CS 437 / CS 5317
Deep Learning

Murtaza Taj
murtaza.taj@lums.edu.pk

Lecture 18: Deep Learning
Wed 27th Mar 2019

Analytical vs. Iterative Solution

$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum_j w_j x_j$$
$$E = \sum_i (t^{(i)} - \mathbf{w}^T\mathbf{x})^2$$

- Analytical Solution

$$\frac{\partial E}{\partial w_j} = -\sum_i (t^{(i)} - y^{(i)}) x_j^{(i)}$$

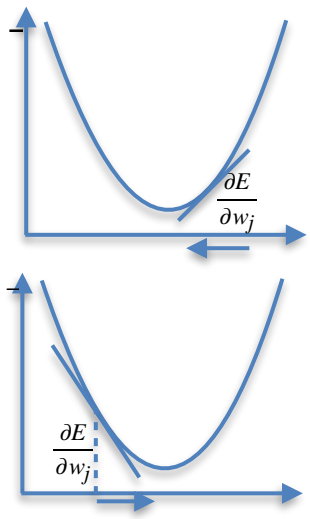
Aw = b

- Iterative Solution

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

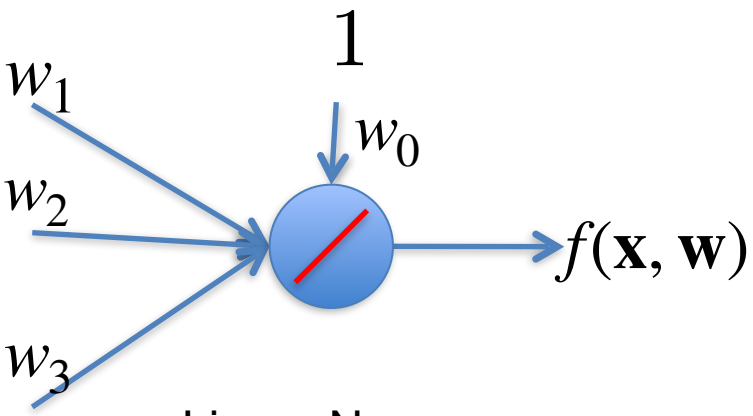
$$w_j \leftarrow w_j + \Delta w_j$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

