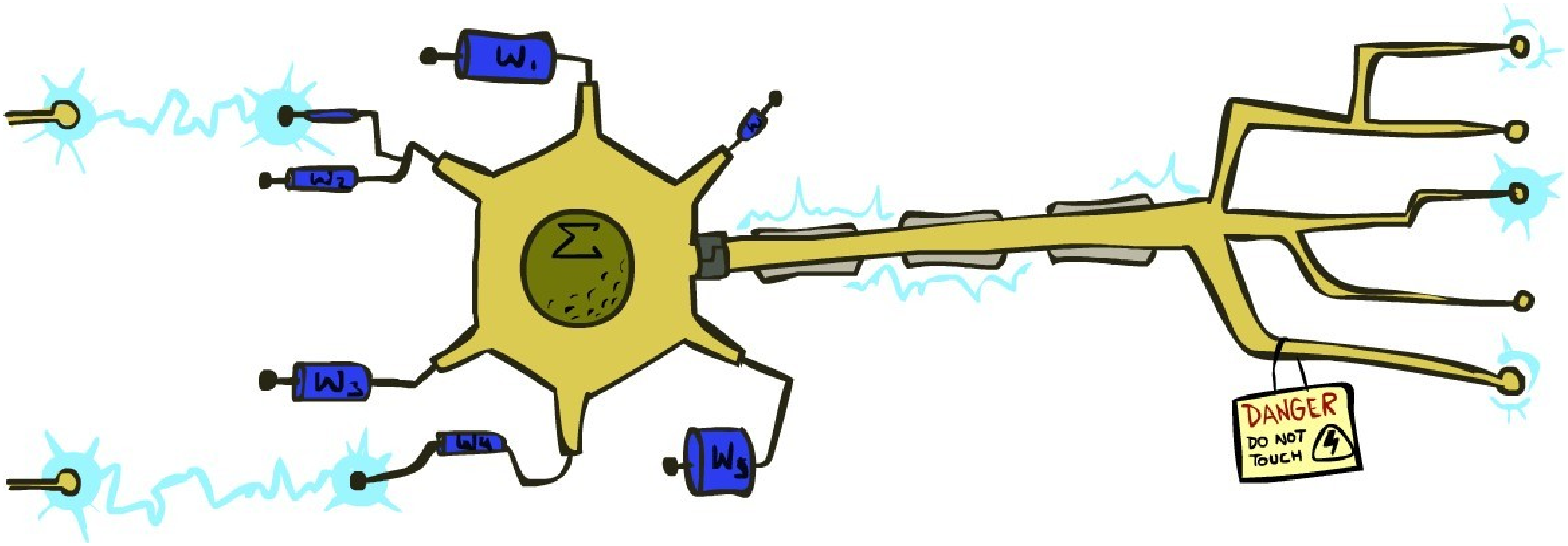
第十八章 样例学习

Linear Regression and Perceptrons

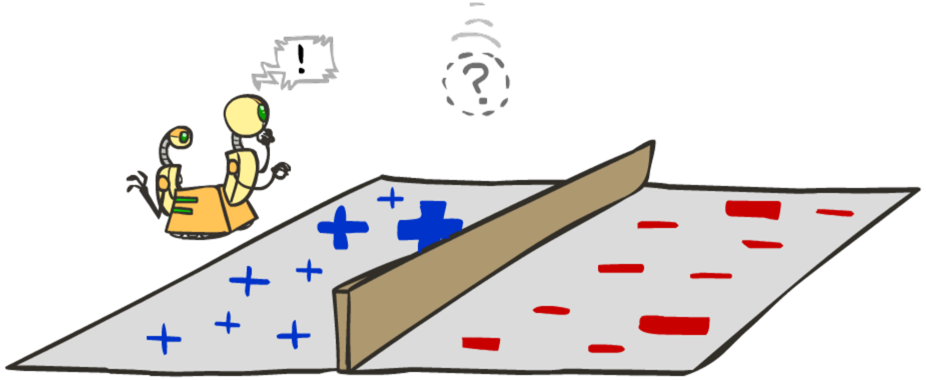


Outline

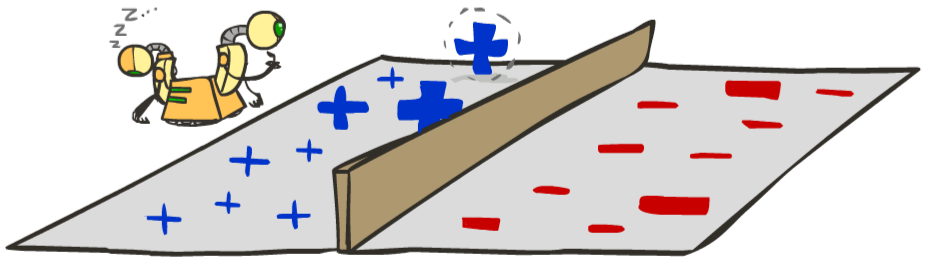
- 18.6 线性回归模型
- 18.7 神经网络
  - M-P 模型、感知机、多层感知机 MLP
  - 神经网络中的学习
    - 感知机的学习规则，MLP 的 BP 算法

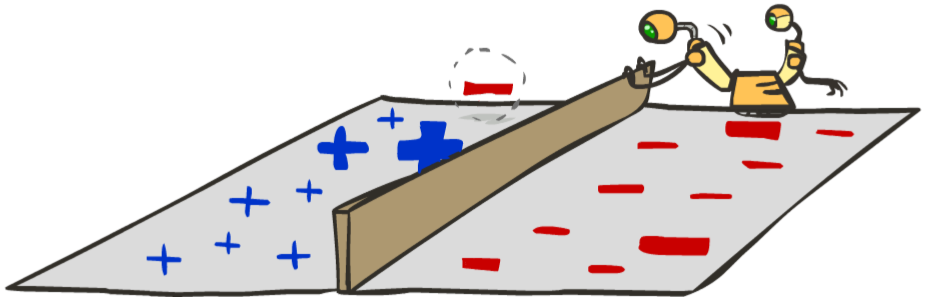
感知机的学习规则

- 随机初始化
- 对于每个训练实例：
  - Classify with current weights



  - If correct (i.e., y=y*), no change!





  - If wrong: adjust the weight vector

Review: Derivatives and Gradients

- What is the derivative of the function $g(x) = x^2 + 3$?

$$\frac{dg}{dx} = 2x$$

- What is the derivative of g(x) at x=5?

$$\frac{dg}{dx}\Big|_{x=5} = 10$$

Review: Derivatives and Gradients

- What is the gradient of the function $g(x, y) = x^2 y$?
  - Recall: Gradient is a vector of partial derivatives with respect to each variable

$$\nabla g = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2xy \\ \\ x^2 \end{bmatrix}$$

- What is the derivative of g(x, y) at x=0.5, y=0.5?

$$\nabla g|_{x=0.5, y=0.5} = \begin{bmatrix} 2(0.5)(0.5) \\ (0.5^2) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix}$$

