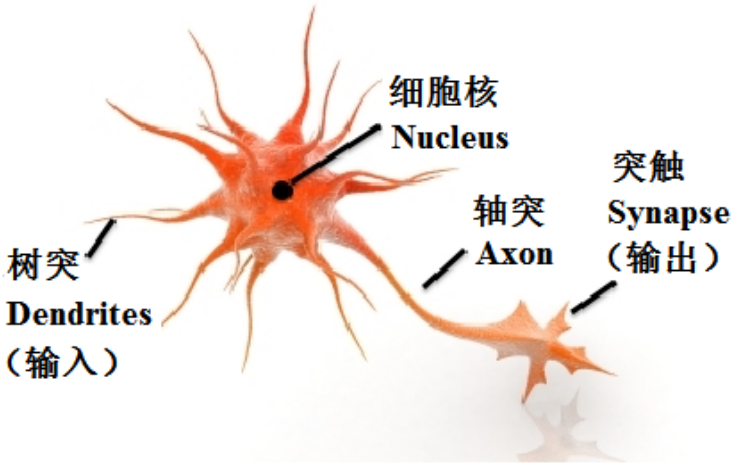
人工神经网络介绍

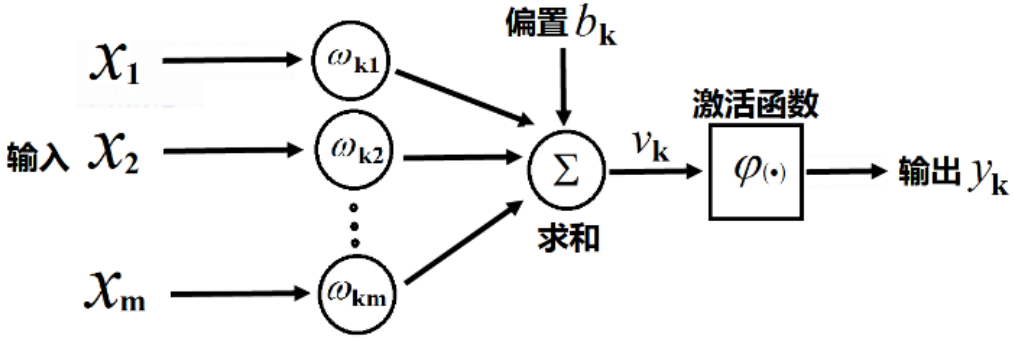
胡浩基
浙江大学信息与电子工程学院
haoji_hu@zju.edu.cn

人工神经网络的历史－生物模型

1943年，心理学家W.S.McCulloch和数理逻辑学家W.Pitts基于神经元的生理特征，建立了单个神经元的数学模型（MP模型）。

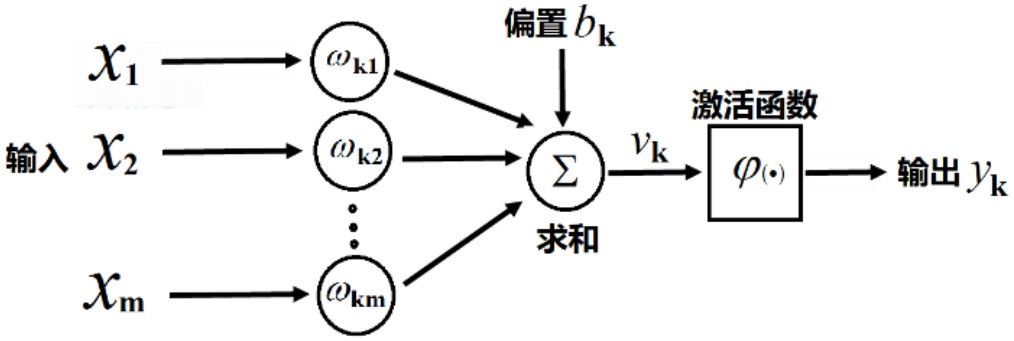


神经元生理结构示意图



神经元的数学模型示意图

人工神经网络的历史 – 数学模型



$$y_k = \varphi\left(\sum_{i=1}^{m} \omega_{ki} x_i + b_k\right) = \varphi\left(W_k^T X + b\right)$$

神经元的数学模型示意图

人工神经网络的历史 – 感知器

1957年，Frank Rosenblatt从纯数学的度重新考察这一模型，指出能够从一些输入输出对(X, y)中通过学习算法获得权重 W 和 b。

问题：给定一些输入输出对 (X, y)，其中 $y = \pm 1$，求一个函数，使：$f(X) = y$

感知器算法：设定 $f(X) = sign(W^T X + b)$，从一堆输入输出中自动学习，获得 W 和 b。

人工神经网络的历史 – 感知器

1957年，Frank Rosenblatt从纯数学的度重新考察这一模型，指出能够从一些输入输出对(X, y)中通过学习算法获得权重 W 和 b。

感知器算法（Perceptron Algorithm）：

（1）随机选择W和 b。

（2）取一个训练样本 (X, y)