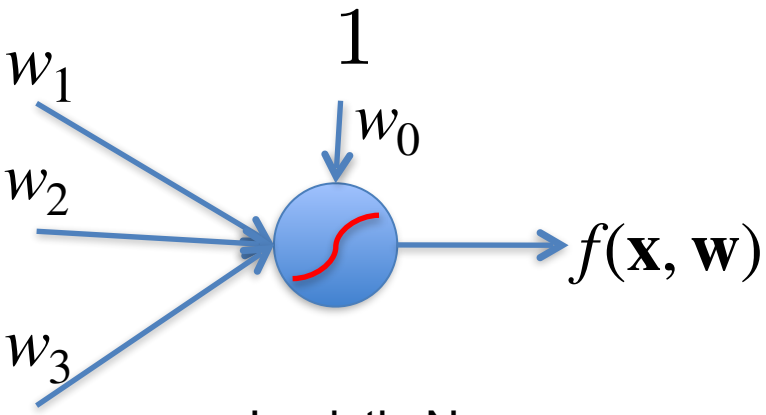


where η is the learning rate

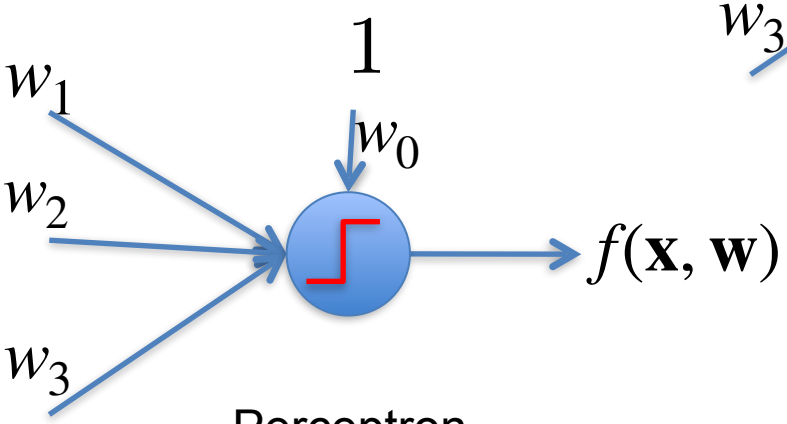
Types of Neurons



Linear Neuron



Logistic Neuron

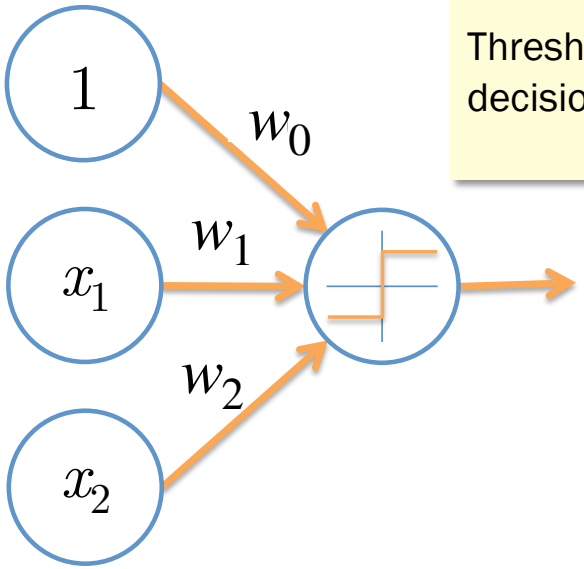


Perceptron

Potentially more. Require a convex loss function for gradient descent training.

Slide Credit: HKUST

Perceptron

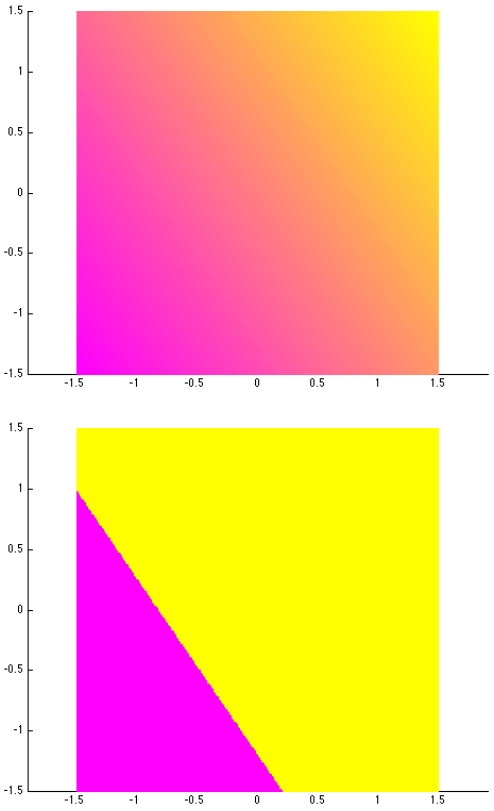


Implements a linear function $f(x_1, x_2)$

Threshold to get linear decision boundary

Example:

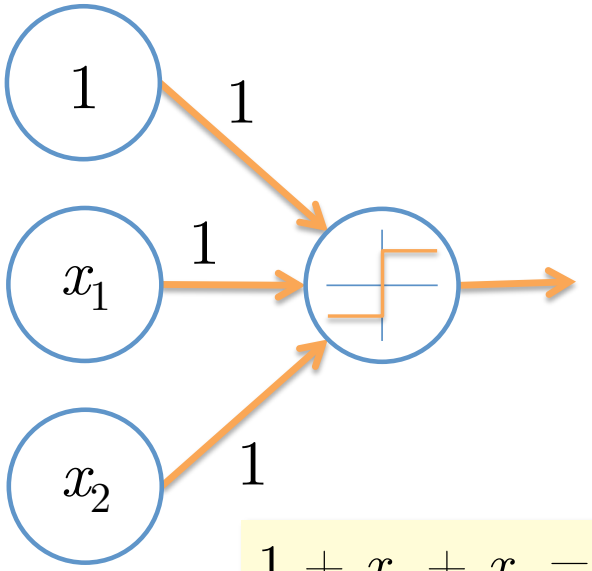
$\mathbf{w} = [0.79, 0.96, 0.66]$



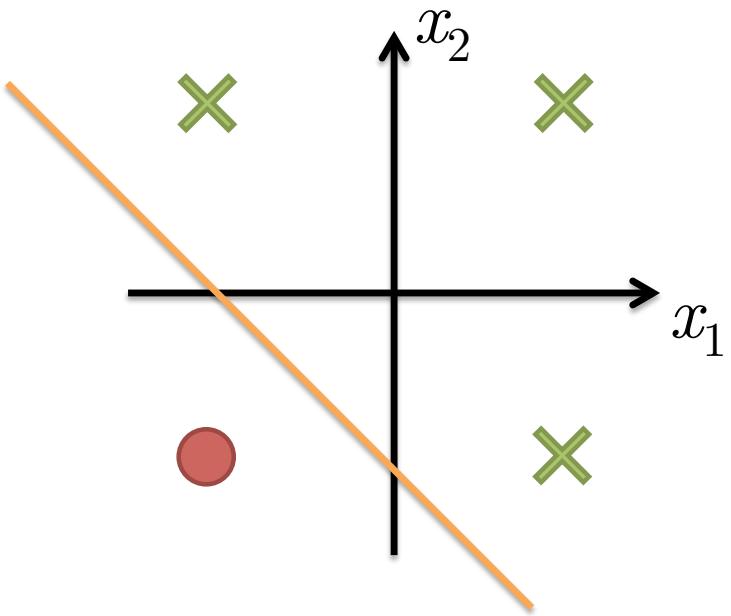
$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 > 0 \\ 0 & \text{otherwise} \end{cases}$$