感知机（ Perceptron ）学习规则

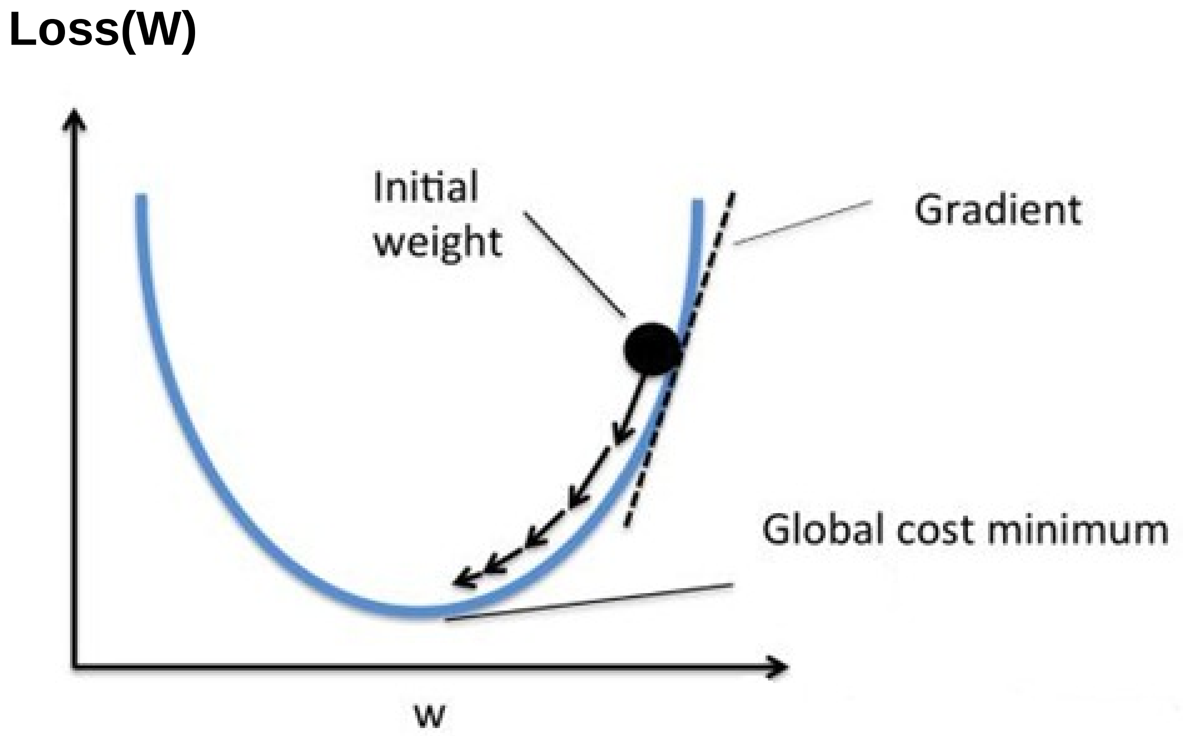
- 采用梯度下降法：沿着被优化函数的梯度搜索，试图最小化损耗值

w ← 参数空间的任何点

loop 直到收敛 **do**

  for w 中的每个 w_i **do**

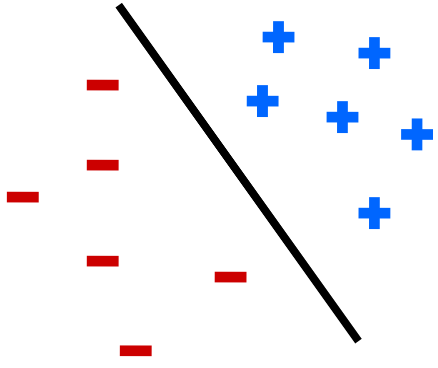
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$



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

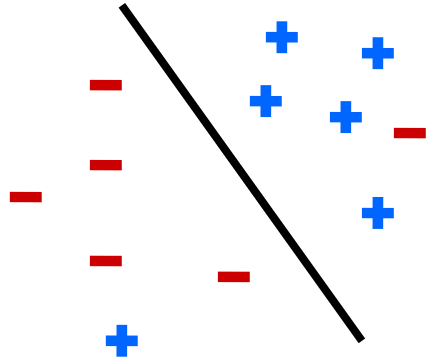
感知机收敛性

Separable



- 一个学习问题是**线性可分的**，当有一个超平面精确地将正反示例分离时
- 收敛性：如果训练数据是线性可分离的，则对训练集重复应用感知机学习，将最终收敛到一个完美的分隔

- 收敛性：如果训练数据是**不可分离的**，如果学习率 □是随着迭代次数 t 递减的 (e.g., $\alpha=1/t$) ，则感知机学习将收敛到最小误差解



