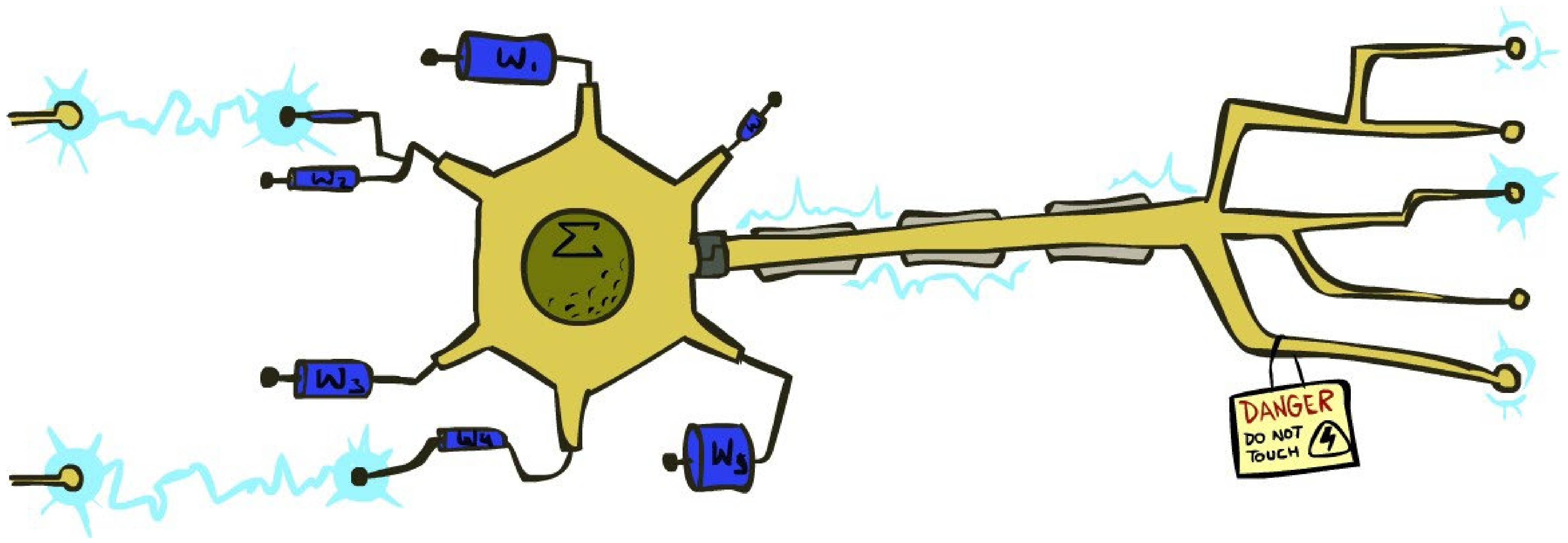
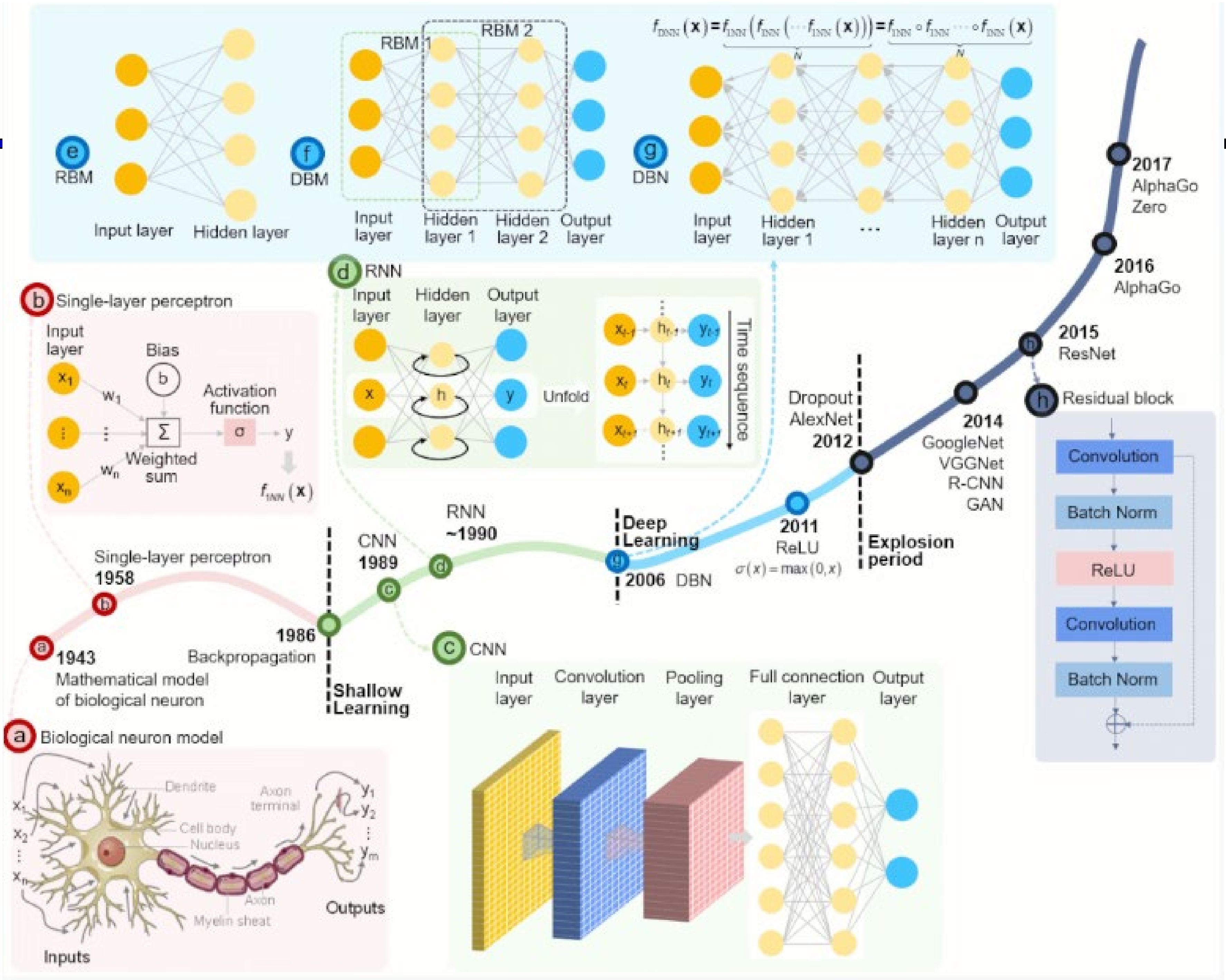
# 第21章 深度学习



# A brief history

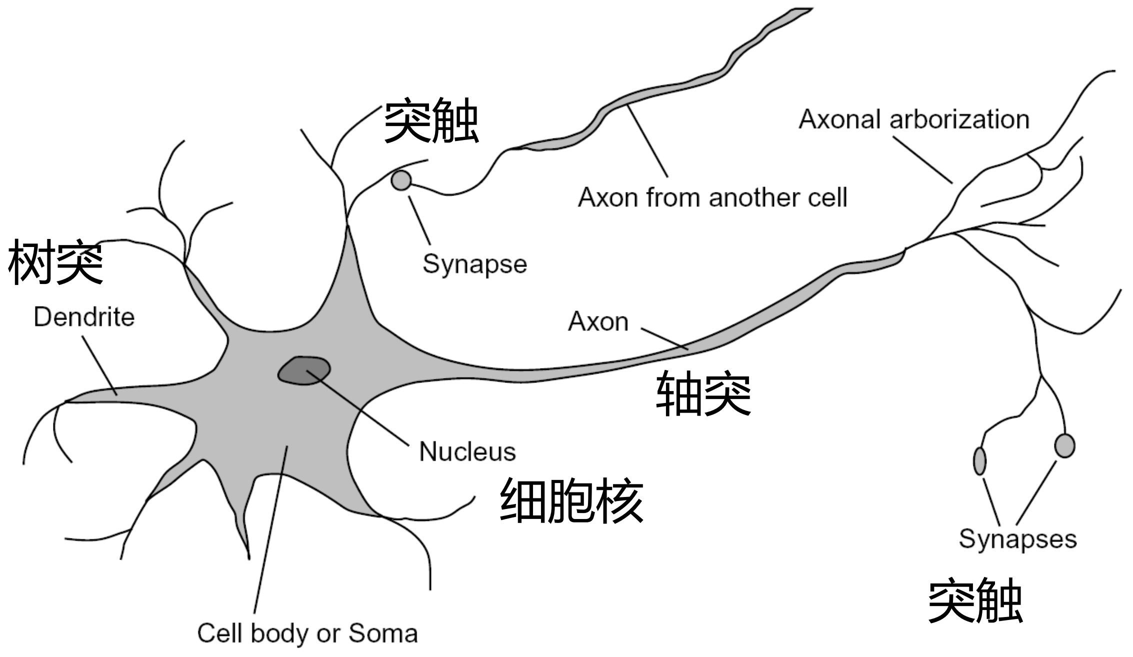


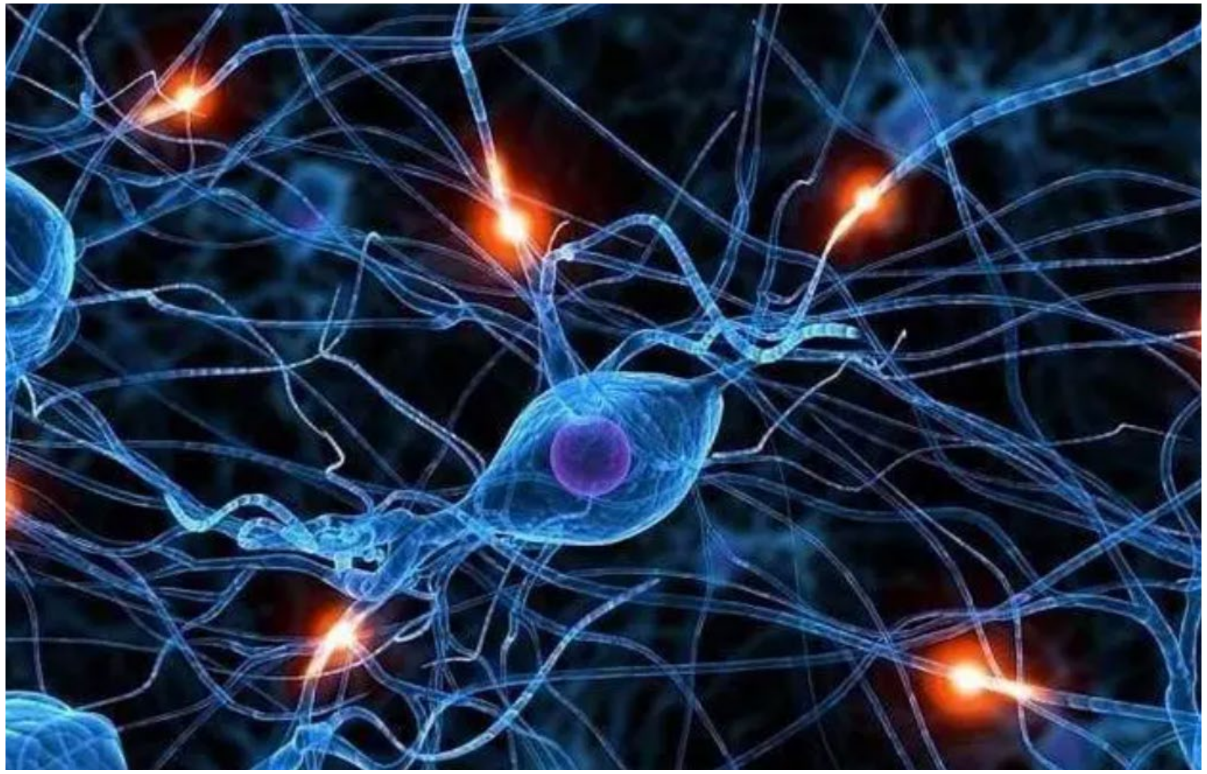
## Outline

- 21.1 简单前馈网络
  - M-P模型、感知机与多层感知机
  - 神经网络的表示能力
  - 神经网络中的学习(BP算法)