OR Function

x_1	x_2	OR(x_1, x_2)
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

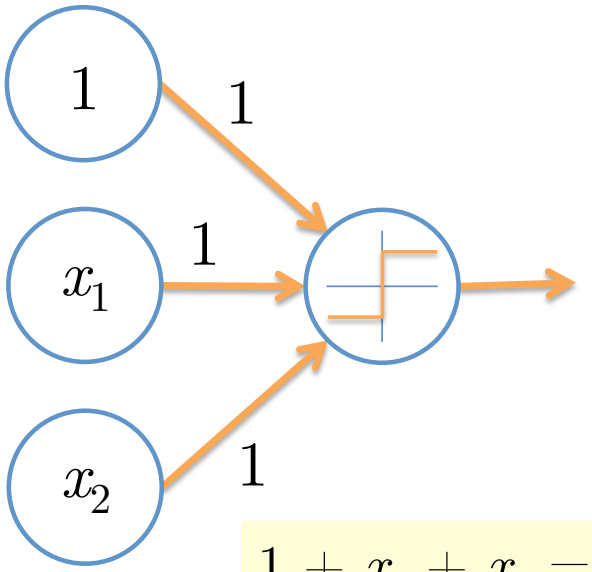


$$1 + x_1 + x_2 = 0$$

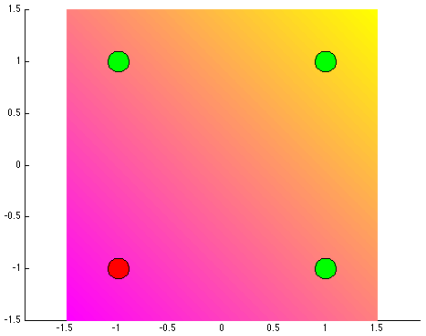


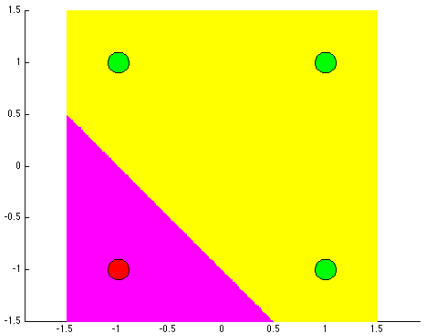
OR Function

x_1	x_2	OR(x_1, x_2)
-1	-1	-1
-1	1	1
1	-1	1
1	1	1



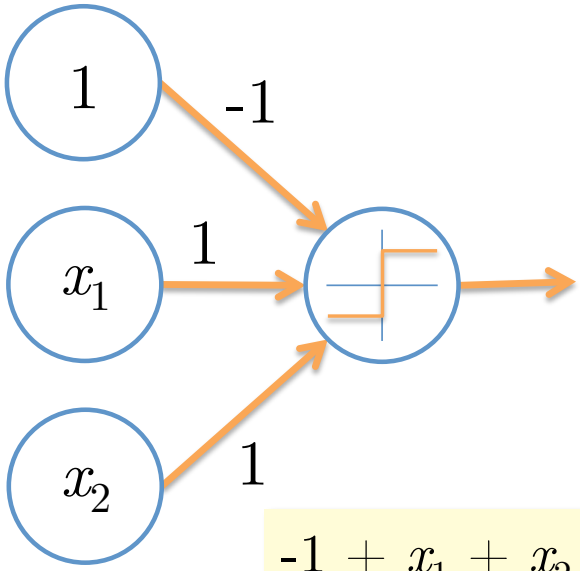
$$1 + x_1 + x_2 = 0$$



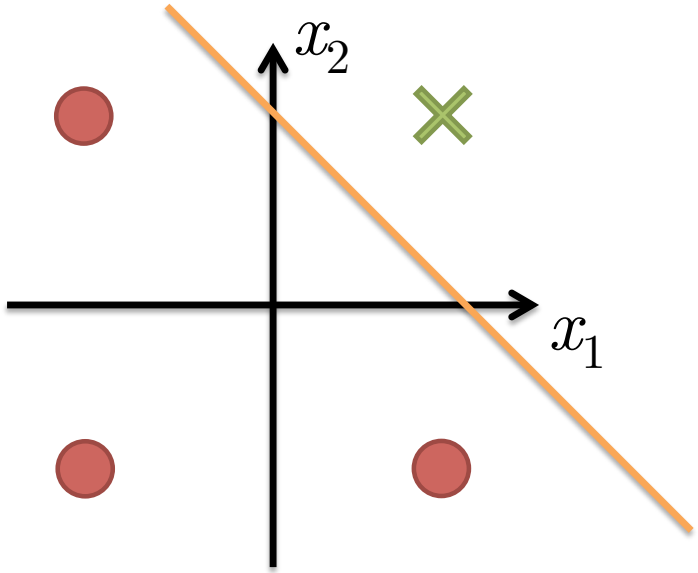


AND Function

x_1	x_2	AND(x_1, x_2)
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

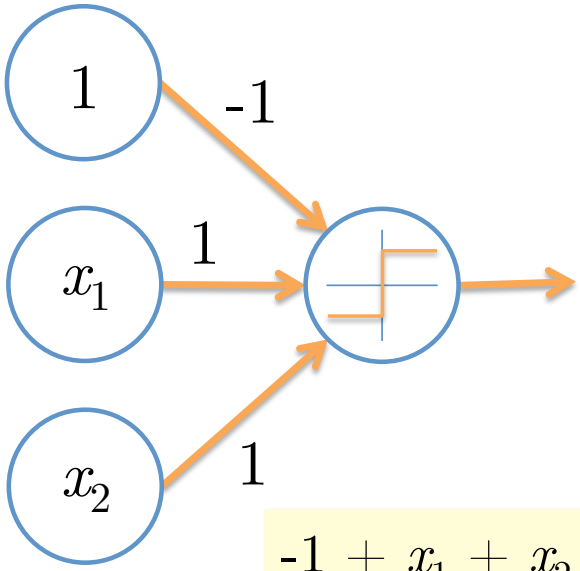


$$-1 + x_1 + x_2 = 0$$

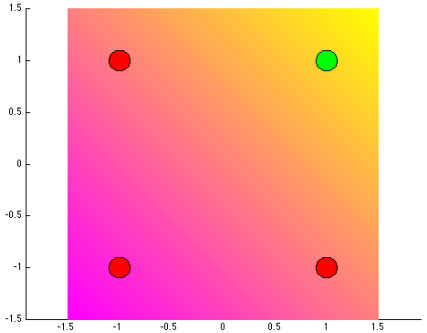


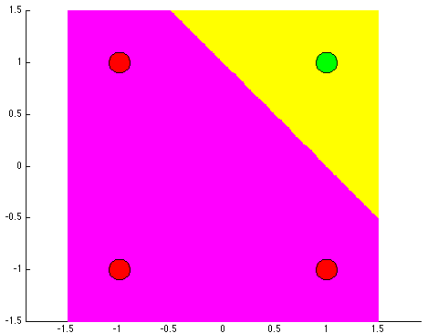
AND Function

x_1	x_2	AND(x_1, x_2)
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1



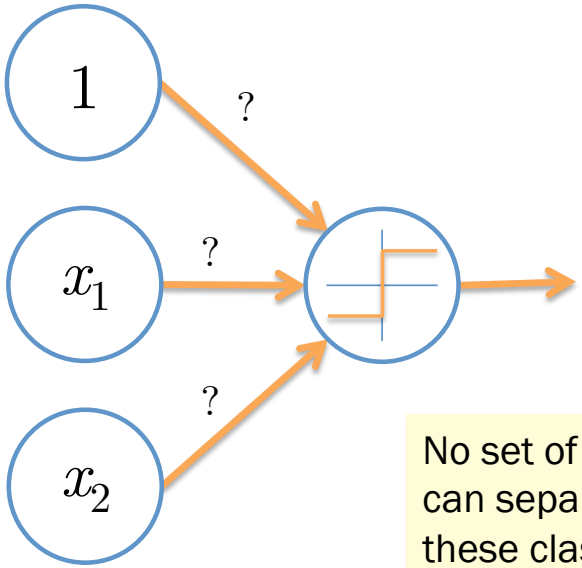
$$-1 + x_1 + x_2 = 0$$



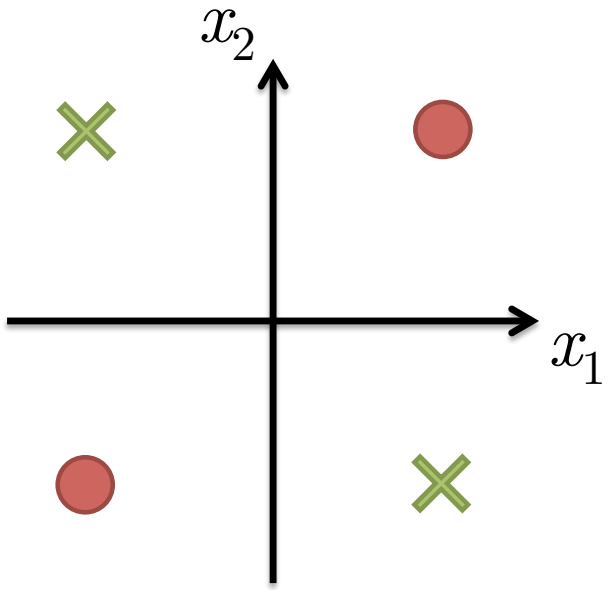


XOR Function

x_1	x_2	XOR(x_1, x_2)
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



No set of weights can separate these classes



XOR

$$\text{XOR}(x_1, x_2) = \text{OR}(\text{AND}(x_1, \neg x_2), \text{AND}(\neg x_1, x_2))$$

x_1	x_2	$\text{AND}(x_1, \neg x_2)$	$\text{AND}(\neg x_1, x_2)$	$\text{OR}(\text{AND}(x_1, \neg x_2), \text{AND}(\neg x_1, x_2))$
-1	-1			
-1	1			
1	-1			
1	1			

XOR

$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$$

x_1	x_2	$\mathrm{AND}(x_1, \neg x_2)$	$\mathrm{AND}(\neg x_1, x_2)$	$\mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$
-1	-1	-1		
-1	1	-1		
1	-1	1		
1	1	-1		

XOR

$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$$

x_1	x_2	$\mathrm{AND}(x_1, \neg x_2)$	$\mathrm{AND}(\neg x_1, x_2)$	$\mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$
-1	-1	-1	-1	
-1	1	-1	1	
1	-1	1	-1	
1	1	-1	-1	

