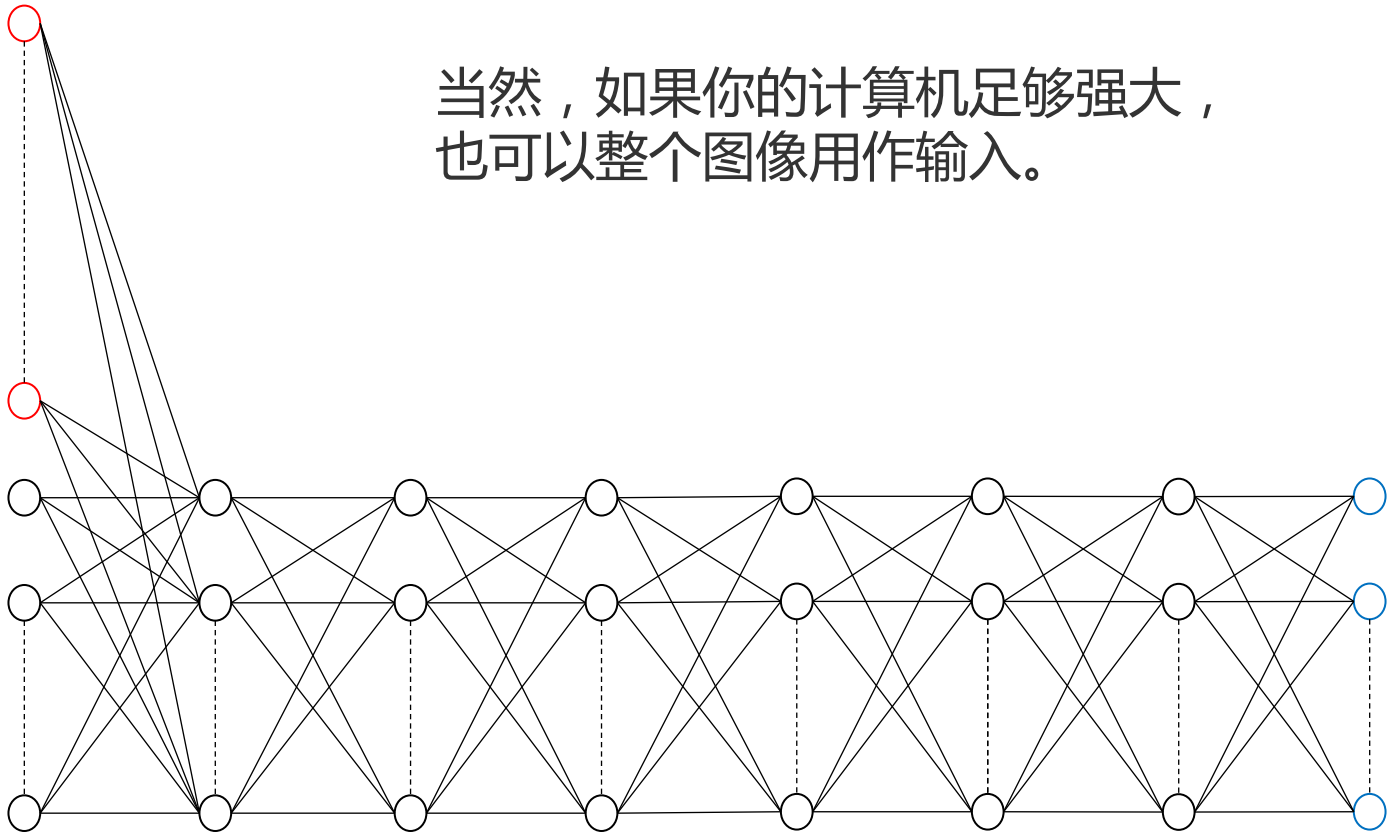




网络输入问题



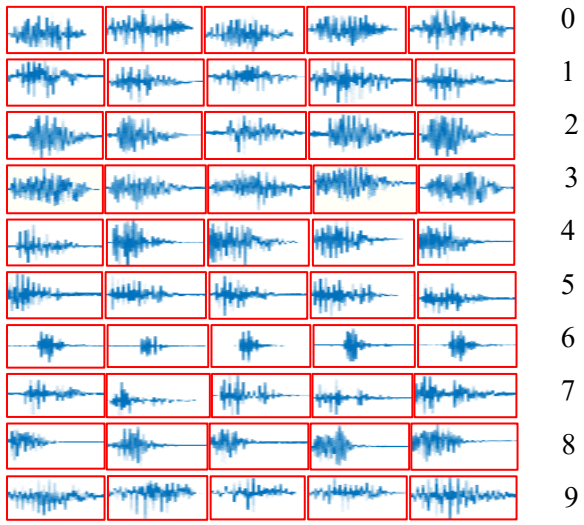
784-dimension



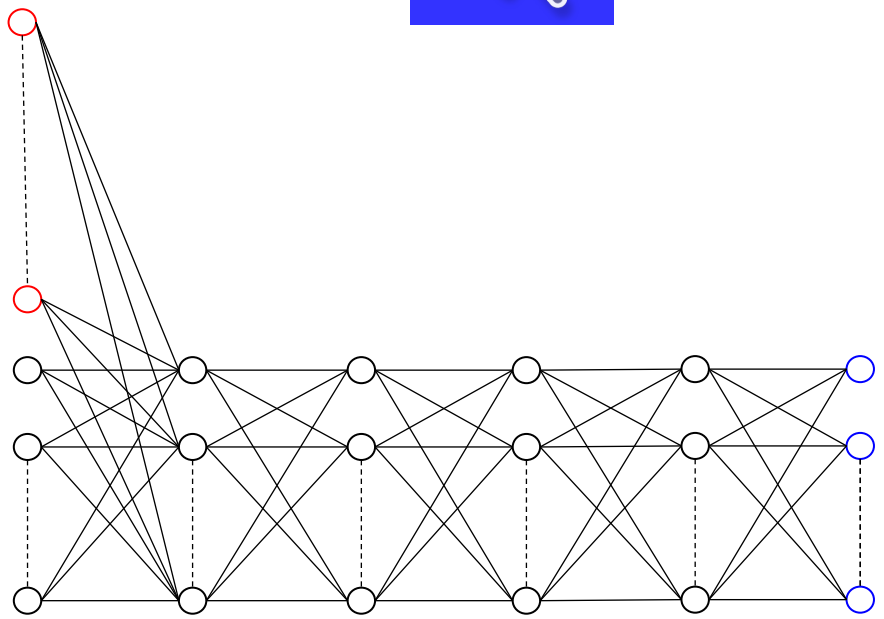
当然，如果你的计算机足够强大，
也可以整个图像用作输入。

网络输入问题

语音数据输入



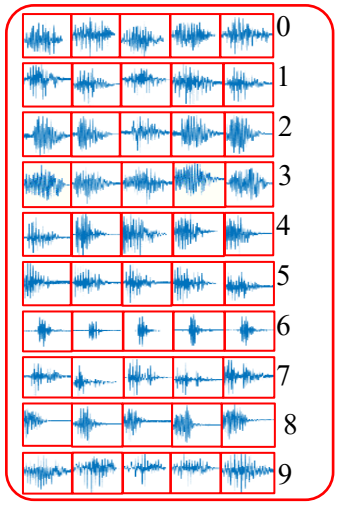
4000-dimension



3

$$\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

网络输入问题

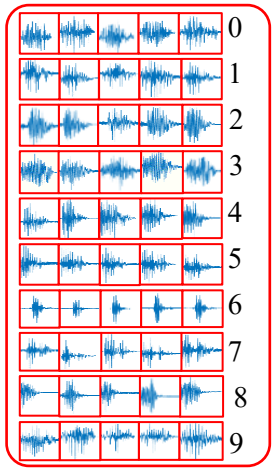


多模态数据输入

4000-dimension

784-dimension

网络输入问题

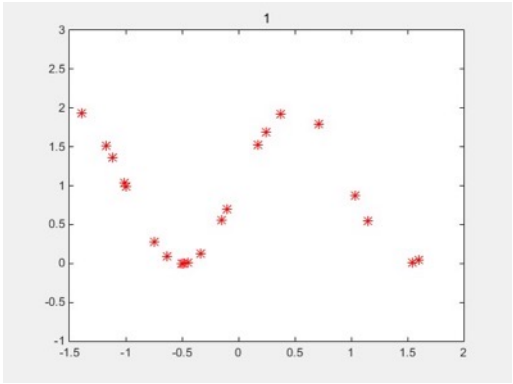


784-dimension

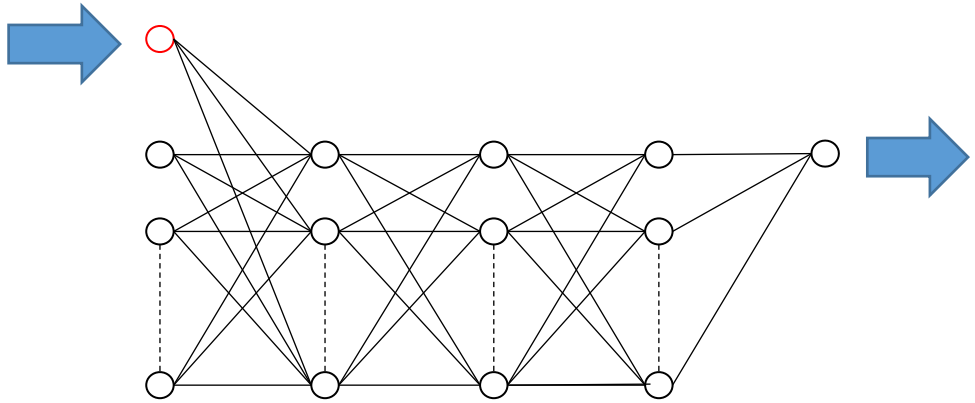
4000-dimension

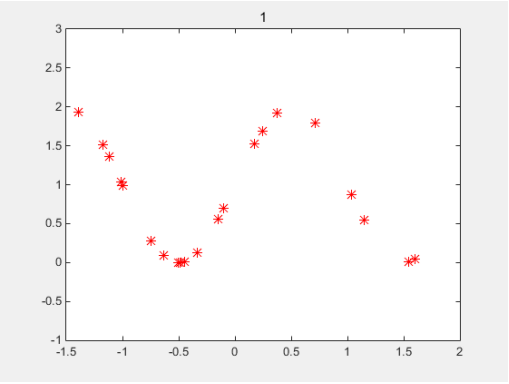
多模态数据输入

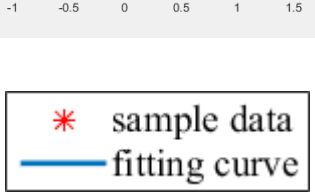
网络输入问题

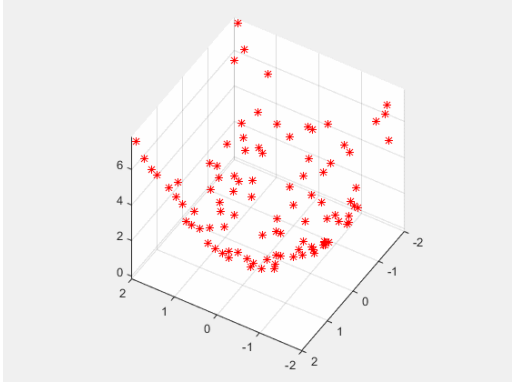


一维数据输入

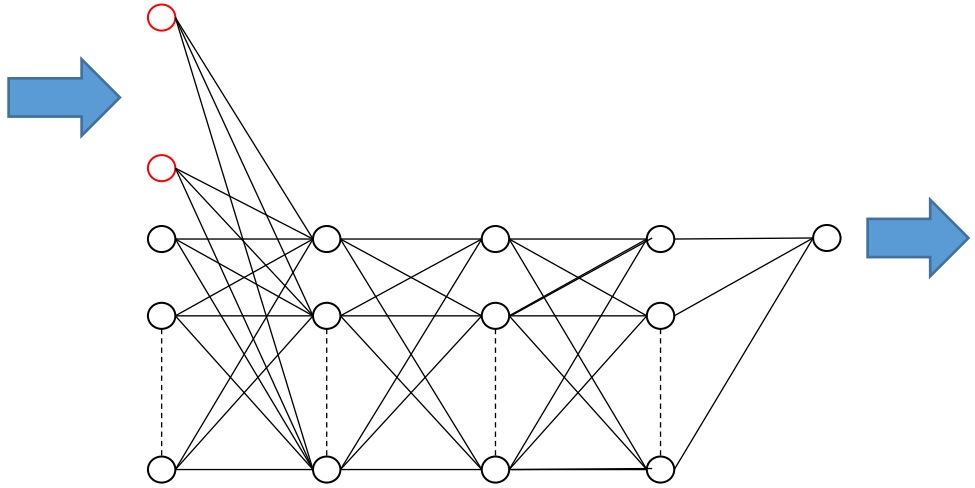


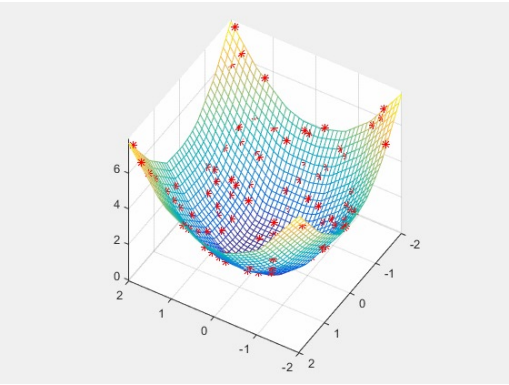






二维数据输入





提纲

神经网络训练 - 问题

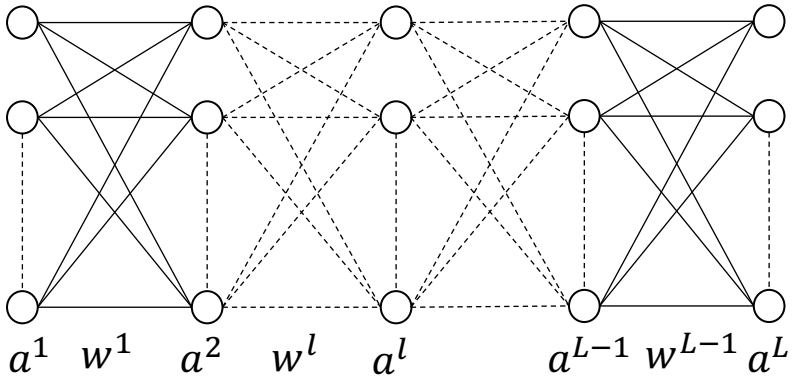
I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

网络预测问题



网络预测

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

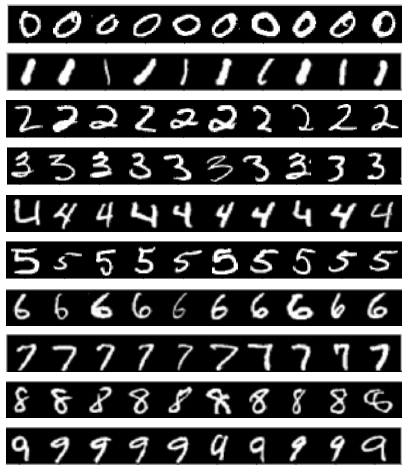
定义最后一层的激活函数 f^L 使得：

- 网络预测向量 a^L 和目标输出向量 y^L 的取值域相匹配
- 需要 f^L是可微的

目标输出

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

$$a_i^L = f_i^L(z_i^L)$$

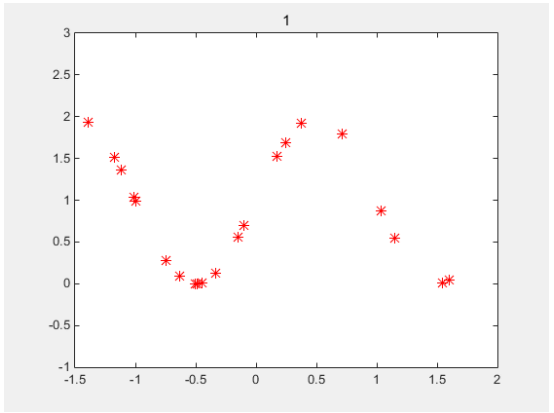


Sigmoid function

$$f(s) = \frac{1}{1+e^{-s}} \in (0,1)$$

$$\begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix} \quad \text{Threshold } \theta \quad \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