## Some (Simplified) Biology

### 灵感：人类神经元





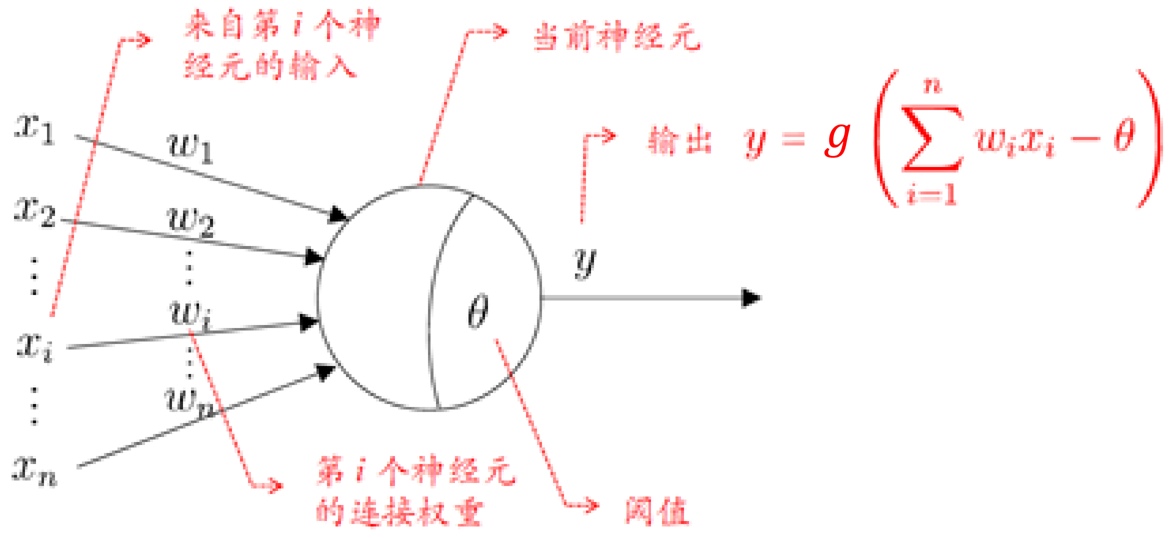
人类的大脑大概有几百亿个神经元，神经元与神经元之间通过突触“链接”在一起

# M-P 神经元模型 [McCulloch and Pitts, 1943]

线性分类

$y=g(w_0+w_1x_1+w_2x_2+\ldots)$

- 输入：来自其他$n$个神经元传递过来的输入信号（特征值）
- 处理：输入信号通过带权重w的连接进行传递, 神经元接受到总输入值将与神经元的阈值θ进行比较
- 输出：通过激活函数$g$的处理得到输出

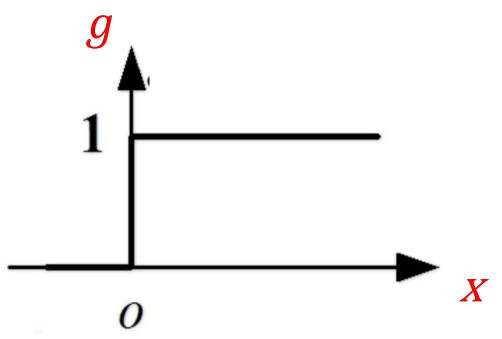


来自第 $i$ 个神经元的输入
当前神经元
输出 $y = g\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$
$x_1$
$x_2$
$x_i$
$x_n$
$w_1$
$w_2$
$w_i$
$w_n$
$\theta$
$y$
第 $i$ 个神经元的连接权重
阈值

M-P 神经元模型

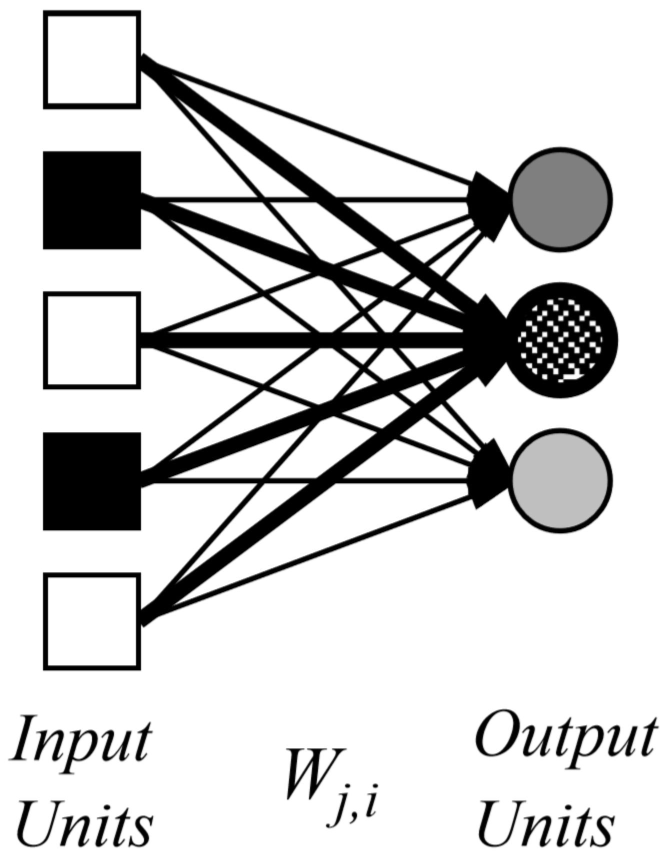
阈值激活函数: *阶跃函数*

*g(x)=sign(x)*



MP模型是一个计算模型。给定的参数w、θ

# 感知机（Perceptron）



一个感知机网络

- Rosebblatt提出感知机，最早的人工神经网络模型。
- 感知机输入层接受外界输入信号传递给输出层, 输出层是多个M-P神经元（阈值逻辑单元）
- 可学习：根据训练数据来调整神经元之间的“连接权”以及每个功能神经元的“阈值”

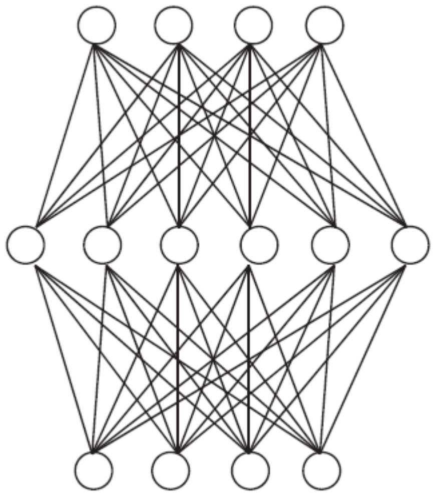
Rosebblatt, The perceptron: a probabilistic model for information storage and organization in the brain[J]. Psychological review, 1958, 65(6): 386-408

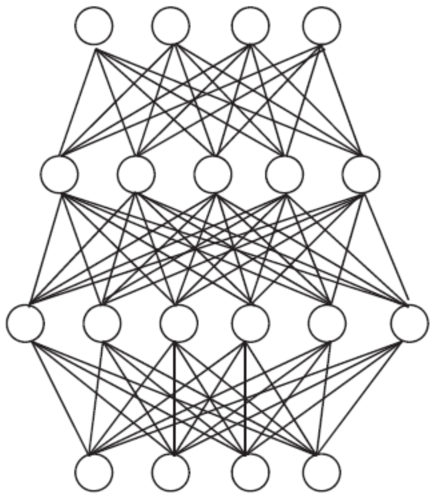
# 多层感知机MLP

## 多层前馈神经网络

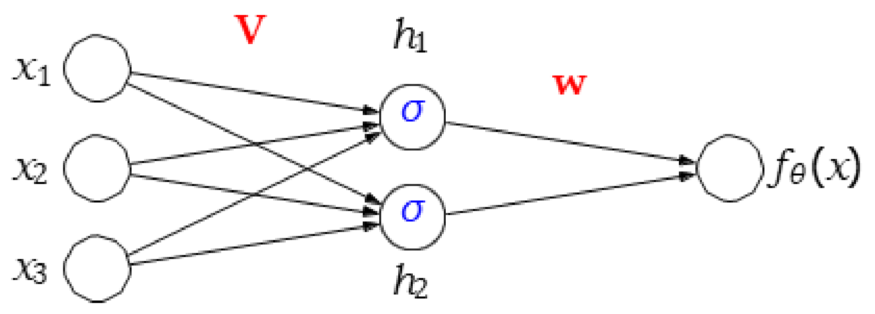
- 定义：每层神经元与下一层神经元全互联，神经元之间不存在同层连接也不存在跨层连接
- 前馈：输入层接受外界输入，隐藏层与输出层神经元对信号进行加工，最终结果由输出层神经元输出
- 学习：根据训练数据来调整神经元之间的"连接权"以及每个功能神经元的"阈值"

- 隐藏层和输出层神经元都是具有激活函数的功能神经元

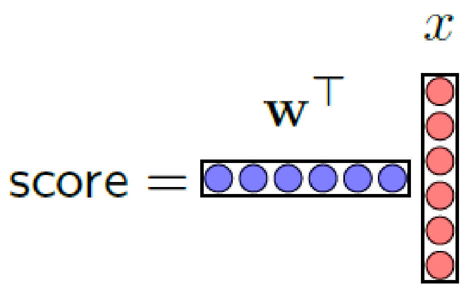

(a) 单隐层前馈网络


(b) 双隐层前馈网络

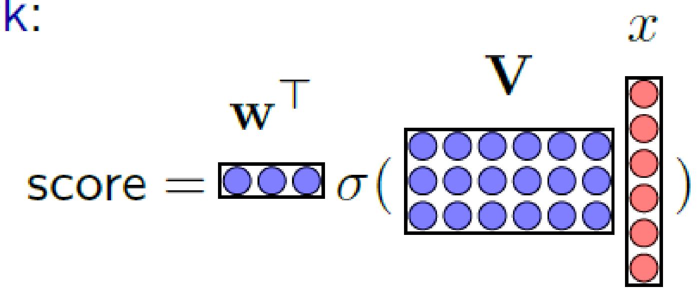
# 多层感知机



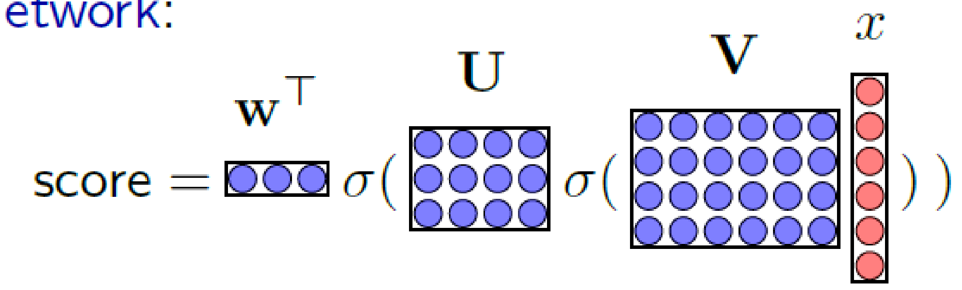
1-layer neural network:



2-layer neural network:



3-layer neural network:



score $= \mathbf{w}^\top \sigma(\mathbf{U}\, \sigma(\mathbf{V}\, x))$

Parameters: $(\mathbf{V}, \mathrm{U}, \ldots\ldots, \mathbf{w})$

… …

# Feature Vectors

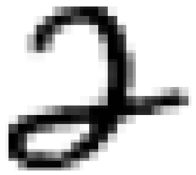
原始数据 $x$ $y$

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

```
# free         : 2
YOUR_NAME      : 0
MISSPELLED     : 2
FROM_FRIEND    : 0
...
```

