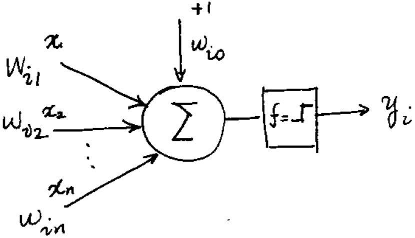
Artificial Neural Networks

Ravi Kothari, Ph.D.
ravi.kothari@ashoka.edu.in

"People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world - Pedro Domingos"

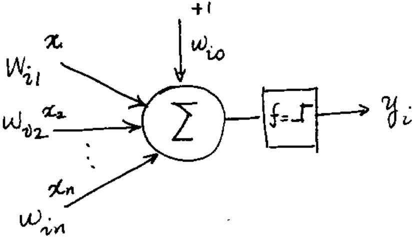
Basic Model of a Neuron

Basic Model of a Neuron



$$y_i = f(S_i) = f\left(\sum_{j=1}^{n} w_{ij} x_j + w_{i0}\right)$$

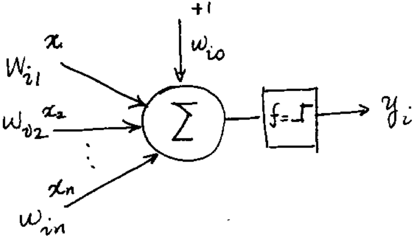
Basic Model of a Neuron



$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{n} w_{ij}x_j + w_{i0}\right)$$

- Abstraction of a biological neuron. Other abstractions are possible including spiking neurons etc.

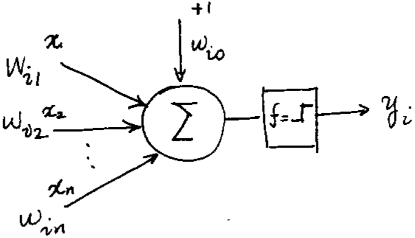
Basic Model of a Neuron



$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{n} w_{ij}x_j + w_{i0}\right)$$

- Abstraction of a biological neuron. Other abstractions are possible including spiking neurons etc.
- The output of a neuron can be the input to another neuron

Basic Model of a Neuron

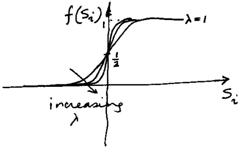


$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{n} w_{ij}x_j + w_{i0}\right)$$

- Abstraction of a biological neuron. Other abstractions are possible including spiking neurons etc.
- The output of a neuron can be the input to another neuron
- $f(\cdot)$ is the activation function, S_i is the weighted sum, w_{ij} is the weight from the j^{th} input and w_{i0} is the "bias"

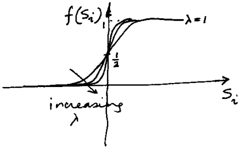
Binary Sigmoid Activation Function

Binary Sigmoid Activation Function



$$y_i = f\left(S_i\right) = \frac{1}{1 + e^{-\lambda S_i}}$$

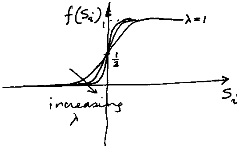
Binary Sigmoid Activation Function



$$y_i = f\left(S_i\right) = \frac{1}{1 + e^{-\lambda S_i}}$$

- Binary Sigmoid - Differentiable

Binary Sigmoid Activation Function

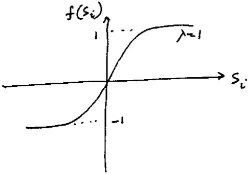


$$y_i = f\left(S_i\right) = \frac{1}{1 + e^{-\lambda S_i}}$$

- Binary Sigmoid - Differentiable
- With increasing λ, the binary sigmoid becomes more and more like the step function