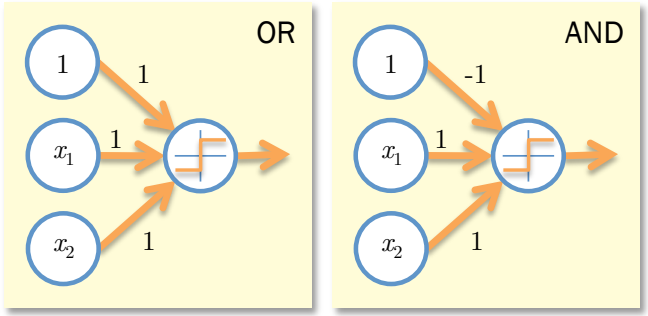
XOR

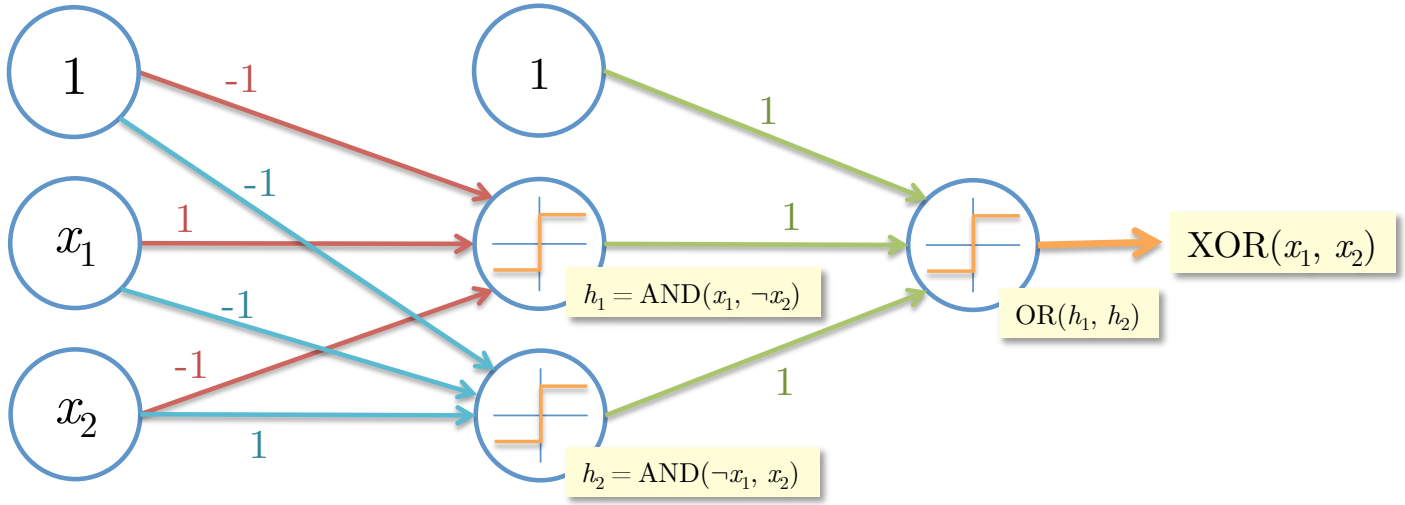
$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$$

x_1	x_2	$\mathrm{AND}(x_1, \neg x_2)$	$\mathrm{AND}(\neg x_1, x_2)$	$\mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$
-1	-1	-1	-1	-1
-1	1	-1	1	1
1	-1	1	-1	1
1	1	-1	-1	-1

XOR

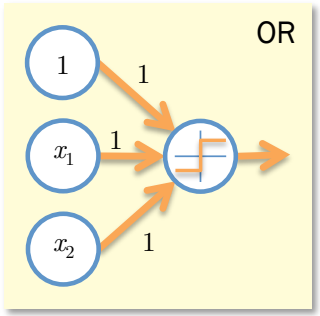
$$\text{XOR}(x_1, x_2) = \text{OR}(\text{AND}(x_1, \neg x_2), \text{AND}(\neg x_1, x_2))$$

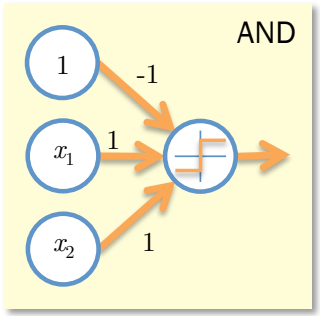


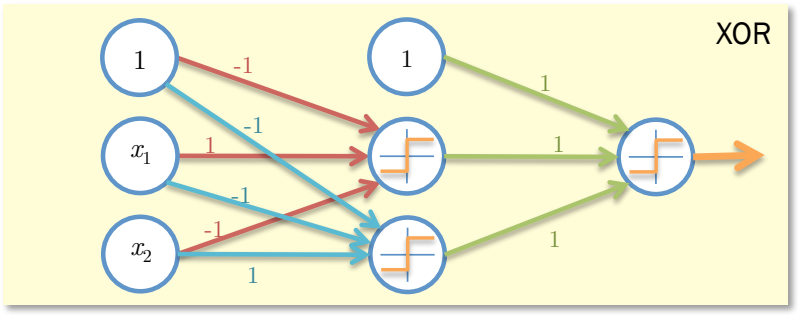


XOR

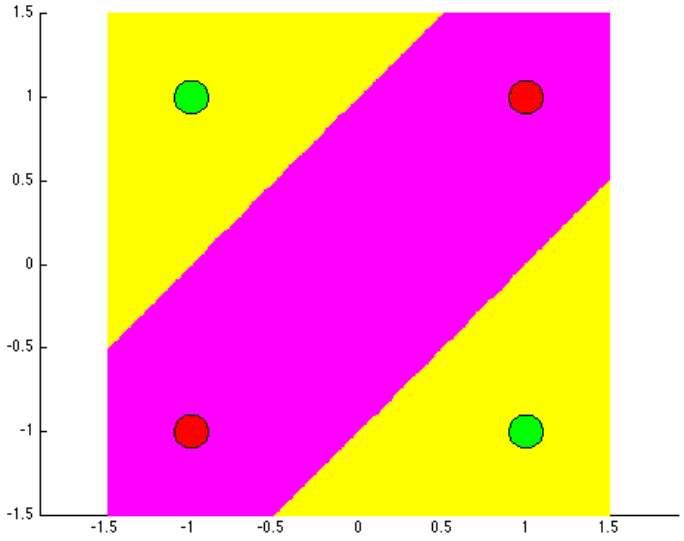
$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$$