（i）若 $W^T X + b > 0$ 且 $y = -1$，则：

$W = W - X$ $b = b - 1$

（ii）若 $W^T X + b < 0$ 且 $y = +1$，则：

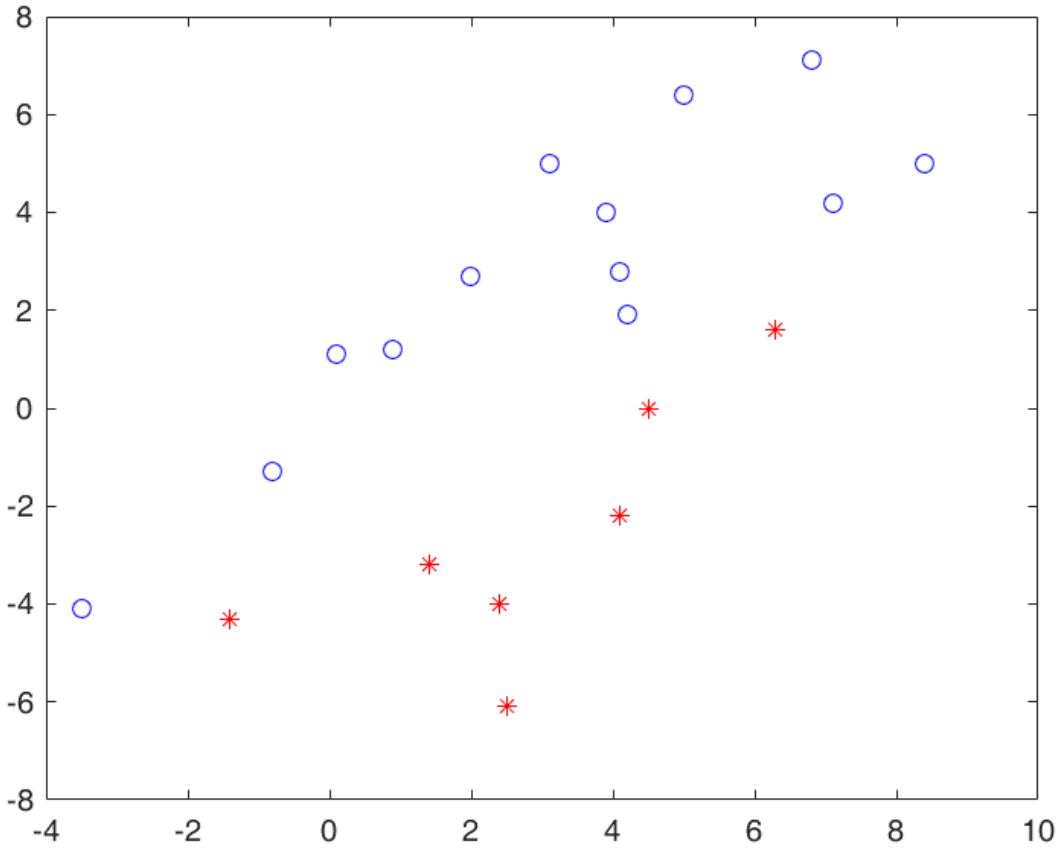
$W = W + X$ $b = b + 1$

（3）再取另一个(X, y)，回到（2）

（4）终止条件：直到所有输入输出对 都不满足（2）中(i)和(ii)之一，退出循环。

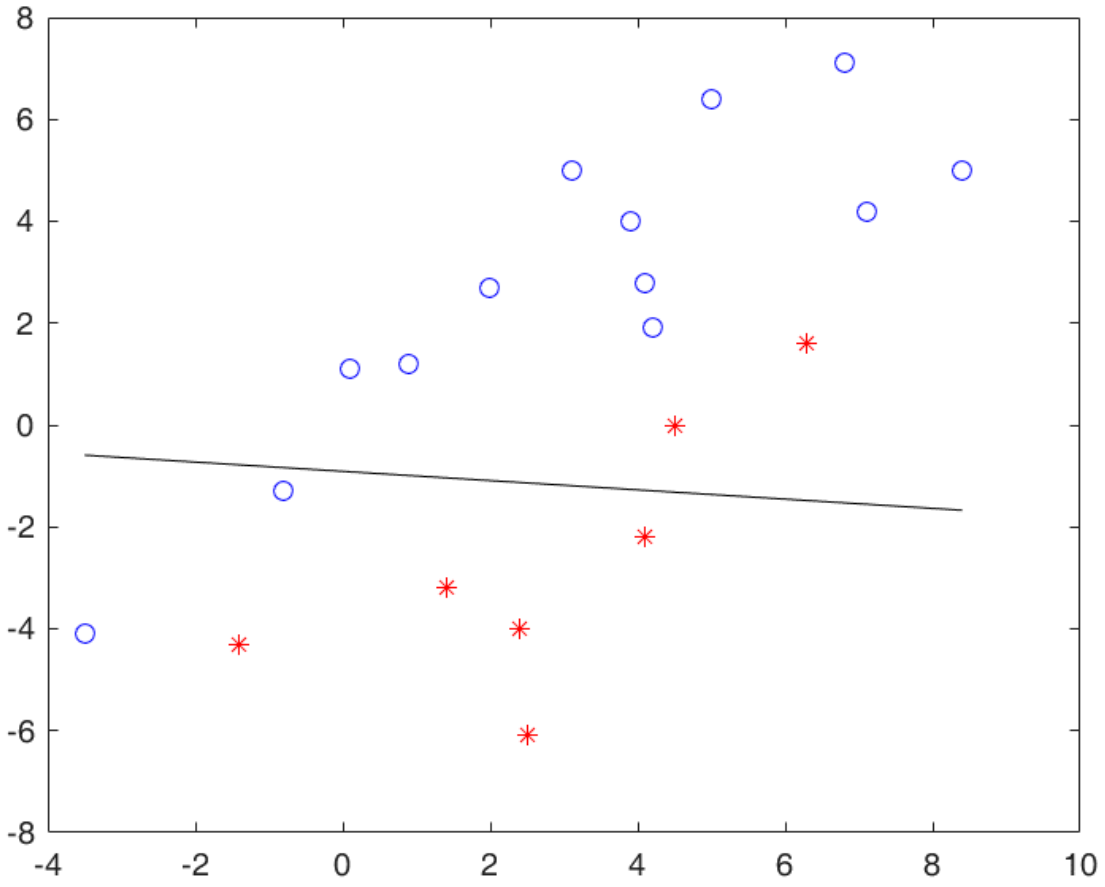
人工神经网络的历史 – 感知器

感知器算法演示：



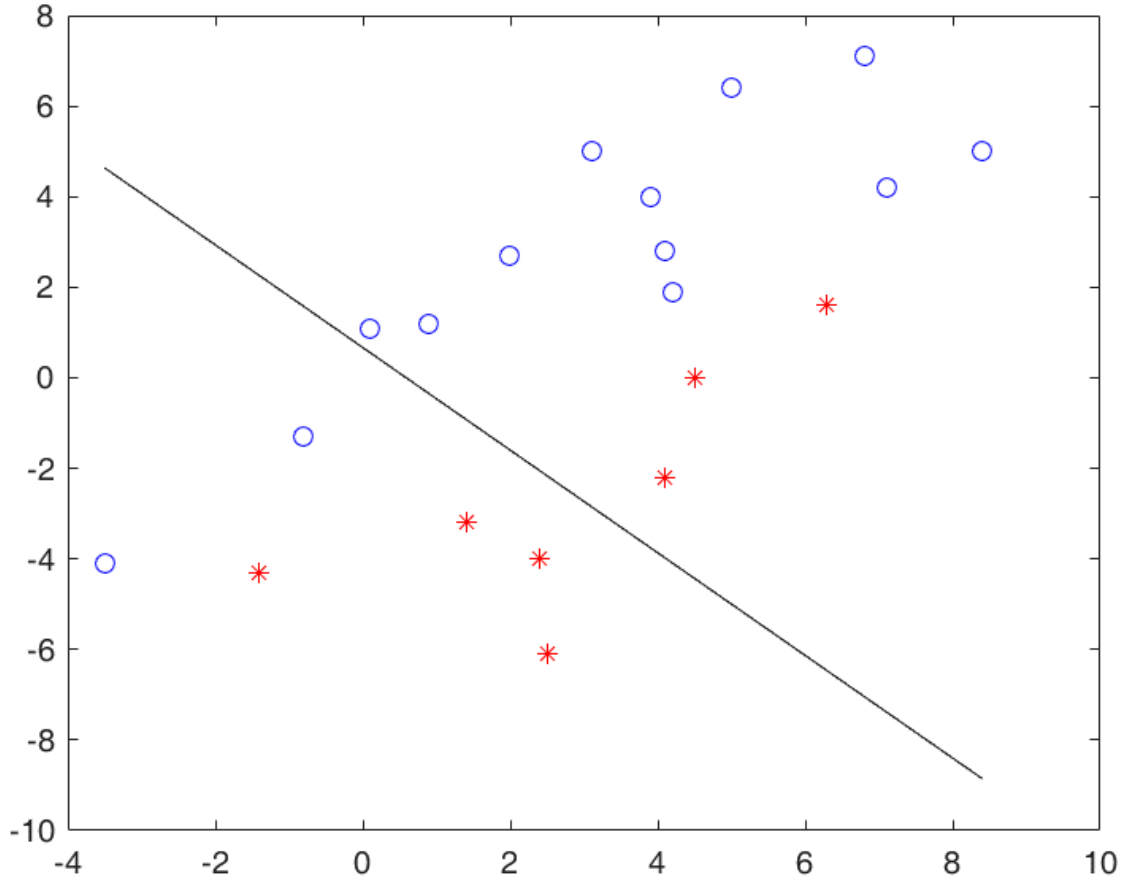
人工神经网络的历史 – 感知器

感知器算法演示：



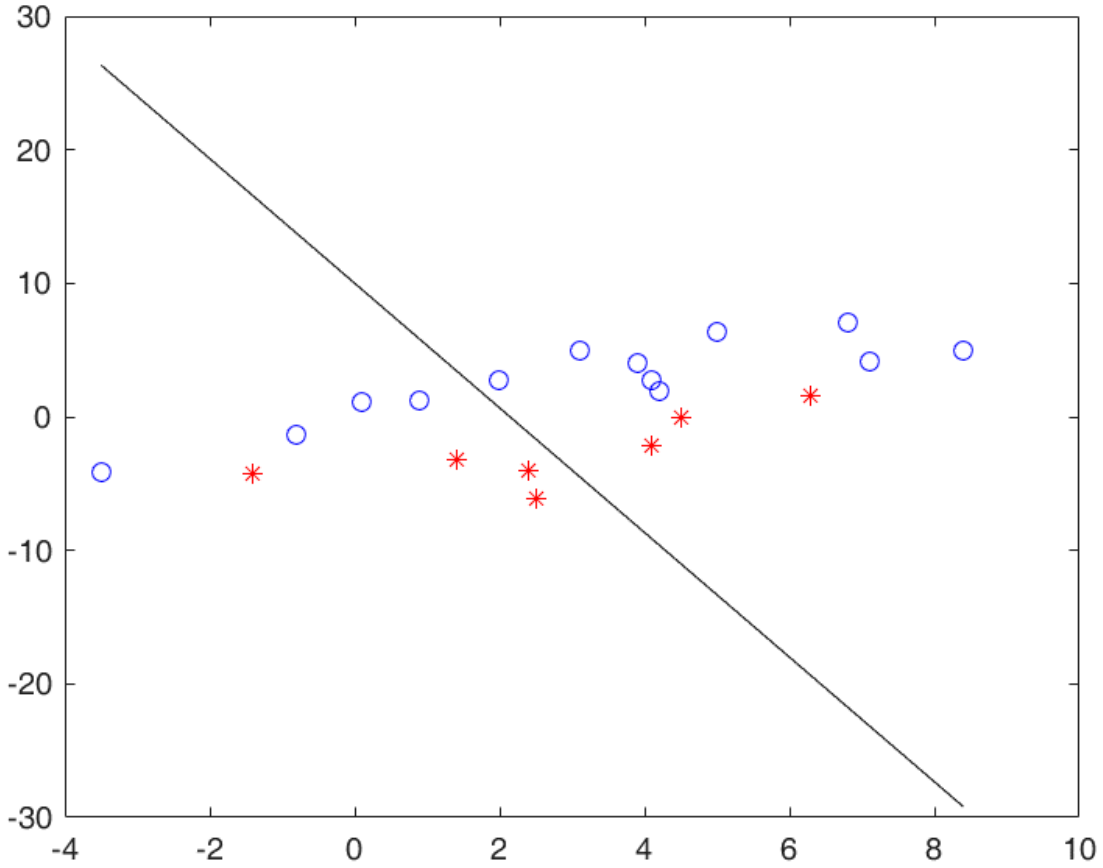
人工神经网络的历史 – 感知器

感知器算法演示：



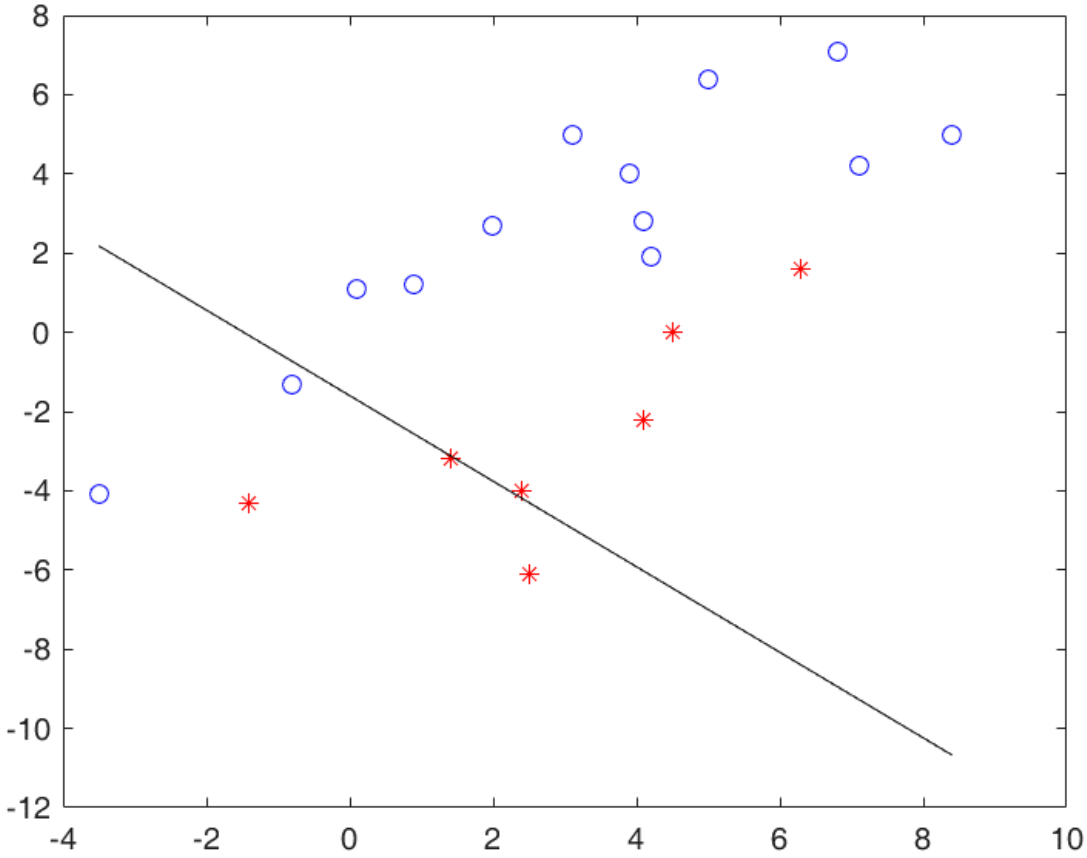
人工神经网络的历史 – 感知器

感知器算法演示：



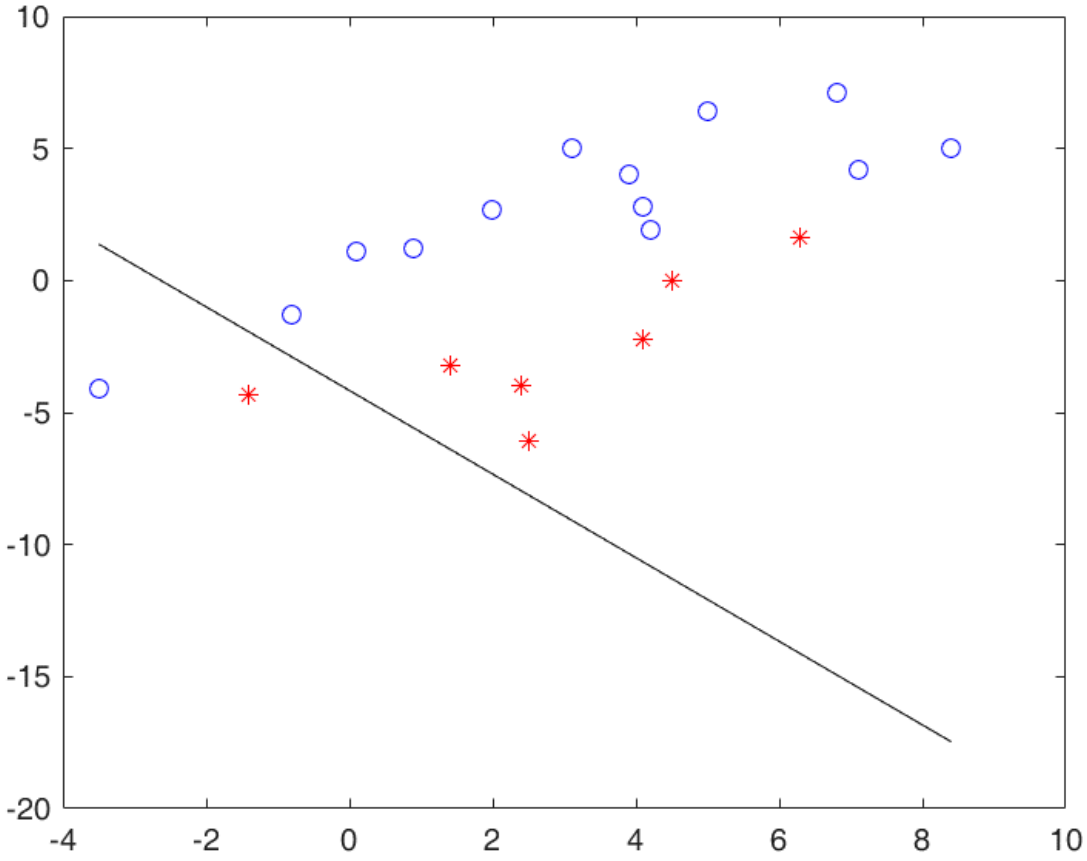
人工神经网络的历史 – 感知器

感知器算法演示：



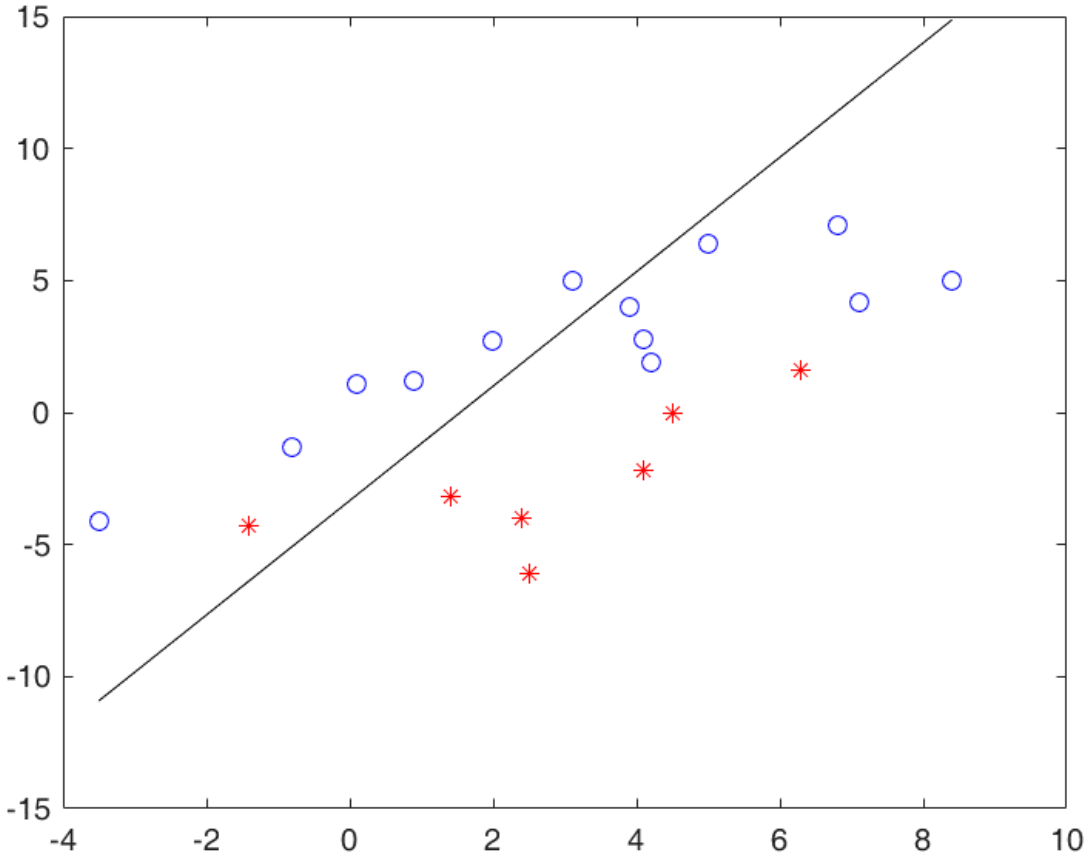
人工神经网络的历史 – 感知器

感知器算法演示：