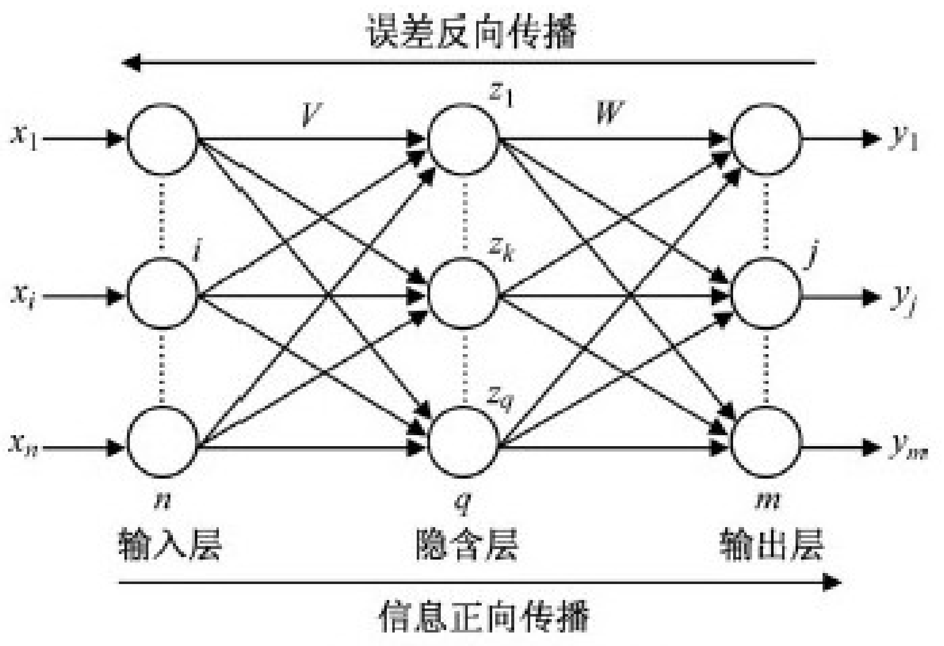
Non-Separable

Outline

- 18.6 线性回归模型
- 18.7 神经网络
  - M-P 模型、感知机、多层感知机 MLP
  - 神经网络中的学习
    - 感知机的学习规则、 MLP 的 BP 算法