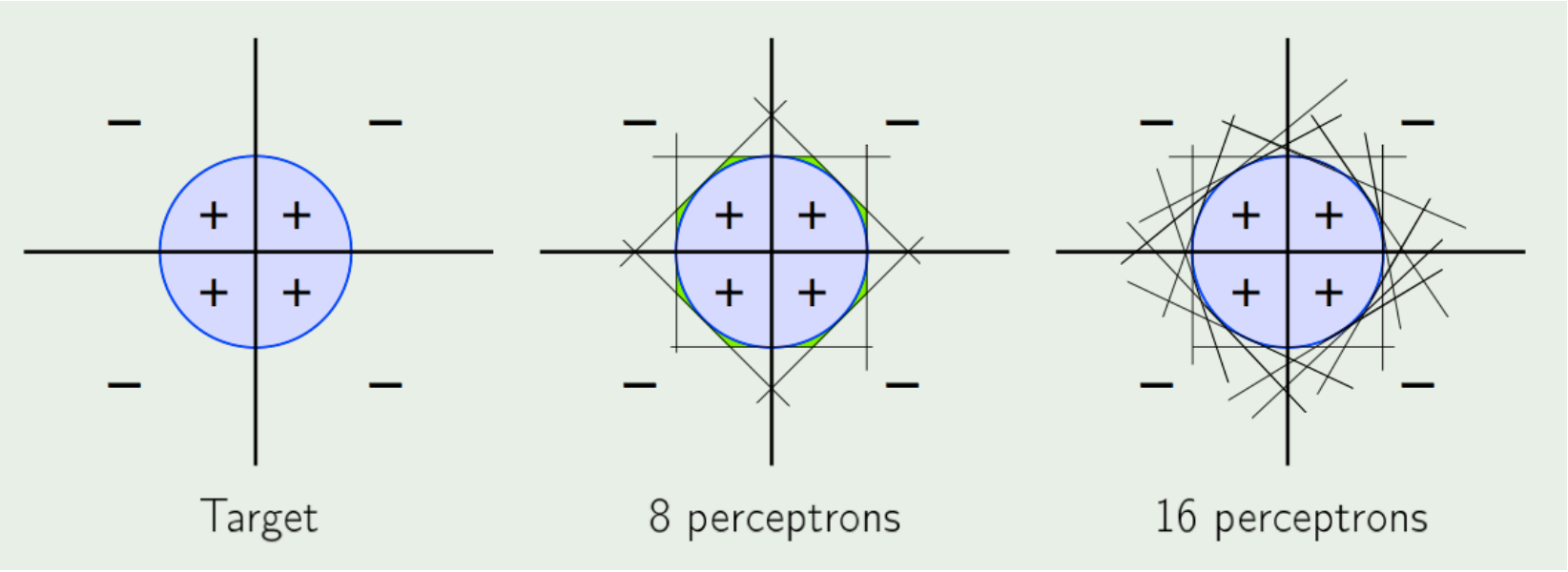




By combining two Perceptrons, we are able to create a non-linear decision boundary