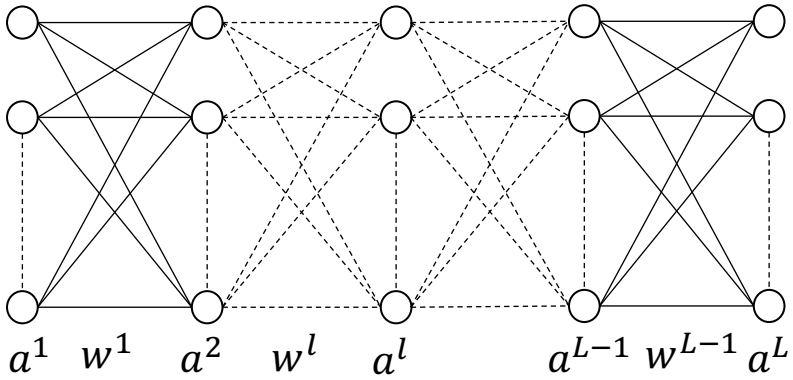
$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



Linear function

$$f(s) = s$$

网络预测问题



目标输出

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

$$0 \le y_i^L \le 1$$

$$\sum_{i=1}^{n_L} y_i^L = 1$$

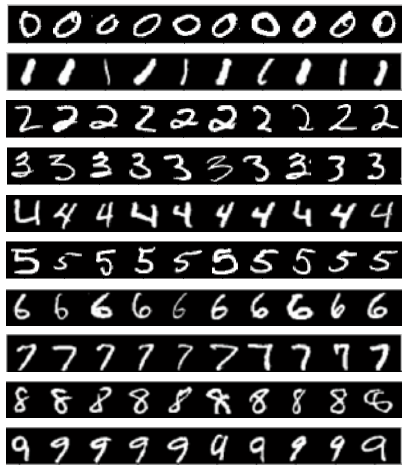
$$a_i^L = \frac{e^{z_i^L}}{e^{z_1^L} + \cdots + e^{z_{n_L}^L}}$$

Softmax function

网络预测

$$0 < a_i^L < 1$$

$$\sum_{i=1}^{n_L} a_i^L = 1$$



0	1	2	3	4	5	6	7	8	9
$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}$

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix} \qquad 0 \le y_i^L \le 1, \sum_{i=1}^{n_L} y_i^L = 1$$

提纲

神经网络训练 - 问题

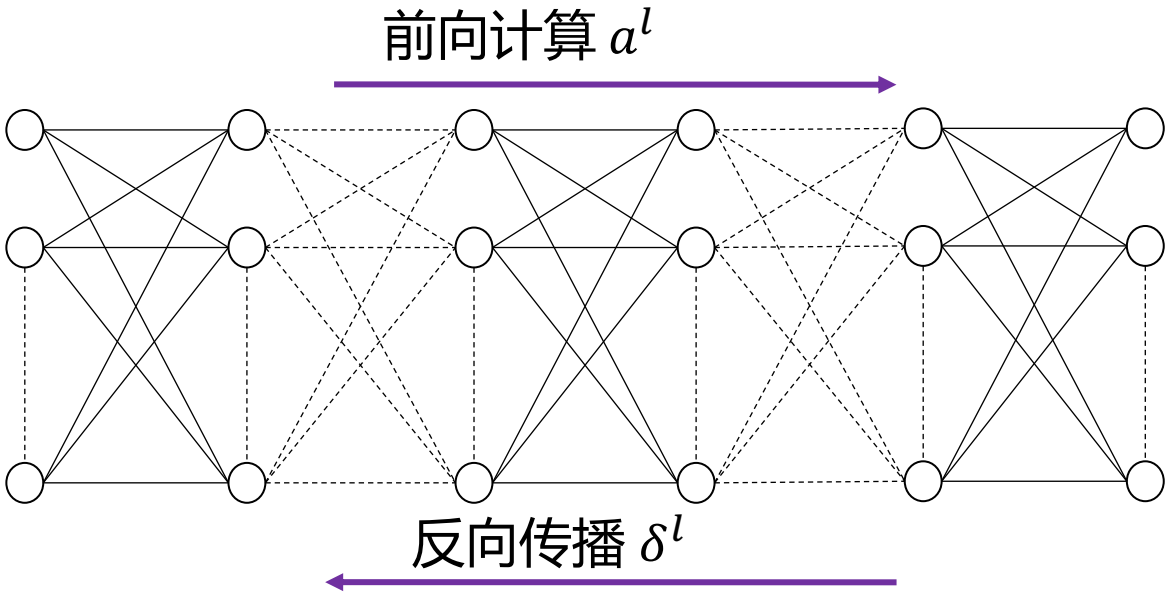
I

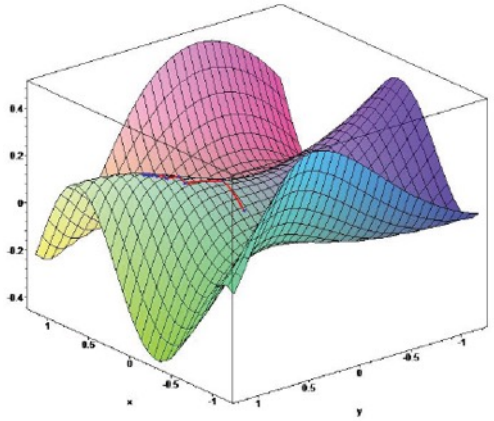
- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

性能函数问题





网络输出

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

目标输出

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

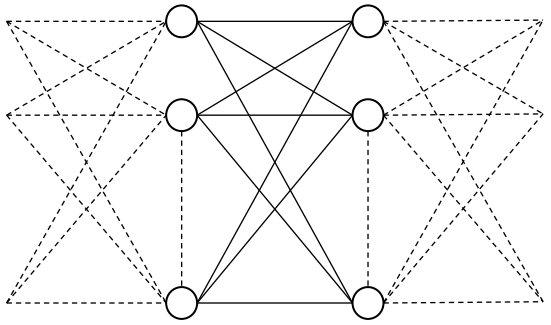
$$J(a^L, y^L)$$

性能函数 $J(a^L, y^L)$ 被用来刻画 a^L 和 y^L 的靠近程度，$J(a^L, y)$ 实际上是$(w^1, \cdots, w^{L-1})$的函数，例如：

$$J = J(w^1, \cdots, w^{L-1})$$

性能函数问题

前向计算 a^l



反向传播 δ^l

欧式距离

$$\begin{cases} J = \frac{1}{2}\sum_{j=1}^{n^L}\left(a_j^L - y_j^L\right)^2 \\ \delta_i^L = \frac{\partial J}{\partial z_i^l} = \left(a_i^L - y_i^L\right)\cdot \dot{f}\left(z_i^L\right) \end{cases}$$

$$0 \le y_i^L \le 1 \ (i = 1, \cdots, n_L)$$

$$a_i^L = f\left(z_i^L\right) = \frac{1}{1 + e^{-z_i^L}}$$

Sigmoid function

网络输出 $a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$

目标输出 $y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$

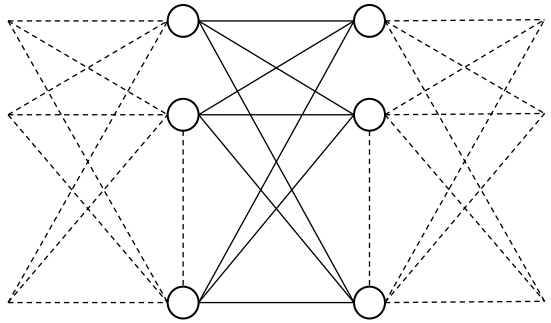
$J(a^L, y^L)$

性能函数 $J(a^L, y^L)$ 被用来刻画 a^L 和 y^L 的靠近程度，$J(a^L, y)$ 实际上是 $(w^1, \cdots, w^{L-1})$的函数，例如：

$$J = J(w^1, \cdots, w^{L-1})$$

性能函数问题

前向计算 a^l



反向传播 δ^l

$$\sum_{j=1}^{n_L} y_i^L = 1$$

$$a_j^L = \frac{e^{z_j^L}}{e^{z_1^L} + \cdots + e^{z_{n_L}^L}}$$

Softmax function

网络输出 $a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$

目标输出 $y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$

$J(a^L, y^L)$

性能函数 $J(a^L, y^L)$ 被用来刻画 a^L 和 y^L 的靠近程度，$J(a^L, y)$ 实际上是$(w^1, \cdots, w^{L-1})$的函数，例如：

$$J = J(w^1, \cdots, w^{L-1}).$$

交叉熵

$$J = -\sum_{j=1}^{n_L} y_j^L \cdot \log(a_j^L) + \lambda \cdot \sum (w_{ij}^l)^2$$

$$a_j^L = \frac{e^{z_j^L}}{\sum_{i=1}^{n_L} e^{z_i^L}}$$

$$\delta_i^L = a_i^L - y_i^L$$

问题：为什么交叉熵可以刻画y^L和a^L的距离？

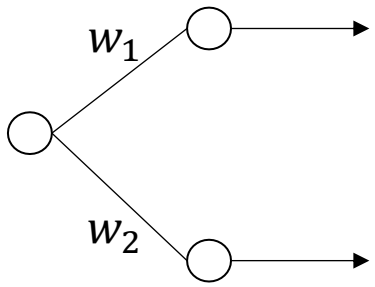
性能函数问题

一个例子

样本数据

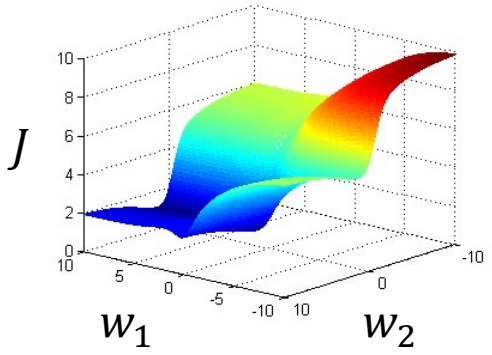
	1	2
x	0.8000	0.2000
y	0	1
	1	0

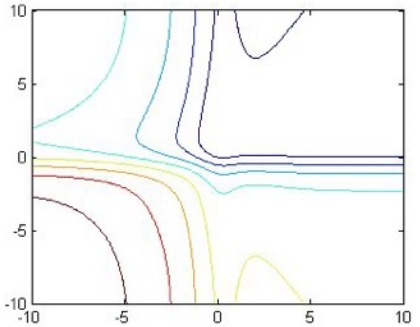
网络



欧式距离

$$\begin{cases} J = \frac{1}{2}\sum_{j=1}^{2}(a_j - y_j)^2 \\ a_j = \frac{1}{1+\exp(-z_j)} \\ z_j = w_j \cdot x \end{cases}$$

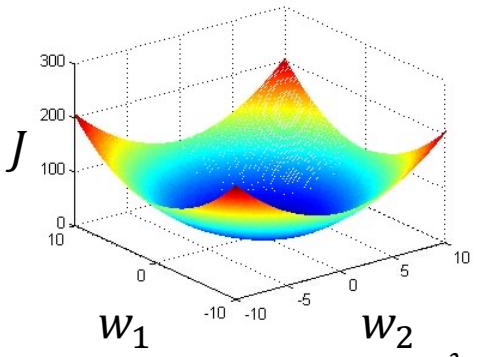


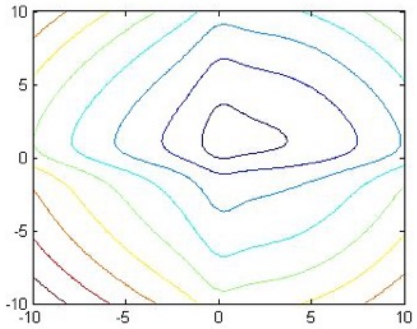


轮廓

交叉熵

$$\begin{cases} J = -\sum_{j=1}^{2} y_j \cdot \log(a_j) + \lambda(w_1^2 + w_2^2) \\ a_j = \frac{e^{z_j}}{\sum_{i=1}^{2} e^{z_j}} \\ z_j = w_j \cdot x \end{cases}$$





$\lambda = 0.05$

提纲

神经网络训练 - 问题

I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

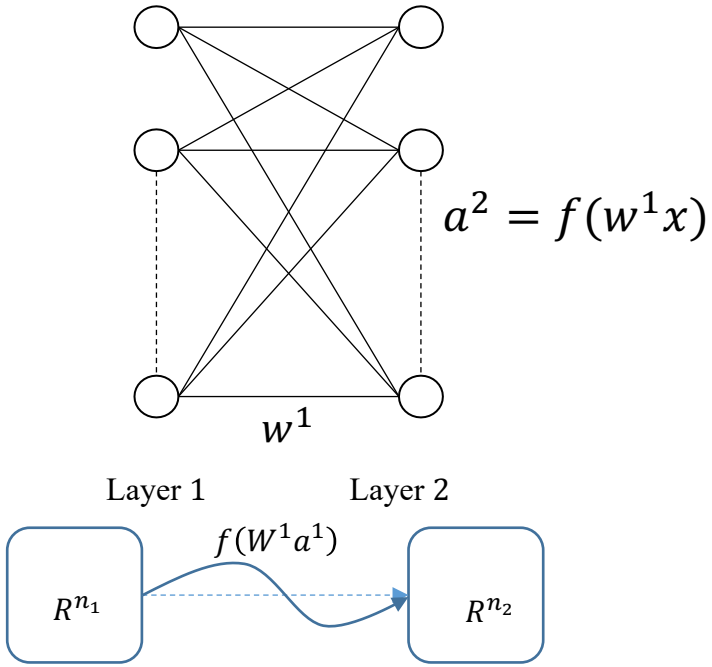
II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

网络深度问题

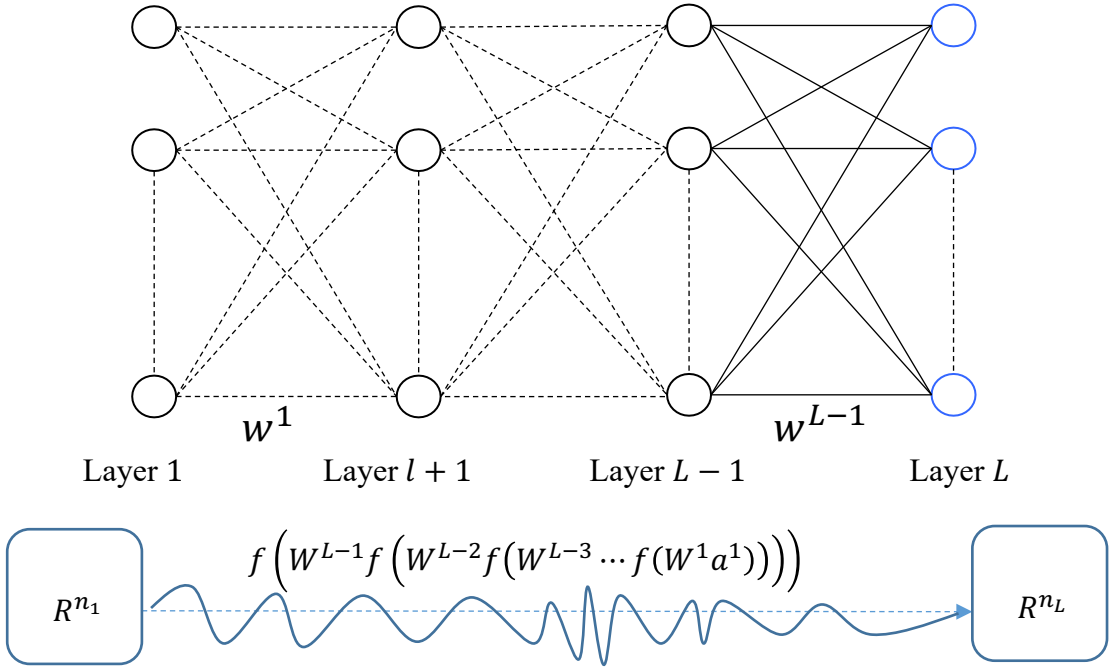
浅层神经网络

- $L = 2$
- 难以学习到复杂的非线性映射



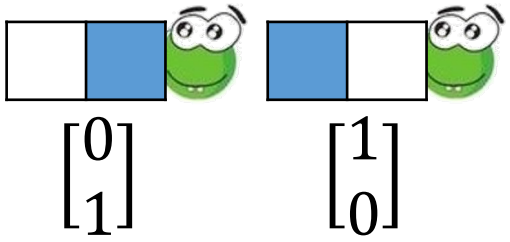
深层神经网络

- $L > 2$
- 只要网络中的神经元数目足够多，可以任意精度拟合任意非线性映射



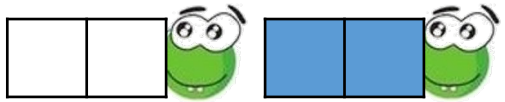
例子：异或问题

有斑点的毛毛虫

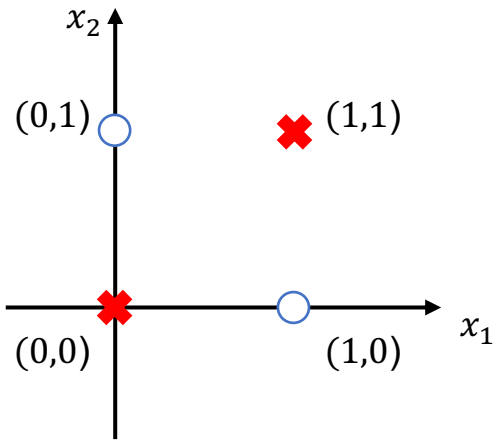


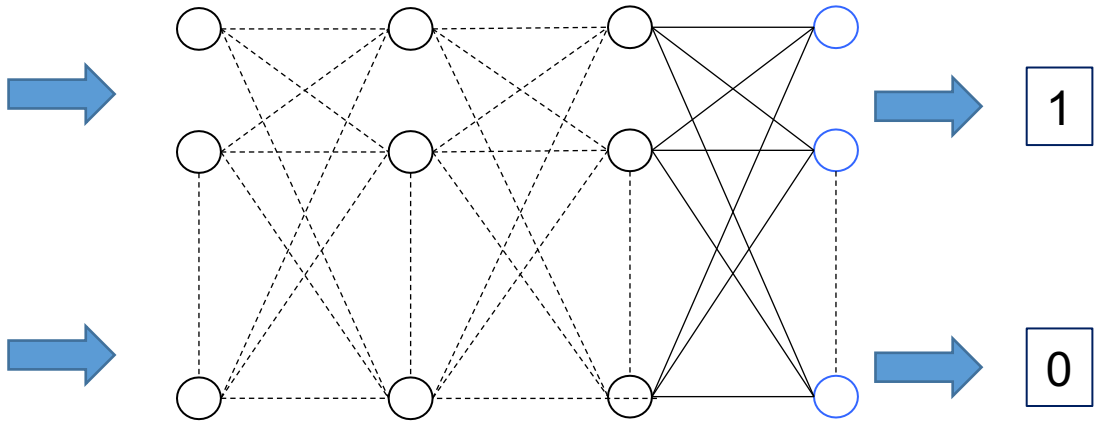
$\begin{bmatrix}0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\end{bmatrix}$