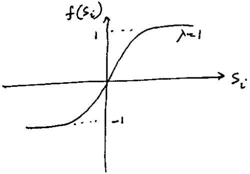
Bipolar Sigmoid Activation Function

Bipolar Sigmoid Activation Function



$$y_i = f\left(S_i\right) = \frac{2}{1 + e^{-\lambda S_i}} - 1$$

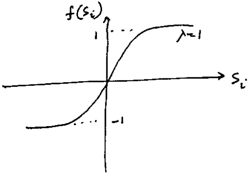
Bipolar Sigmoid Activation Function



$$y_i = f\left(S_i\right) = \frac{2}{1 + e^{-\lambda S_i}} - 1$$

- Bipolar Sigmoid - Differentiable

Bipolar Sigmoid Activation Function

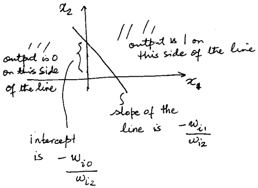


$$y_i = f\left(S_i\right) = \frac{2}{1 + e^{-\lambda S_i}} - 1$$

- Bipolar Sigmoid - Differentiable
- With increasing λ, the bipolar sigmoid becomes more and more steeper

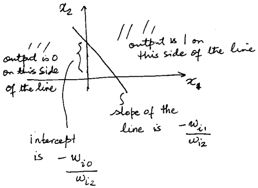
What does a Simple Neuron Do?

What does a Simple Neuron Do?



$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{2} w_{ij}x_j + w_{i0}\right) = f\left(w_{i1}x_1 + w_{i2}x_2 + w_{i0}\right)$$

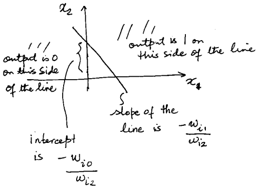
What does a Simple Neuron Do?



$$y_i = f(S_i) = f\left(\sum_{j=1}^{2} w_{ij}x_j + w_{i0}\right) = f\left(w_{i1}x_1 + w_{i2}x_2 + w_{i0}\right)$$

$$x_2 = -\frac{w_{i1}}{w_{i2}}x_1 - \frac{w_{i0}}{w_{i2}} \qquad ;\text{at } S_i = 0$$

What does a Simple Neuron Do?

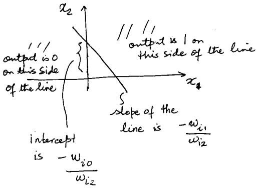


$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{2} w_{ij}x_j + w_{i0}\right) = f\left(w_{i1}x_1 + w_{i2}x_2 + w_{i0}\right)$$

$$x_2 = -\frac{w_{i1}}{w_{i2}}x_1 - \frac{w_{i0}}{w_{i2}} \qquad ;\text{at } S_i = 0$$

- On one side of the line, $S_i > 0$ and on the other side $S_i < 0$. So, this simple neuron establishes a decision boundary in the input space

What does a Simple Neuron Do?



$$y_i = f\left(S_i\right) = f\left(\sum_{j=1}^{2} w_{ij}x_j + w_{i0}\right) = f\left(w_{i1}x_1 + w_{i2}x_2 + w_{i0}\right)$$

$$x_2 = -\frac{w_{i1}}{w_{i2}}x_1 - \frac{w_{i0}}{w_{i2}} \qquad ;\text{at } S_i = 0$$

- On one side of the line, $S_i > 0$ and on the other side $S_i < 0$. So, this simple neuron establishes a decision boundary in the input space
- If the bias (w_{i0}) was not there, the line would always be constrained to pass through the origin. Hence the bias

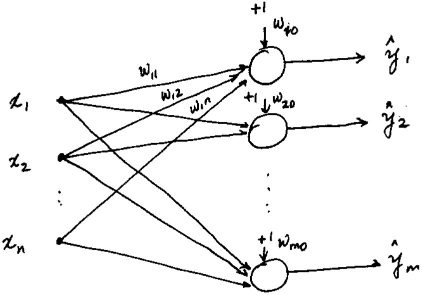
Learning (Adjusting the Weights)

Learning (Adjusting the Weights)

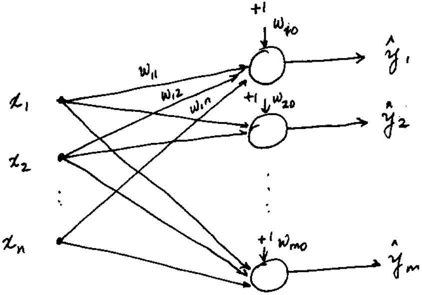
- Based on the training data, we would like to come up with an algorithm that would “evolve” the weights such that the network starts producing the desired output