SPAM
or
+

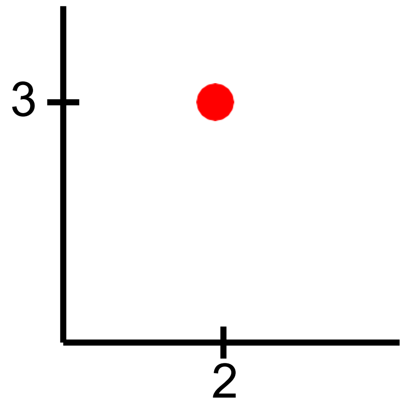


```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

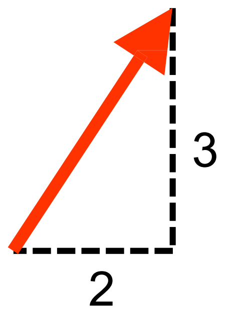
"2"

# Review: Vectors

- A tuple like (2,3) can be interpreted two different ways:



A **point** on a coordinate grid

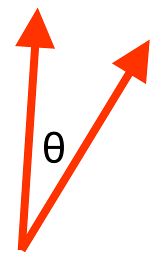


A **vector** in space. Notice we are not on a coordinate grid.

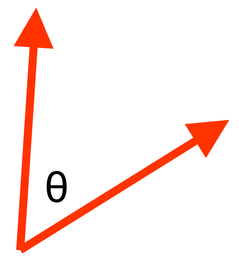
- A tuple with more elements like (2, 7, -3, 6) is a point or vector in higher-dimensional space (hard to visualize)

# Review: Vectors

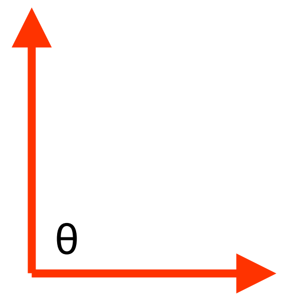
- Definition of dot product:
  - a · b = |a| |b| cos(θ)
  - θ is the angle between the vectors a and b
- Consequences of this definition:
  - Vectors closer together
    = “similar” vectors
    = smaller angle θ between vectors
    = larger (more positive) dot product
  - If θ < 90°, then dot product is positive
  - If θ = 90°, then dot product is zero
  - If θ > 90°, then dot product is negative



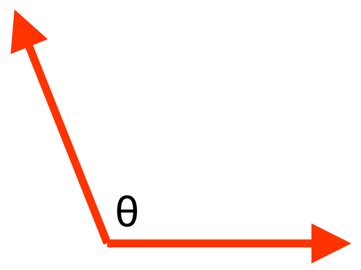
a · b large, positive



a · b small, positive



a · b zero



a · b negative

# Weights

*Dot product* $w \cdot x$ *positive means the positive class (spam)*

$$w \cdot x_1$$

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$$w \cdot x_2$$

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```

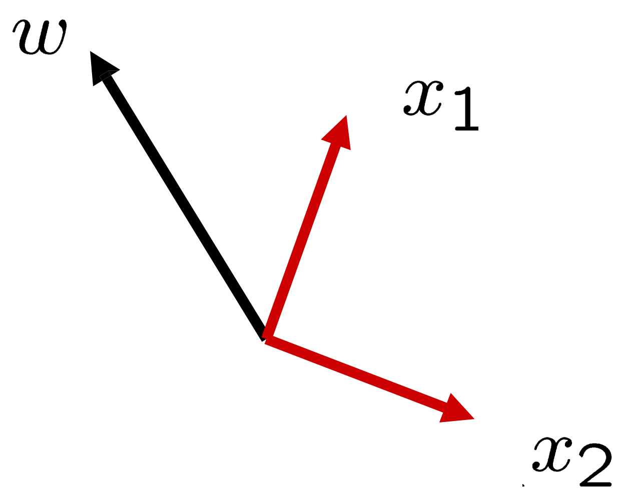
```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

Do these weights make sense for spam classification?

# Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```



```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

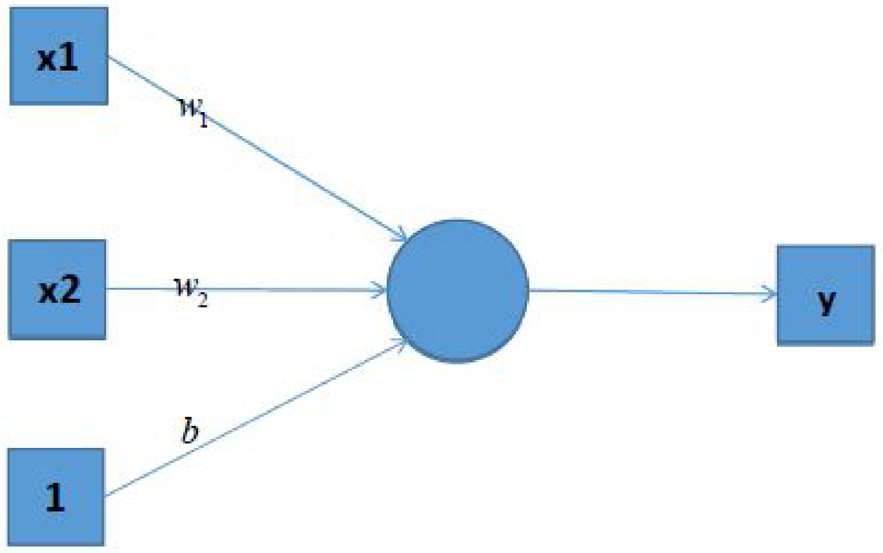
*Dot product positive means the positive class*

# Outline

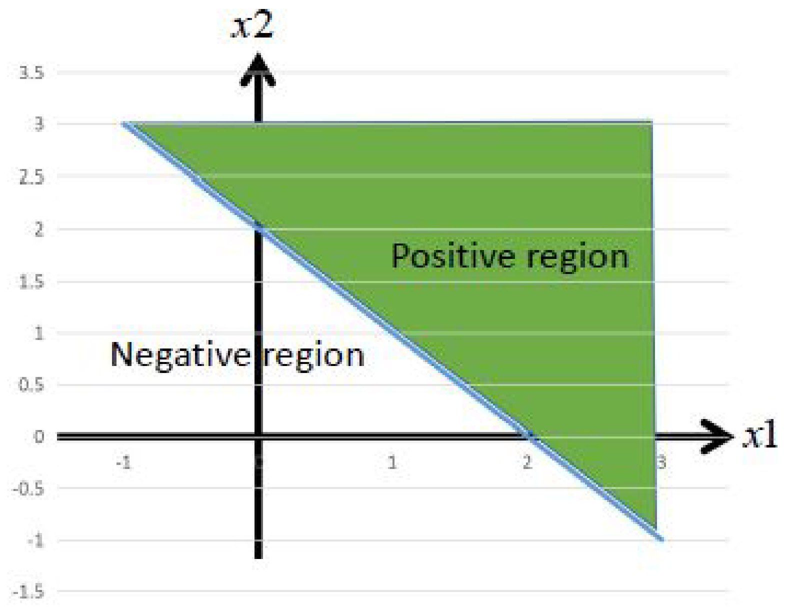
- 21.1 简单前馈网络
  - M-P模型、感知机与多层感知机
  - 神经网络的表示能力
  - 神经网络中的学习(BP算法)

# 感知机的表示能力

Perceptron



$$y = w_1x_1 + w_2x_2 + b$$



$$w_1 = 1, w_2 = 1, b = -2$$

一个采用阈值激活函数的感知机，是线性可分的函数

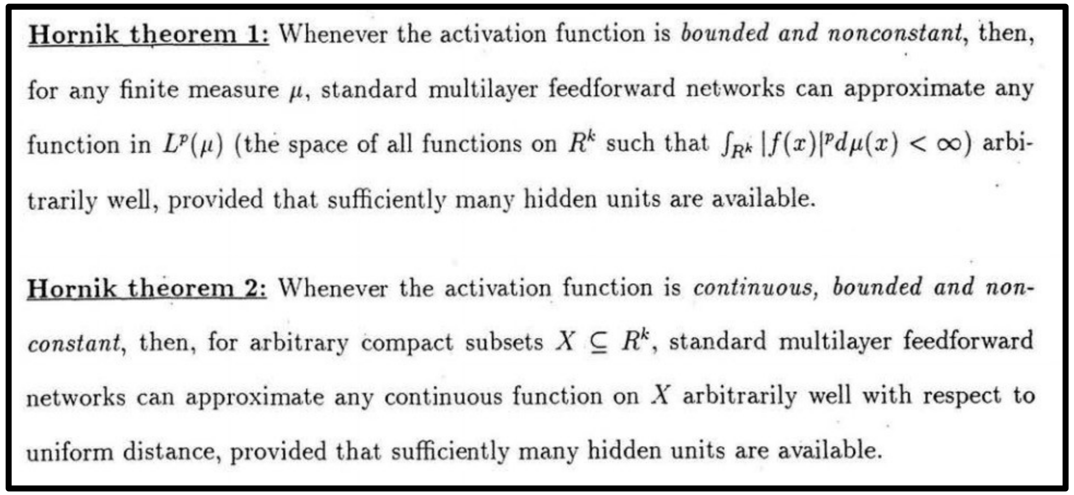
# 神经网络的表示能力

- 万能近似定理

只需要一个包含足够多神经元的隐层, 多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数[Hornik et al.， 1989]

利用两个隐藏层，甚至能表示非连续的函数。

不幸的是，对于特定的网络结构，很难精确刻画什么函数可以表示、什么函数不能表示。



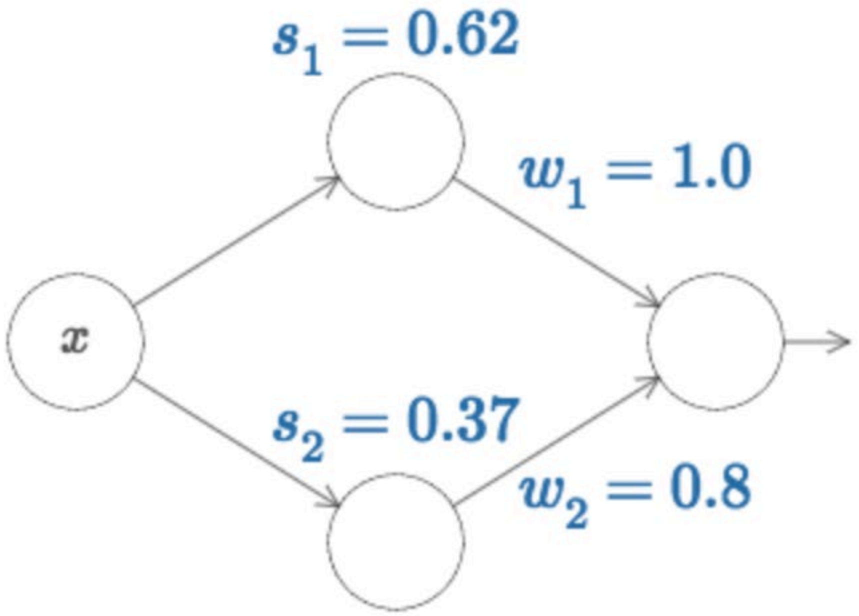
Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure $\mu$, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on $R^k$ such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

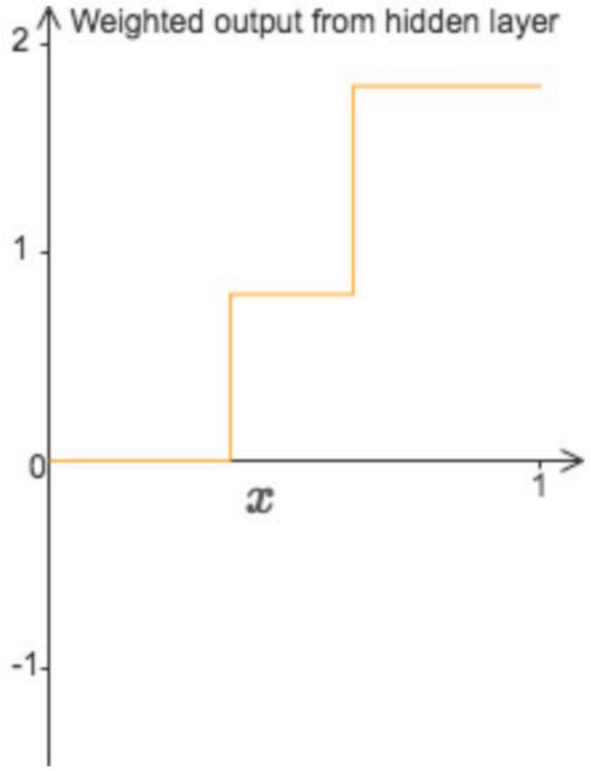
Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on $X$ arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

Hornik, K., Stinchcombe, M. B., and White, H. (1989). Multilayer feedforward networks are universal approximators.
Cybenko (1989) “Approximations by superpositions of sigmoidal functions”

# 神经网络可以模拟任意函数

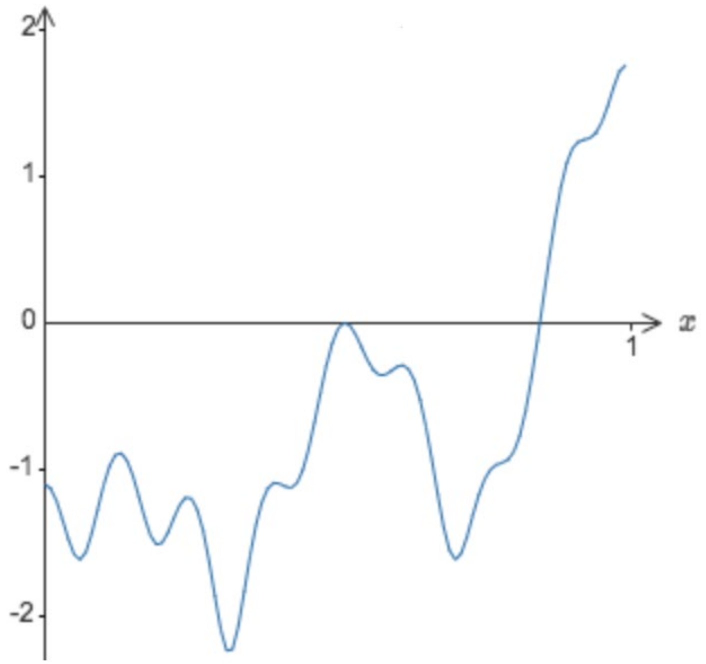
一个阶跃函数

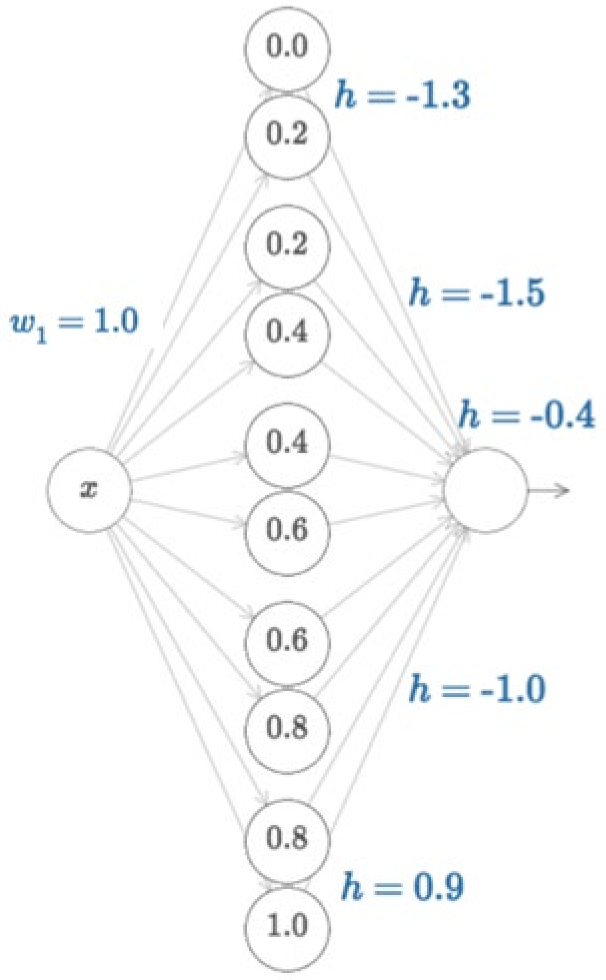


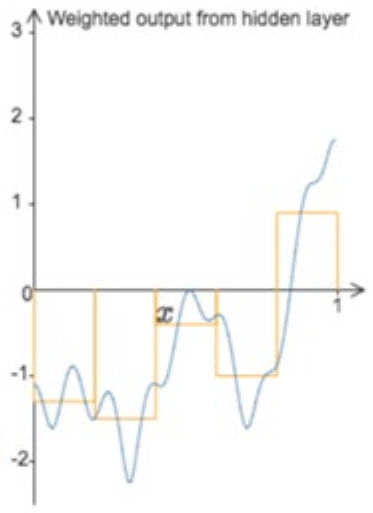


# 神经网络模拟函数

曲线函数:





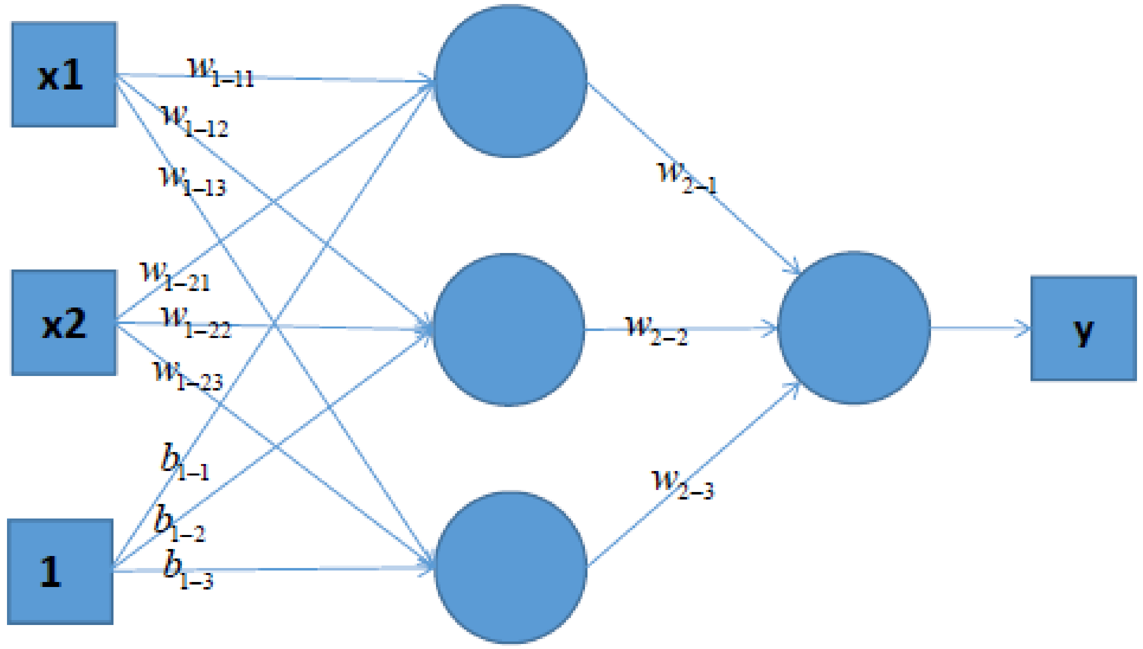


阈值激活函数: *sign(x)*

本质上，使用一个单层神经网络构建了一个lookup表，不同区间对应不同的值，区间分的越细小，就越准确。

# 多层感知机的表示能力

## Perceptron with one hidden layer

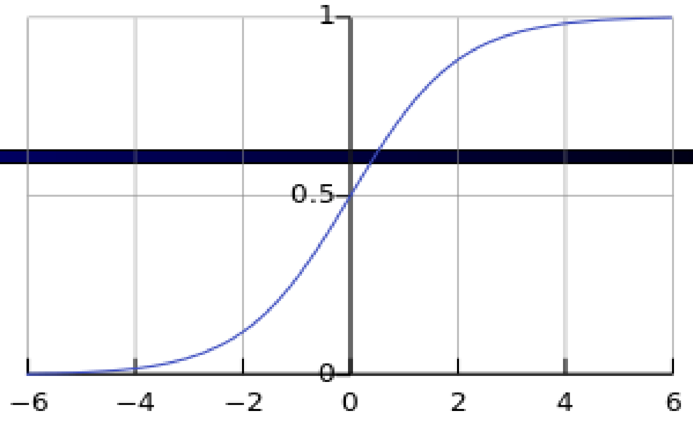


$$
\begin{aligned}