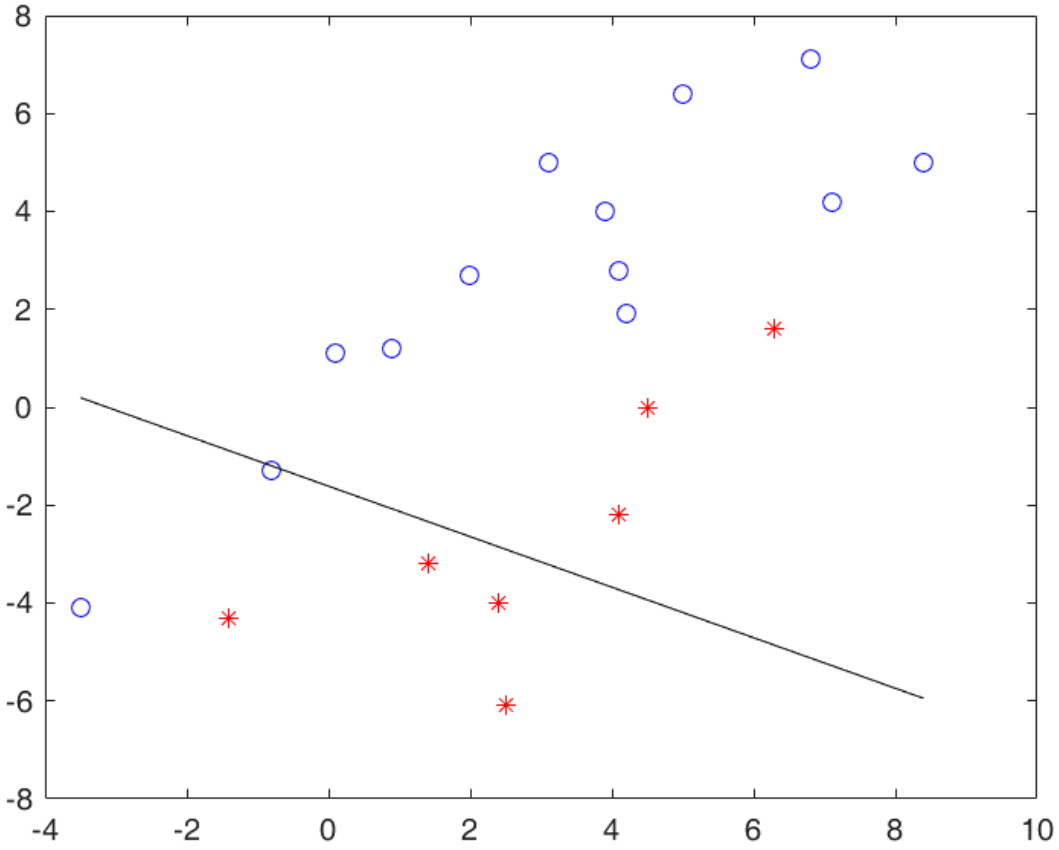
人工神经网络的历史 – 感知器

感知器算法演示：



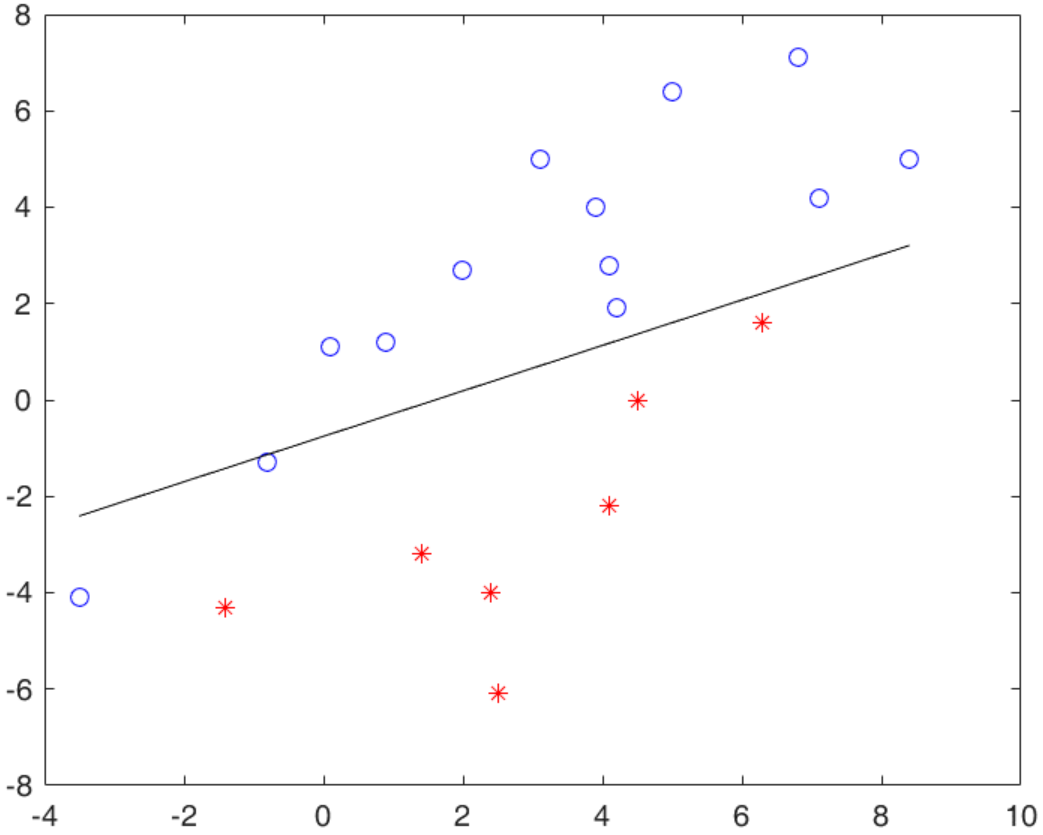
人工神经网络的历史 – 感知器

感知器算法演示：



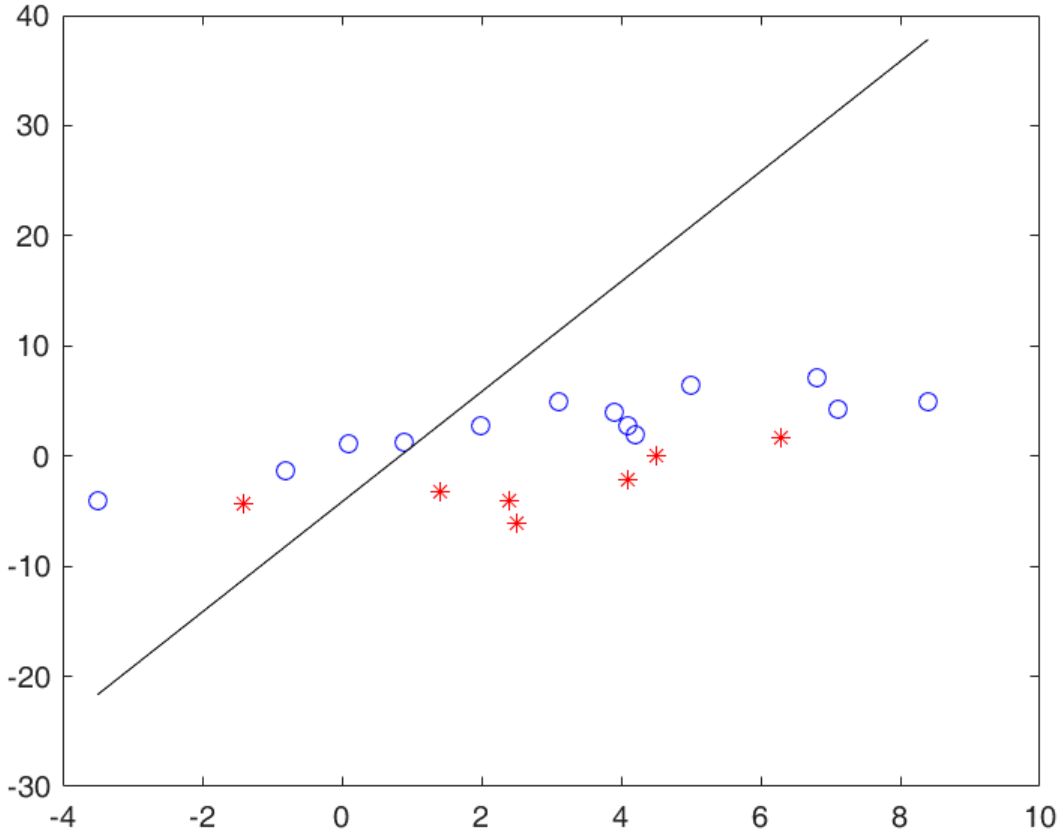
人工神经网络的历史 – 感知器

感知器算法演示：



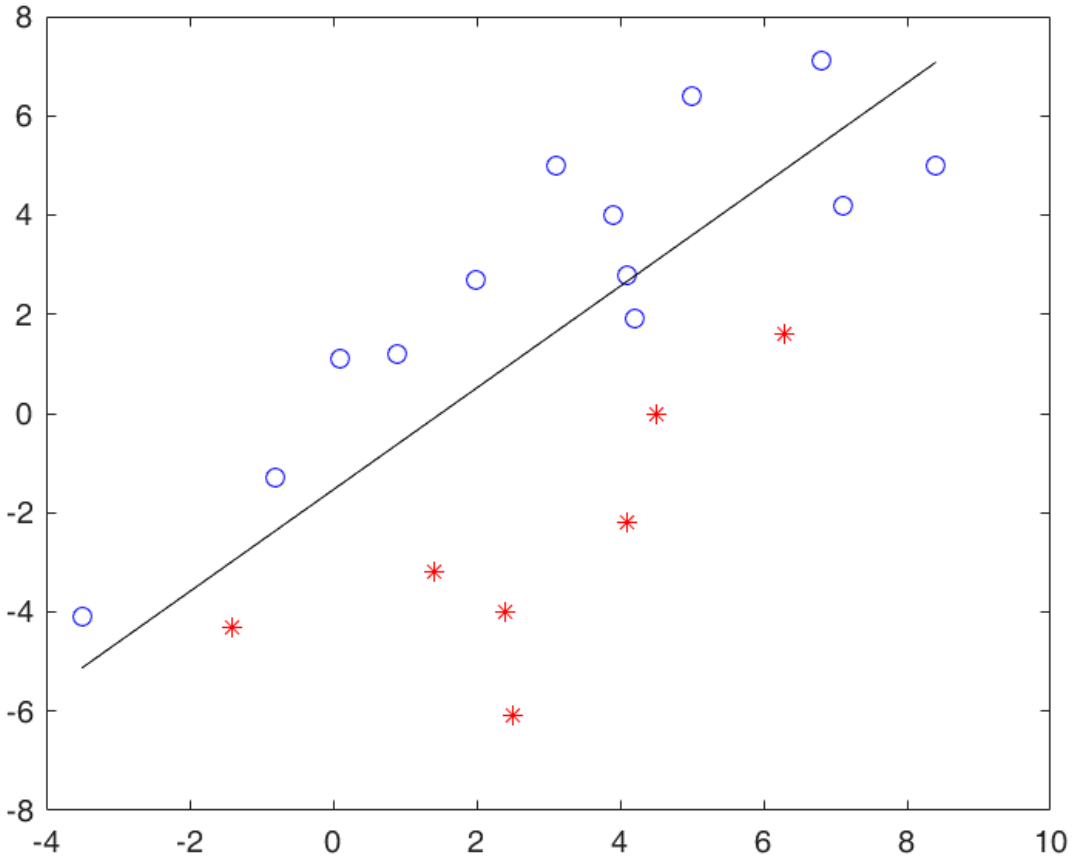
人工神经网络的历史 – 感知器

感知器算法演示：



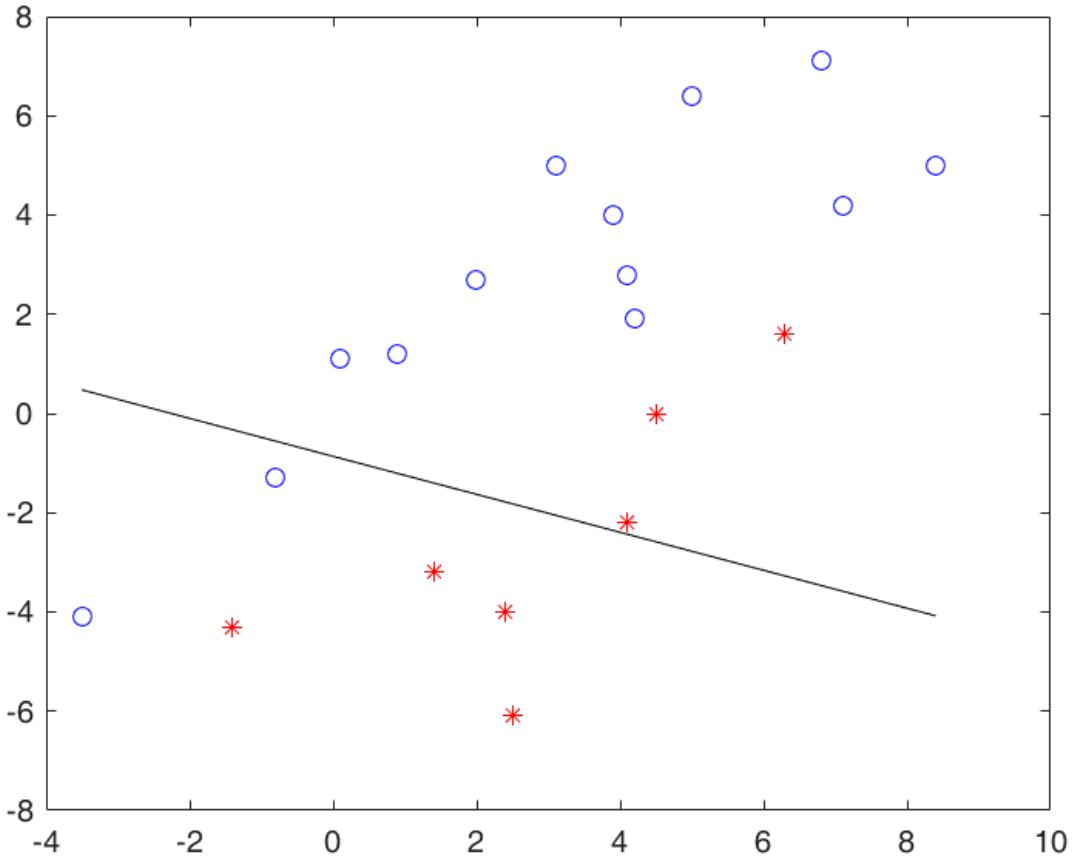
人工神经网络的历史 – 感知器

感知器算法演示：



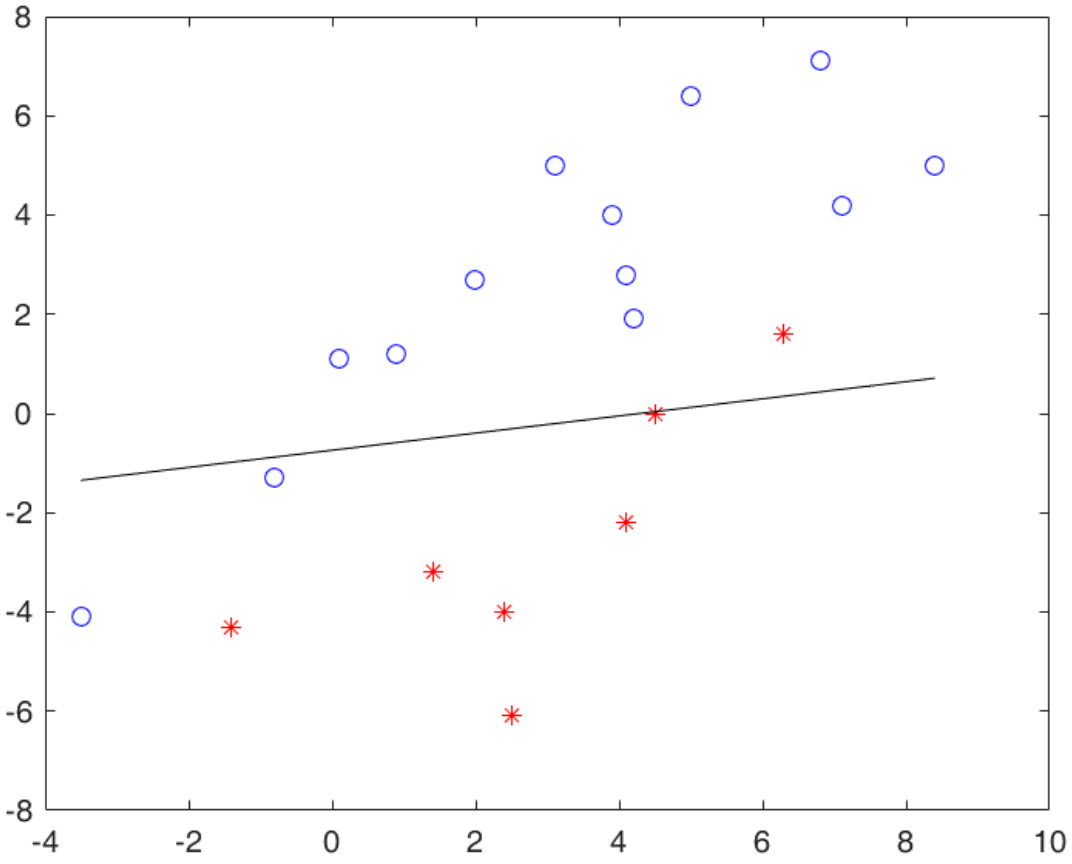
人工神经网络的历史 – 感知器

感知器算法演示：



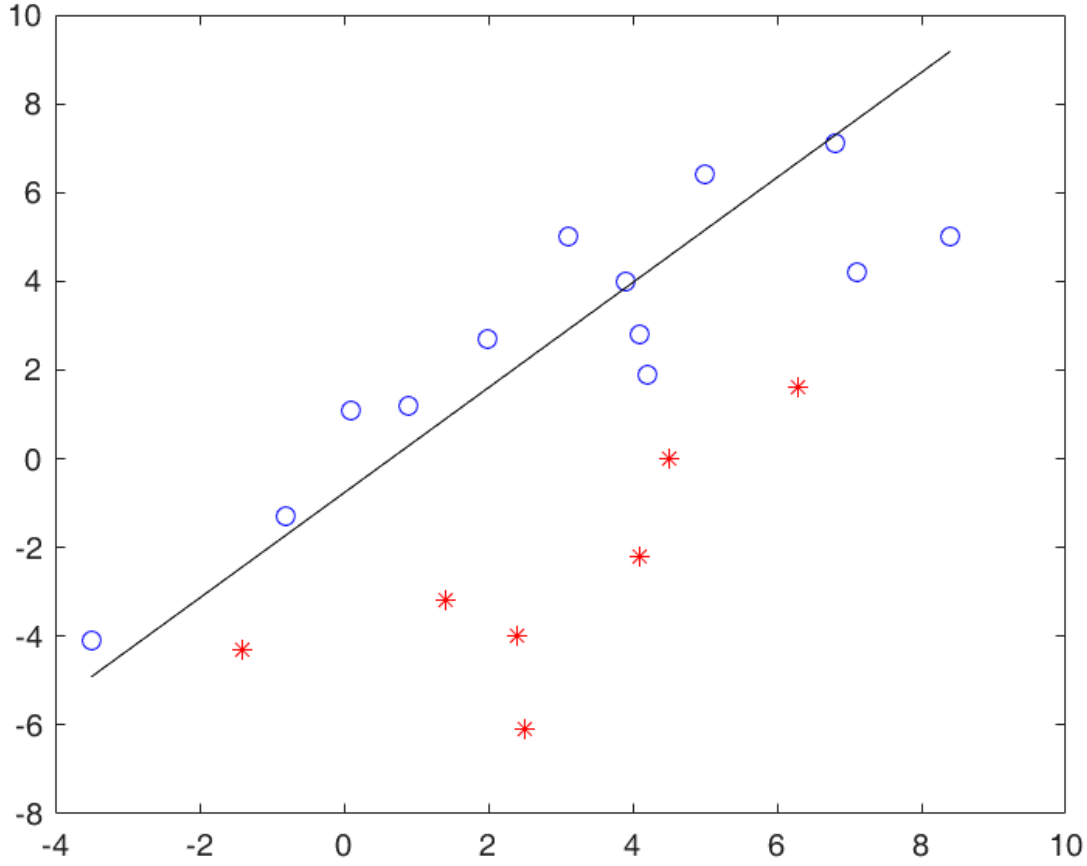
人工神经网络的历史 – 感知器

感知器算法演示：



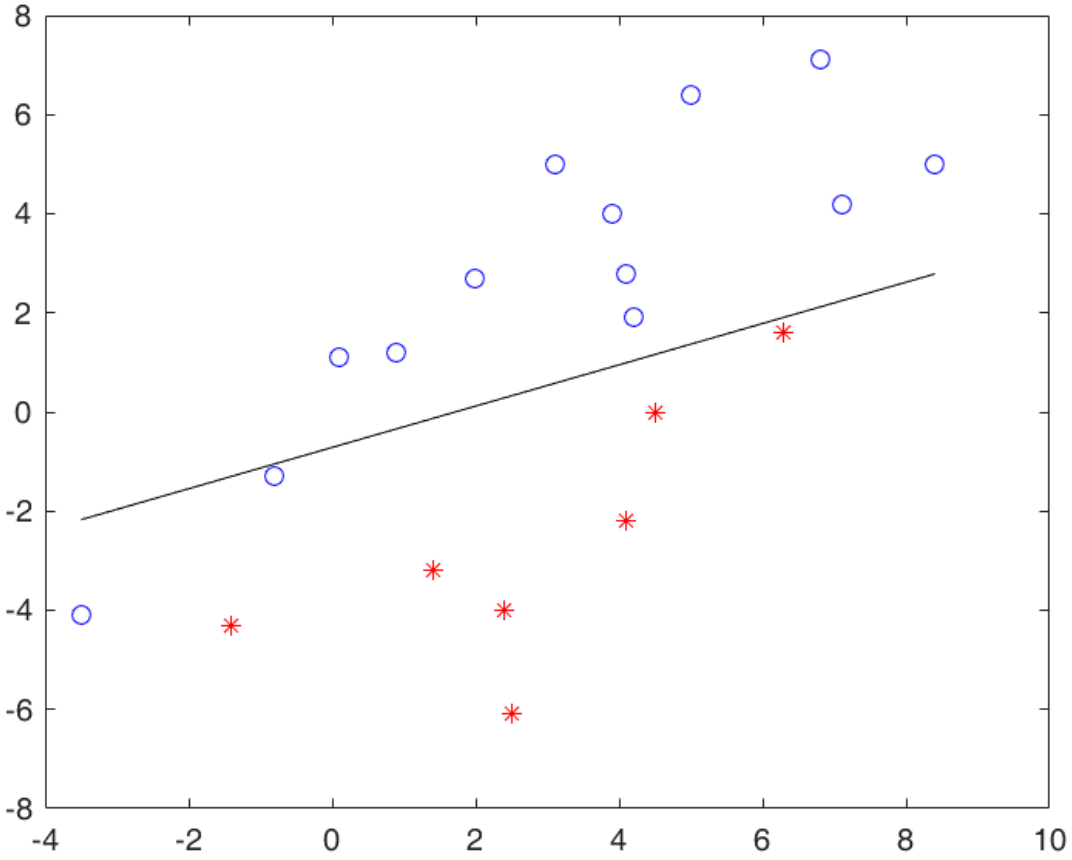
人工神经网络的历史 – 感知器

感知器算法演示：



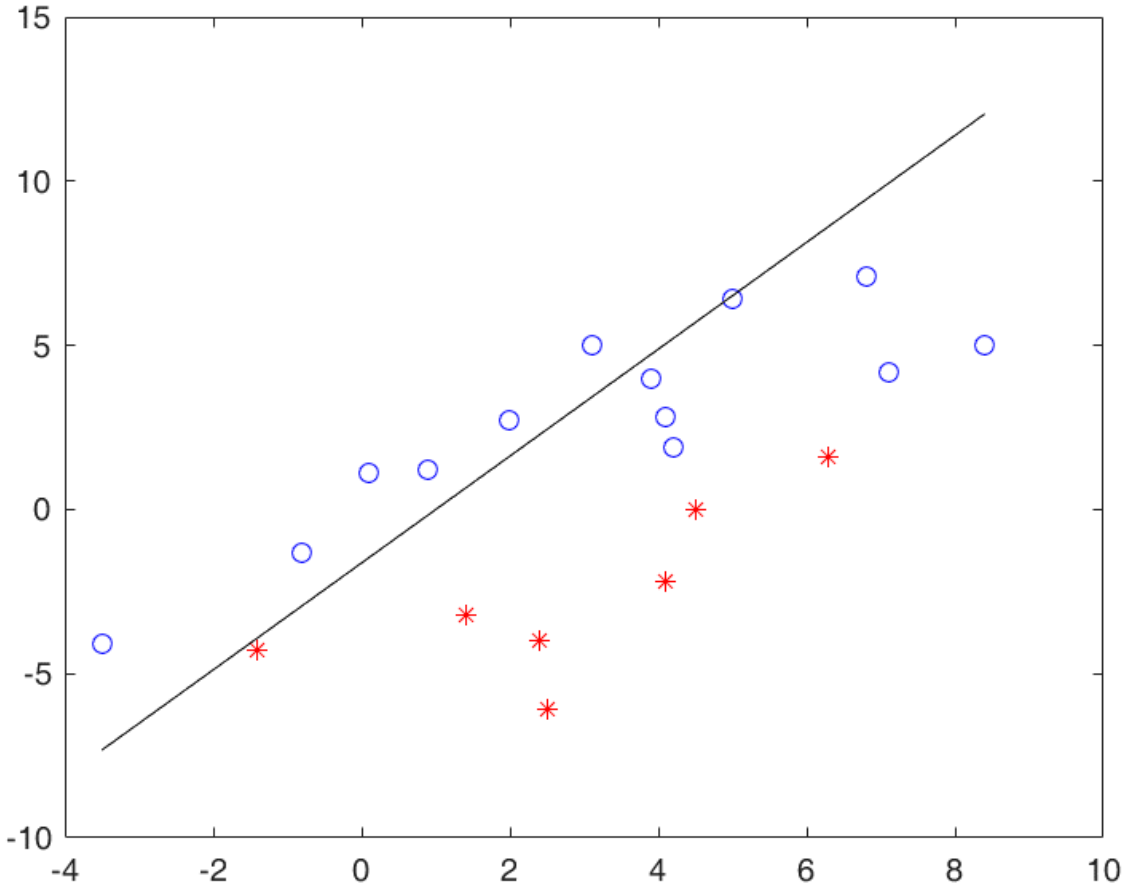
人工神经网络的历史 – 感知器

感知器算法演示：