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



- 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

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

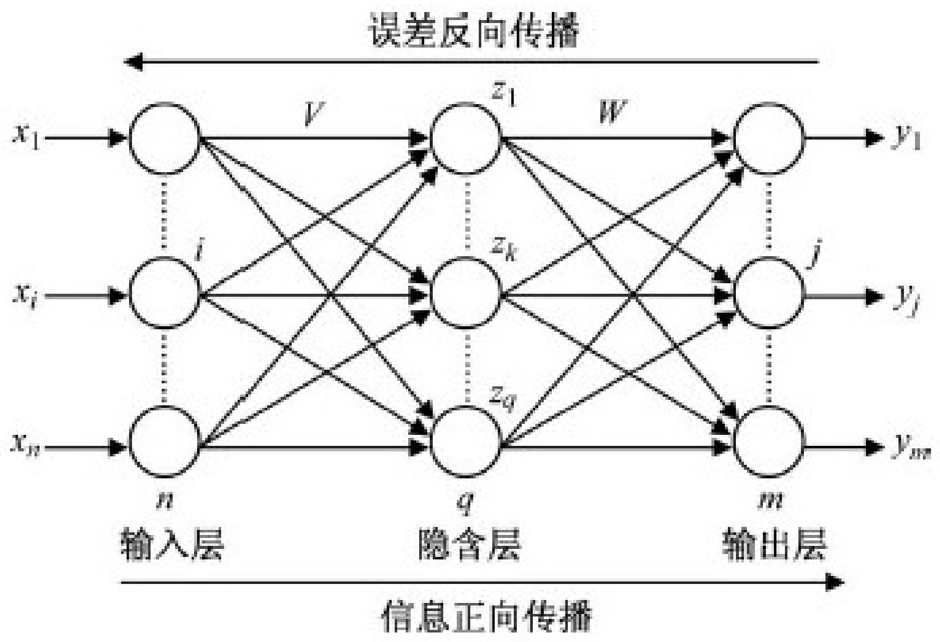
  step2：计算输出层的神经元

  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

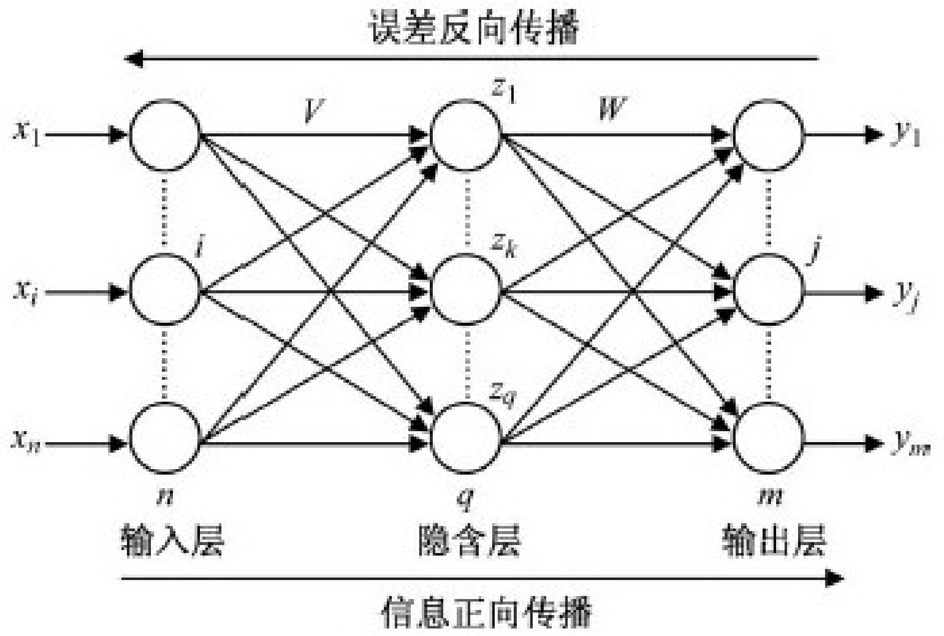
- 反向计算

  step1：计算输出层的梯度

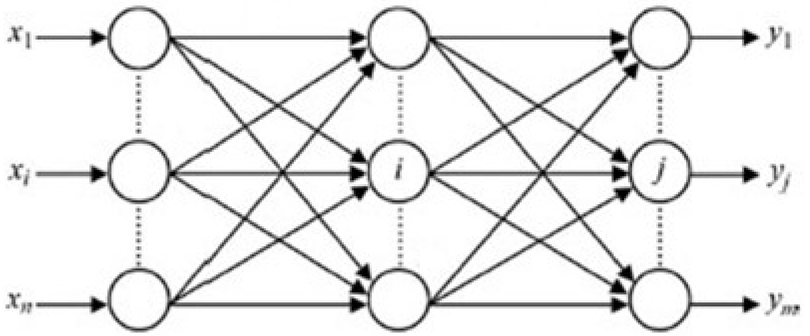
  step2：从输出层开始，循环直到最早的隐藏层

  2.1： 将误差传播回前一层

  2.2： 更新这两层间的权重与阈值



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: Δ, a vector of errors, indexed by network node