The Delta Rule

The Delta Rule



The Delta Rule



$$\left\{ \left(x^{(l)}, y^{(l)} \right) \right\}_{l=1}^{N} \qquad ; x^{(l)} \in \mathcal{R}^n, \quad y^{(l)} \in \mathcal{R}^m$$

The Cost Function

The Cost Function

$$J^{(l)} = \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2$$

The Cost Function

$$J^{(l)} = \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2$$

$$\begin{aligned} J &= \sum_{l=1}^{N} J^{(l)} \\ &= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 \end{aligned} \tag{1}$$

A Simplification

A Simplification

From now on, we will include the bias term implicitly, i.e.,

A Simplification

From now on, we will include the bias term implicitly, i.e.,

$$\mathrm{x}^{(l)} = \begin{bmatrix} 1 \\ x_1^{(l)} \\ x_2^{(l)} \\ \vdots \\ x_n^{(l)} \end{bmatrix} \qquad \mathrm{w}_i = \begin{bmatrix} w_{i0} \\ w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix}$$

A Simplification

From now on, we will include the bias term implicitly, i.e.,

$$\mathrm{x}^{(l)} = \begin{bmatrix} 1 \\ x_1^{(l)} \\ x_2^{(l)} \\ \vdots \\ x_n^{(l)} \end{bmatrix} \qquad \mathrm{w}_i = \begin{bmatrix} w_{i0} \\ w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix}$$

$$\hat{y}_i^{(l)} = f\left(\sum_{j=0}^{n} w_{ij} x_j^{(l)}\right)$$

The Cost Function

The Cost Function

$$
\begin{aligned}
J &= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 \\
&= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - f \left(\sum_{j=0}^{n} w_{ij} x_j \right) \right)^2
\end{aligned}
$$

The Cost Function

$$J = \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2$$

$$= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - f \left(\sum_{j=0}^{n} w_{ij} x_j \right) \right)^2$$

So, we can start with random values of w's and adopt the weights by moving opposite to the gradient computed from the equation above, i.e.

$$\Delta w = -\eta \nabla J$$

The Cost Function

The Cost Function

In batch learning, we aggregate the gradients computed for each pattern, i.e.,

$$\nabla J = \nabla J^{(1)} + \nabla J^{(2)} + \ldots + \nabla J^{(N)}$$

The Cost Function

In batch learning, we aggregate the gradients computed for each pattern, i.e.,

$$\nabla J = \nabla J^{(1)} + \nabla J^{(2)} + \ldots + \nabla J^{(N)}$$

In online learning, we make the approximation that as long as η is small, we can,

- Feed pattern l
- Compute $\nabla J^{(l)}$
- Update w's and repeat

The Delta Rule Derived

The Delta Rule Derived

$$
\begin{aligned}
J &= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - f\left(\sum_{j=0}^{n} w_{ij} x_j \right) \right)^2 \\
&= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 = \sum_{l=1}^{N} \sum_{i=1}^{m} e_i^{(l)^2}
\end{aligned}
$$

The Delta Rule Derived

$$J = \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - f\left(\sum_{j=0}^{n} w_{ij} x_j \right) \right)^2$$
$$= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 = \sum_{l=1}^{N} \sum_{i=1}^{m} e_i^{(l)2}$$

$$\frac{\partial J^{(l)}}{\partial w_{ij}} = \frac{\partial J^{(l)}}{\partial e_i^{(l)}} \frac{\partial e_i^{(l)}}{\partial \hat{y}_i^{(l)}} \frac{\partial \hat{y}_i^{(l)}}{\partial S_i^{(l)}} \frac{\partial S_i^{(l)}}{\partial w_{ij}}$$
$$= 2e_i^{(l)} \cdot (-1) \cdot f'(S_i^{(l)}) \cdot x_j^{(l)}$$
$$= -2 \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f'(S_i^{(l)}) \cdot x_j^{(l)}$$

What is $f'(S_i)$

What is $f'(S_i)$

Recall that,

$$\hat{y}_i^{(l)} = f\left(S_i^{(l)}\right) \tag{2}$$

What is $f'(S_i)$

Recall that,

$$\hat{y}_i^{(l)} = f\left(S_i^{(l)}\right) \tag{2}$$

If the activation function is linear, $\hat{y}_i^{(l)} = S_i^{(l)}$

$$f'\left((S_i^{(l)}\right) = \frac{\hat{y}_i^{(l)}}{S_i^{(l)}} = 1 \tag{3}$$

If the activation function is a binary sigmoid, $\hat{y}_i^{(l)} = 1/(1 + e^{-S_i^{(l)}})$

$$f'\left(S_i^{(l)}\right) = \hat{y}_i^{(l)}\left(1 - \hat{y}_i^{(l)}\right) \tag{4}$$