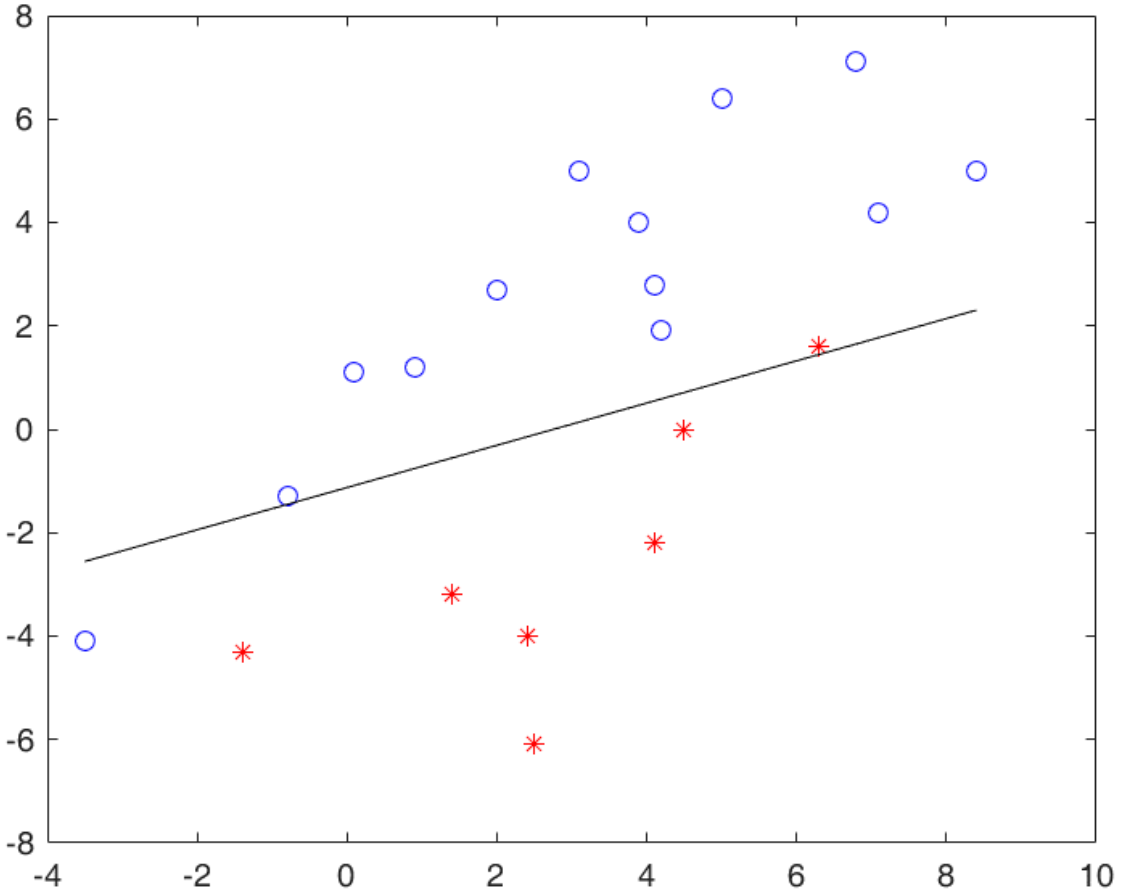
人工神经网络的历史 – 感知器

感知器算法演示：



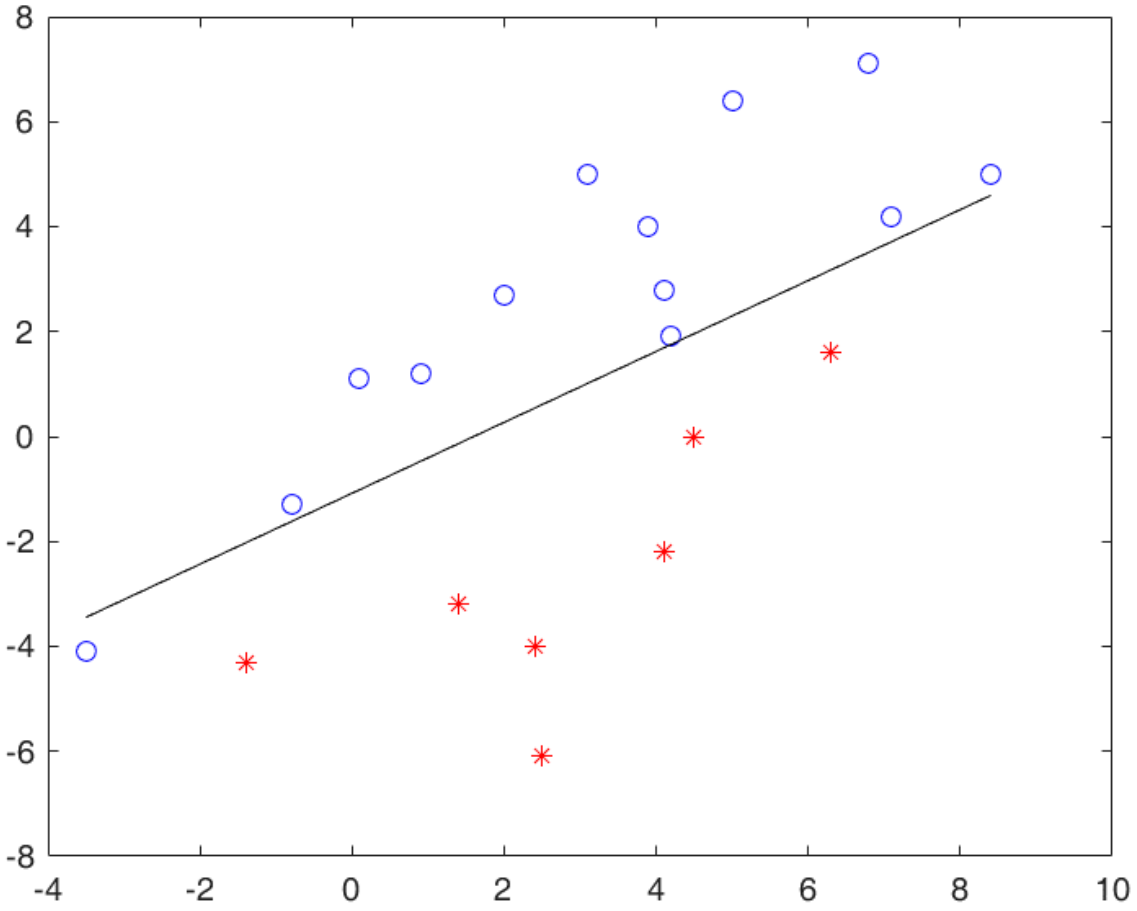
人工神经网络的历史 – 感知器

感知器算法演示：



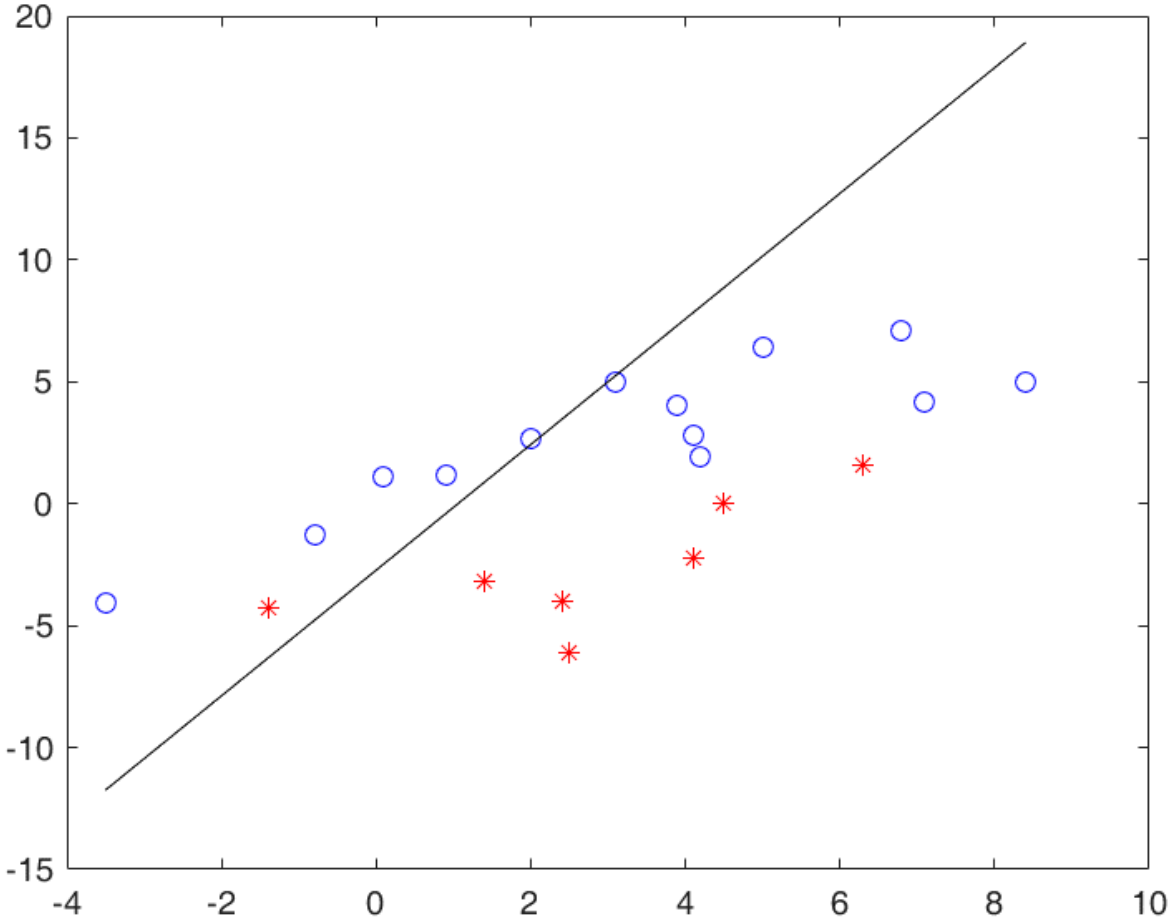
人工神经网络的历史 – 感知器

感知器算法演示：



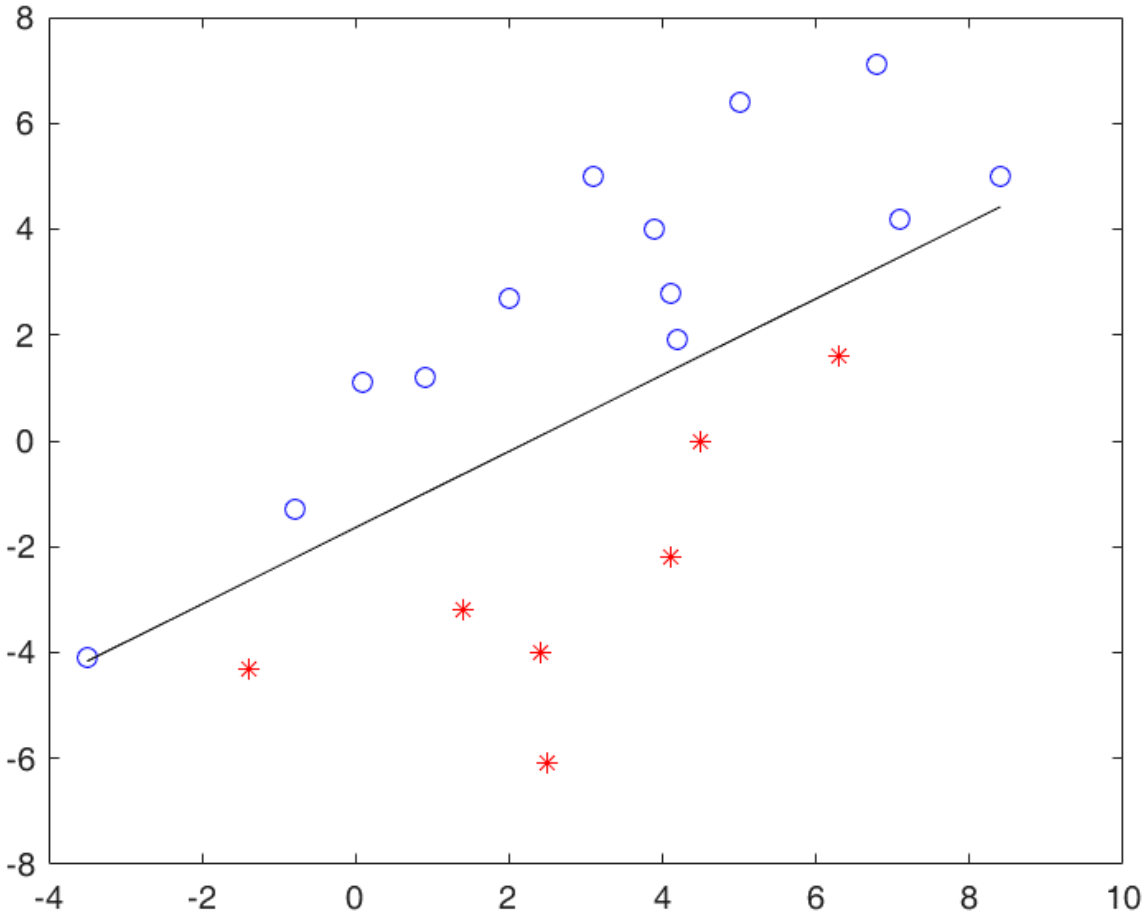
人工神经网络的历史 – 感知器

感知器算法演示：

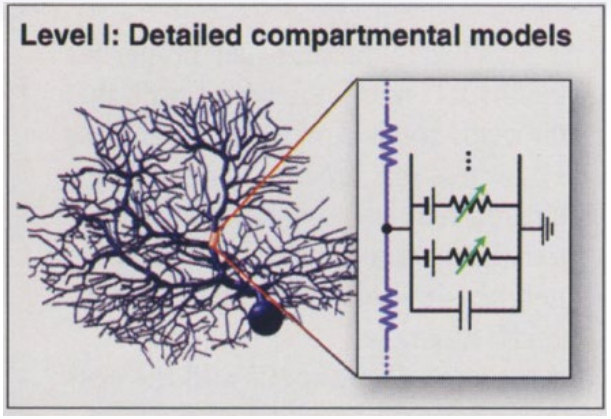


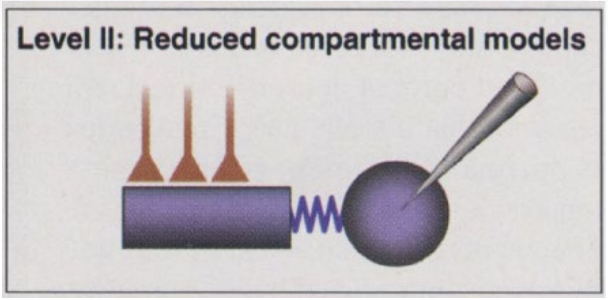
人工神经网络的历史 – 感知器

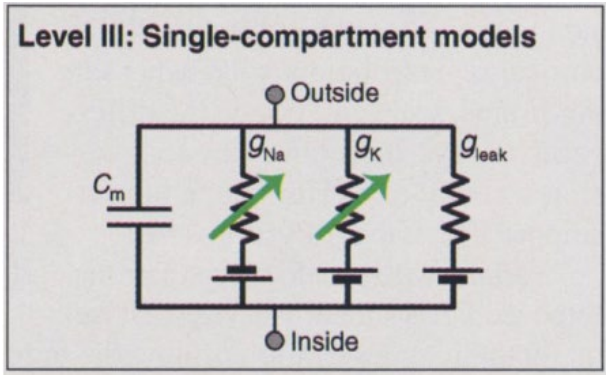
感知器算法演示：

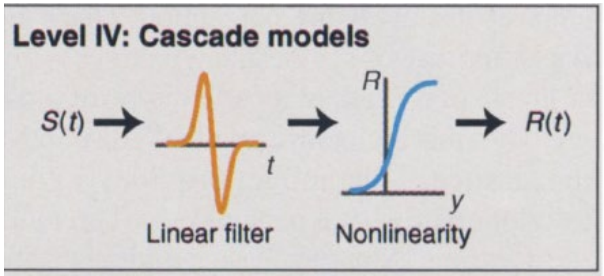


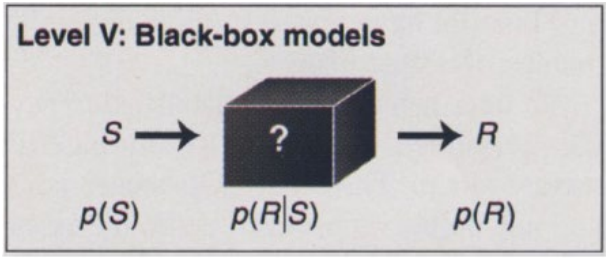
更复杂的神经元模型







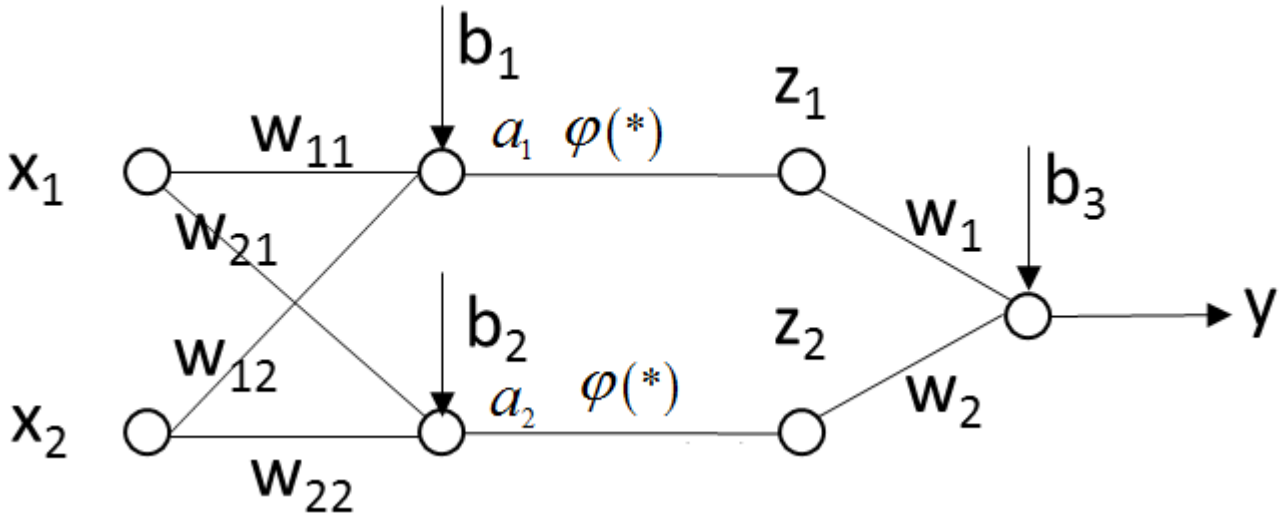




Andreas V. M. Herz et al. Modeling single-neuron dynamics and computation: A balance of detail and abstraction, Science 2014