光滑毛毛虫



$\begin{bmatrix}0\\0\end{bmatrix}$ $\begin{bmatrix}1\\1\end{bmatrix}$





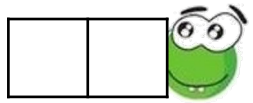
例子：异或问题

有斑点的毛毛虫

$\begin{bmatrix}0\\1\end{bmatrix}$

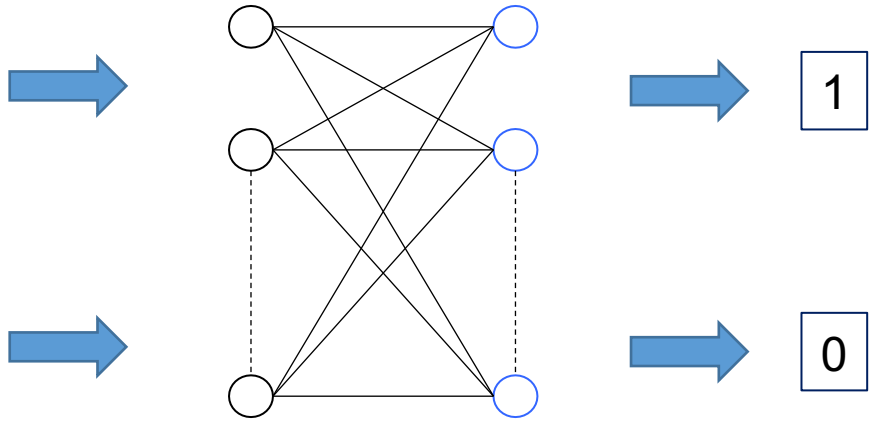
$\begin{bmatrix}1\\0\end{bmatrix}$

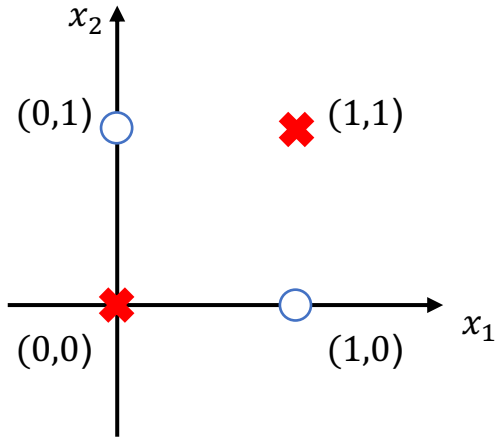
光滑毛毛虫



$\begin{bmatrix}0\\0\end{bmatrix}$

$\begin{bmatrix}1\\1\end{bmatrix}$



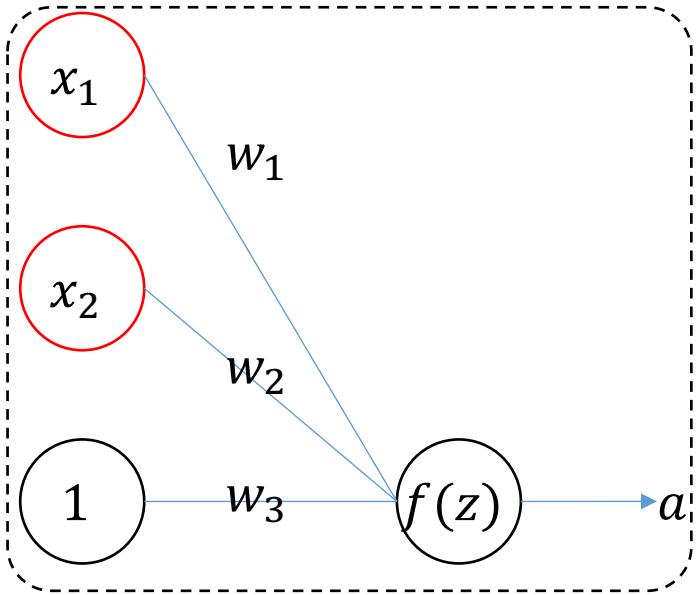


两层网络在激活函数是单调函数的情况下无法完成XOR分类任务

例子：异或问题

$\begin{bmatrix}0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\end{bmatrix}$

$\begin{bmatrix}0\\0\end{bmatrix}$ $\begin{bmatrix}1\\1\end{bmatrix}$



等价

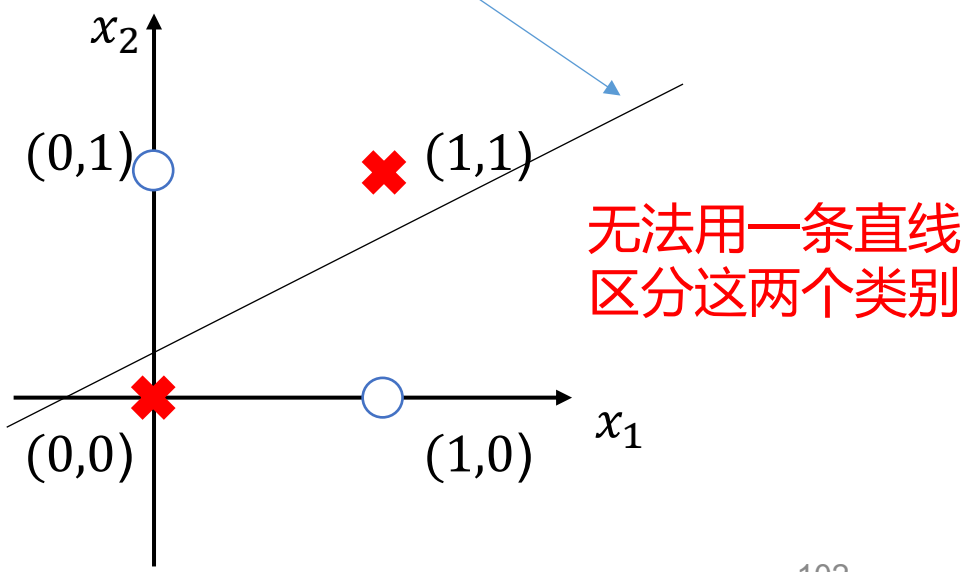
f是单调函数

$a = f(z)$

$z = w_1x_1 + w_2x_2 + w_3$

决策线

$w_1x_1 + w_2x_2 + w_3 = 0$



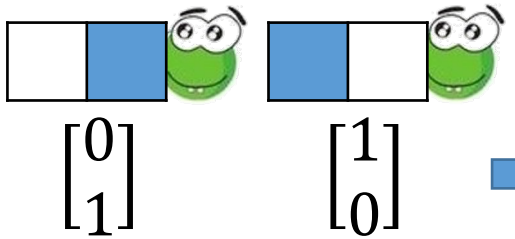
无法用一条直线
区分这两个类别

结论：两层网络在激活函数是单调函数的情况下无法完成XOR分类任务

问题：两层网络在激活函数为非单调函数的情况下能否完成XOR分类任务？

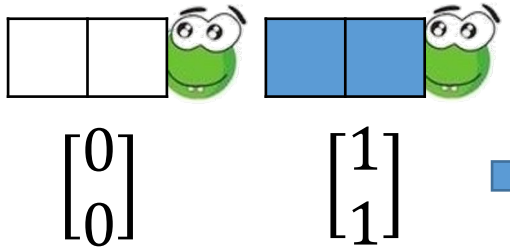
例子：异或问题

不光滑毛毛虫

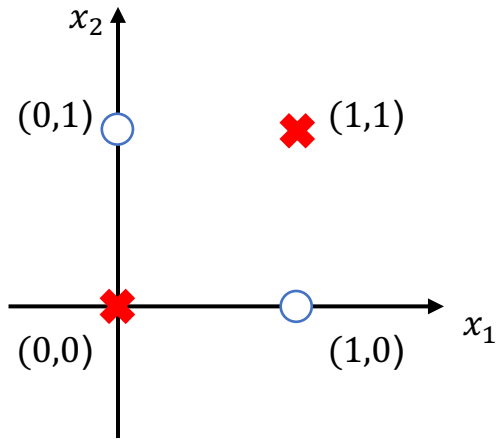


$\begin{bmatrix}0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\end{bmatrix}$

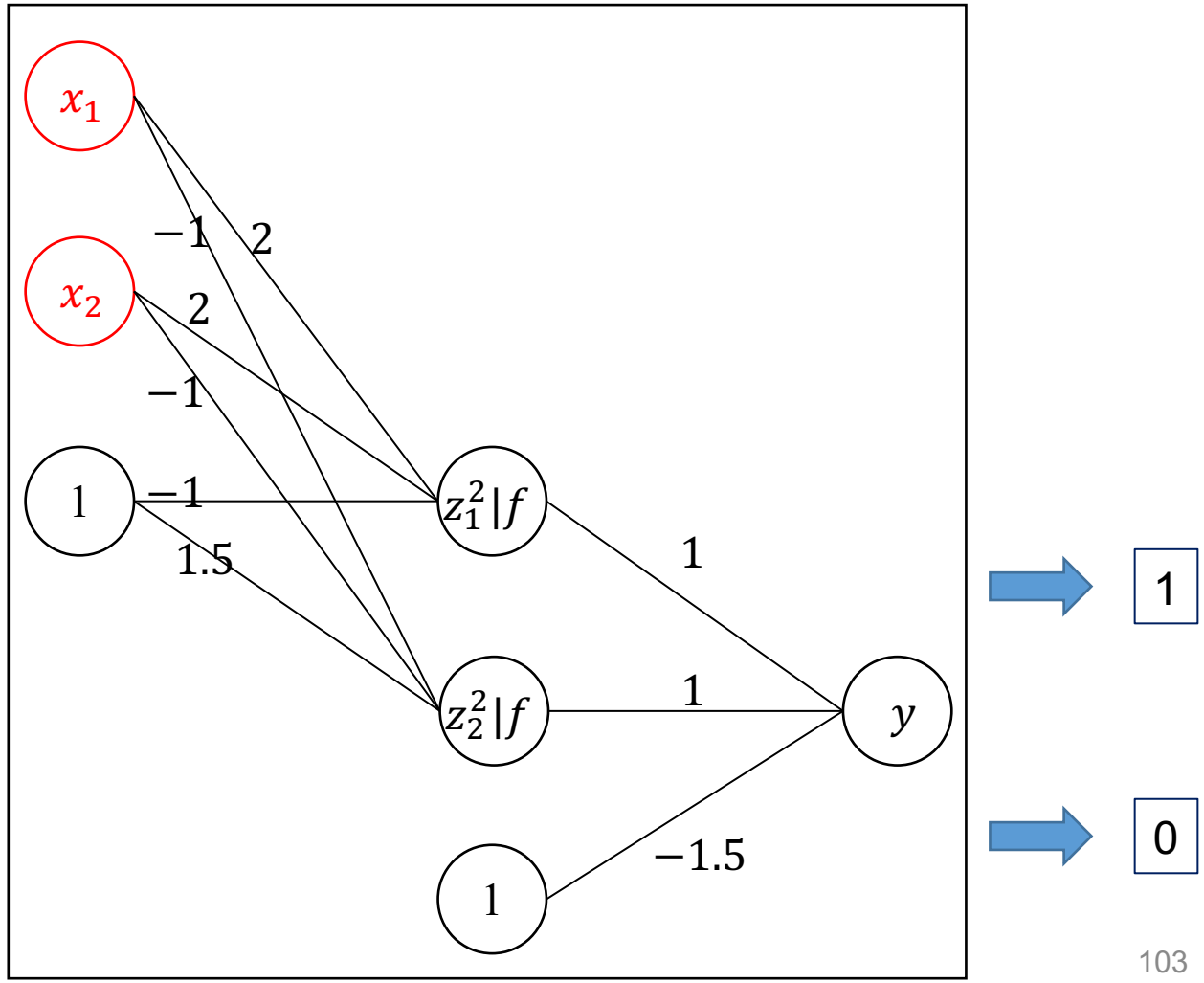
光滑毛毛虫



$\begin{bmatrix}0\\0\end{bmatrix}$ $\begin{bmatrix}1\\1\end{bmatrix}$



三层网络可以解决异或问题



网络深度问题

梯度消失问题

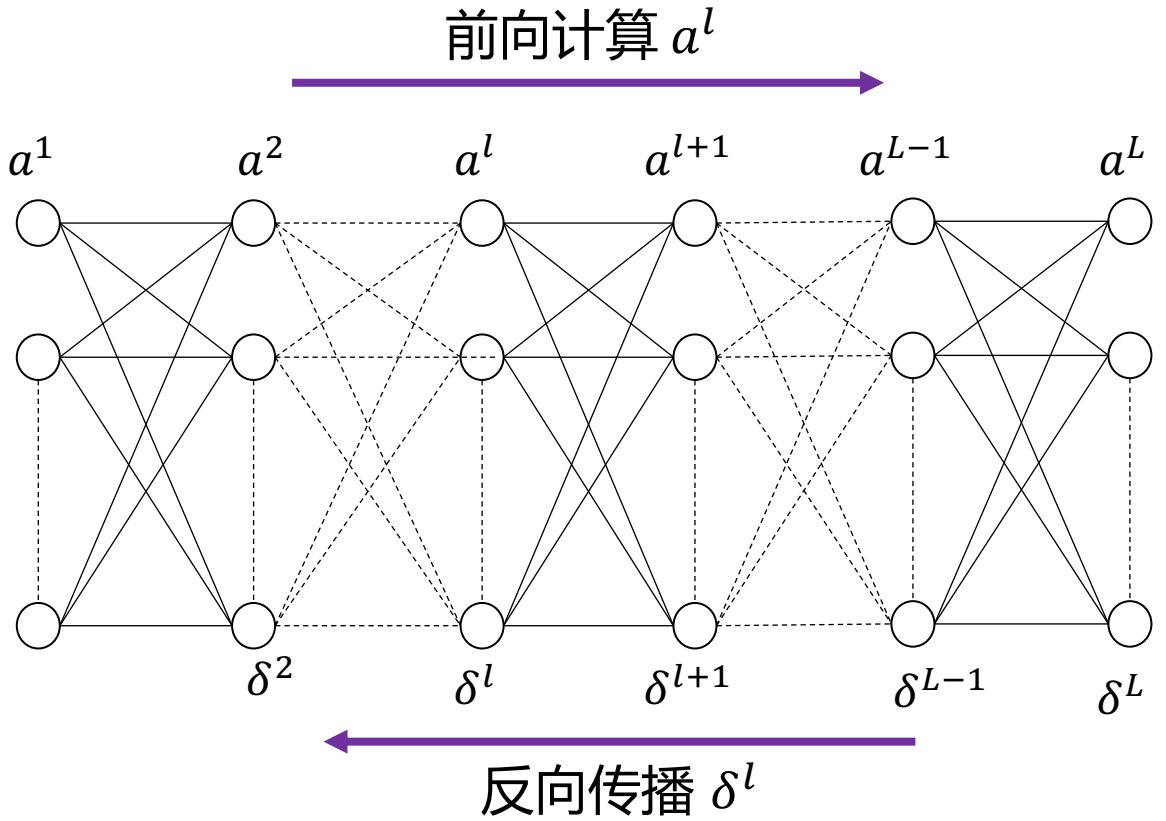
代价函数: $J(w^1, \cdots, w^{L-1})$

更新规则: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

关系: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

关键:

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$



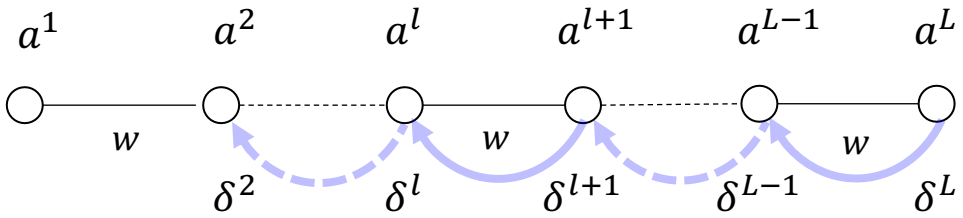
网络深度问题

梯度消失问题

一个简单的例子

$$w = w^{\,l}$$

$$\delta^l = \dot{f}\left(z^l\right) \cdot w \cdot \delta^{l+1}$$

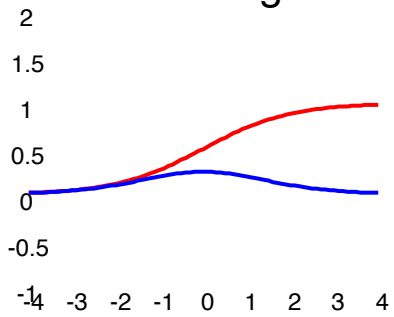


$$\begin{aligned}\delta^l &= \dot{f}\left(z^l\right) \cdot w \cdot \delta^{l+1} \\ &= \dot{f}\left(z^l\right) \cdot w \cdot \dot{f}\left(z^{l+1}\right) \cdot w \cdot \delta^{l+2} \\ &= w \cdot \dot{f}\left(z^l\right) \cdot w \cdot \dot{f}\left(z^{l+1}\right) \cdots w \cdot \dot{f}\left(z^{L-1}\right) \cdot \delta^L \\ &= \prod_{m=L-1}^{l} \left(w \cdot \dot{f}\left(z^m\right)\right) \cdot \delta^L \to 0\end{aligned}$$

$$\left|\frac{\partial \delta^l}{\partial \delta^L}\right| = \prod_{m=L-1}^{l} \left|w \cdot \dot{f}\left(z^m\right)\right| \le |w|^{L-l+1} \cdot (0.25)^{L-l+1}$$

$$\dot{f}\left(z^m\right) \le 0.25$$

Sigmoid

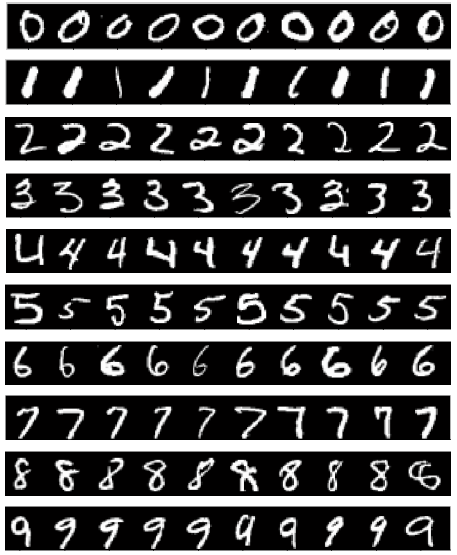


Notes：

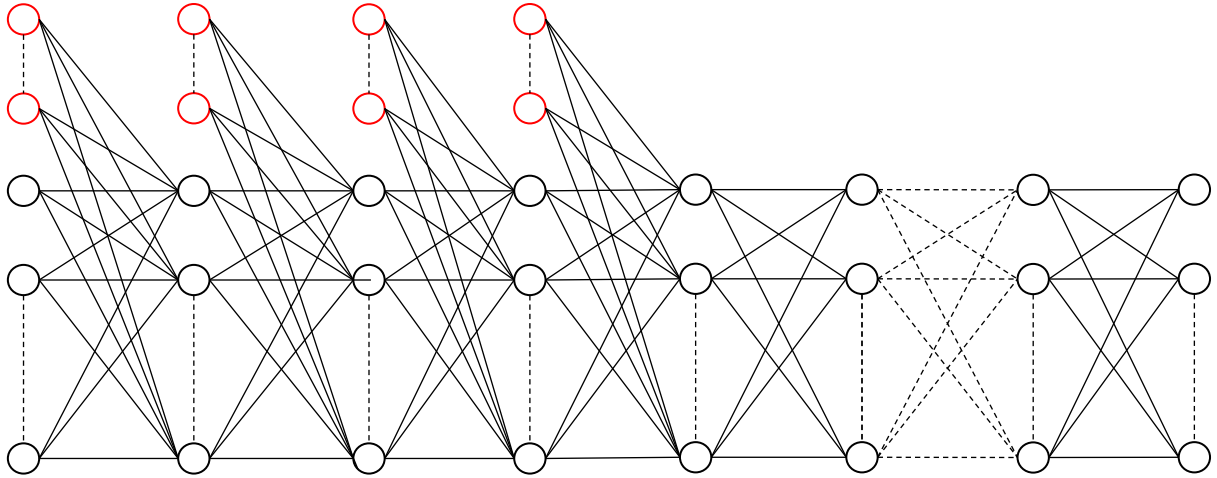
δ^l 的指数下降导致梯度消失问题。

网络深度问题

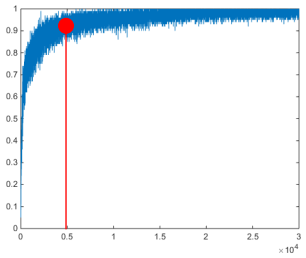
网络的深度与具体问题相关



手写体数字识别问题



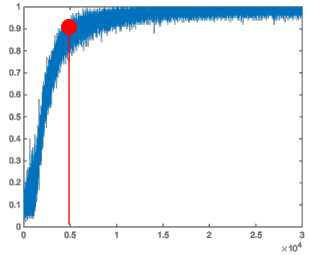
5 layers



准确率

- Training=97.55%
- Testing=95.25%

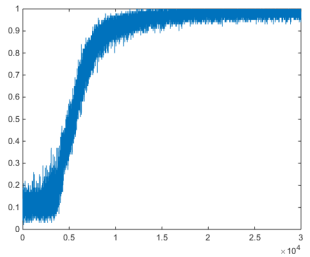
8 layers



准确率

- Training= 98.65%
- Testing= 95.10%

9 layers



准确率

- Training=98.45%
- Testing=93.20%

提纲

神经网络训练 - 问题

I

- 网络结构问题
- 学习算法问题
- 目标输出问题
- 网络输入问题

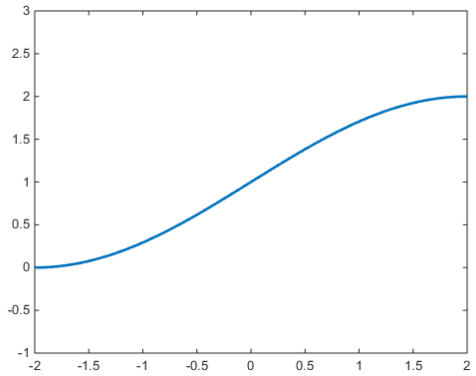
II

- 网络预测问题
- 性能函数问题
- 网络深度问题
- 训练数据问题

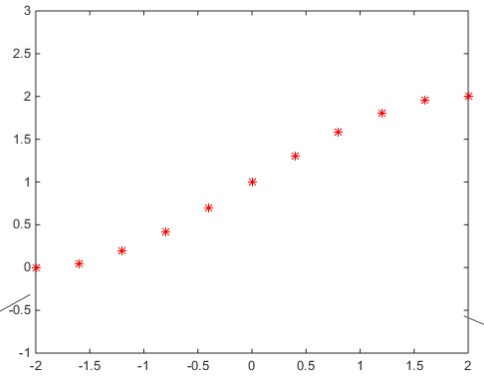
训练数据问题

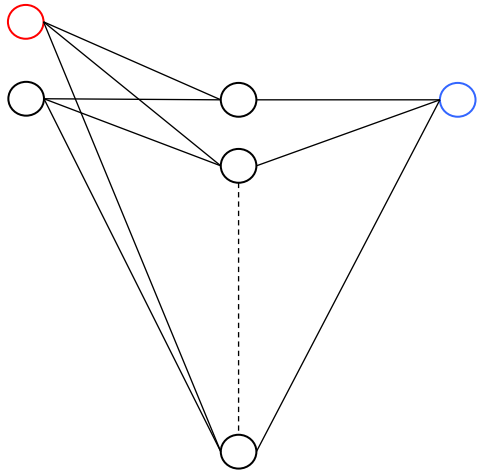
用 2-9-1 网络拟合一个部分正弦曲线

$$y = g(x) = 1 + \sin\left(\frac{\pi}{4}x\right), x \in [-2,2]$$



采样





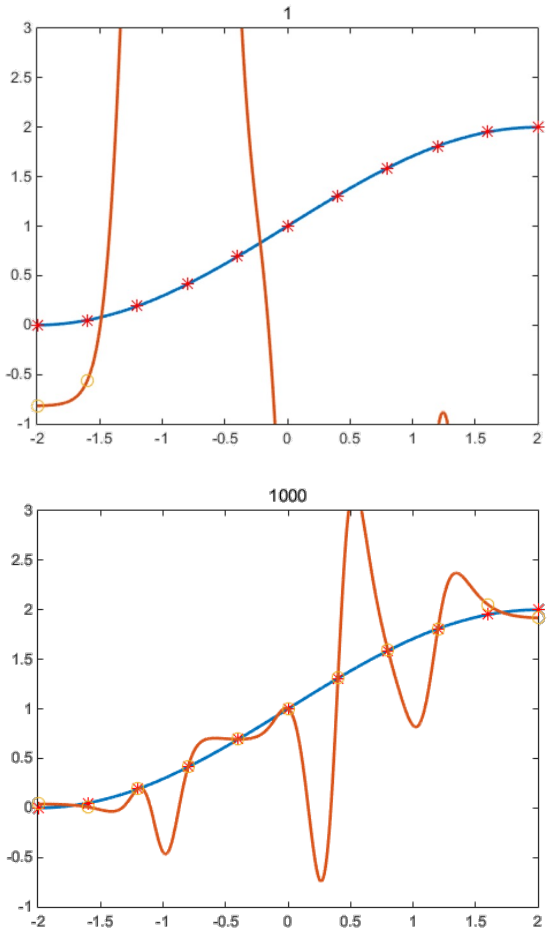
(1,1) → (0,9) → (0,1)

	1	2	3	4	5	6	7	8	9	10	11
x	-2	-1.6000	-1.2000	-0.8000	-0.4000	0	0.4000	0.8000	1.2000	1.6000	2
y	0	0.0489	0.1910	0.4122	0.6910	1	1.3090	1.5878	1.8090	1.9511	2

11 samples

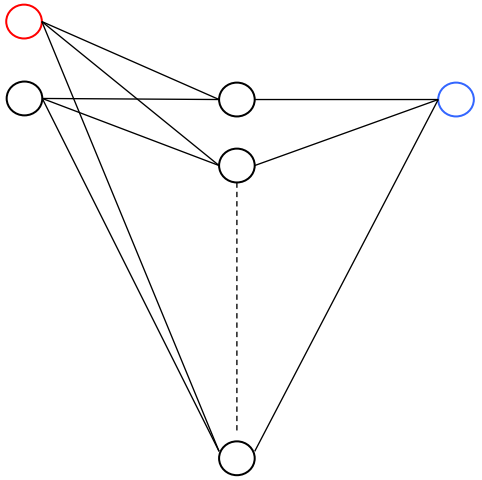
训练数据问题

过拟合



11 个数据样本

用2-9-1网络拟合一个部分正弦曲线

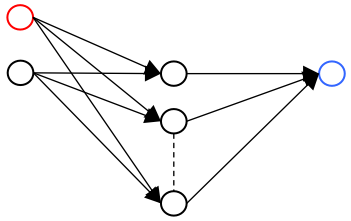


2-9-1 网络有 27 个权重需要调优。一般来说，需要比参数数量更多的样本来进行训练。

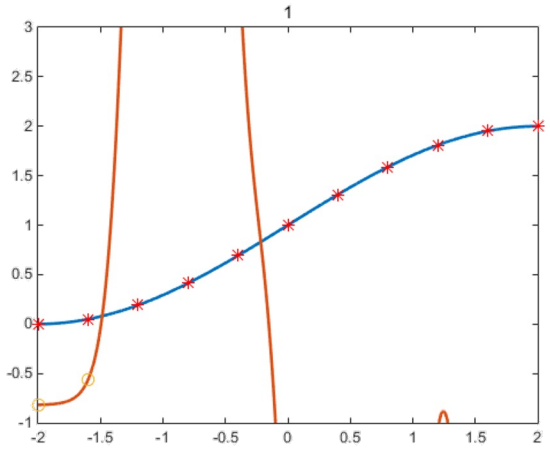
网络与数据样本拟合得很好，但在曲线的其他部分表现不好！过拟合!

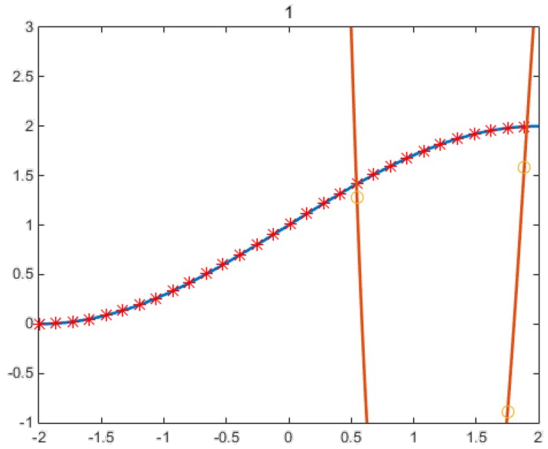
- 在训练数据上拟合的很好
- 无法良好拟合测试数据
- 需要**更多**的数据！

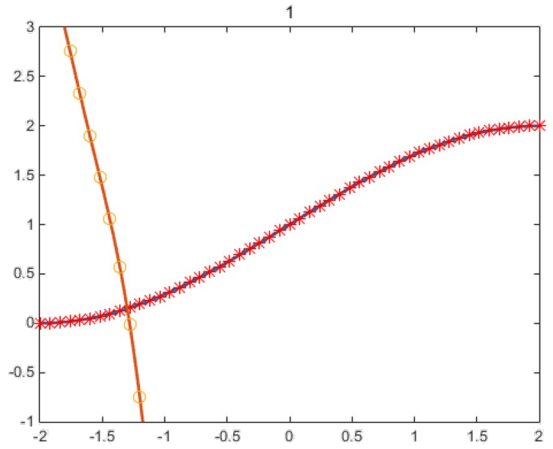
训练数据问题

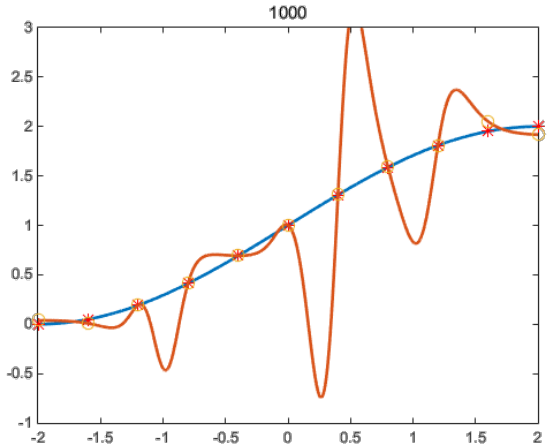


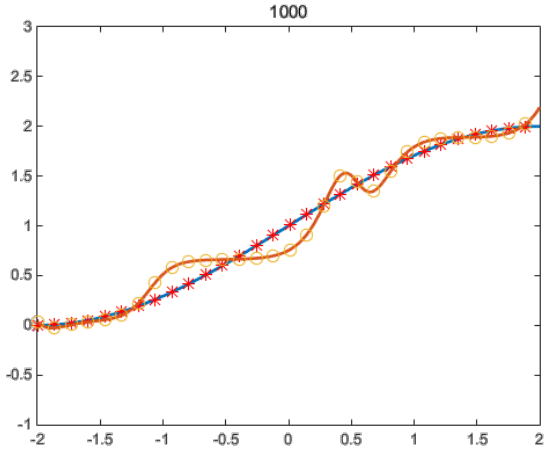
(1,1) → (0,9) → (0,1)