y = {} & w_{2\text{-}1}(w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}) \\
& + w_{2\text{-}2}(w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}) \\
& + w_{2\text{-}3}(w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3})
\end{aligned}
$$

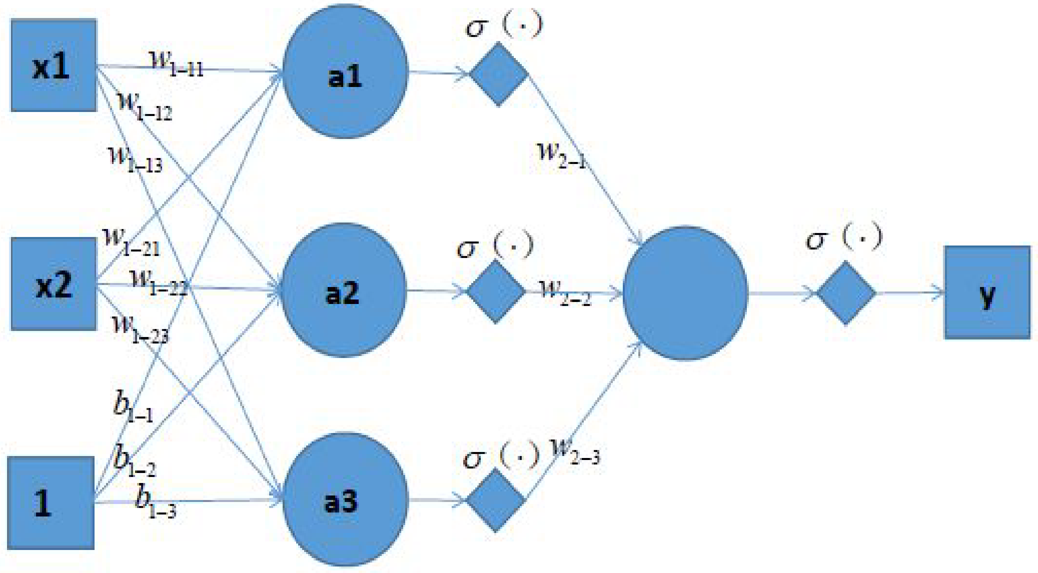
采用阈值激活函数的MLP（复杂的线性组合），输出还是一个线性方程

问题本身是非线性的，如何解决？

# MLP+非线性激活函数



Perceptron with non-linear activation function



*σ: sigmoid function*

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

$$a1 = w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}$$

$$a2 = w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}$$

$$a3 = w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3}$$

$$y = \sigma(w_{2-1}\sigma(a1) + w_{2-2}\sigma(a2) + w_{2-3}\sigma(a3))$$

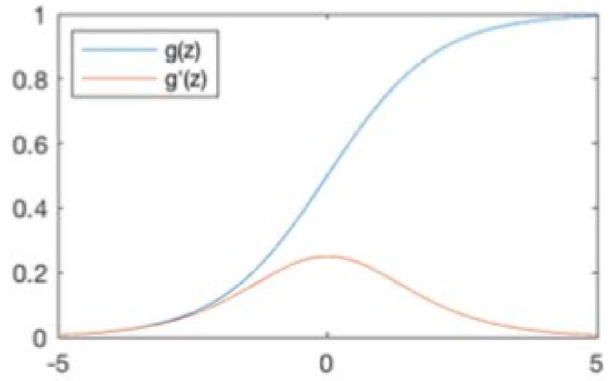
具有非线性激活函数,使得神经网络的表达能力更加强大

# 非线性激活函数

线性分类

$y=g(w_0+w_1x_1+w_2x_2+\dots)$

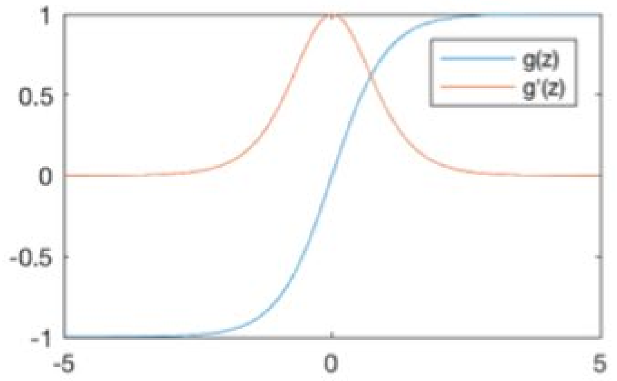
Sigmoid Function



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

$$g'(z)=g(z)(1-g(z))$$

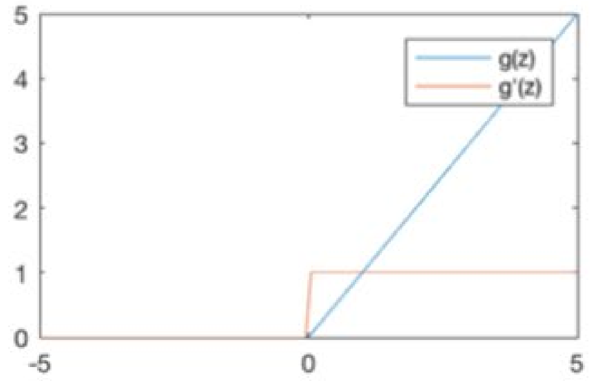
Hyperbolic Tangent



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

$$g'(z)=1-g(z)^2$$

Rectified Linear Unit (ReLU)

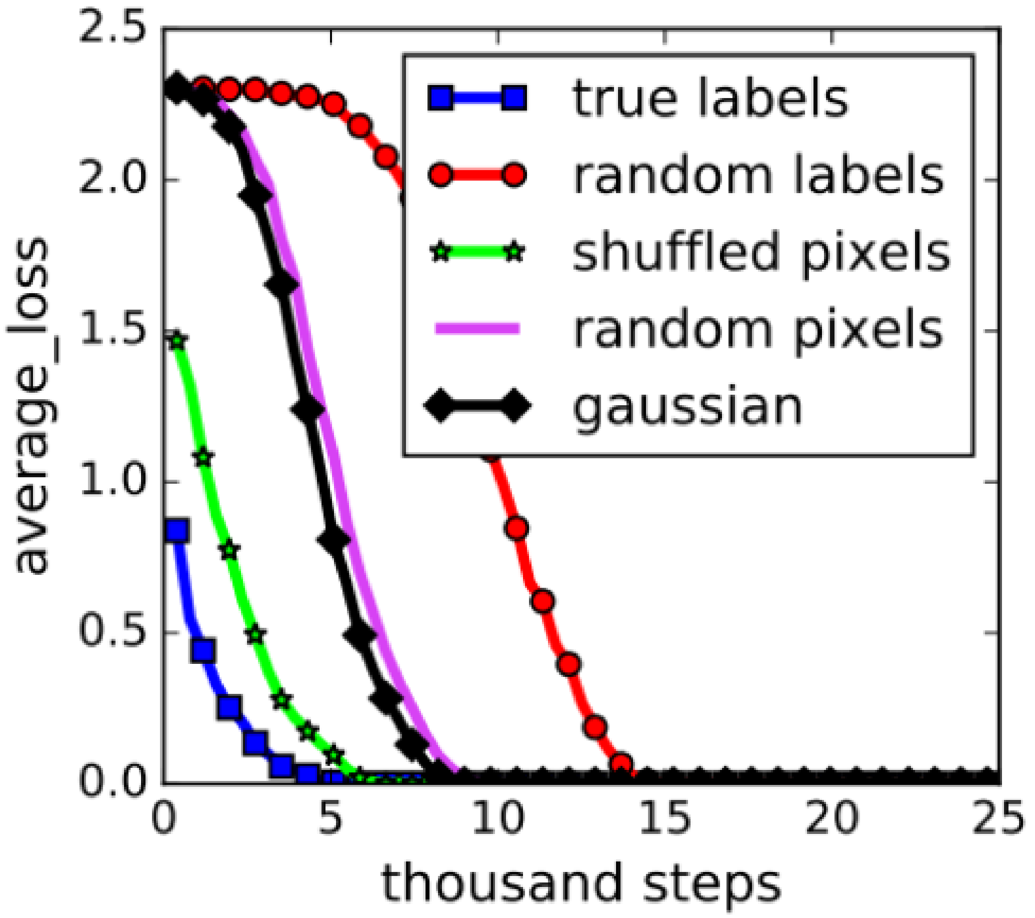


$$g(z)=\max(0,z)$$

$$g'(z)=\begin{cases}1, & z>0\\ 0, & \text{otherwise}\end{cases}$$

# 拟合能力

- **"深层神经网络轻松拟合随机标签"**。可以使任何组的输入拟合任何组的输出，并实现0训练错误。 这使我们得出结论，一个足够大的DNN可以简单地拟合任意数据。



**泛化效果**：Random Label 在测试集不可能有任何提升，50%上下

《Understanding deep learning requires rethinking generalization》，ICLR2017

## Outline

- 21.1 简单前馈网络
  - M-P模型、感知机与多层感知机
  - 神经网络的表示能力
  - 神经网络中的学习(BP算法)

# optimization

Regression:

$$\text{Loss}(x, y, \theta) = (f_\theta(x) - y)^2$$