人工神经网络的历史 --多层网络

两层神经网络例子：



$$a_1 = \omega_{11}x_1 + \omega_{12}x_2 + b_1$$

$$a_2 = \omega_{21}x_1 + \omega_{22}x_2 + b_2$$

$$z_1 = \varphi(a_1)$$

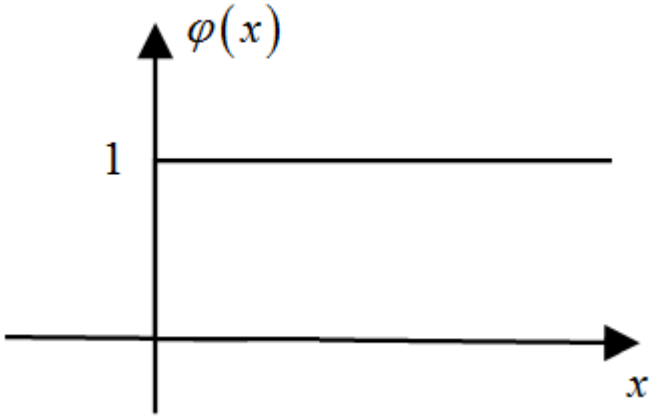
$$z_2 = \varphi(a_2)$$

$$y = \omega_1 z_1 + \omega_2 z_2 + b_3$$

（注意：其中$\varphi(*)$为非线性函数）

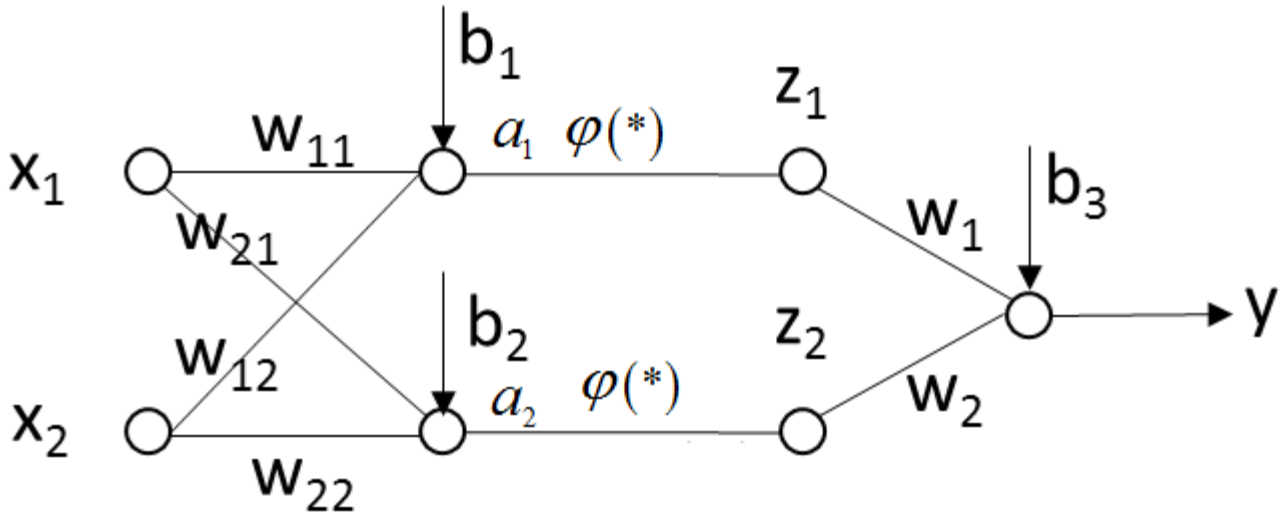
人工神经网络的历史 --多层网络

定理：当$\varphi(x)$为阶跃函数时，三层网络可以模拟任意决策面。



人工神经网络的历史 --多层网络

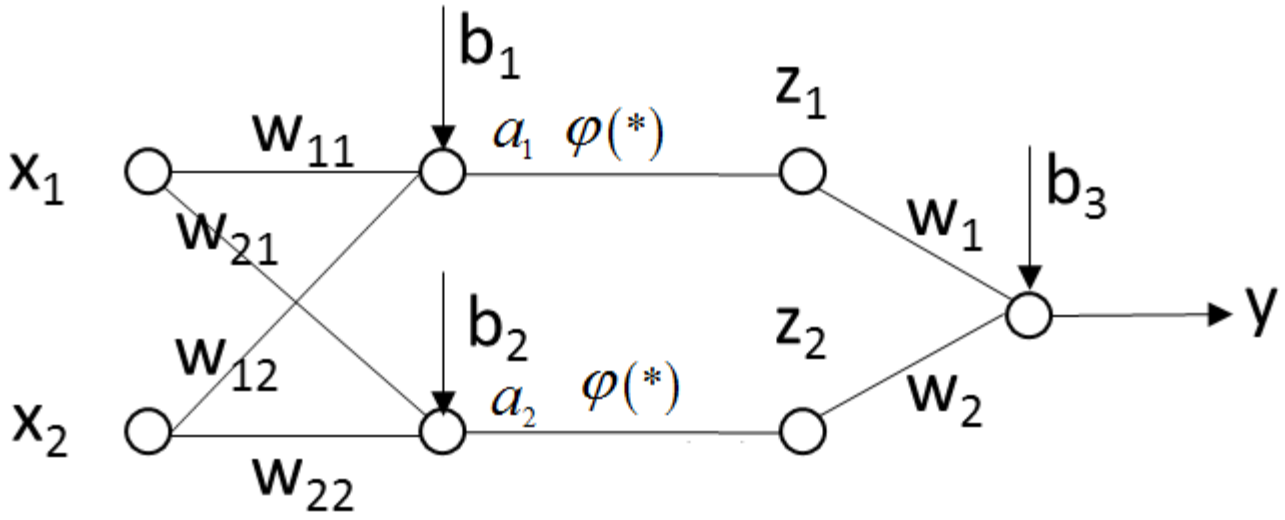
学习算法：后向传播（Back Propogation Algorithm）



输入 (X,Y)，其中 $X=\begin{bmatrix}x_1\\x_2\end{bmatrix}$，$Y$ 是标签值（label），即我们希望改变 ω 和 b ，使得标签值 Y 与实际的网络输出值 y 尽量接近。