If the activation function is a bipolar sigmoid, $\hat{y}_i^{(l)} = \left(2/(1 + e^{-S_i^{(l)}})\right) - 1$

$$f'\left(S_i^{(l)}\right) = \frac{1}{2}\left(1 - \hat{y}^{(l)2}_i\right) \tag{5}$$

The Overall Algorithm (Online Learning)

Initialize all the weights to small random values
while J is large
    for $l = 1, 2, \ldots, N$
        Do a forward pass i.e. compute $\hat{y}_1^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_m^{(l)}$
        for $i = 1, 2, \ldots, n$
            for $j = 1, 2, \ldots, m$

$$\frac{\partial J^{(l)}}{\partial w_{ij}} = -\left(y_i^{(l)} - \hat{y}_i^{(l)}\right) \cdot f'\left(S_i^{(l)}\right) \cdot x_j^{(l)}$$

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) - \eta \frac{\partial J^{(l)}}{\partial w_{ij}}$$

            end for
        end for
    end for
end while

The Overall Algorithm (Batch Learning)

Initialize all the weights to small random values

while J is large

$\Delta w_{ij} = 0; \quad i = 1, 2, \ldots n; \; j = 1, 2, \ldots, m$

for $l = 1, 2, \ldots, N$

Do a forward pass i.e. compute $\hat{y}_1^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_m^{(l)}$

for $i = 1, 2, \ldots, n$

for $j = 1, 2, \ldots, m$

$$\frac{\partial J^{(l)}}{\partial w_{ij}} = -\left(y_i^{(l)} - \hat{y}_i^{(l)}\right) \cdot f'\left(S_i^{(l)}\right) \cdot x_j^{(l)}$$

$$\Delta w_{ij}(\text{new}) = \Delta w_{ij}(\text{old}) - \eta \frac{\partial J^{(l)}}{\partial w_{ij}}$$

end for

end for

end for

$w_{ij}(\text{new}) = w_{ij}(\text{old}) | \Delta w_{ij}$

end while

Additional Comments

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule
- Obviously, the activation function needs to be differentiable for gradient descent based optimizations. The sigmoid is a nice choice for this reason (more reasons later)

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule
- Obviously, the activation function needs to be differentiable for gradient descent based optimizations. The sigmoid is a nice choice for this reason (more reasons later)
- The loss function is quadratic

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule
- Obviously, the activation function needs to be differentiable for gradient descent based optimizations. The sigmoid is a nice choice for this reason (more reasons later)
- The loss function is quadratic
- The activation function needs to be differentiable

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule
- Obviously, the activation function needs to be differentiable for gradient descent based optimizations. The sigmoid is a nice choice for this reason (more reasons later)
- The loss function is quadratic
- The activation function needs to be differentiable
- For classification, we can interpret an output of greater than 0 as Class 1 and an output of less than 0 as Class 2 if we are using a bipolar sigmoid

Additional Comments

- The Delta Rule is also called the LMS Rule or the Perceptron Learning Rule
- Obviously, the activation function needs to be differentiable for gradient descent based optimizations. The sigmoid is a nice choice for this reason (more reasons later)
- The loss function is quadratic
- The activation function needs to be differentiable
- For classification, we can interpret an output of greater than 0 as Class 1 and an output of less than 0 as Class 2 if we are using a bipolar sigmoid
- Note that a neuron establishes a linear decision boundary. Thus, such networks can only perform linearly separable pattern classification

An Illustrative Example – Online Learning

Assume we are given the following training data,

	Input	Desired Output	
$x^{(1)} \rightarrow$	1	1	$\leftarrow y^{(1)}$
$x^{(2)} \rightarrow$	-0.5	-1	$\leftarrow y^{(2)}$
$x^{(3)} \rightarrow$	3	1	$\leftarrow y^{(3)}$
$x^{(4)} \rightarrow$	-1	-1	$\leftarrow y^{(4)}$