   for each weight w_{i,j} in network do
      w_{i,j} ← 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 ← x_i
         for ℓ = 2 to L do
            for each node j in layer ℓ do
               in_j ← Σ_i w_{i,j} a_i
               a_j ← g(in_j)
         /* Propagate deltas backward from output layer to input layer */
         for each node j in the output layer do
            Δ[j] ← g'(in_j) × (y_j − a_j)
         for ℓ = L − 1 to 1 do
            for each node i in layer ℓ do
               Δ[i] ← g'(in_i) Σ_j w_{i,j} Δ[j]
         /* Update every weight in network using deltas */
         for each weight w_{i,j} in network do
            w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
   until some stopping criterion is satisfied
   return network
```

前向计算

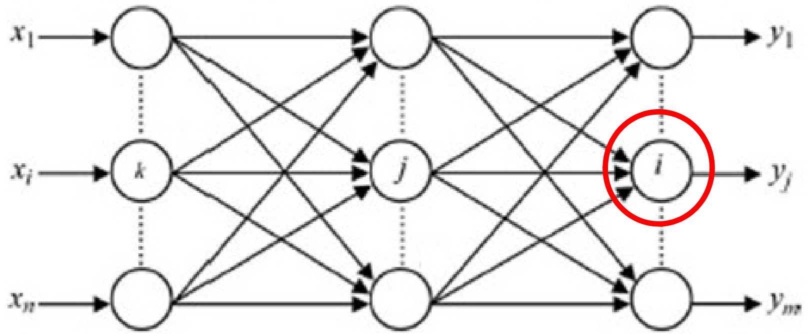
反向计算误差

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：

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



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

其中，求和是对输出层的所有节点进行的。

输出层的权值更新：