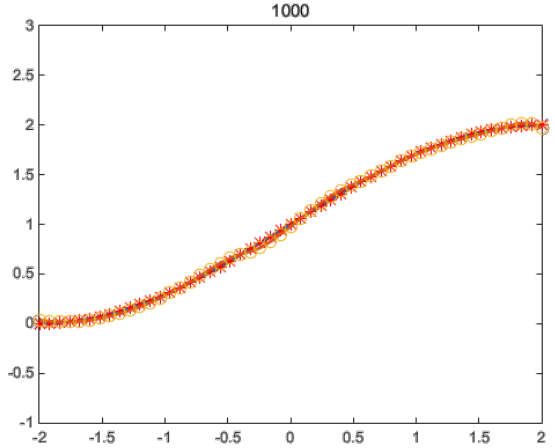






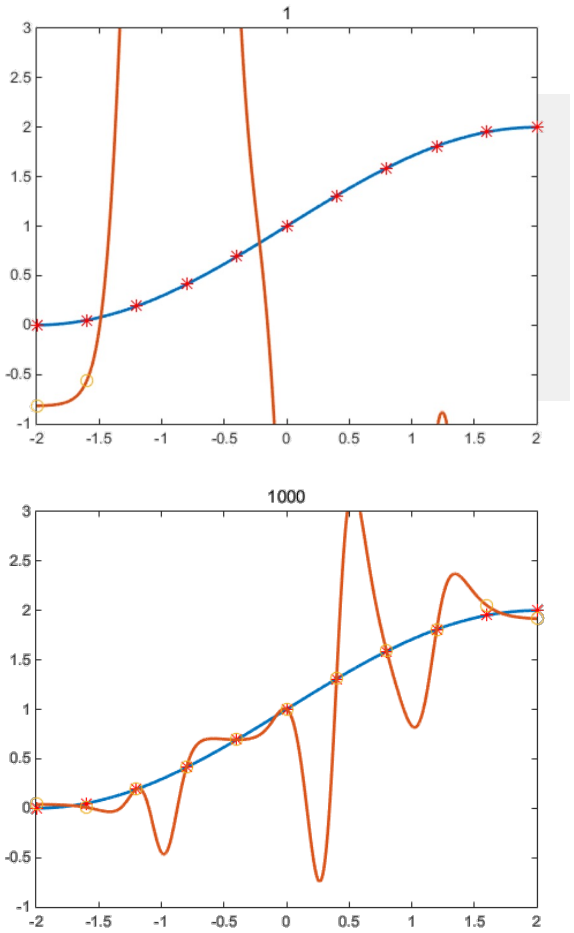




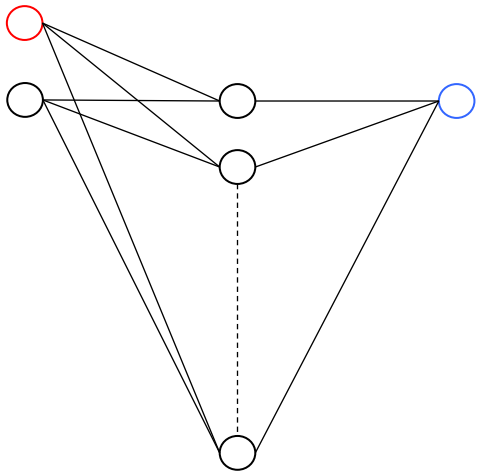


11 个数据样本　　23 个数据样本　　51 个数据样本

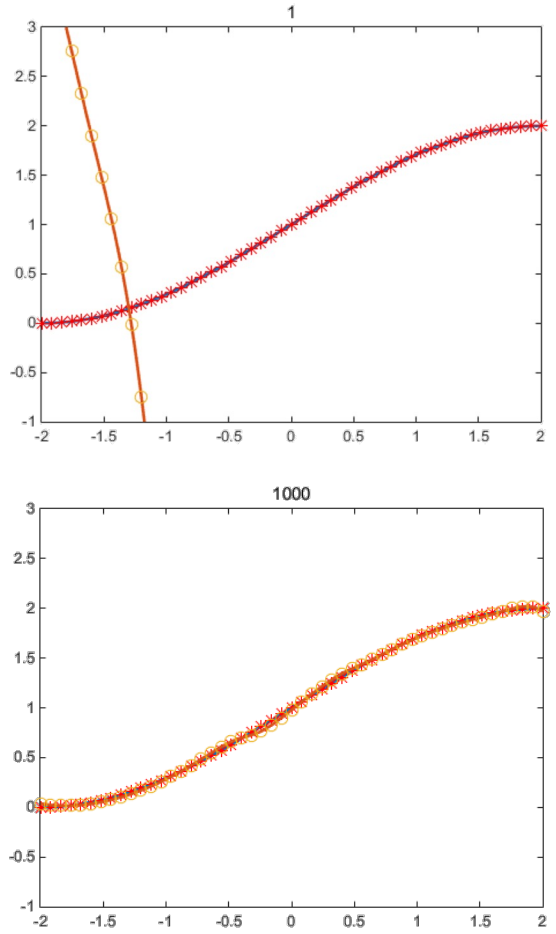
训练数据问题



11 个数据样本



为了使网络具有泛化性，网络参数个数应该少于训练集中样本数据的个数。



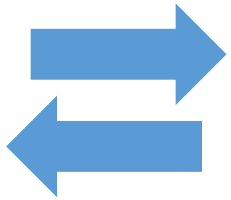
51 个数据样本

训练数据问题

大数据

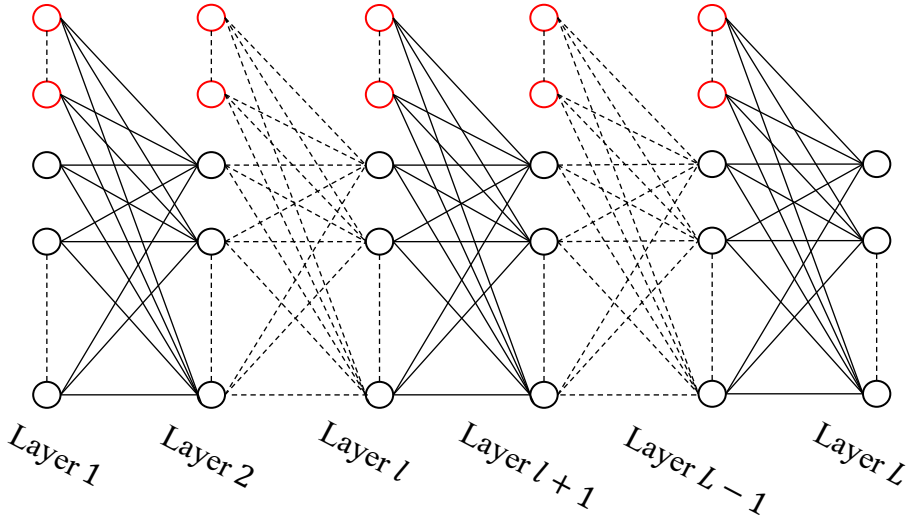
大数据中的复杂模式需要复杂的模型来处理。

丰富的数据可以用于模型训练。
(样本)



高度非线性，灵活，可训练的模型。
(复杂性)

深度神经网络



深度神经网络模型中有大量的参数需要训练。

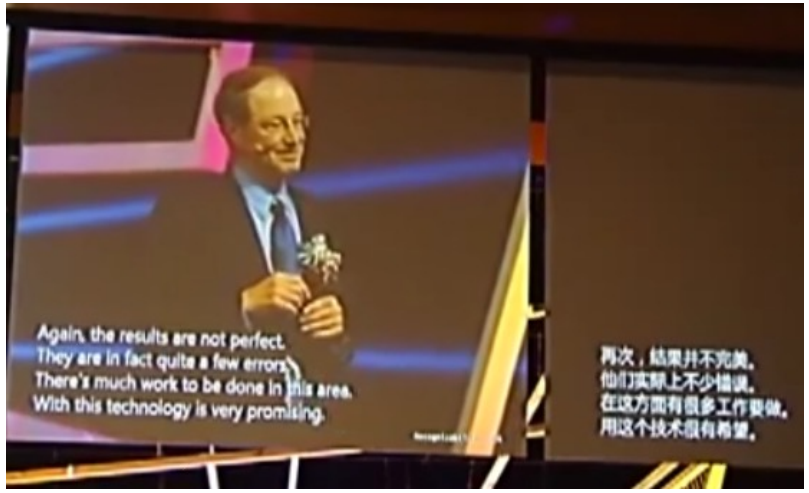
大数据+ 深度神经网示例

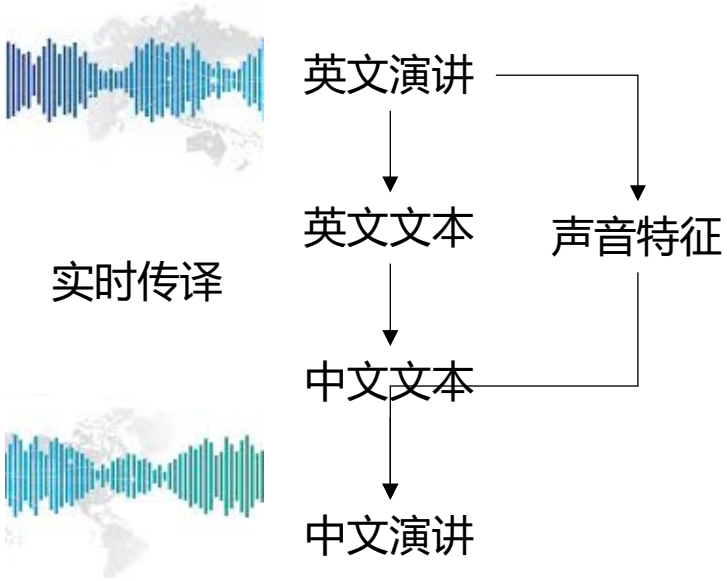
语音识别

1950s　语音波形 + 模式识别 = 识别少量词汇

1970s　高斯混合模型 + 隐马尔可夫模型 = ~80%的识别率

2011　深度神经网络建模的语音模型 = 令人惊讶的实时识别!

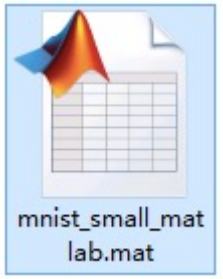


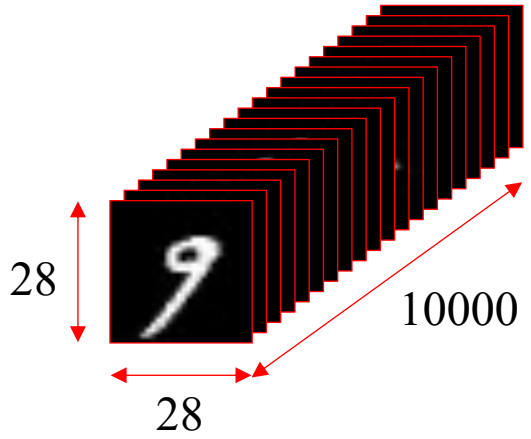


作业

- 使用BP算法实现一个简单的手写数字识别。

  - 提供MATLAB模版和Python模版
  - 可以使用MATLAB或Python
  - 下周上课前交（10/9）





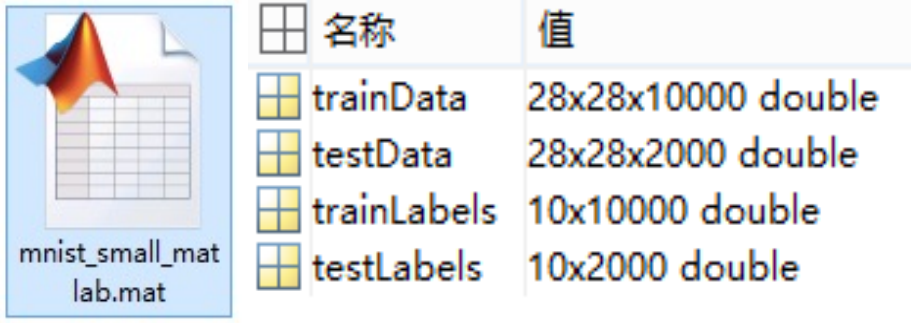
提纲

Appendix 1: Reference Code by Using MATLAB

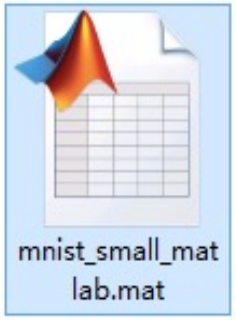
Appendix 2: Reference Code by Using Python

Appendix 1: Reference Code by Using MATLAB

Reference Code for Handwritten Digits Recognition by Using MATLAB



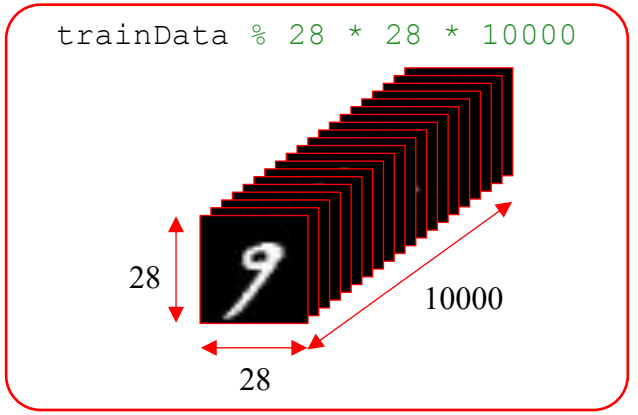
Step 1: Data Preparation



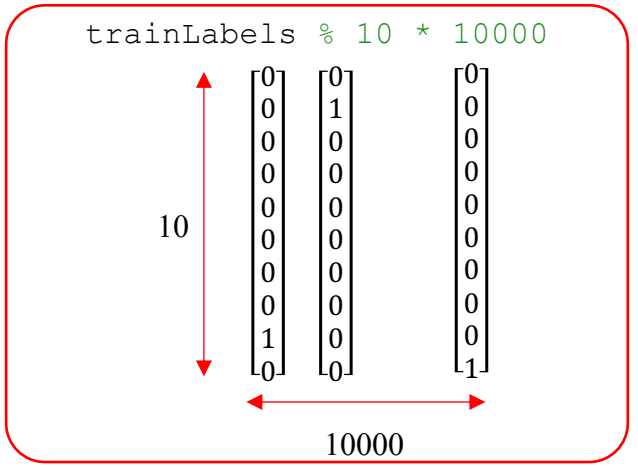
名称	值
trainData	28x28x10000 double
testData	28x28x2000 double
trainLabels	10x10000 double
testLabels	10x2000 double

```
%% Step 1: Data Preparation

% loading dataset
load mnist_small_matlab.mat
% trainData: a matrix with size of 28x28x10000
% trainLabels: a matrix with size of 10x10000
% testData: a matrix with size of 28x28x2000
% testLabels: a matrix with size of 10x2000
```

Training Data

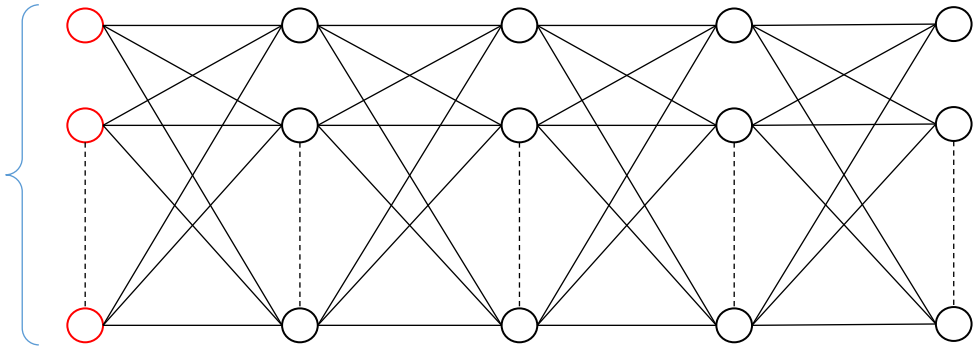


Label



Step 1: Data Preparation

```
% prepare the external input for training set

train_size = 10000; % number of training samples
% input in the 1st layer
X_train = reshape(trainData, 784, train_size);
```



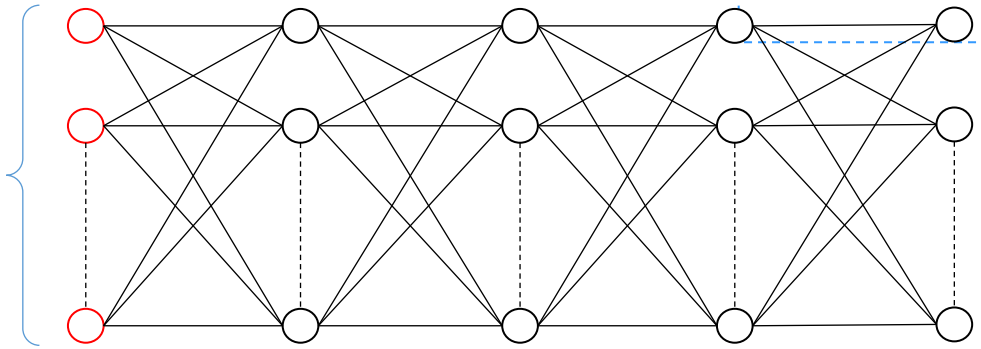
```
% prepare the external input for
testing set

test_size = 2000; % number of testing
samples
% external input in the 1st layer
X_test = reshape(testData,
                 784,
                 test_size);
```