人工神经网络的历史 --多层网络

学习算法：后向传播（Back Propogation Algorithm）

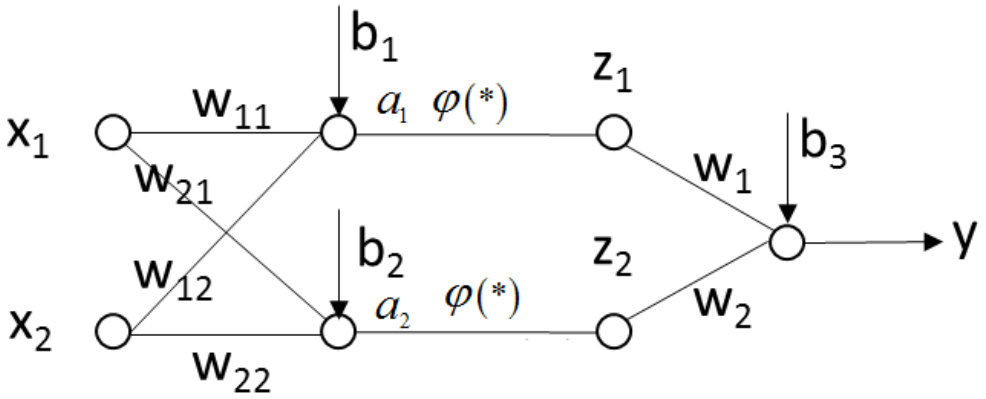


定义目标函数

$$Minimize: E(\omega, b) = E_{(X,Y)}[(Y - y)^2]$$

人工神经网络的历史 --多层网络

学习算法：后向传播（Back Propogation Algorithm）



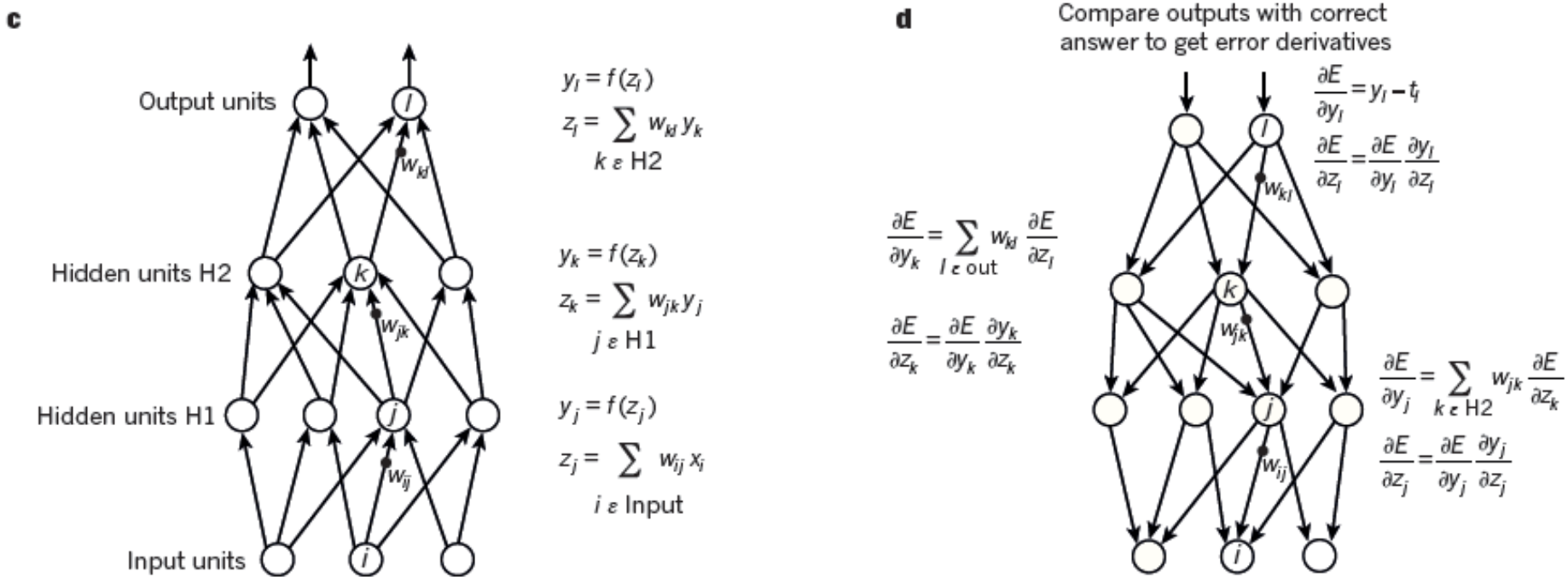
$$Minimize: E(\omega, b) = E_{(X,Y)}[(Y - y)^2]$$

优化算法：最简单的梯度下降法（Gradient Descent Method）

$$\omega^{(new)} = \omega^{(old)} - \alpha \frac{\partial E}{\partial \omega}|_{\omega^{(old)}, b^{(old)}}$$

$$b^{(new)} = b^{(old)} - \alpha \frac{\partial E}{\partial b}|_{\omega^{(old)}, b^{(old)}}$$

人工神经网络的历史 --多层网络

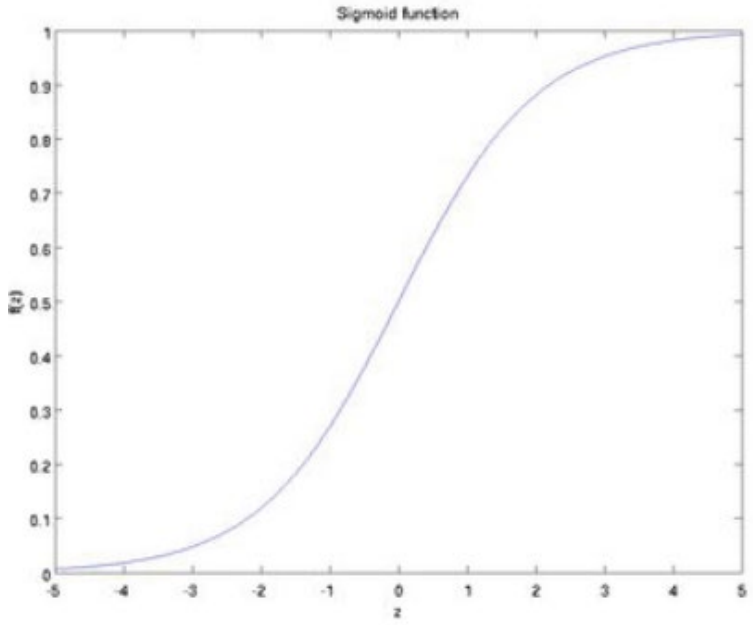


Picture extracted from Hinton et al 2016.

人工神经网络的历史 --多层网络

常见非线性函数 $\varphi(x)$ 的选择。

（1）Sigmoid

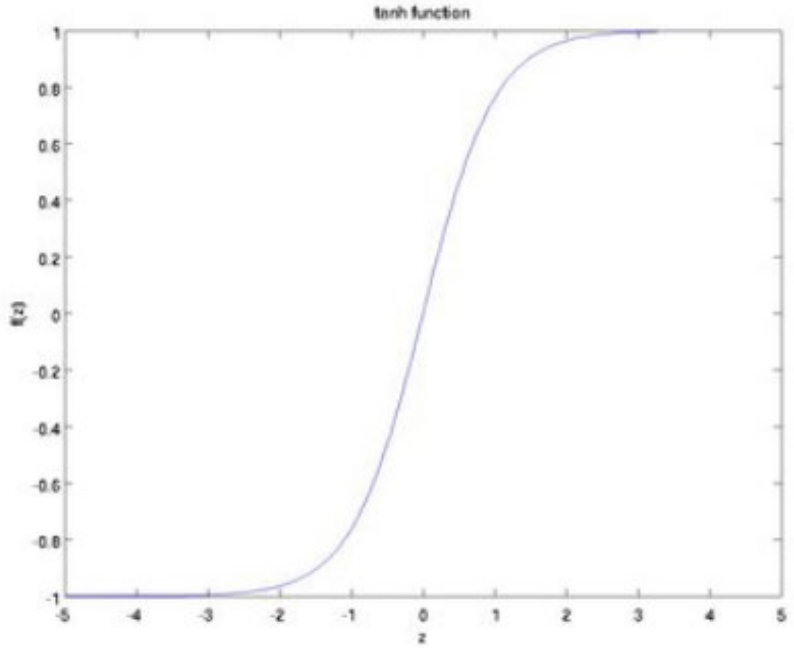


$$\varphi(x) = \frac{1}{1+e^{-x}} \qquad \varphi'(x) = \varphi(x)[1-\varphi(x)]$$

人工神经网络的历史 --多层网络

常见非线性函数 $\varphi(x)$ 的选择。

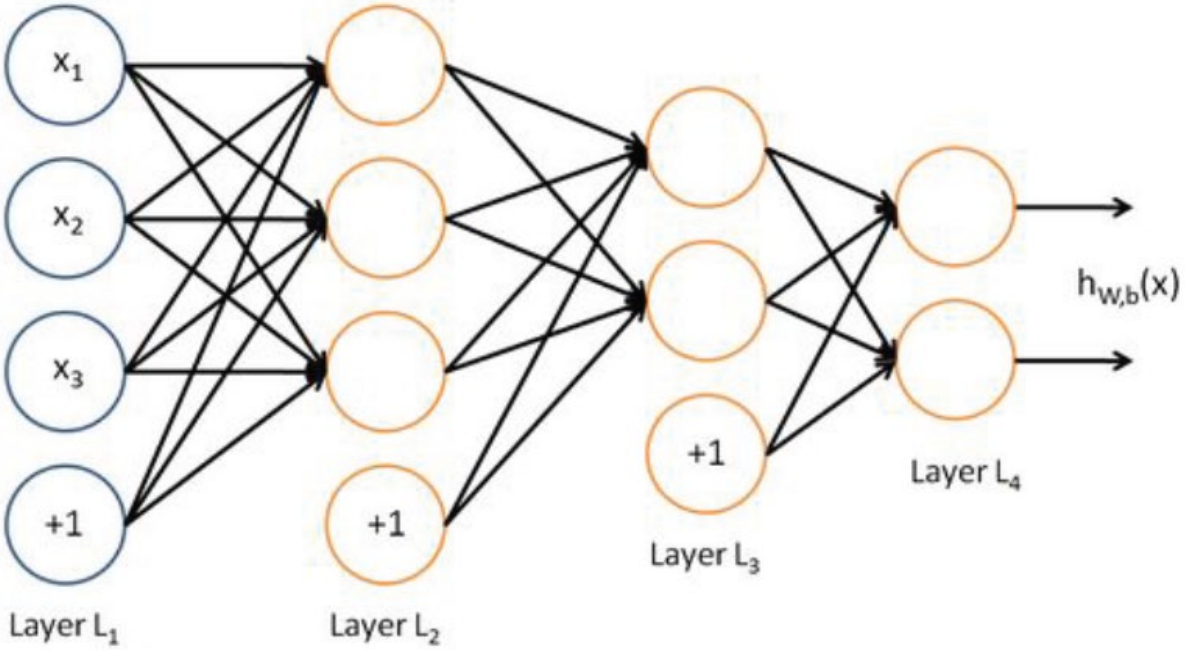
（2）tanh



$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad \varphi'(x) = 1 - [\varphi(x)]^2$$

人工神经网络的历史 --多层网络

大于两层的神经网络



人工神经网络的历史 --多层网络

多层神经网络的优势：

（1）基本单元简单，多个基本单元可扩展为非常复杂的非线性函数。因此易于构建，同时模型有很强的表达能力。

（2）训练和测试的计算并行性非常好，有利于在分布式系统上的应用。

（3）模型构建来源于对人脑的仿生，话题丰富，各种领域的研究人员都有兴趣，都能做贡献。

人工神经网络的历史 --多层网络

多层神经网络的劣势：

（1）数学不漂亮，优化算法只能获得局部极值，算法性能与初始值有关。

（2）不可解释。训练神经网络获得的参数与实际任务的关联性非常模糊。

（2）模型可调整的参数很多 （网络层数、每层神经元个数、非线性函数、学习率、优化方法、终止条件等等），使得训练神经网络变成了一门“艺术”。

（3）如果要训练相对复杂的网络，需要大量的训练样本。

训练建议

（1）一般情况下，在训练集上的目标函数的平均值（cost）会随着训练的深入而不断减小，如果这个指标有增大情况，停下来。有两种情况：第一是采用的模型不够复杂，以致于不能在训练集上完全拟合；第二是已经训练很好了。

（2）分出一些验证集（Validation Set），训练的本质目标是在验证集上获取最大的识别率。因此训练一段时间后，必须在验证集上测试识别率，保存使验证集上识别率最大的模型参数，作为最后结果。

（3）注意调整学习率（Learning Rate），如果刚训练几步cost就增加，一般来说是学习率太高了；如果每次cost变化很小，说明学习率太低。

参数设置

1. 随机梯度下降（Stochastic Gradient Descent, SGD）
2. 激活函数选择
3. 训练数据初始化
4. （W, b)的初始化
5. Batch normalization
6. 目标函数选择
7. 参数更新策略
8. 训练建议