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i-a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i-a_i)\frac{\partial g(in_i)}{\partial W_{j,i}}$$

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

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

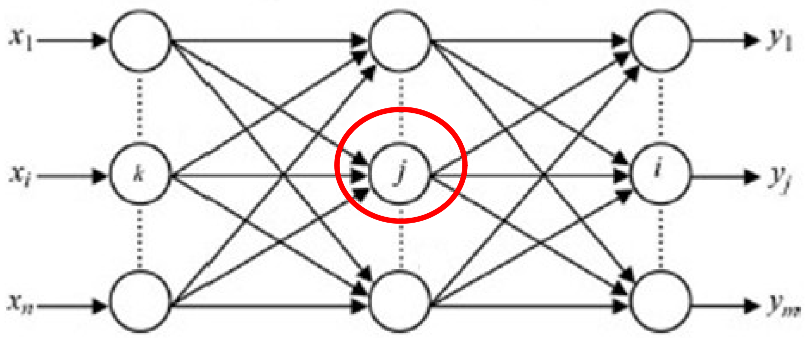
修正误差 $\Delta_i = (y_i-a_i)g'(in_i)$

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：

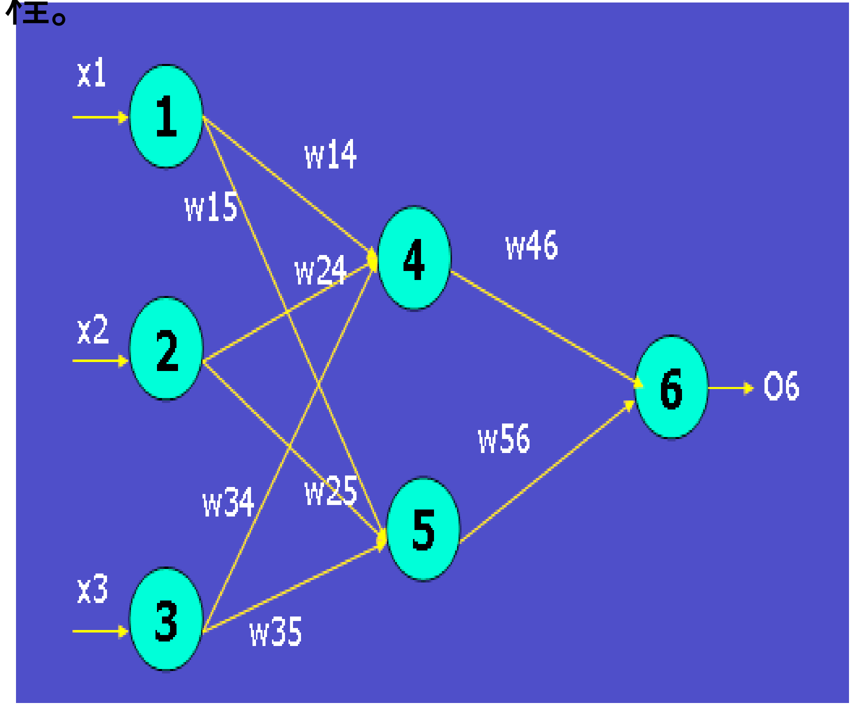
$$
\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\
&= -\sum_i \underline{(y_i - a_i)g'(in_i)}\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \underline{\Delta_i}\frac{\partial}{\partial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



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

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

2.2 计算权重和阈值的更新

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

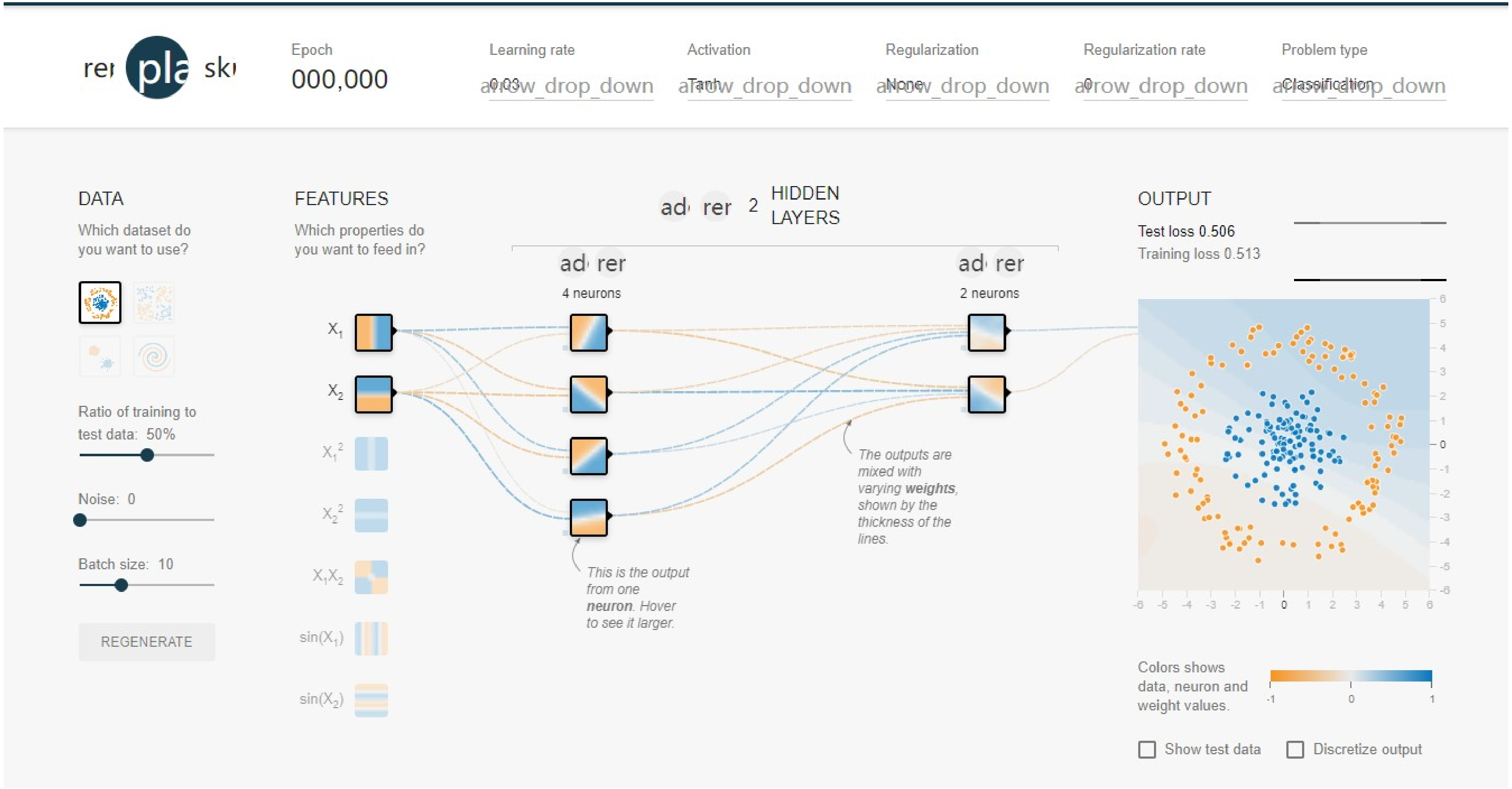
训练样本 **x={1,0,1}**

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

A demo from Google



http://playground.tensorflow.org/