TIPs

- X_train and X_test are cells in Matlab
- reshape and zeros are built-in functions in Matlab

Step 2: Design Network Architecture

```
% define number of layers
L = 5;

% define number of neurons in each layer
% - 1st column: external neurons
% - 2nd column: internal neurons
layer_size = [784 % number of neurons in 1st layer
              256 % number of neurons in 2nd layer
              128 % number of neurons in 3rd layer
              64  % number of neurons in 4th layer
              10];% number of neurons in 5th layer
```



Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
784	256	128	64	10

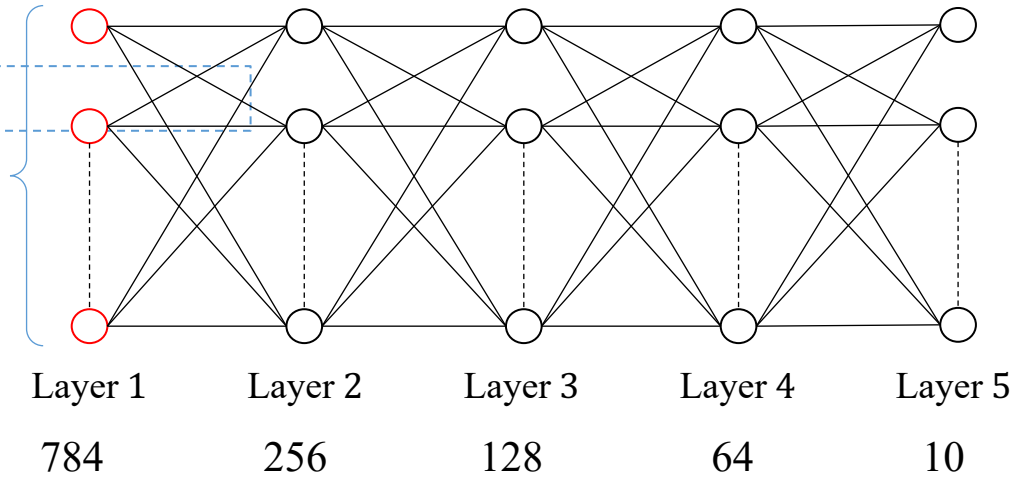
Step 3: Initialize Parameters

Gaussian distribution: $w_{ij}^{l} \sim N(0,1)$

```
% initialize weights
for l = 1:L-1
    w{l} = 0.1*randn(layer_size(l+1,2), sum(layer_size(l,:)));
end
```

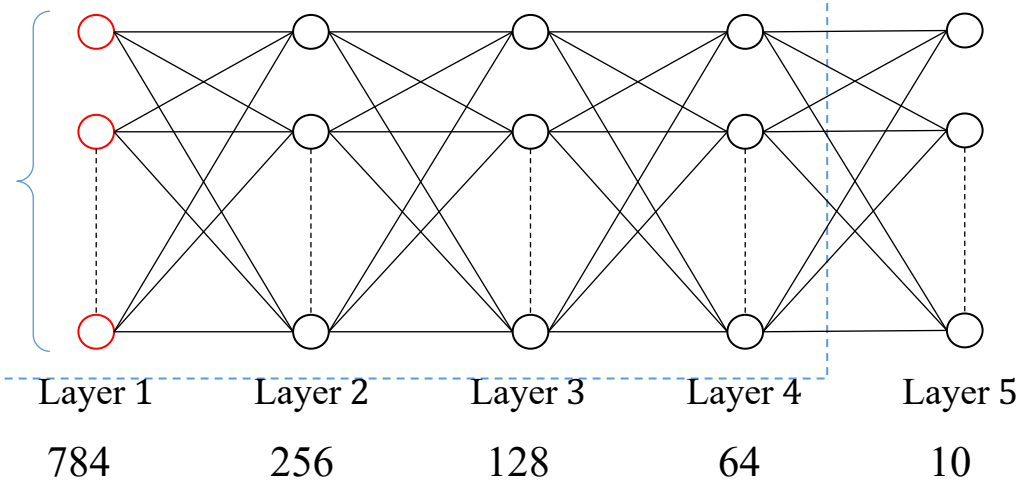
Initialize learning rate

```
alpha = 0.005; % initialize learning rate
```



Step 4, 5: Define Cost Function and Evaluation Index

```
%% Step 4: Define Cost Function
function [J] = cost(a, y)
    J = 1/2 * sum((a - y).^2);
end

%% Step 5: Define Evaluation Index
function [ acc ] = accuracy(a, y)
    mini_batch = size(a, 2);
    [~,idx_a] = max(a);
    [~,idx_y] = max(y);
    acc = sum(idx_a==idx_y) / mini_batch;
end
```



Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2$$

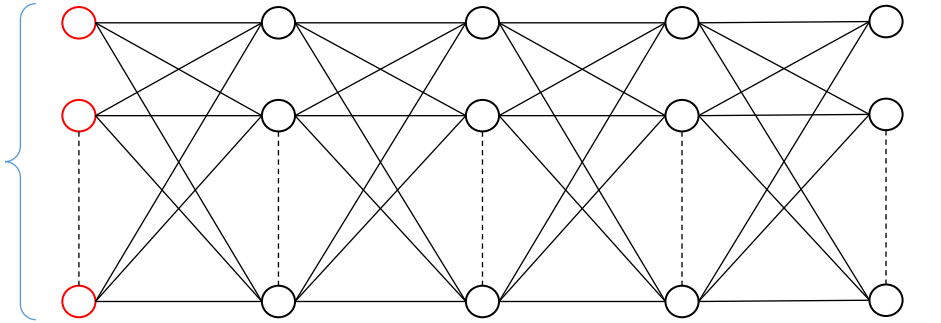
$$\text{Acc} = \frac{\text{number of correct prediction}}{\text{number of samples}}$$

Step 6: Train the Network

```
%% Step 6: Train the Network
J = []; % array to store cost of each mini batch
Acc = []; % array to store accuracy of each mini batch
max_epoch = 200; % number of training epochs
mini_batch = 100; % number of sample in each mini batch
```

```
% loop until converge
for iter = 1:max_epoch
    % for each mini-batch
        % batch forward computation
        % batch backward computation
        % cumulate and update weight
end
```

Mini-batch BP implement



BP Algorithm:

Step 1. Input the training data set $D = \{(x, y)\}$

Step 2. Initialize each w_{ij}^{l}, and choose a learning rate α.

Step 3. for each mini-batch sample $D_m \subseteq D$

for each $x \in D_m$

$a^1 \leftarrow x \in D_m$;

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

$\frac{\partial J}{\partial w_{ji}^{l}} \leftarrow \frac{\partial J}{\partial w_{ji}^{l}} + \delta_j^{l+1} \cdot a_i^l$;

end

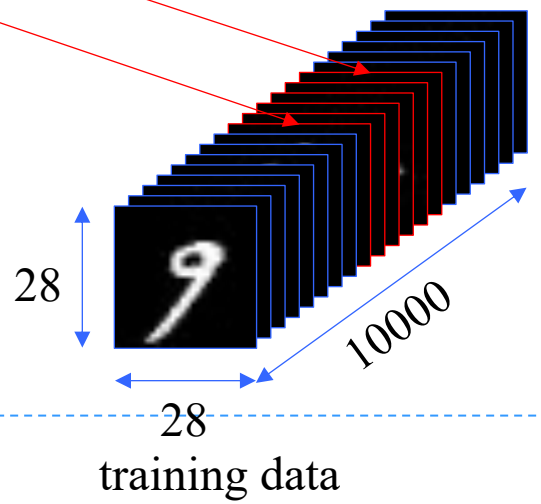
$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$;

end

Step 4. Return to Step 3 until each w^l converge.

Step 6: Train the Network

```
% randomly permutate the indexes of samples in training set
idxs = randperm(train_size);
% for each mini-batch
for k = 1:ceil(train_size/mini_batch)
    % prepare internal inputs in 1st layer denoted by a{1}

    start_idx = (k-1)*mini_batch+1;           % start index of kth mini-batch
    end_idx = min(k*mini_batch, train_size); % end index of kth mini-batch
    a{1} = X_train(:,idxs(start_idx:end_idx));

    % prepare labels
    y = trainLabels(:, idxs(start_idx:end_idx));
    % forward computation
    % ...
    % cost function
    % ...
    % backward computation
    % ...
    % update weight
    % ...
end
```



TIPs

- randperm is a built-in function in Matlab

Step 6: Train the Network

```
% forward computation
for l=1:L-1
    [a{l+1}, z{l+1}] = fc(w{l}, a{l});
end

% Compute delta of last layer
delta{L} = (a{L} - y).* a{L} .*(1-a{L});

% backward computation
for l=L-1:-1:2
    delta{l} = bc(w{l}, z{l}, delta{l+1});
end

% update weight
for l=1:L-1
    % compute the gradient
    grad_w = delta{l+1} * a{l}';
    w{l} = w{l} - alpha*grad_w;
end
```

BP Algorithm:

Step 1. Input the training data set $D = \{(x, y)\}$

Step 2. Initialize each w_{ij}^{l}, and choose a learning rate α.

Step 3. for each mini-batch sample $D_m \subseteq D$

for each $x \in D_m$

$a^1 \leftarrow x \in D_m$;

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

$\frac{\partial J}{\partial w_{ji}^{l}} \leftarrow \frac{\partial J}{\partial w_{ji}^{l}} + \delta_j^{l+1} \cdot a_i^l$;

end

$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$;

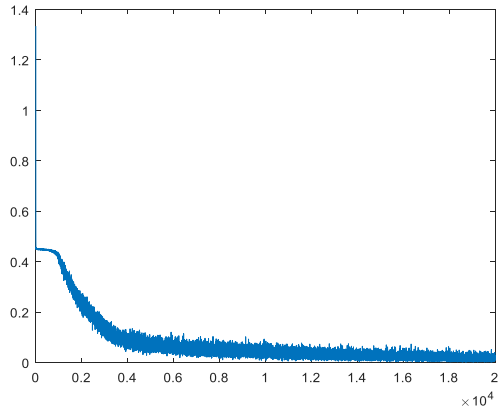
end

Step 4. Return to Step 3 until each w^l converge.

Step 6: Train the Network

Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

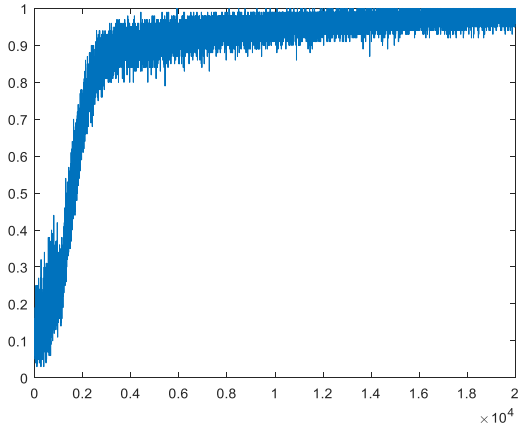



```
% training cost on training batch
J = [J 1/mini_batch*cost(a{L}, y)];
figure
plot(J);
```

Accuracy

$$\text{Acc} = \frac{number\ of\ correct\ prediction}{number\ of\ samples}$$

Use max output as prediction




```
% accuary on training batch
Acc =[Acc accuracy(a{L}, y)];
figure
plot(Acc);
```

Step 7: Test the Network

```
%test on training set
a{1} = X_train;
for l = 1:L-1
  a{l+1} = fc(w{l}, a{l});
end
train_acc = accuracy(a{L}, trainLabels);
fprintf('Accuracy on training dataset is %f%%\n', train_acc*100);
```

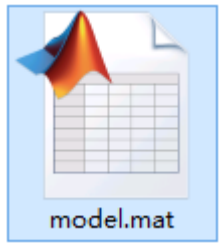
Accuracy on training dataset is 98.160000%

```
%test on testing set
a{1} = X_test;
for l = 1:L-1
   a{l+1} = fc(w{l}, a{l});
end
test_acc = accuracy(a{L}, testLabels);
fprintf('Accuracy on testing dataset is %f%%\n', test_acc*100);
```

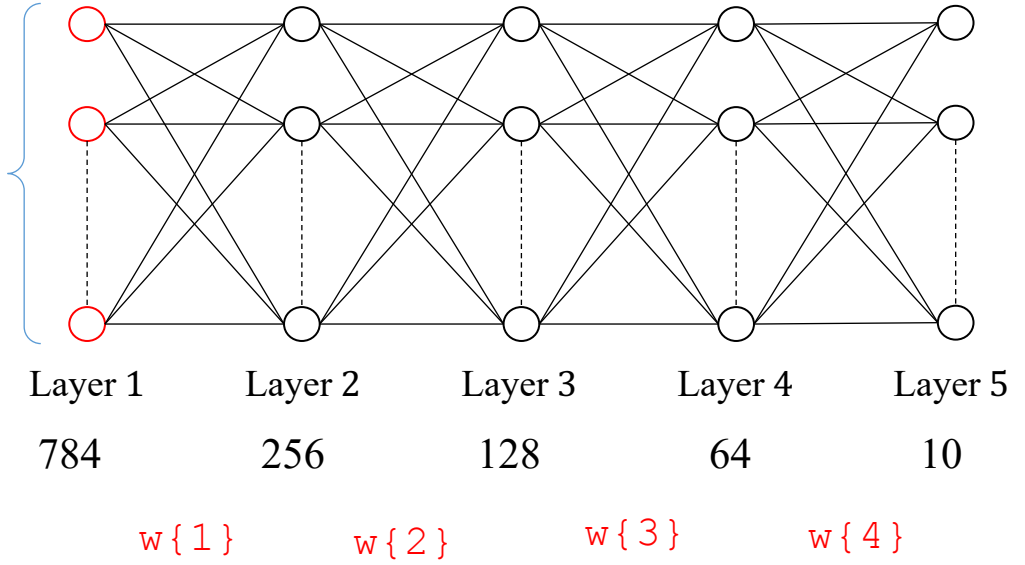
Accuracy on testing dataset is 95.550000%

Experiments: Store the Network Parameters

```
% save model
save model.mat w layer_size
```



This is very important!