An Illustrative Example – Online Learning

Assume we are given the following training data,

	Input	Desired Output	
$x^{(1)} \rightarrow$	1	1	$\leftarrow y^{(1)}$
$x^{(2)} \rightarrow$	-0.5	-1	$\leftarrow y^{(2)}$
$x^{(3)} \rightarrow$	3	1	$\leftarrow y^{(3)}$
$x^{(4)} \rightarrow$	-1	-1	$\leftarrow y^{(4)}$

An Illustrative Example – Online Learning

Assume we are given the following training data,

	Input	Desired Output	
$x^{(1)} \rightarrow$	1	1	$\leftarrow y^{(1)}$
$x^{(2)} \rightarrow$	-0.5	-1	$\leftarrow y^{(2)}$
$x^{(3)} \rightarrow$	3	1	$\leftarrow y^{(3)}$
$x^{(4)} \rightarrow$	-1	-1	$\leftarrow y^{(4)}$

So, we have $n = 1$ input and $m = 1$ output. Let us initialize the parameters as: $w_{10} = -0.5$, $w_{11} = 0.5$, take bipolar activations (outputs are $+1$ and -1), and $\eta = 0.1$

An Illustrative Example – Online Learning (contd.)

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1 + e^{-0})) - 1 = 0$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1 + e^{-0})) - 1 = 0$
  - Adjust the weights

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1+e^{-0})) - 1 = 0$
  - Adjust the weights
    - ★ $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1+e^{-0})) - 1 = 0$
  - Adjust the weights
    - ★ $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$
    - ★ $w_{10}(new) = -0.5 + 0.1\,(1-0)) \cdot \frac{1}{2}\,(1-0) \cdot 1 = -0.45$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1 + e^{-0})) - 1 = 0$
  - Adjust the weights
    - $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$
    - $w_{10}(new) = -0.5 + 0.1\,(1 - 0)) \cdot \frac{1}{2}\,(1 - 0) \cdot 1 = -0.45$
    - $w_{11}(new) = 0.5 + 0.1\,(1 - 0)) \cdot \frac{1}{2}\,(1 - 0) \cdot 1 = 0.55$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1+e^{-0})) - 1 = 0$
  - Adjust the weights
    - $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$
    - $w_{10}(new) = -0.5 + 0.1\,(1-0)) \cdot \frac{1}{2}\,(1-0) \cdot 1 = -0.45$
    - $w_{11}(new) = 0.5 + 0.1\,(1-0)) \cdot \frac{1}{2}\,(1-0) \cdot 1 = 0.55$
- Present $x^{(2)}$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0.$ $\hat{y}^{(1)} = (2/(1+e^{-0})) - 1 = 0$
  - Adjust the weights
    - $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$
    - $w_{10}(new) = -0.5 + 0.1\,(1-0)) \cdot \frac{1}{2}\,(1-0) \cdot 1 = -0.45$
    - $w_{11}(new) = 0.5 + 0.1\,(1-0)) \cdot \frac{1}{2}\,(1-0) \cdot 1 = 0.55$
- Present $x^{(2)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times (-0.5) =$ -3.2. $\hat{y}^{(1)} = (2/(1+e^{-3.2})) - 1 = \ldots.$

An Illustrative Example – Online Learning (contd.)

- Present $x^{(1)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times 1 = 0$. $\hat{y}^{(1)} = (2/(1 + e^{-0})) - 1 = 0$
  - Adjust the weights
    - ★ $w_{ij}(new) = w_{ij}(old) + \eta \left(y_i^{(l)} - \hat{y}_i^{(l)} \right) \cdot f' \left(S_i^{(l)} \right) \cdot x_j^{(l)}$
    - ★ $w_{10}(new) = -0.5 + 0.1\,(1 - 0)) \cdot \frac{1}{2}\,(1 - 0) \cdot 1 = -0.45$
    - ★ $w_{11}(new) = 0.5 + 0.1\,(1 - 0)) \cdot \frac{1}{2}\,(1 - 0) \cdot 1 = 0.55$
- Present $x^{(2)}$
  - $S_1 = w_{10} \times 1 + w_{11} \times (-0.5) =$ -3.2. $\hat{y}^{(1)} = (2/(1 + e^{-3.2})) - 1 = ...$
  - ...