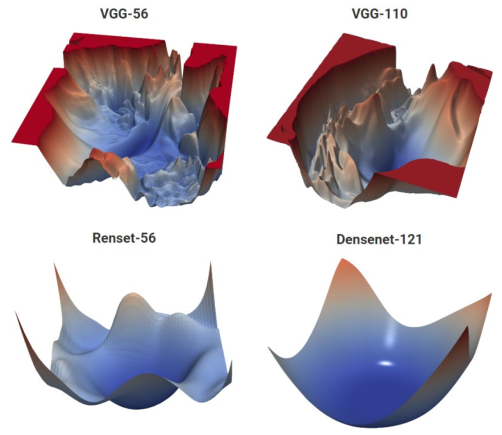
**Key idea: minimize training loss**

$$\text{TrainLoss}(\theta) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \theta)$$

$$\min_{\theta\in\mathbb{R}^d} \text{TrainLoss}(\theta)$$

**Algorithm: stochastic gradient descent**

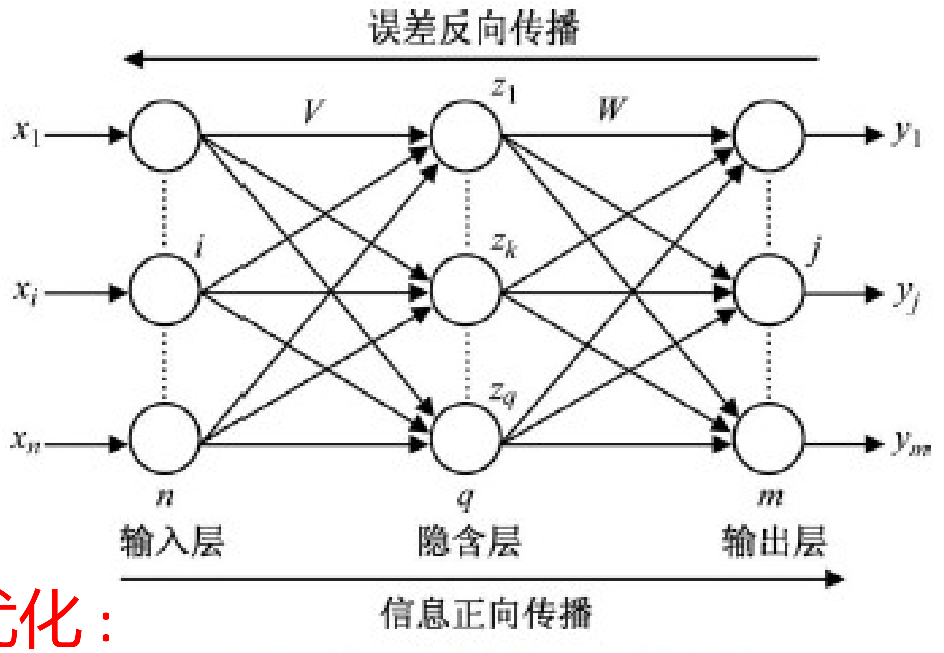
```
For $t = 1, \ldots, T$:
    For $(x, y) \in \mathcal{D}_{\text{train}}$:
        $\theta \leftarrow \theta - \eta_t \nabla_\theta \text{Loss}(x, y, \theta)$
```

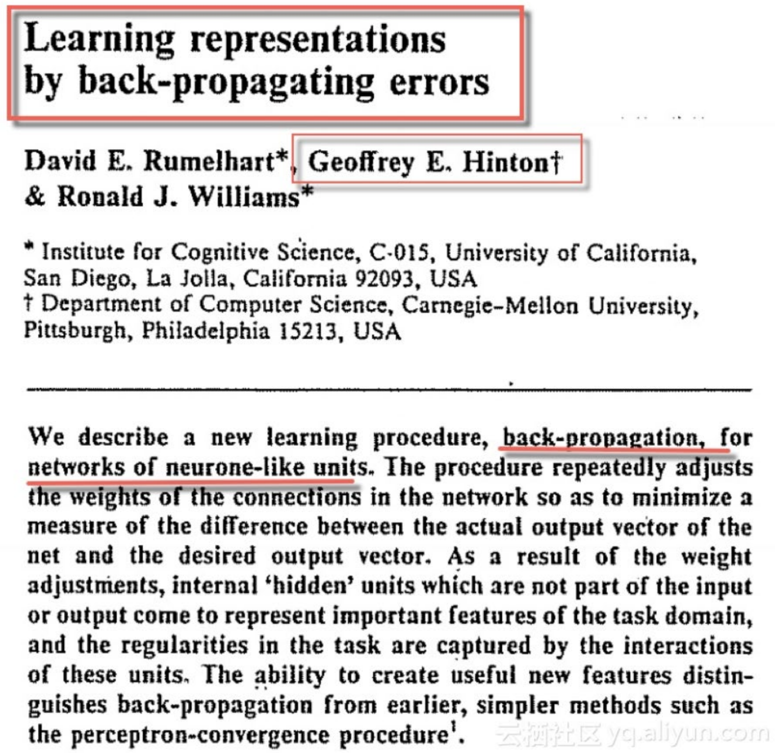


- Non-convex optimization, No theoretical guarantees that it works
- Before 2000s, empirically very difficult to get working

## 误差反向传播算法（Error BackPropagation, 简称BP）

BP算法[Rumelhart & Hinton, 1986]是最成功的训练多层前馈神经网络的学习算法





**Learning representations by back-propagating errors**

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

- 参数优化：
  - BP是一个迭代学习算法, 在迭代的每一轮中对参数进行一次更新估计

# 误差反向传播算法（Error BackPropagation, 简称BP）

BP算法[Rumelhart & Hinton, 1986] 是最成功的训练多层前馈神经网络的学习算法

- 前向计算

  step1: 计算隐层的神经元

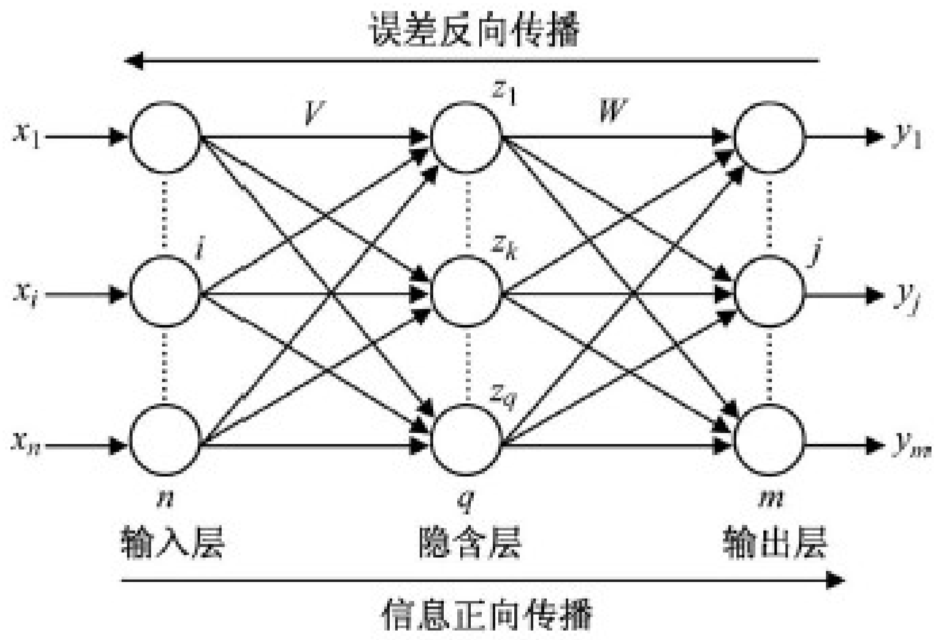
  step2: 计算输出层的神经元

  step3: 计算误差 $E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$

- 网络参数

  参数包括：权重，阈值

  网络训练的过程就是参数优化的过程



# 反向传播算法（Error BackPropagation, 简称BP）

BP算法是最成功的训练多层前馈神经网络的学习算法

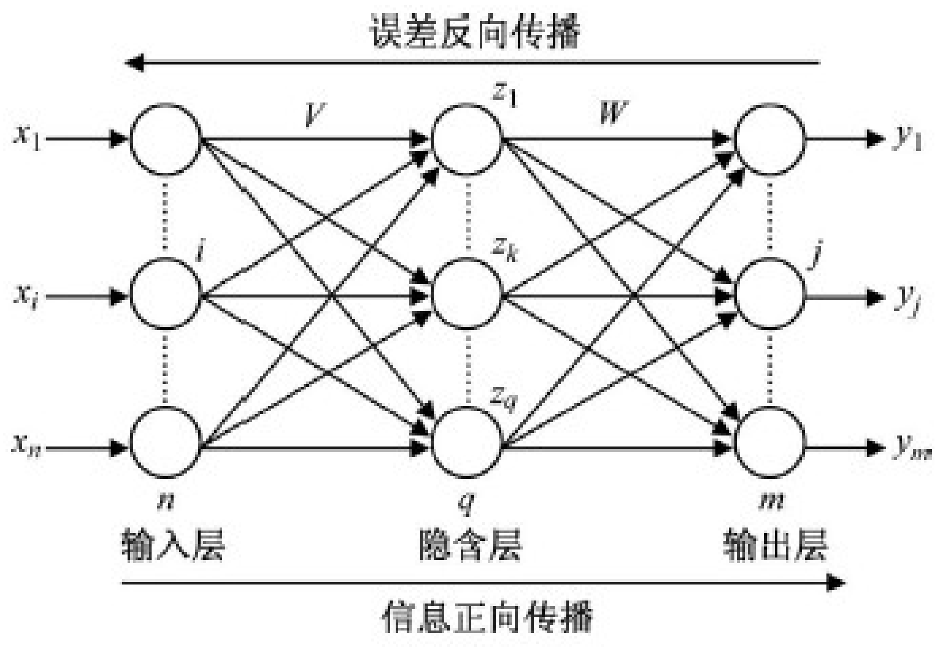
- 反向计算

  step1: 计算输出层的梯度

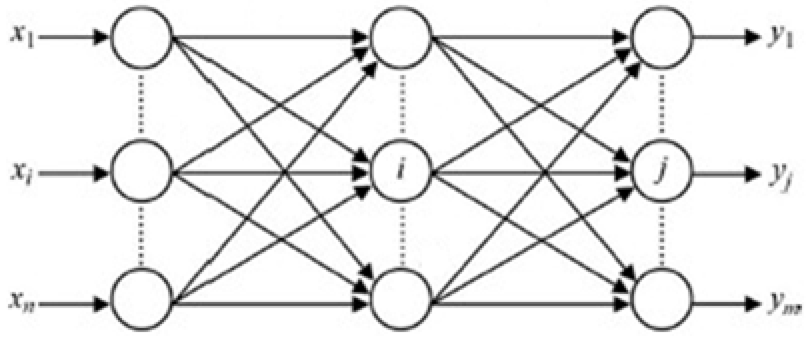
  step2: 从输出层开始，循环直到最早的隐藏层

  2.1: 将误差传播回前一层

  step3: 更新权重与阈值



# BP算法



前向计算 $a_j = g(\sum_i w_{ij} a_i)$

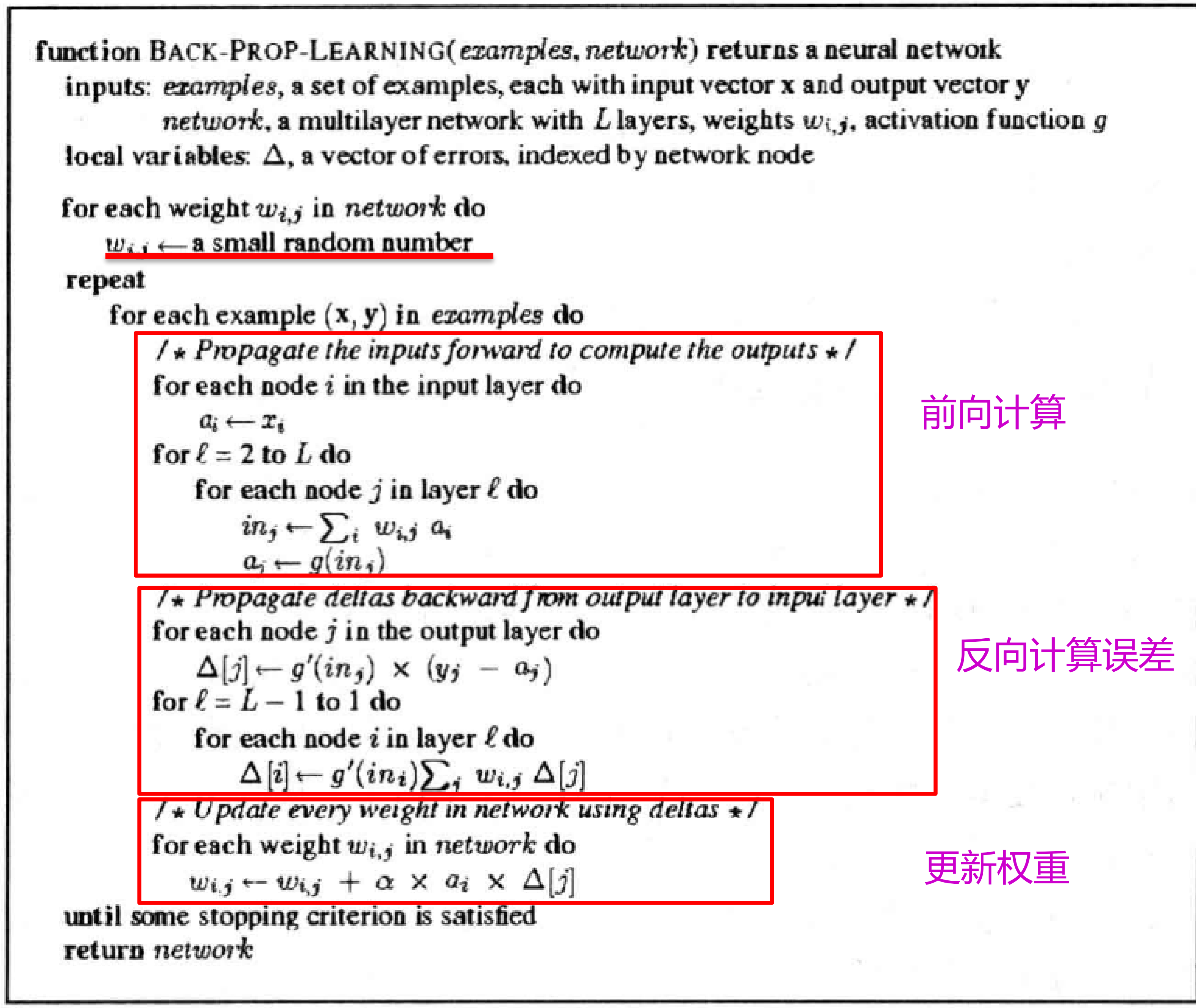
反向计算误差

$$\Delta_j = g'(in_j)(y_j - a_j)$$

$$\Delta_i = g'(in_i)\sum_j \Delta_j W_{i,j}$$

更新权重

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$