Experiments: Code Structure

The main running script

Forward computing function

Backward computing function

Cost function

Evaluation Index

Experiments: Code—main.m

```
%% Step 1: Data Preparation

% loading dataset
load mnist_small_matlab.mat
% trainData: a matrix with size of 28x28x10000
% trainLabels: a matrix with size of 10x10000
% testData: a matrix with size of 28x28x2000
% testLabels: a matrix with size of 10x2000
```

```
% prepare the external input for training set

train_size = 10000; % number of training samples
% input in the 1st layer
X_train = reshape(trainData, 784, train_size);
```

Experiments: Code—main.m

```
% prepare the external input for testing set
test_size = 2000; % number of testing samples
% external input in the 1st layer
X_test = reshape(testData, 784, test_size);
```

```
%% Step 2: Design Network Architecture
% define number of layers
L = 5;
% define number of neurons in each layer
% - 1st column: external neurons
% - 2nd column: internal neurons
layer_size = [784 % number of neurons in 1st layer
              256 % number of neurons in 2nd layer
              128 % number of neurons in 3rd layer
              64  % number of neurons in 4th layer
              10];% number of neurons in 5th layer
```

Experiments: Code—main.m

```
%% Step 3: Initial Parameters

% initialize weights in each layer with Gaussian distribution
for l = 1:L-1
    w{l} = 0.1 * randn(layer_size(l+1,2), sum(layer_size(l,:)));
end

alpha = 0.005; % initialize learning rate

%% Step 4: Define Cost Function
% cost function is defined in cost.m

%% Step 5: Define Evaluation Index
% accuracy defined in accuracy.m
```

Experiments: Code—main.m

```
%% Step 6: Train the Network
J = []; % array to store cost of each mini batch
Acc = []; % array to store accuracy of each mini batch
max_epoch = 200; % number of training epoch
mini_batch = 100; % number of sample of each mini batch

for iter=1:max_epoch
    % randomly permute the indexes of samples in training set
    idxs = randperm(train_size);
    % for each mini-batch
    for k = 1:ceil(train_size/mini_batch)
        % prepare internal inputs in 1st layer denoted by a{1}

        start_idx = (k-1)*mini_batch+1;           % start index of kth mini-batch
        end_idx = min(k*mini_batch, train_size); % end index of kth mini-batch
       a{1} = X_train(:,idxs(start_idx:end_idx));
    % prepare labels
        y = trainLabels(:, idxs(start_idx:end_idx));
```

Experiments: Code—main.m

```
% forward computation
for l=1:L-1
    [a{l+1}, z{l+1}] = fc(w{l}, a{l});
end

% Compute delta of last layer
delta{L} = (a{L} - y).* a{L} .*(1-a{L}); %delta{L}={partial J}/{partial z^L}

% backward computation
for l=L-1:-1:2
    delta{l} = bc(w{l}, z{l}, delta{l+1});
end

% update weight
for l=1:L-1
    % compute the gradient
    grad_w = delta{l+1} * a{l}';
    w{l} = w{l} - alpha*grad_w;
end
```

Experiments: Code—main.m

```
        % training cost on training batch
        J = [J 1/mini_batch*sum(cost(a{L}, y))];
        Acc =[Acc accuracy(a{L}, y)];
    end
end
% end training

% plot training error and accuracy
figure
plot(J);
figure
plot(Acc);
```

Experiments: Code—main.m

```
%% Step 7: Test the Network
%test on training set
a{1} = X_train;
for l = 1:L-1
  a{l+1} = fc(w{l}, a{l});
end
train_acc = accuracy(a{L}, trainLabels);
fprintf('Accuracy on training dataset is %f%%\n', train_acc*100);

%test on testing set
a{1} = X_test;
for l = 1:L-1
   a{l+1} = fc(w{l}, a{l});
end
test_acc = accuracy(a{L}, testLabels);
fprintf('Accuracy on testing dataset is %f%%\n', test_acc*100);

%% Step 8: Store the Network Parameters
save model.mat w layer_size
```

Experiments: Code—fc.m and bc.m

```
% fc.m
% This is forward computation function

function [a_next, z_next] = fc(w, a)
    % define the activation function
    f = @(s) 1 ./ (1 + exp(-s));

    % forward computing
    z_next = w * a;
    a_next = f(z_next);
end
```

```
% bc.m
% This is backward computation function

function delta = bc(w, z, delta_next)

    % activation function
    f = @(s) 1 ./ (1 + exp(-s));

    % derivative of activation  function
    df = @(s) f(s) .* (1 - f(s));

    % backward computing
    delta = (w' * delta_next) .* df(z);

end
```

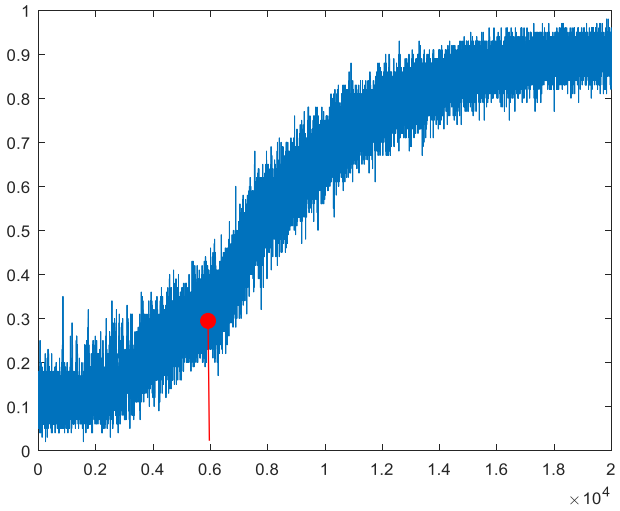
Experiments: Code—cost.m and accuracy.m

```
%% Step 4: Define Cost Function
function [J] = cost(a, y)
    J = 1/2 * sum((a - y).^2);
end
```

```
%% Step5: Define Evaluation Index
function [ acc ] = accuracy( a, y )
    mini_batch = size(a, 2);
    [~,idx_a] = max(a);
    [~,idx_y] = max(y);
    acc = sum(idx_a==idx_y) / mini_batch;
end
```

Results: Learning Rate

```
% learning rate
alpha = 0.001;
```

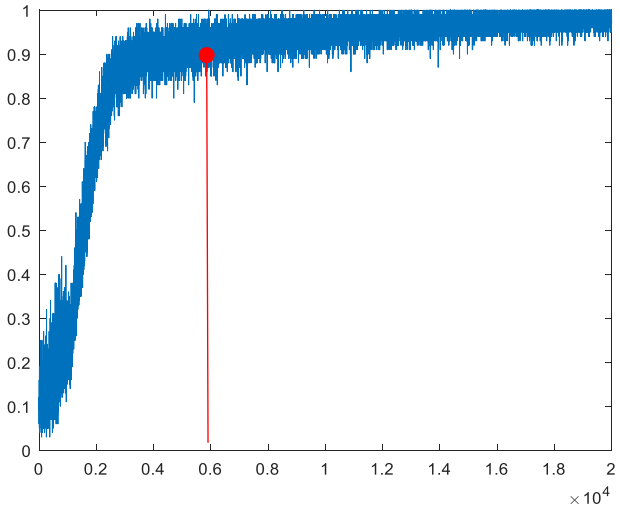


Accuracy

- Training=90.24%
- Testing=89.10%

Too Slow

```
% learning rate
alpha = 0.005;
```

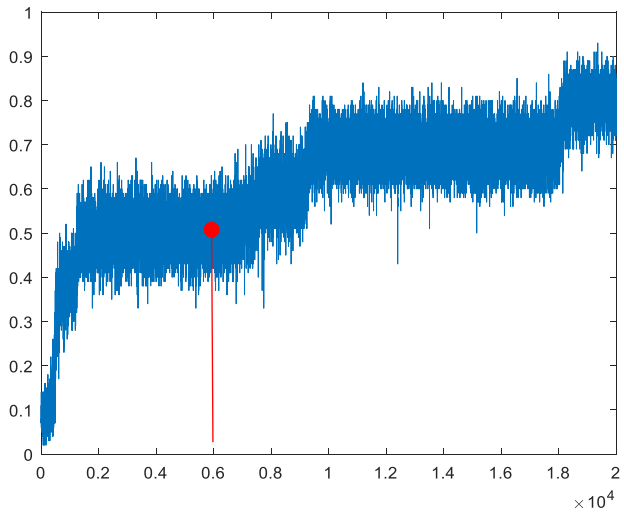


Accuracy

- Training=98.15%
- Testing=95.25%

Good

```
% learning rate
alpha = 0.08;
```



Accuracy

- Training=79.66%
- Testing=76.65%

Too Fast

提纲

Appendix 1: Reference Code by Using MATLAB

Appendix 2: Reference Code by Using Python

Appendix 2: Reference Code by Using Python

Reference Code for Handwritten Digits Recognition by Using Python

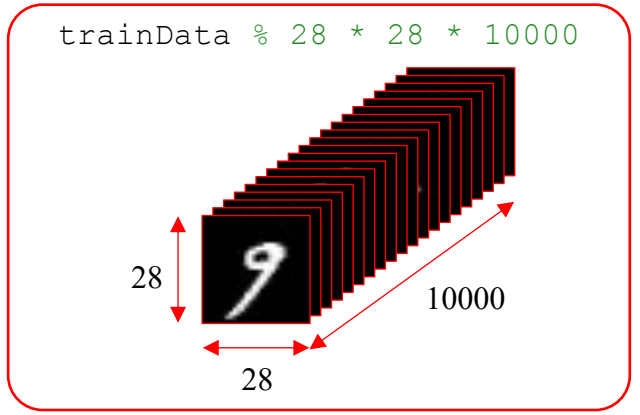
名称	值
trainData	28x28x10000 double
testData	28x28x2000 double
trainLabels	10x10000 double
testLabels	10x2000 double

Step 1: Data Preparation

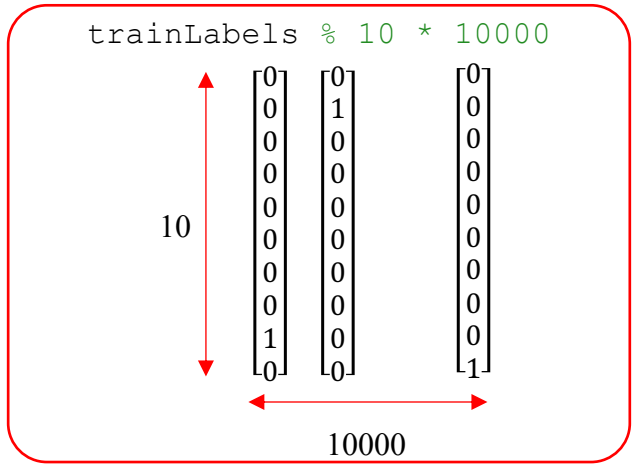
trainData	28x28x10000 double
testData	28x28x2000 double
trainLabels	10x10000 double
testLabels	10x2000 double

```
import math
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
## Step 1: Data Preparation

# loading dataset
m = loadmat("./mnist_small_matlab.mat")
trainData, trainLabels = m['trainData'], m['trainLabels']
testData, testLabels = m['testData'], m['testLabels']
# trainData: a matrix with size of 28x28x10000
# trainLabels: a matrix with size of 10x10000
# testData: a matrix with size of 28x28x2000
# testLabels: a matrix with size of 10x2000
```

Training Data



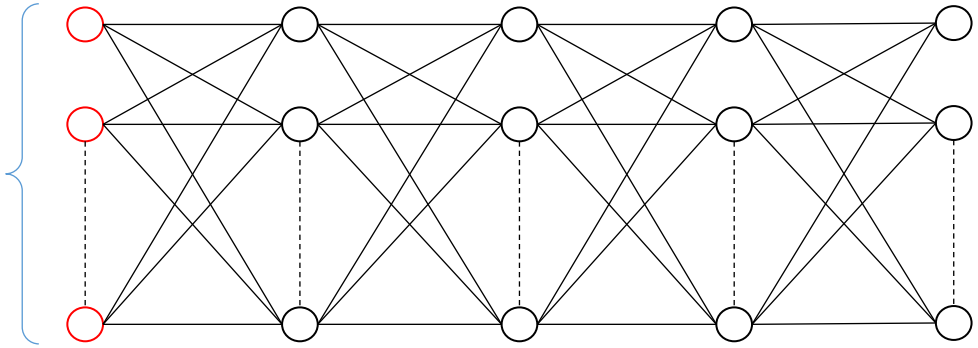
Label



Step 1: Data Preparation

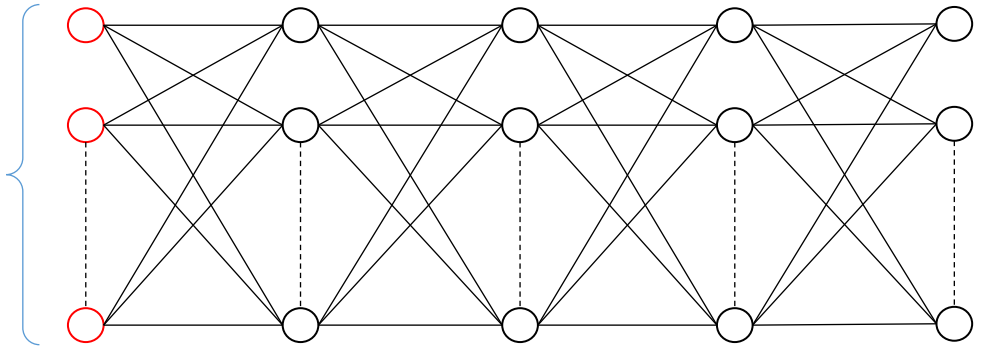
```
# prepare the external input for training set

train_size = 10000 # number of training samples
# input in the 1st layer
X_train = trainData.reshape(-1, train_size)
```

```
# prepare the external input for testing set

test_size = 2000 # number of testing samples
# external input in the 1st layer
X_test = testData.reshape(-1, test_size)
```



Step 2: Network Architecture Design

```
# define number of layers
L = 5

# define number of neurons in each layer
# - 1st column: external neurons
# - 2nd column: internal neurons
layer_size = [784, # number of neurons in 1st layer
              256, # number of neurons in 2nd layer
              128, # number of neurons in 3rd layer
              64,  # number of neurons in 4th layer
              10]  # number of neurons in 5th layer
```



Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
784	256	128	64	10

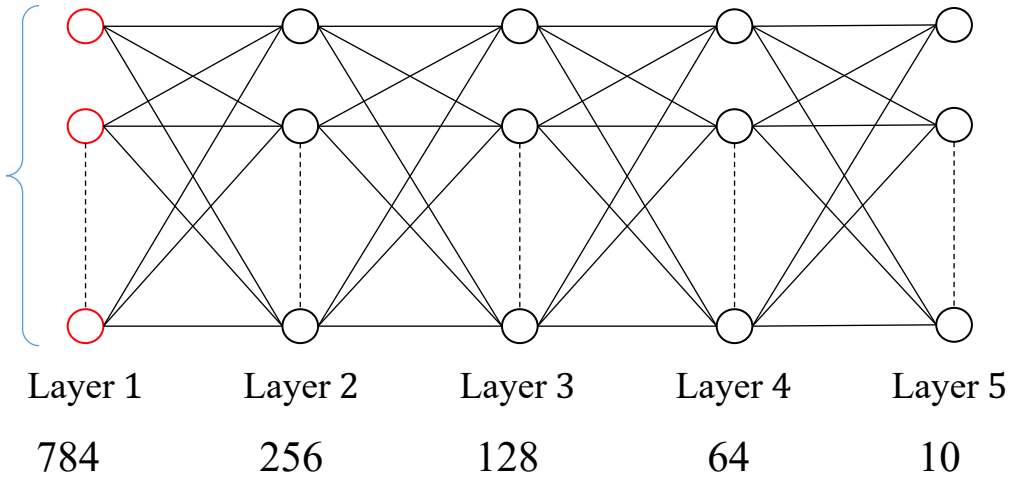
Step 3: Initializing Network Parameters

Gaussian distribution: $w_{ij}^{l} \sim N(0,1)$

```
# initialize weights
w = {}
for l in range(1, L):
    w[l] = 0.1 * np.random.randn(layer_size[l], layer_size[l-1])
```

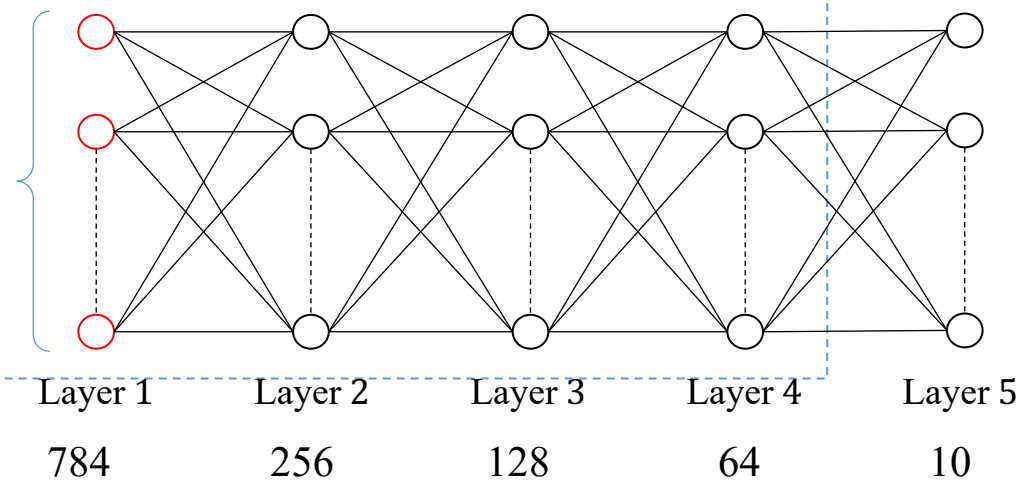
Initialize learning rate

```
alpha = 0.05 # initialize learning rate
```



Step 4, 5: Define Cost Function and Evaluation Index

```python
## Step 4: Define Cost Function
def cost(a, y):
    J = 1/2 * np.sum((a - y)**2)
    return J

## Step 5: Define Evaluation Index
def accuracy(a, y):
    mini_batch = a.shape[1]
    idx_a = np.argmax(a, axis=0)
    idx_y = np.argmax(y, axis=0)
    acc = sum(idx_a==idx_y) / mini_batch
    return acc
```



Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2$$

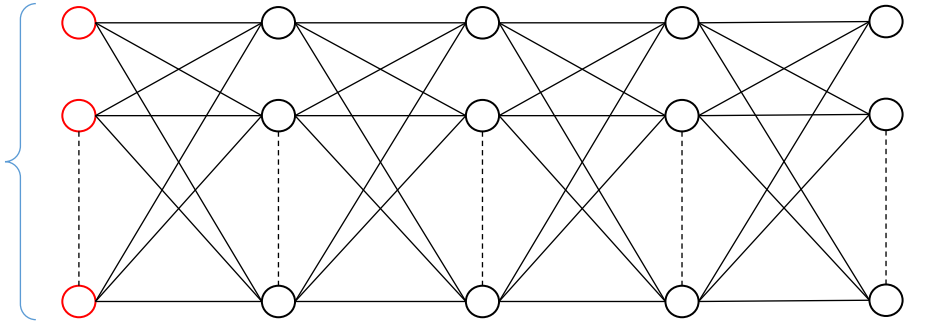
$$\text{Acc} = \frac{\text{number of correct prediction}}{\text{number of samples}}$$

Step 6: Train the Network

```
## Step 6: Train the Network
J = [] # array to store cost of each mini batch
Acc = [] # array to store accuracy of each mini batch
max_epoch = 100 # number of training epochs
mini_batch = 100 # number of sample in each mini batch
```

```
# loop until converge
for epoch_num in range(max_epoch):
    # for each mini-batch
        # batch forward computation
        # batch backward computation
        # cumulate and update weight
```

Mini-batch BP implement



BP Algorithm:

Step 1. Input the training data set $D = \{(x, y)\}$

Step 2. Initialize each w_{ij}^{l}, and choose a learning rate α.