随机梯度下降

随机梯度下降

（1）不用每输入一个样本就去变换参数，而是输入一批样本（叫做一个BATCH或MINI-BATCH），求出这些样本的梯度平均值后，根据这个平均值改变参数。

（2）在神经网络训练中，BATCH的样本数大致设置为50-200不等。

随机梯度下降

```
batch_size = option.batch_size;
m = size(train_x,1);
num_batches = m / batch_size;
for k = 1 : iteration
    kk = randperm(m);
    for l = 1 : num_batches
        batch_x = train_x(kk((l - 1) * batch_size
+ 1 : l * batch_size), :);
        batch_y = train_y(kk((l - 1) * batch_size
+ 1 : l * batch_size), :);
        nn = nn_forward(nn,batch_x,batch_y);
        nn = nn_backpropagation(nn,batch_y);
        nn = nn_applygradient(nn);
    end
end
```

随机梯度下降

```
m = size(batch_x,2);
```

前向计算

```
nn.cost(s) = 0.5 / m * sum(sum((nn.a{k} - batch_y).^2))
+ 0.5 * nn.weight_decay * cost2;
```

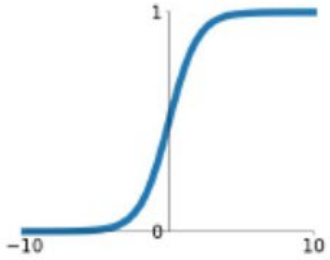
后向传播

```
nn.W_grad{nn.depth-1} =
nn.theta{nn.depth}*nn.a{nn.depth-1}'/m +
nn.weight_decay*nn.W{nn.depth-1};
nn.b_grad{nn.depth-1} = sum(nn.theta{nn.depth},2)/m;
```

激活函数

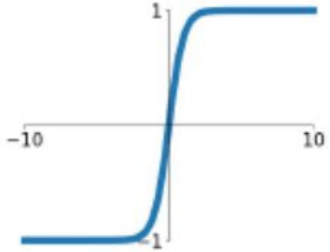
Sigmoid

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



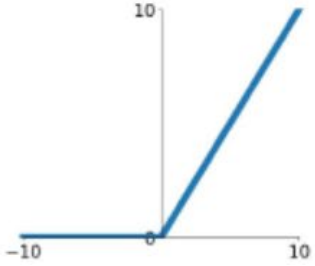
tanh

$$\tanh(x)$$



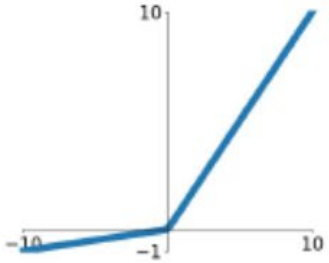
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

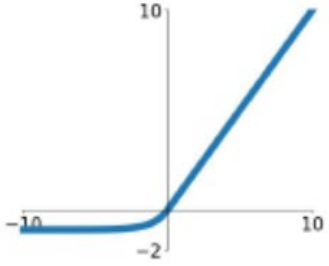


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

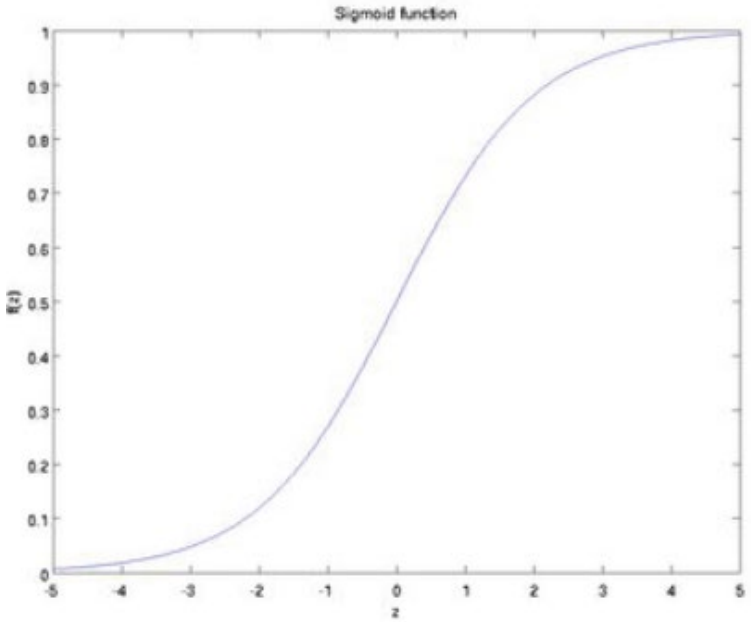
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



激活函数

常见非线性函数 $\varphi(x)$ 的选择。

（1）Sigmoid

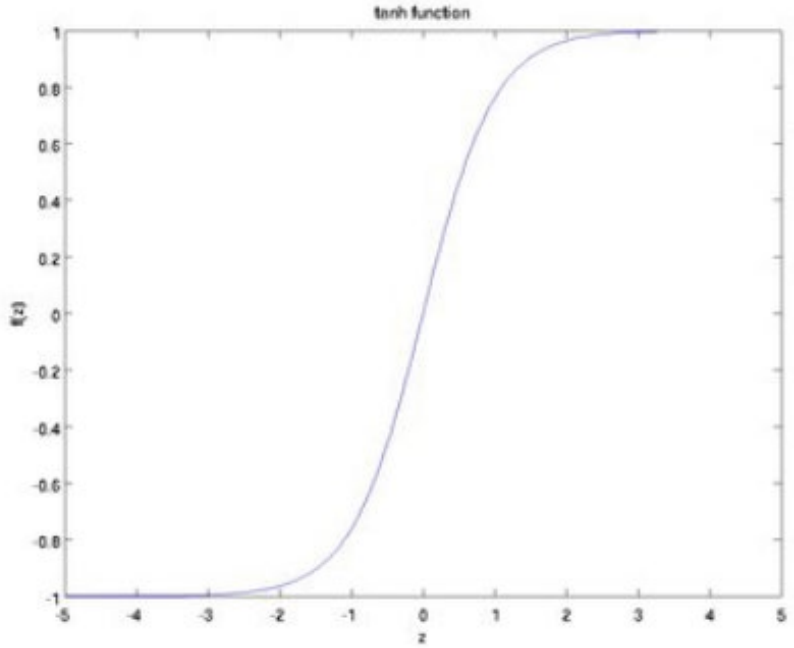


$$\varphi(x)=\frac{1}{1+e^{-x}} \qquad \varphi'(x)=\varphi(x)[1-\varphi(x)]$$

激活函数

常见非线性函数 $\varphi(x)$ 的选择。

（2）tanh



$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad \varphi'(x) = 1 - [\varphi(x)]^2$$

目标函数

(1) Mean Square Error (MSE)

$$E = \frac{1}{2}\|y - Y\|^2$$

其中 y是预测值， Y是真实值（Ground Truth）。

目标函数

(2) Cross Entropy (交叉熵)

$$E = -\sum_{i=1}^{M} Y_i \, log(y_i)$$

其中 y是预测值， Y是真实值（Ground Truth）。

要求：

a) Y是概率向量。即每一个分量$0 \leq Y_i \leq 1$，且 $\sum_{i=1}^{M} Y_i = 1$。

b) 最后一层输出z与y的关系为：

$$z = softmax(y)$$

即

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{M} exp(z_j)}$$

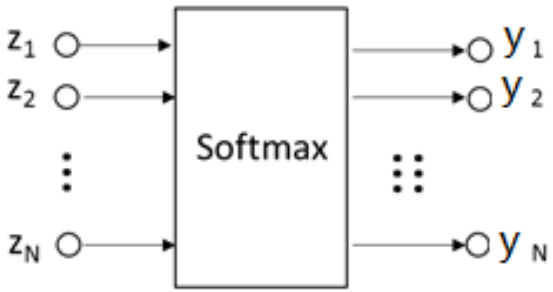
目标函数

(2) Cross Entropy (交叉熵)

$$E = -\sum_{i=1}^{M} Y_i \, log(y_i)$$

定理1. $0 \le E \le \log M$

定理2. $\frac{\partial E}{\partial z_i} = y_i - Y_i$



目标函数

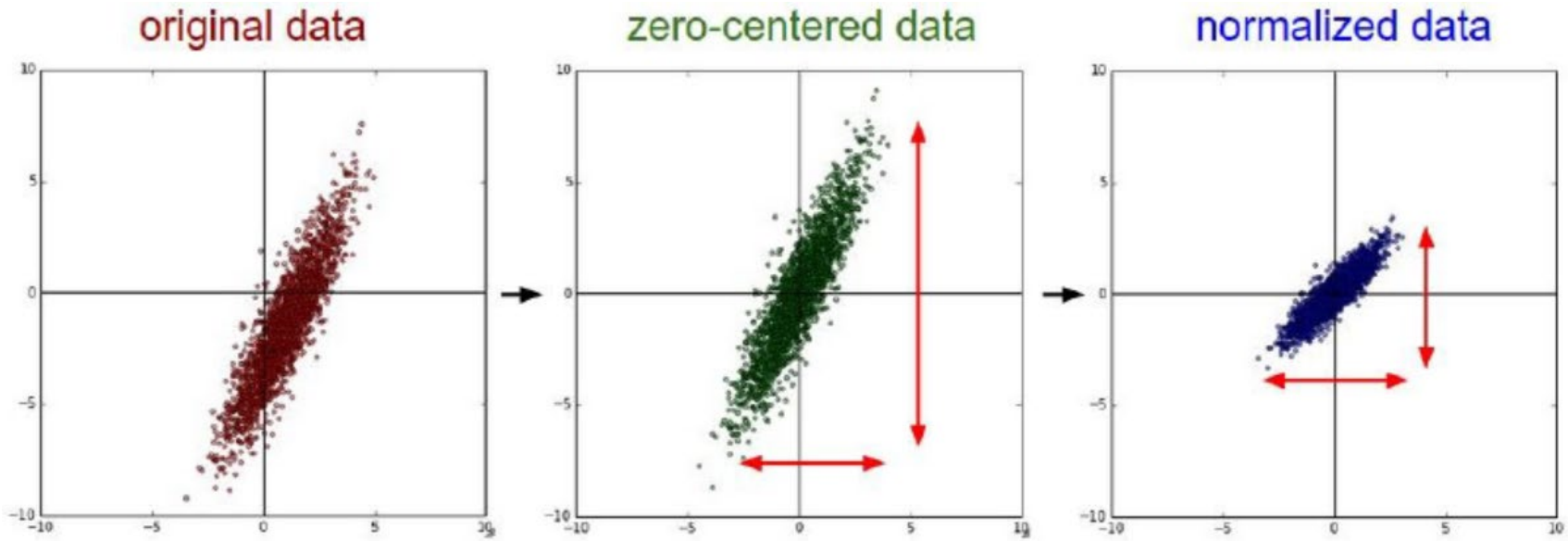
代码：

```
if strcmp(nn.objective_function,'MSE')
    nn.cost(s) = 0.5 / m * sum(sum((nn.a{k} -
batch_y).^2)) + 0.5 * nn.weight_decay * cost2;

elseif strcmp(nn.objective_function,'Cross Entropy')
   nn.cost(s) = -0.5*sum(sum(batch_y.*log(nn.a{k})))/m
+ 0.5 * nn.weight_decay * cost2;
```

训练数据初始化

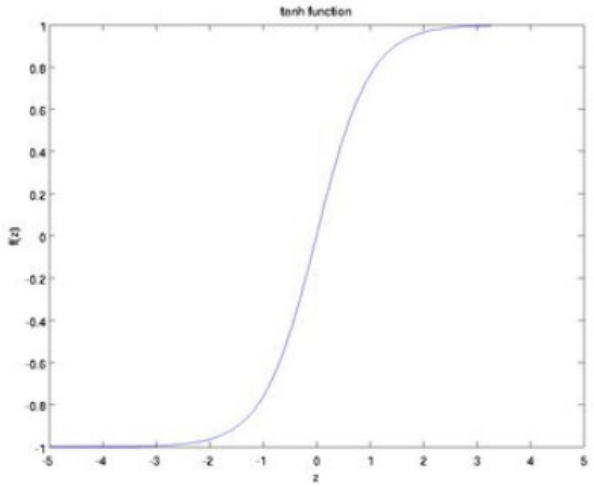
建议：做均值和方差归一化。。



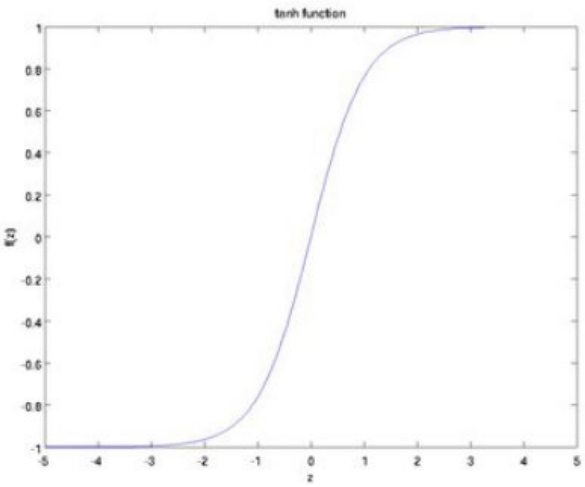
$$newX = \frac{X - mean(X)}{std(X)}$$

```
[U,V] = size(xTraining);
avgX = mean(xTraining);
sigma = std(xTraining);
xTraining = (xTraining -
repmat(avgX,U,1))./repmat(sigma,U,1);
```

（W,b）的初始化



Sigmoid



tanh

梯度消失现象：如果 W^TX+b 一开始很大或很小，那么梯度将趋近于0，反向传播后前面与之相关的梯度也趋近于0，导致训练缓慢。

因此，我们要使 W^TX+b 一开始在零附近。

（W,b）的初始化

一种比较简单有效的方法是：（W,b）初始化从区间 $\left(-\frac{1}{\sqrt{d}},\frac{1}{\sqrt{d}}\right)$ 均匀随机取值。其中 d 为（W,b）所在层的神经元个数。

可以证明，如果X服从正态分布，均值0，方差1，且各个维度无关，而（W,b）是 $\left(-\frac{1}{\sqrt{d}},\frac{1}{\sqrt{d}}\right)$ 的均匀分布，则 W^TX+b 是均值为0， 方差为1/3的正态分布。

```
nn.W{k} = 2*rand(height, width)/sqrt(width)-1/sqrt(width);
nn.b{k} = 2*rand(height, 1)/sqrt(width)-1/sqrt(width);
```

（W,b）的初始化

参数初始化是一个热点领域，相关论文包括：

Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

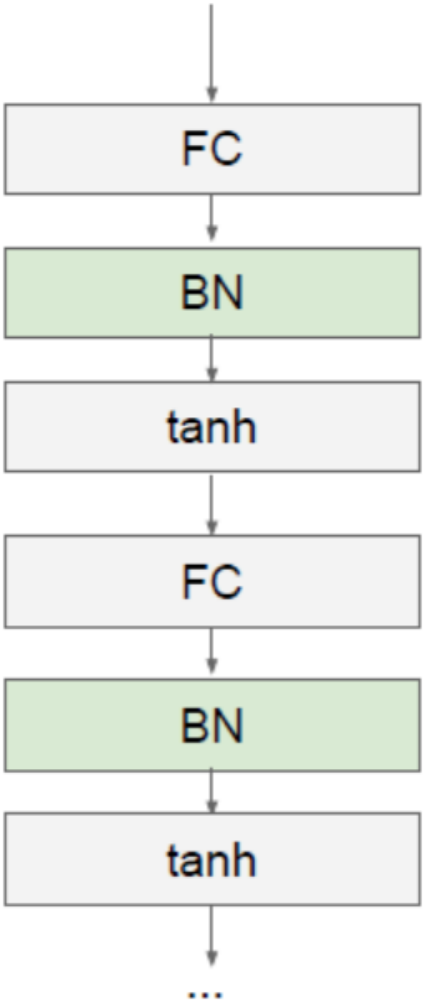
…

Batch Normalization

论文：Batch normalization accelerating deep network training by reducing internal covariate shift (2015)