```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
  local variables: $\Delta$, a vector of errors, indexed by network node

  for each weight $w_{i,j}$ in network do
      $w_{i,j} \leftarrow$ a small random number
  repeat
      for each example (x, y) in examples do
          /* Propagate the inputs forward to compute the outputs */
          for each node $i$ in the input layer do
              $a_i \leftarrow x_i$
          for $\ell = 2$ to $L$ do
              for each node $j$ in layer $\ell$ do
                  $in_j \leftarrow \sum_i w_{i,j} a_i$
                  $a_j \leftarrow g(in_j)$
          /* Propagate deltas backward from output layer to input layer */
          for each node $j$ in the output layer do
              $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
          for $\ell = L - 1$ to 1 do
              for each node $i$ in layer $\ell$ do
                  $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
          /* Update every weight in network using deltas */
          for each weight $w_{i,j}$ in network do
              $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return network
```

前向计算

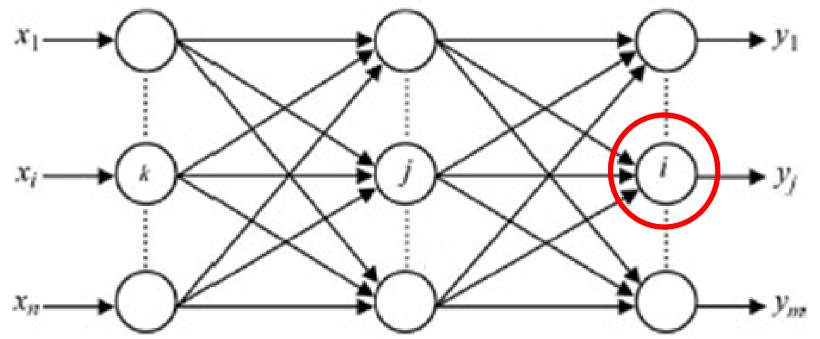
反向计算误差

更新权重

# 反向传播公式推导

- 在单个样本上的平方误差定义为：

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2$$



其中，求和是对输出层的所有节点进行的。

前向计算 $a_i = g(in_i)$
$= g(\sum_j w_{ji} a_j)$

输出层的权值更新：

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}}$$
$$= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$
$$= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i$$

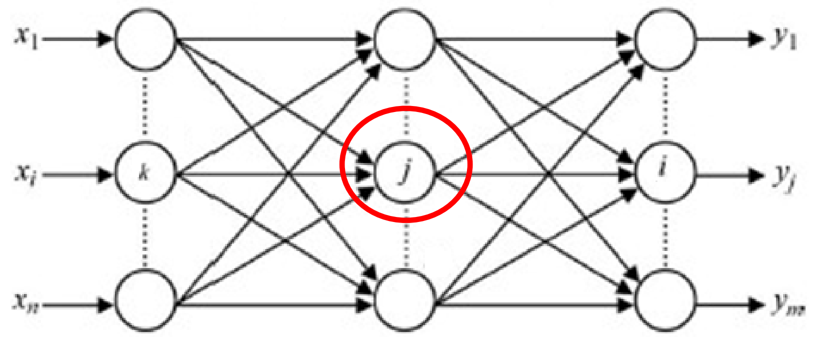
修正误差 $\Delta_i = (y_i - a_i)g'(in_i)$

函数求导(链式法则)

# 反向传播公式推导（续）

隐藏层的权值更新：

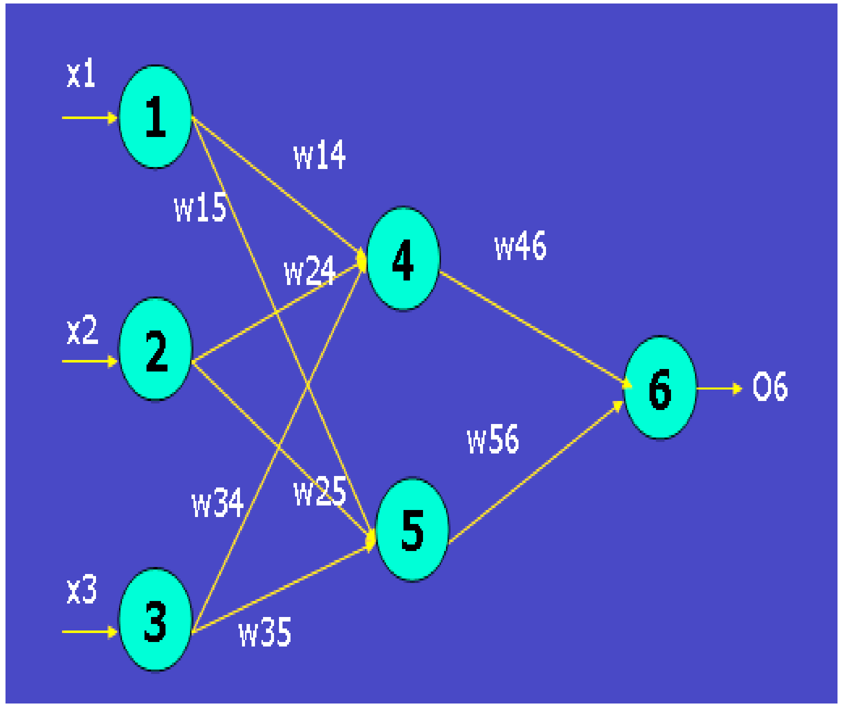
$$
\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\
&= -\sum_i \underline{(y_i - a_i)g'(in_i)}\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \underline{\Delta_i}\frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right) \\
&= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial in_j}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j}a_k\right) \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)a_k = -a_k\Delta_j
\end{aligned}
$$



修正误差$\Delta_j = \sum_i \Delta_i W_{j,i} g'(in_j)$

# 课堂练习

用一个训练样本，示例了网络学习过程中的一次迭代过程。



**Step 1 前向传播** $y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$

**Step 2 反向传播计算修正误差**

$$\boxed{\Delta_i = g'(in_i)(y_i - a_i)}$$

$$\boxed{\Delta_j = g'(in_j)\sum_i \Delta_i W_{j,i}}$$

**Step 3 计算权重和阈值的更新**

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$

训练样本x={1, 0, 1}

类标号(标签)为1

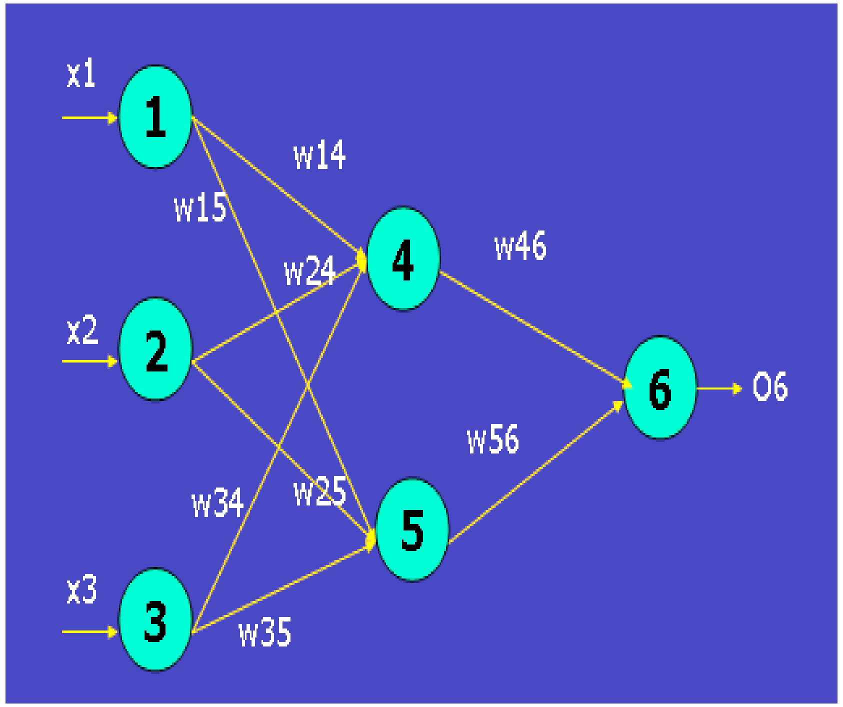
激活函数为sigmoid函数

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

设 $w_0=-\theta$，$x_0=1$，前向计算公式就可以表示为：$y=f(\sum_{i=0}^{n} w_i * x_i)$

# 课堂练习

用一个训练样本，示例了网络学习过程中的一次迭代过程。



$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

**Step 1 前向传播**

| 神经元 | 输入 net | 输出o |
|---|---|---|
| 4 | | |
| 5 | | |
| 6 | | |

训练样本x={1, 0, 1}

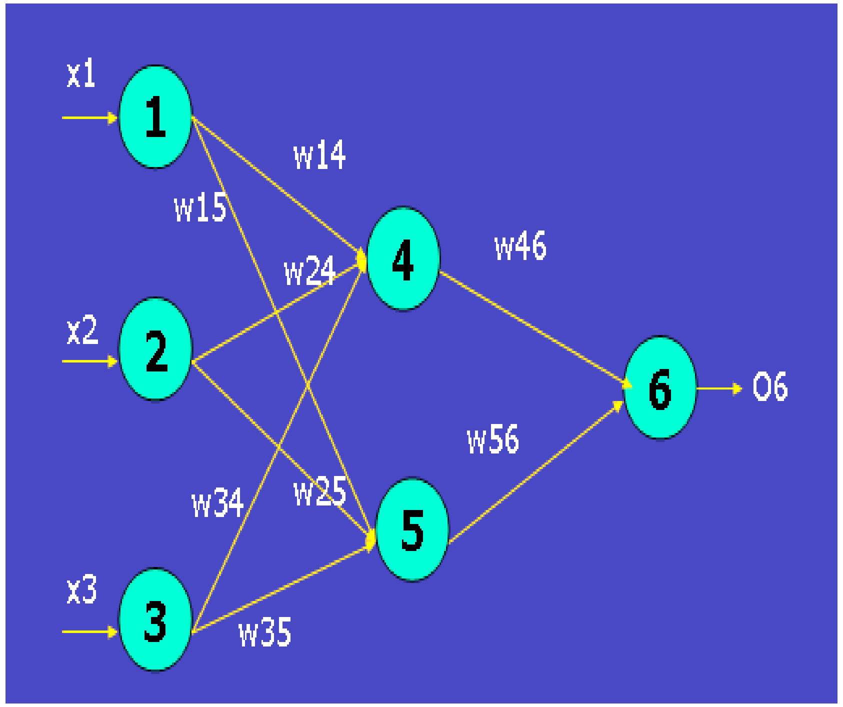
类标号(标签)为1

激活函数为sigmoid函数

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

# 课堂练习

用一个训练样本，示例了网络学习过程中的一次迭代过程。



$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

**Step 1 前向传播**

| 神经元 | 输入 in | 输出out |
|---|---|---|
| 4 | 0.2*1+0.4*0+(-0.5)*1-0.4=-0.7 | 1/(1+e-(-0.7))=0.332 |
| 5 | (-0.3)*1+0.1*0+(0.2)*1-(-0.2)=0.1 | 1/(1+e(-0.1))=0.525 |
| 6 | (-0.3)*0.332+(-0.2)*0.525-(-0.1)=-0.105 | 1/(1+e-(-0.105))=0.474 |

训练样本x={1, 0, 1}

类标号(标签)为1

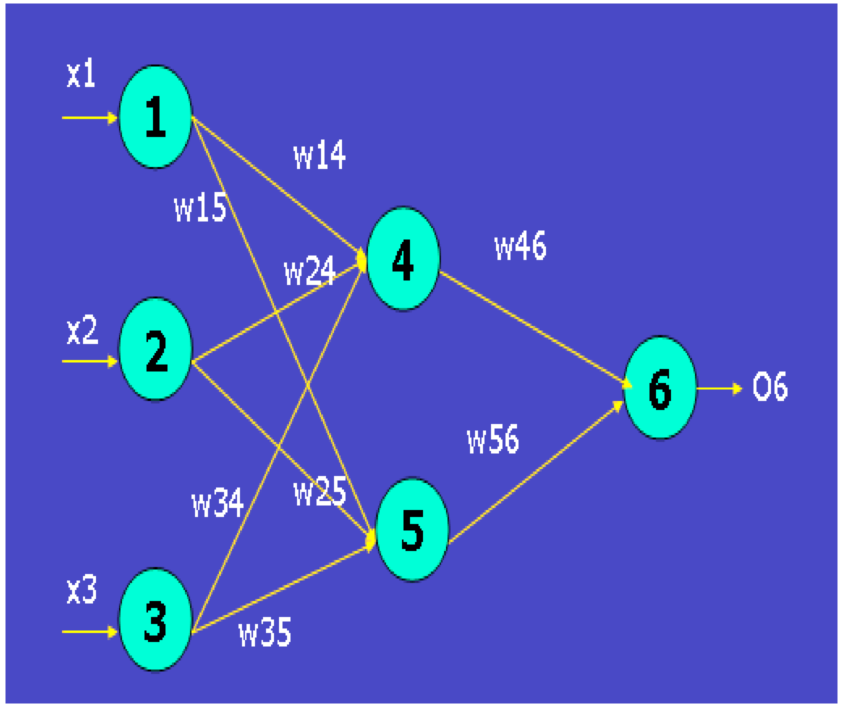
激活函数为sigmoid函数

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

# 课堂练习

| 神经元 | 输入 in | 输出out |
|---|---|---|
| 4 | -0.7 | 0.332 |
| 5 | 0.1 | 0.525 |
| 6 | 0.105 | 0.474 |

用一个训练样本，示例了网络学习过程中的一次迭代过程。



**Step 2 反向传播**

$$\boxed{\Delta_i = g'(in_i)(y_i - a_i)}$$

**2.1 计算修正误差**

$$\Delta_j = g'(in_j)\sum_i \Delta_i W_{j,i}$$

| 神经元 | 修正误差 |
|---|---|
| 6 | |
| 5 | |
| 4 | |

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

$$g'(z) = g(z)(1-g(z))$$

训练样本x={1, 0, 1}

类标号(标签)为1

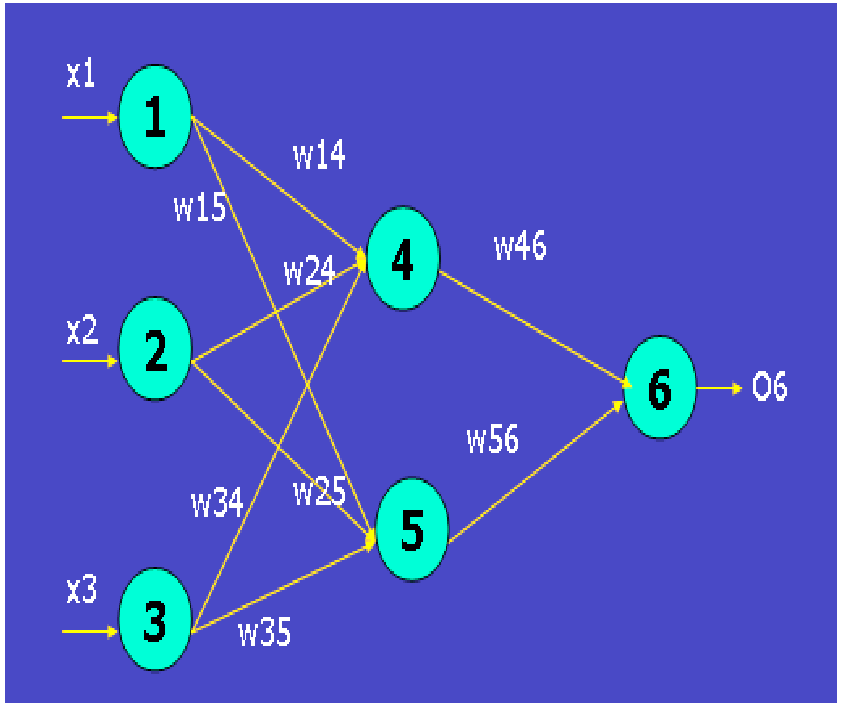
激活函数为sigmoid函数

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

# 课堂练习

| 神经元 | 输入 in | 输出out |
|---|---|---|
| 4 | -0.7 | 0.332 |
| 5 | 0.1 | 0.525 |
| 6 | 0.105 | 0.474 |

用一个训练样本，示例了网络学习过程中的一次迭代过程。



**Step 2 反向传播**

**计算修正误差**

$$\Delta_i = g'(in_i)(y_i - a_i)$$

$$\Delta_j = g'(in_j)\sum_i \Delta_i W_{j,i}$$

| 神经元 | 修正误差 |
|---|---|
| 6 | 0.474*(1-0.474) *(1-0.474) =0.1311 |
| 5 | 0.525*(1-0.525) *(0.1311*(-0.2)) =-0.0065 |
| 4 | 0.332*(1-0.332)*(0.1311*(-0.3))=-0.0087 |

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

$$g'(z) = g(z)(1-g(z))$$

训练样本x={1, 0, 1}

类标号(标签)为1

激活函数为sigmoid函数

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

## Step 3 反向传播

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.2 | -0.3 | 0.4 | 0.1 | -0.5 | 0.2 | -0.3 | -0.2 | 0.4 | -0.2 | -0.1 |

### 计算权重和阈值的更新

| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 | θ4 | θ5 | θ6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.192 | -0.306 | 0.4 | 0.1 | -0.508 | 0.194 | -0.216 | -0.138 | 0.408 | -0.194 | -0.218 |

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$

学习率α，此处取值为0.9

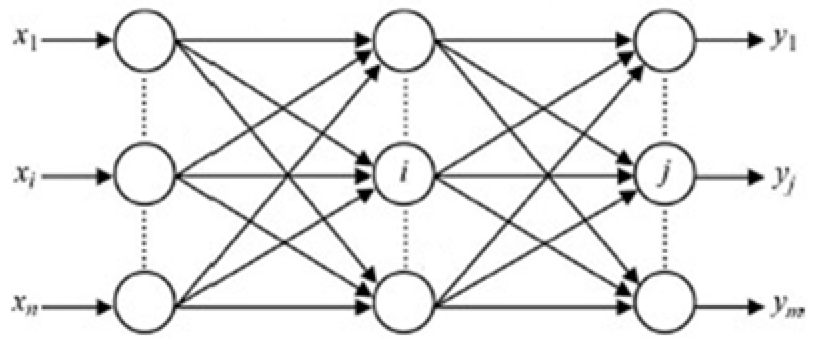
x={1, 0, 1}

| 神经元 | 输入 in | 输出out |
|---|---|---|
| 4 | -0.7 | 0.332 |
| 5 | 0.1 | 0.525 |
| 6 | 0.105 | 0.474 |

| 神经元 | 修正误差 $\Delta_j$ |
|---|---|
| 6 | (1-0.474)*0.474*(1-0.474) =0.1311 |
| 5 | (0.1311*(-0.2))*0.525*(1-0.525)=-0.0065 |
| 4 | (0.1311*(-0.3))*0.332*(1-0.332)=-0.0087 |

| | |
|---|---|
| w46 | -0.3+0.9*0.332*0.1311=-0.216 |
| w56 | -0.2+0.9*0.525*0.1311=-0.138 |
| w14 | 0.2+0.9*1*(-0.0087)=0.192 |
| w15 | -0.3+0.9*1*(-0.0065) =-0.306 |
| w24 | 0.4+0.9*0*(-0.0087)=0.4 |
| w25 | 0.1+0.9*0*(-0.0065)=0.1 |
| w34 | -0.5+0.9*1*(-0.0087)=-0.508 |
| w35 | 0.2+0.9*1*(-0.0065)=-0.194 |
| -θ6 | 0.1+0.9*1*0.1311=0.218 |
| -θ5 | 0.2+0.9*1*(-0.0065)=0.194 |
| -θ4 | -0.4+0.9*1*(-0.0087)=-0.408 |

# BP算法



前向计算 $a_j = g(\sum_i w_{ij} a_i)$

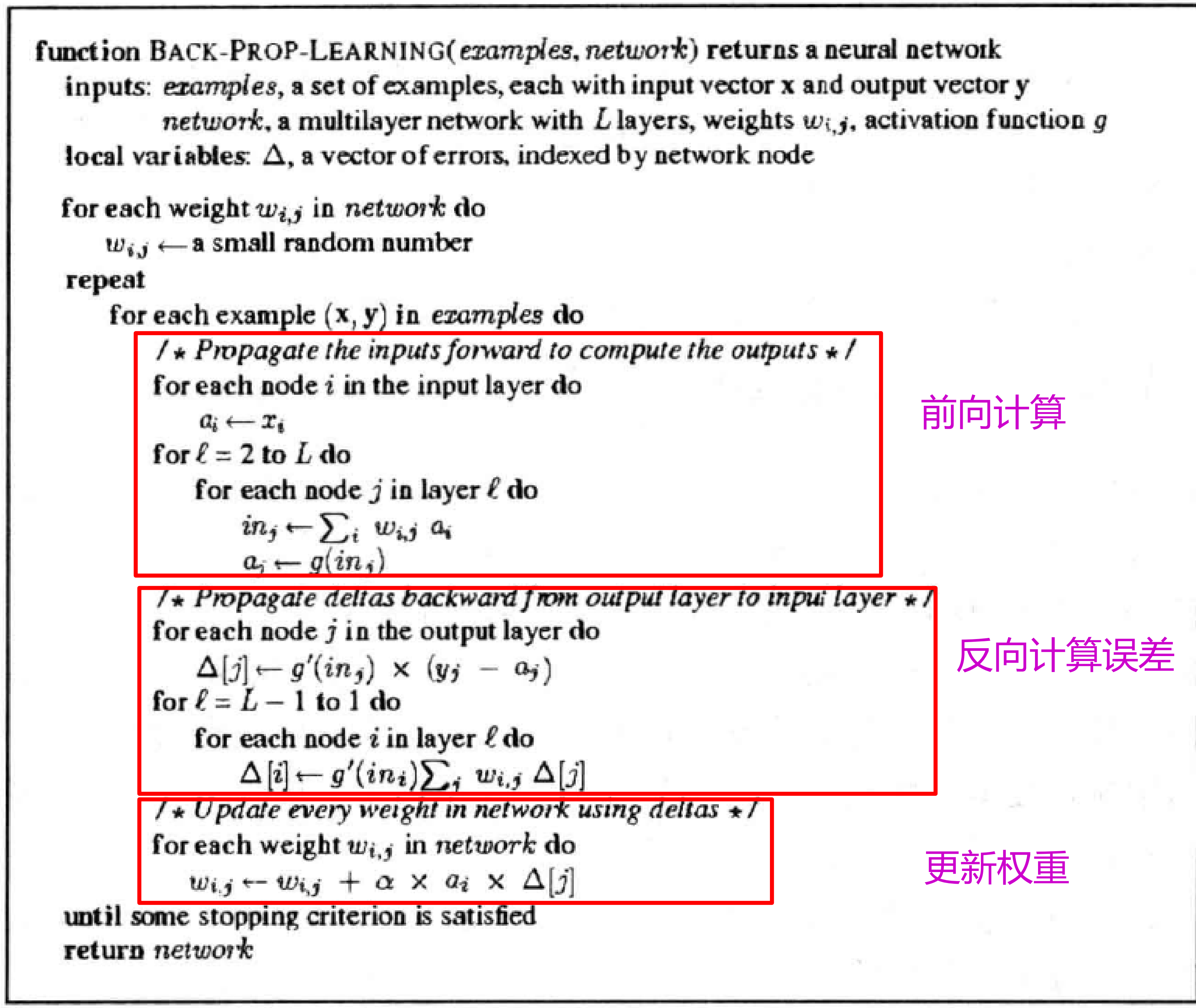
反向计算误差

$$\Delta_j = g'(in_j)(y_j - a_j)$$

$$\Delta_i = g'(in_i)\sum_j \Delta_j W_{i,j}$$

更新权重

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$