Step 3. for each mini-batch sample $D_m \subseteq D$

for each $x \in D_m$

$a^1 \leftarrow x \in D_m$;

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

$\frac{\partial J}{\partial w_{ji}^l} \leftarrow \frac{\partial J}{\partial w_{ji}^l} + \delta_j^{l+1} \cdot a_i^l$;

end

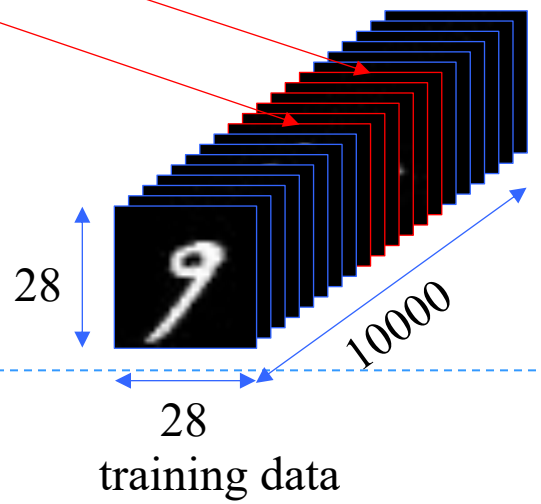
$w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$;

end

Step 4. Return to Step 3 until each w^l converge.

Step 6: Train the Network

```
# randomly permutate the indexes of samples in training set
idxs = np.random.permutation(train_size)
# for each mini-batch
for k in range(math.ceil(train_size//mini_batch)):    # prepare internal inputs
in 1st layer denoted by a{1}
    a, z, delta = {}, {}, {}
    start_idx = k*mini_batch                          # start index of kth mini-batch
    end_idx = min((k+1)*mini_batch, train_size) # end index of kth mini-batch
    a[1] = X_train[:,idxs[start_idx:end_idx]]

    # prepare labels
    y = trainLabels[:, idxs[start_idx:end_idx]]
    # forward computation
    # ...
    # cost function
    # ...
    # backward computation
    # ...
    # update weight
    # ...
```



Step 6: Train the Network

```
# forward computation
for l in range(1, L):
  a[l+1], z[l+1] = fc(w[l], a[l])


# Compute delta of last layer
delta[L] = (a[L] - y) * a[L] * (1-a[L])


# backward computation
for l in range(L-1, 1, -1):
  delta[l] = bc(w[l], z[l], delta[l+1])


# update weight
for l in range(1, L):
  # compute the gradient
  grad_w = np.dot(delta[l+1], a[l].T)
  w[l] = w[l] - alpha*grad_w
```

BP Algorithm:

Step 1. Input the training data set $D = \{(x, y)\}$

Step 2. Initialize each w_{ij}^{l}, and choose a learning rate α.

Step 3. for each mini-batch sample $D_m \subseteq D$

for each $x \in D_m$

$a^1 \leftarrow x \in D_m$;

for $l = 2: L$

$a^{l+1} \leftarrow fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

$\frac{\partial J}{\partial w_{ji}^{l}} \leftarrow \frac{\partial J}{\partial w_{ji}^{l}} + \delta_j^{l+1} \cdot a_i^l$;

end

$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$;

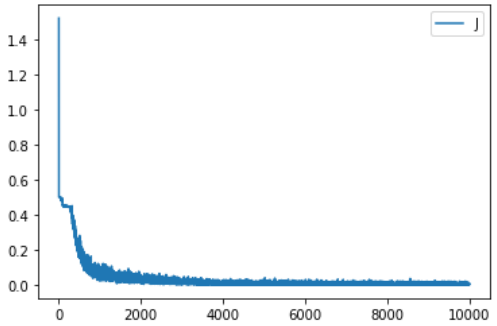
end

Step 4. Return to Step 3 until each w^l converge.

Step 6: Train the Network

Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

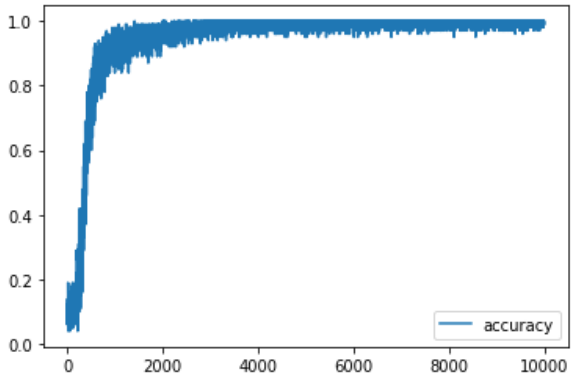



```
# training cost on training batch
J.append(cost(a[L], y)/mini_batch)
plt.figure()
plt.plot(J)
```

Accuracy

$$\text{Acc} = \frac{number\ of\ correct\ prediction}{number\ of\ samples}$$

Use max output as prediction




```
# accuary on training batch
Acc.append(accuracy(a[L], y))
plt.figure()
plt.plot(Acc)
```

Step 7: Test the Network

```
#test on training set
a[1] = X_train
for l in range(1,L):
  a[l+1],_ = fc(w[l], a[l])
train_acc = accuracy(a[L], trainLabels)
print('Accuracy on training dataset is {}%'.format(train_acc*100))
```

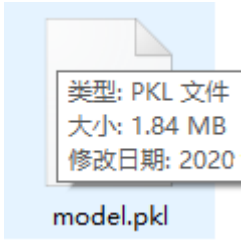
Accuracy on training dataset is 100.000000%

```
#test on testing set
a[1] = X_test
for l in range(1,L):
   a[l+1],_ = fc(w[l], a[l])
test_acc = accuracy(a[L], testLabels)
print('Accuracy on testing dataset is {}%'.format(test_acc*100))
```

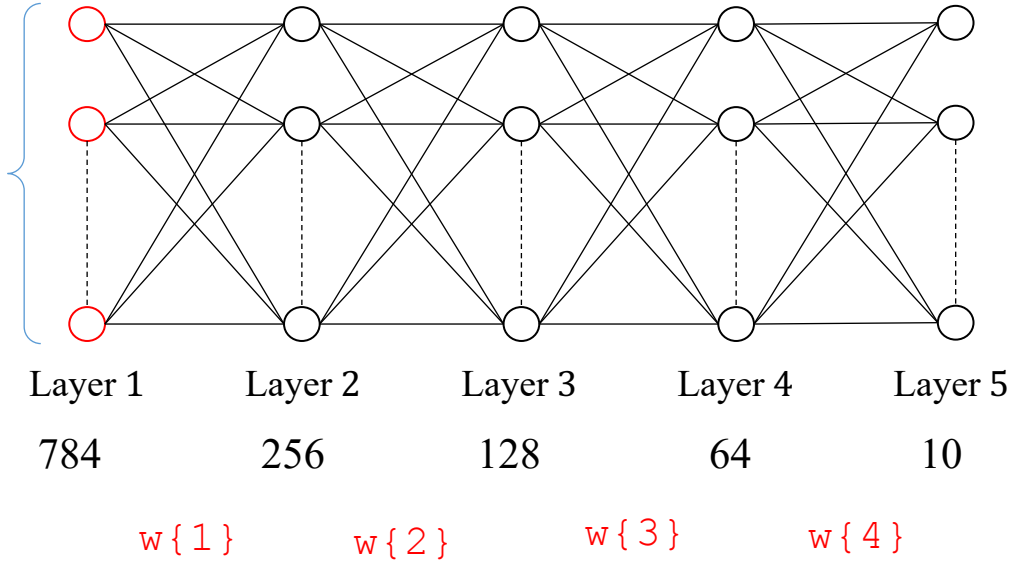
Accuracy on testing dataset is 96.250000%

Step 8: Store the Network Parameters

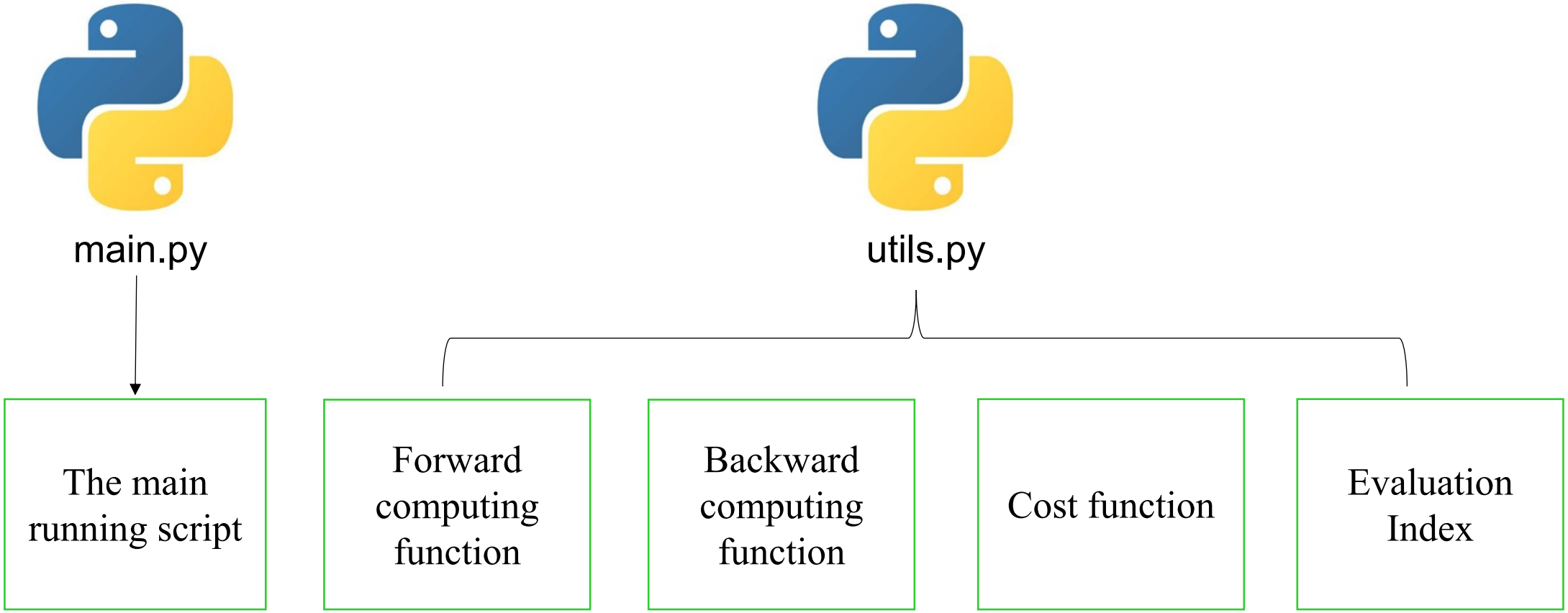
```
# save model
import pickle
with open('model.pkl', 'wb') as f:
    pickle.dump([w, layer_size], f)
```



This is very important!



Experiments: Code Structure



main.py
Line 1-20

```python
import math, pickle
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from utils import cost, accuracy, fc, bc
m = loadmat("./mnist_small_matlab.mat")
trainData, trainLabels = m['trainData'], m['trainLabels']
testData, testLabels = m['testData'], m['testLabels']
train_size ,test_size = 10000, 2000
X_train = trainData.reshape(-1, train_size)
X_test = testData.reshape(-1, test_size)
layer_size = [784, 256, 128, 64, 10]
L = len(layer_size)
w = {}
for l in range(1, L):
    w[l] = 0.1 * np.random.randn(layer_size[l], layer_size[l-1])
alpha = 0.05
J, Acc = [], []
max_epoch = 20
mini_batch = 100
```

```python
for epoch_num in range(max_epoch):
    idxs = np.random.permutation(train_size)
    for k in range(math.ceil(train_size/mini_batch)):
        start_idx = k*mini_batch
        end_idx = min((k+1)*mini_batch, train_size)
        a, z, delta = {}, {}, {}
        batch_indices = idxs[start_idx:end_idx]
        a[1] = X_train[:, batch_indices]
        y = trainLabels[:, batch_indices]
        for l in range(1, L):
            a[l+1], z[l+1] = fc(w[l], a[l])
        delta[L] = (a[L] - y) * (a[L]*(1-a[L]))
        for l in range(L-1, 1, -1):
            delta[l] = bc(w[l], z[l], delta[l+1])
        for l in range(1, L):
            grad_w = np.dot(delta[l+1], a[l].T)
            w[l] = w[l] - alpha*grad_w
        J.append(cost(a[L], y)/mini_batch)
        Acc.append(accuracy(a[L], y))
    a[1] = X_test
    y = testLabels
    for l in range(1, L):
        a[l+1], z[l+1] = fc(w[l], a[l])
    print(epoch_num, "training acc:", Acc[-1], 'test acc:', accuracy(a[L], y))
```

main.py
Line 12-44

```
plt.figure()
plt.plot(J)
plt.savefig("J.png")
plt.close()
plt.figure()
plt.plot(Acc)
plt.savefig("Acc.png")
plt.close()
model_name = 'model.pkl'
with open(model_name, 'wb') as f:
    pickle.dump([w, layer_size], f)
print("model saved to {}".format(model_name))
```

main.py
Line 45-56

```
import numpy as np
f = lambda s : 1 / (1 + np.exp(-s))
df = lambda s : f(s) * (1-f(s))
def cost(a, y):
    J = 1/2 * np.sum((a - y)**2)
    return J
def accuracy(a, y):
    mini_batch = a.shape[1]
    idx_a = np.argmax(a, axis=0)
    idx_y = np.argmax(y, axis=0)
    acc = sum(idx_a==idx_y) / mini_batch
    return acc
def fc(w, a):
    z_next = np.dot(w, a)
    a_next = f(z_next)
    return a_next, z_next
def bc(w, z, delta_next):
    delta = np.dot(w.T, delta_next) * df(z)
    return delta
```

utils.py

课程信息

四川大学 SICHUAN UNIVERSITY 教务处

首页 部门介绍 教师发展 教研教改 人才培养 实践教学 合作交流 教学运行 学籍管理 信息公开 资料下载

当前位置：首页 > 通知公告 > 正文

关于2022年中秋节、国庆节放假本科教学安排有关事项的通知

添加时间：2022-09-09 发布者： 点击次数：23550

校内各开课单位：

根据学校《关于2022年中秋节、国庆节放假有关事项的通知》精神，我校本科教学安排如下：

一、关于放假及课程调整说明

1.中秋节：9月10日至12日放假，共3天。

2.国庆节：10月1日至7日放假调休，共7天。

10月8日（星期六）、10月9日（星期日）行课，学生课程作相应调整，课表上10月6日（星期四）的课程调至10月8日（星期六），10月7日（星期五）的课程调至10月9日（星期日）。

二、法定节假日课程停课原则不补，如需补课请联系教务处运行科协调补课的时间和教室。

教务处

2022.9.9

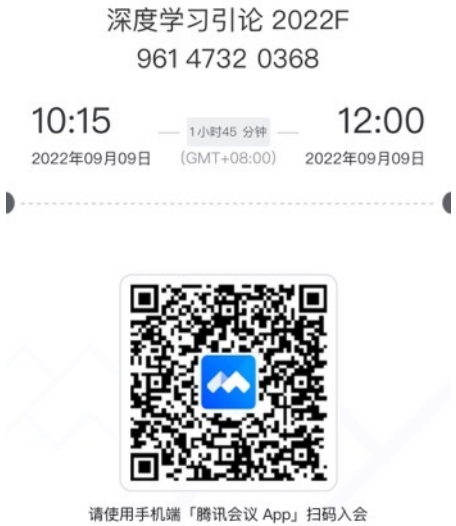
（星期五）的课程调至10月9日（星期日）。

课程信息

时间：2022年秋季学期 1-8周 周五 3-4节

线下：江安文科楼三区203

线上：



http://www.machineilab.org/

http://guoquan.net/

Thanks