基本思想：既然我们希望每一层获得的值都在0附近，从而避免梯度消失现象，那么我们为什么不直接把每一层的值做基于均值和方差的归一化呢？

Batch Normalization



每一层FC（Fully Connected Layer）
接一个BN（Batch Normalization）层

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

算法流程

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Batch Normalization

前向计算

```
y = nn.W{k-1} * nn.a{k-1} + repmat(nn.b{k-1},1,m);
if nn.batch_normalization
    nn.E{k-1} = nn.E{k-1}*nn.vecNum + sum(y,2);
    nn.S{k-1} = nn.S{k-1}.^2*(nn.vecNum-1) + (m-
1)*std(y,0,2).^2;
    nn.vecNum = nn.vecNum + m;
    nn.E{k-1} = nn.E{k-1}/nn.vecNum;
    nn.S{k-1} = sqrt(nn.S{k-1}/(nn.vecNum-1));
    y = (y - repmat(nn.E{k-1},1,m))./repmat(nn.S{k-
1}+0.0001*ones(size(nn.S{k-1})),1,m);
    y = nn.Gamma{k-1}*y+nn.Beta{k-1};
end;
switch nn.activaton_function
    case 'sigmoid'
        nn.a{k} = sigmoid(y);
    case 'tanh'
        nn.a{k} = tanh(y);
```

Batch Normalization

后向传播

```
if nn.batch_normalization
    x = nn.W{k-1} * nn.a{k-1} + repmat(nn.b{k-1},1,m);
    x = (x - repmat(nn.E{k-1},1,m))./repmat(nn.S{k-
1}+0.0001*ones(size(nn.S{k-1})),1,m);
    temp = nn.theta{k}.*x;
    nn.Gamma_grad{k-1} = sum(mean(temp,2));
    nn.Beta_grad{k-1} = sum(mean(nn.theta{k},2));
    nn.theta{k} = nn.Gamma{k-1}*nn.theta{k}./repmat((nn.S{k-
1}+0.0001),1,m);
end;
nn.W_grad{k-1} = nn.theta{k}*nn.a{k-1}'/m +
nn.weight_decay*nn.W{k-1};
nn.b_grad{k-1} = sum(nn.theta{k},2)/m;
```

目标函数选择

1. 正则项 （Regulation Term）

$$L(W) = F(W) + R(W)$$

$$= \frac{1}{2}\left(\sum_{i=1}^{batch\_size} \|y_i - Y_i\|^2 + \beta \sum_k \sum_l W_{k,l}^2 \right)$$

前向计算

```
cost2 = cost2 +  sum(sum(nn.W{k-1}.^2));
nn.cost(s) = 0.5 / m * sum(sum((nn.a{k} - batch_y).^2))
+ 0.5 * nn.weight_decay * cost2;
```

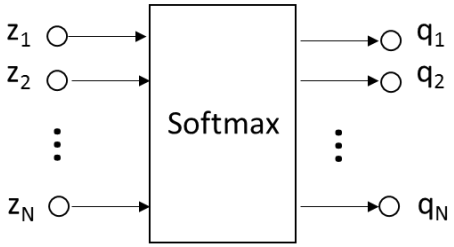
后向传播

```
nn.W_grad{k-1} = nn.theta{k}*nn.a{k-1}'/m +
nn.weight_decay*nn.W{k-1};
```

目标函数选择

2. 如果是分类问题，F(W)可以采用SOFTMAX函数和交叉熵的组合。

（a）SOFTMAX函数：



$$q_i = \frac{exp(z_i)}{\sum_{j=1}^{N} exp(z_j)}$$

我们希望通过此网络学习由 $Z = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ z_N \end{bmatrix}$ 到 $P = \begin{bmatrix} p_1 \\ p_2 \\ . \\ . \\ p_N \end{bmatrix}$ 的映射。其中 $\sum_{i=1}^{N} p_i = 1$

（b）交叉熵（Cross Entropy）：

定义目标函数为 $E = -\sum_{i=1}^{N} p_i * log(q_i)$

目标函数选择

如果F(W)是SOFTMAX函数和交叉熵的组合，那么求导将会有非常简单的形式：

$$\frac{\partial E}{\partial z_i} = q_i - p_i$$

前向计算

```
if strcmp(nn.objective_function,'Cross Entropy')
    nn.cost(s) = -0.5*sum(sum(batch_y.*log(nn.a{k})))/m
+ 0.5 * nn.weight_decay * cost2;
```

后向传播

```
  case 'softmax'
      y = nn.W{nn.depth-1} * nn.a{nn.depth-1} +
repmat(nn.b{nn.depth-1},1,m);
      nn.theta{nn.depth} = nn.a{nn.depth} - batch_y;
```

参数更新策略

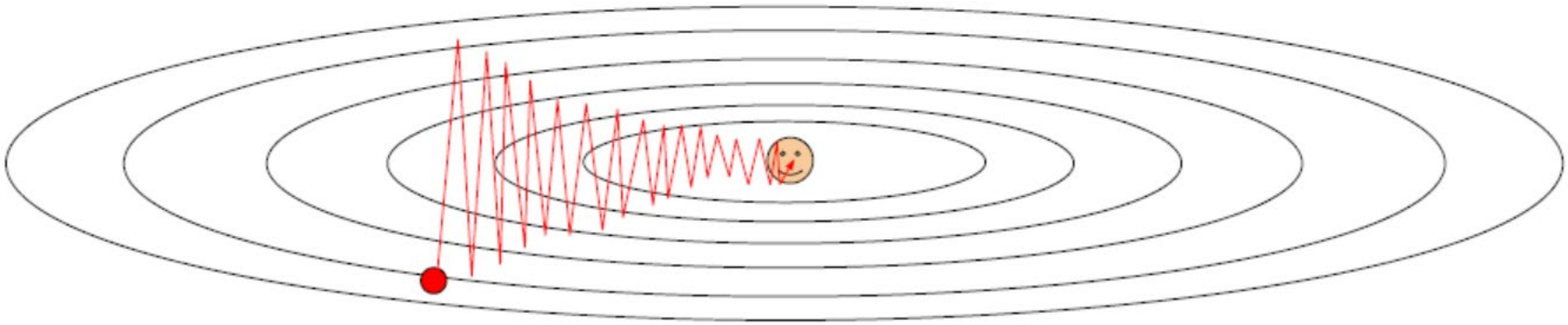
（1）常规的更新 （Vanilla Stochastic Gradient Descent）

```
nn.W{k} = nn.W{k} - nn.learning_rate*nn.W_grad{k};
nn.b{k} = nn.b{k} - nn.learning_rate*nn.b_grad{k};
```

参数更新策略

SGD的问题

（1）(W,b)的每一个分量获得的梯度绝对值有大有小，一些情况下，将会迫使优化路径变成Z字形状。



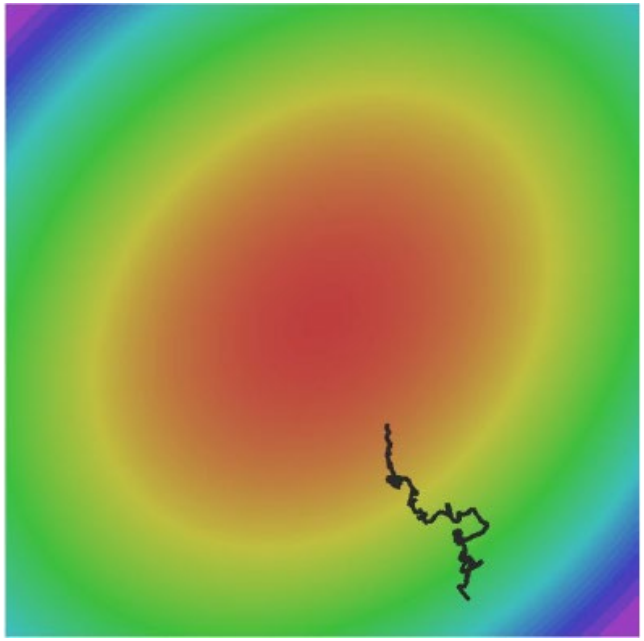
参数更新策略

SGD的问题

（2）SGD求梯度的策略过于随机，由于上一次和下一次用的是完全不同的BATCH数据，将会出现优化的方向随机的情况。

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



参数更新策略

解决各个方向梯度不一致的方法：

（1）AdaGrad

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

参数更新策略

解决各个方向梯度不一致的方法：

（1）AdaGrad

```
if strcmp(nn.optimization_method, 'AdaGrad')

nn.rW{k} = nn.rW{k} + nn.W_grad{k}.^2;
nn.rb{k} = nn.rb{k} + nn.b_grad{k}.^2;

nn.W{k} = nn.W{k} -
nn.learning_rate*nn.W_grad{k}./(sqrt(nn.rW{k})+0.001);

nn.b{k} = nn.b{k} -
nn.learning_rate*nn.b_grad{k}./(sqrt(nn.rb{k})+0.001);
```

参数更新策略

解决各个方向梯度不一致的方法：

（2）RMSProp

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

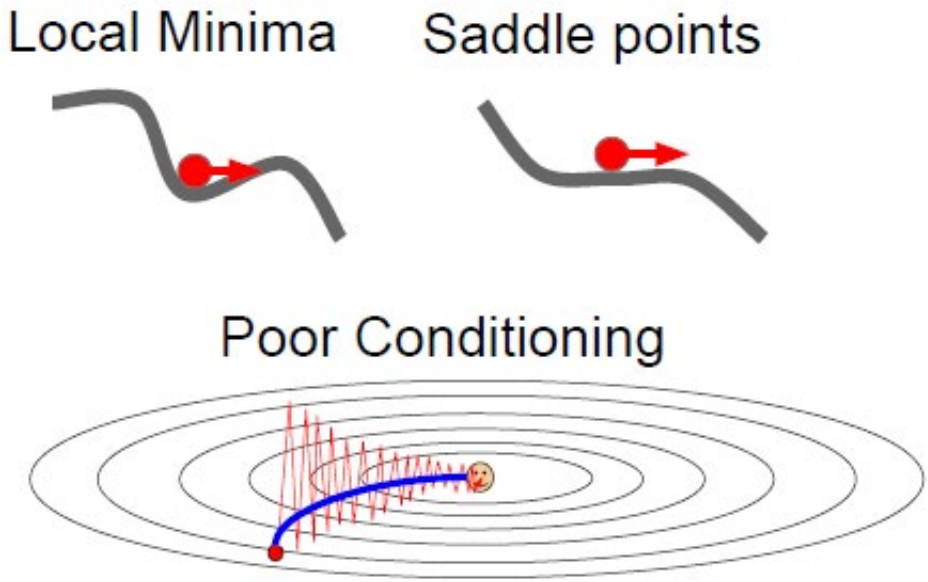
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

参数更新策略

解决梯度随机性问题：

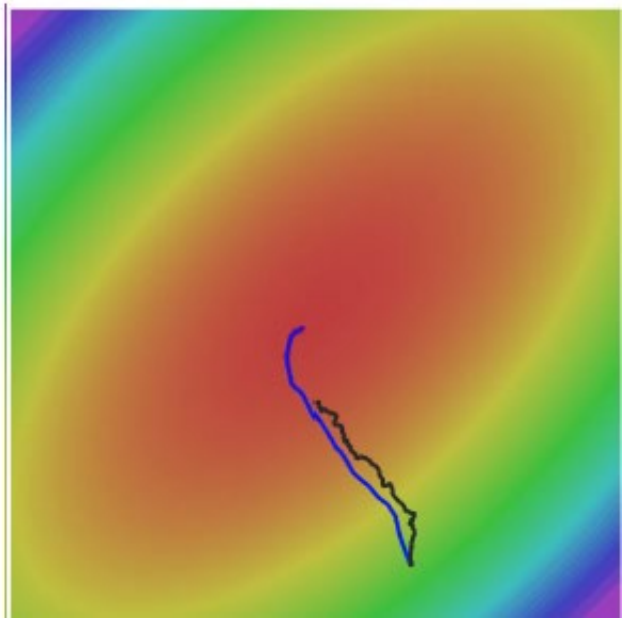
（3）Momentum



SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$



参数更新策略

解决梯度随机性问题：

（3）Momentum

```
if strcmp(nn.optimization_method, 'Momentum')
nn.vW{k} = 0.5*nn.vW{k} + nn.learning_rate*nn.W_grad{k};

nn.vb{k} = 0.5*nn.vb{k} + nn.learning_rate*nn.b_grad{k};

nn.W{k} = nn.W{k} - nn.vW{k};
nn.b{k} = nn.b{k} - nn.vb{k}; %rho = 0.5;
```

参数更新策略

同时两个问题：

（4）Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

参数更新策略

同时两个问题：

（4）Adam

```
if strcmp(nn.optimization_method, 'Adam')
nn.sW{k} = 0.9*nn.sW{k} + 0.1*nn.W_grad{k};
nn.sb{k} = 0.9*nn.sb{k} + 0.1*nn.b_grad{k};
nn.rW{k} = 0.999*nn.rW{k} + 0.001*nn.W_grad{k}.^2;
nn.rb{k} = 0.999*nn.rb{k} + 0.001*nn.b_grad{k}.^2;

nn.W{k} = nn.W{k} -
10*nn.learning_rate*nn.sW{k}./sqrt(1000*nn.rW{k}+0.00001);

nn.b{k} = nn.b{k} -
10*nn.learning_rate*nn.sb{k}./sqrt(1000*nn.rb{k}+0.00001);
%rho1 = 0.9, rho2 = 0.999, delta = 0.00001
```

参数更新策略

解决：

（2）RMSProp

```
if strcmp(nn.optimization_method, 'RMSProp')
nn.rW{k} = 0.9*nn.rW{k} + 0.1*nn.W_grad{k}.^2;
nn.rb{k} = 0.9*nn.rb{k} + 0.1*nn.b_grad{k}.^2;

nn.W{k} = nn.W{k} -
nn.learning_rate*nn.W_grad{k}./(sqrt(nn.rW{k})+0.001);

nn.b{k} = nn.b{k} -
nn.learning_rate*nn.b_grad{k}./(sqrt(nn.rb{k})+0.001);
%rho = 0.9
```

训练建议

（4）近年来流行的深度学习 （Deep Learning）的经验告诉我们，在数据足够的前提下，神经网络层数更多，即深度更深，模型效果更好。

2013年 ALEXNET: 8层
2014年 GOOGLENET INCEPTION: 12层
2015年 RESNET： 152层
2016年 商汤人脸识别系统：1207层

不是越深越好，最近几年，模型复杂度有下降趋势。

训练建议

（5） 分类问题，目标函数尽量用CROSS ENTROPY，类别很多时，如几千类，CROSS ENTROPY的输出过多，性能会下降。因此尽量不要做这么多类。

（6）Batch Normalization 比较好用，用了这个后，对学习率、参数更新策略等不敏感。

（7）如果不用Batch Normalization, 我的经验是，合理变换其他参数组合，也可以达到目的。

（8）由于梯度累积效应，AdaGrad, RMSProp, Adam三种更新策略到了训练的后期会很慢，可以采用提高学习率的策略来补偿这一效应。

Thank you and comments are welcomed