```
function BACK-PROP-LEARNING(examples, network) returns a neural network
   inputs: examples, a set of examples, each with input vector x and output vector y
           network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
   local variables: $\Delta$, a vector of errors, indexed by network node

   for each weight $w_{i,j}$ in network do
      $w_{i,j} \leftarrow$ a small random number
   repeat
      for each example (x, y) in examples do
         /* Propagate the inputs forward to compute the outputs */
         for each node $i$ in the input layer do
            $a_i \leftarrow x_i$
         for $\ell = 2$ to $L$ do
            for each node $j$ in layer $\ell$ do
               $in_j \leftarrow \sum_i w_{i,j} a_i$
               $a_j \leftarrow g(in_j)$
         /* Propagate deltas backward from output layer to input layer */
         for each node $j$ in the output layer do
            $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
         for $\ell = L - 1$ to 1 do
            for each node $i$ in layer $\ell$ do
               $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
         /* Update every weight in network using deltas */
         for each weight $w_{i,j}$ in network do
            $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
   until some stopping criterion is satisfied
   return network
```

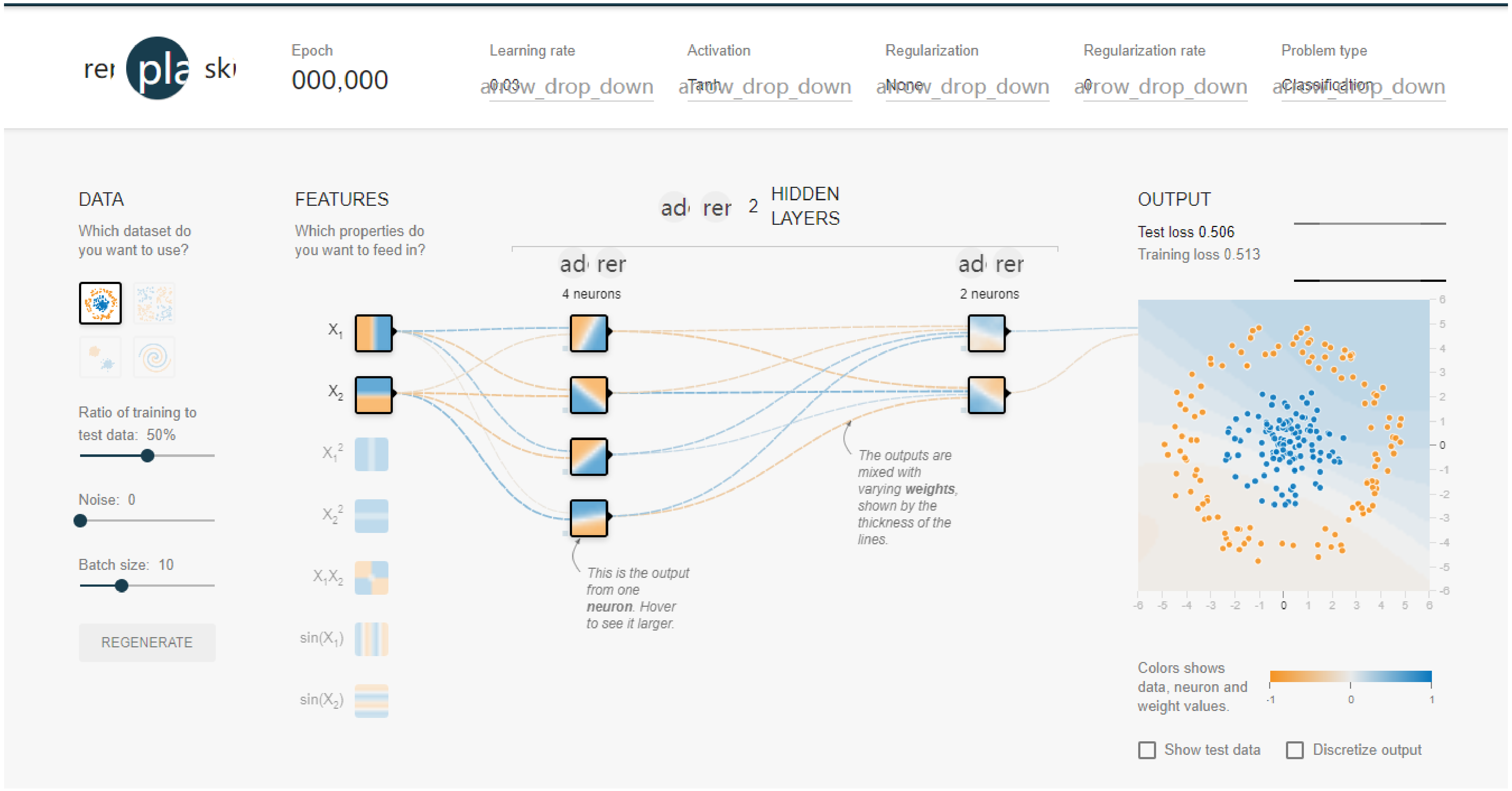
前向计算

反向计算误差

更新权重

谢谢！

# A demo from Google



rei pla ski

Epoch
000,000

Learning rate
0.03

Activation
Tanh

Regularization
None

Regularization rate
0

Problem type
Classification

DATA
Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

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

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

OUTPUT
Test loss 0.506
Training loss 0.513

Colors shows data, neuron and weight values.

Show test data
Discretize output

http://playground.tensorflow.org/