A Powerful Model



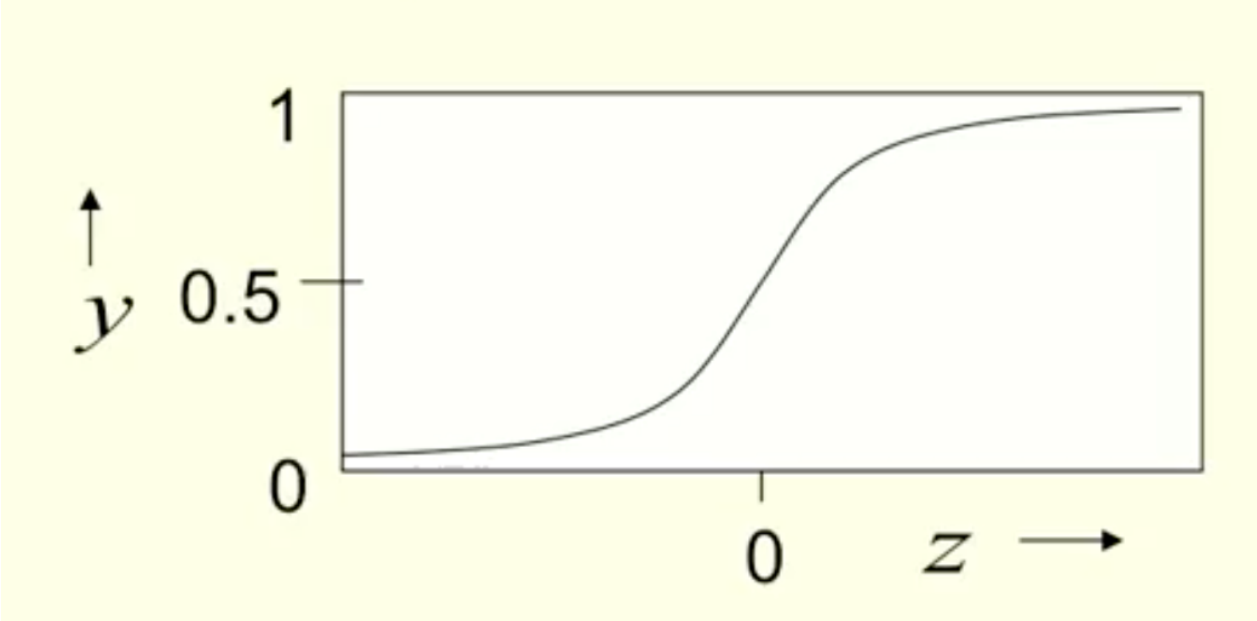
Logistic Neuron

Logistic Neuron

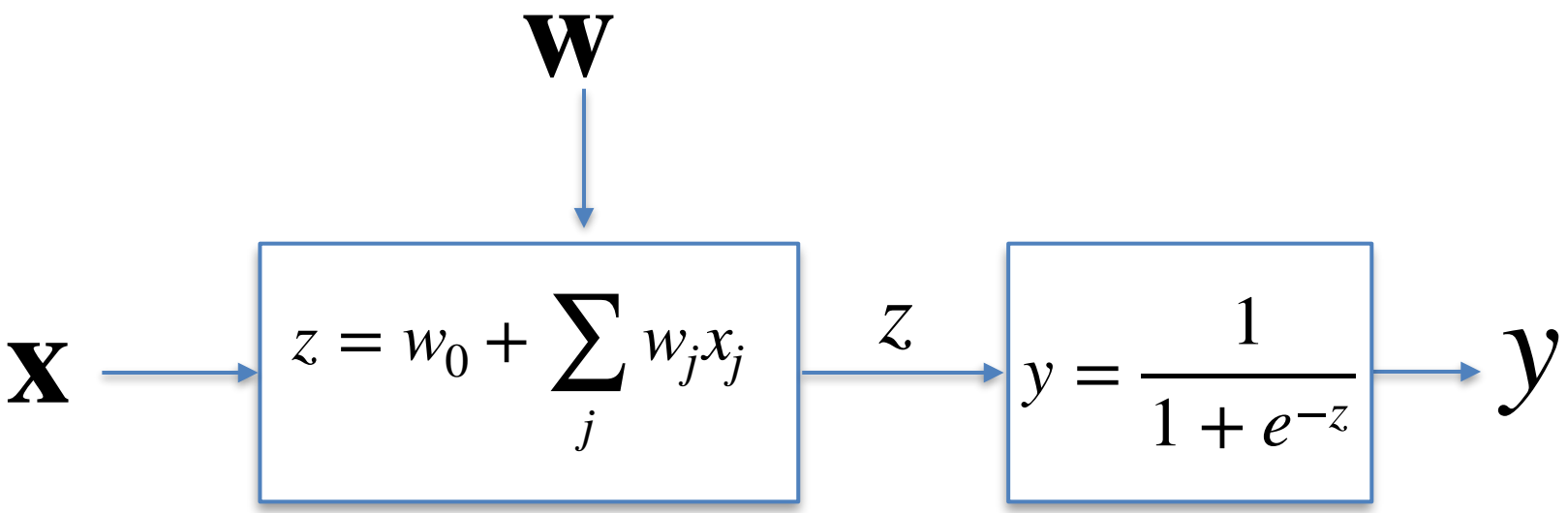
- These give a real-valued output that is a smooth and bounded function of their total input
  - They have nice derivates which make learning easy

$$z = w_0 + \sum_j x_j w_j$$

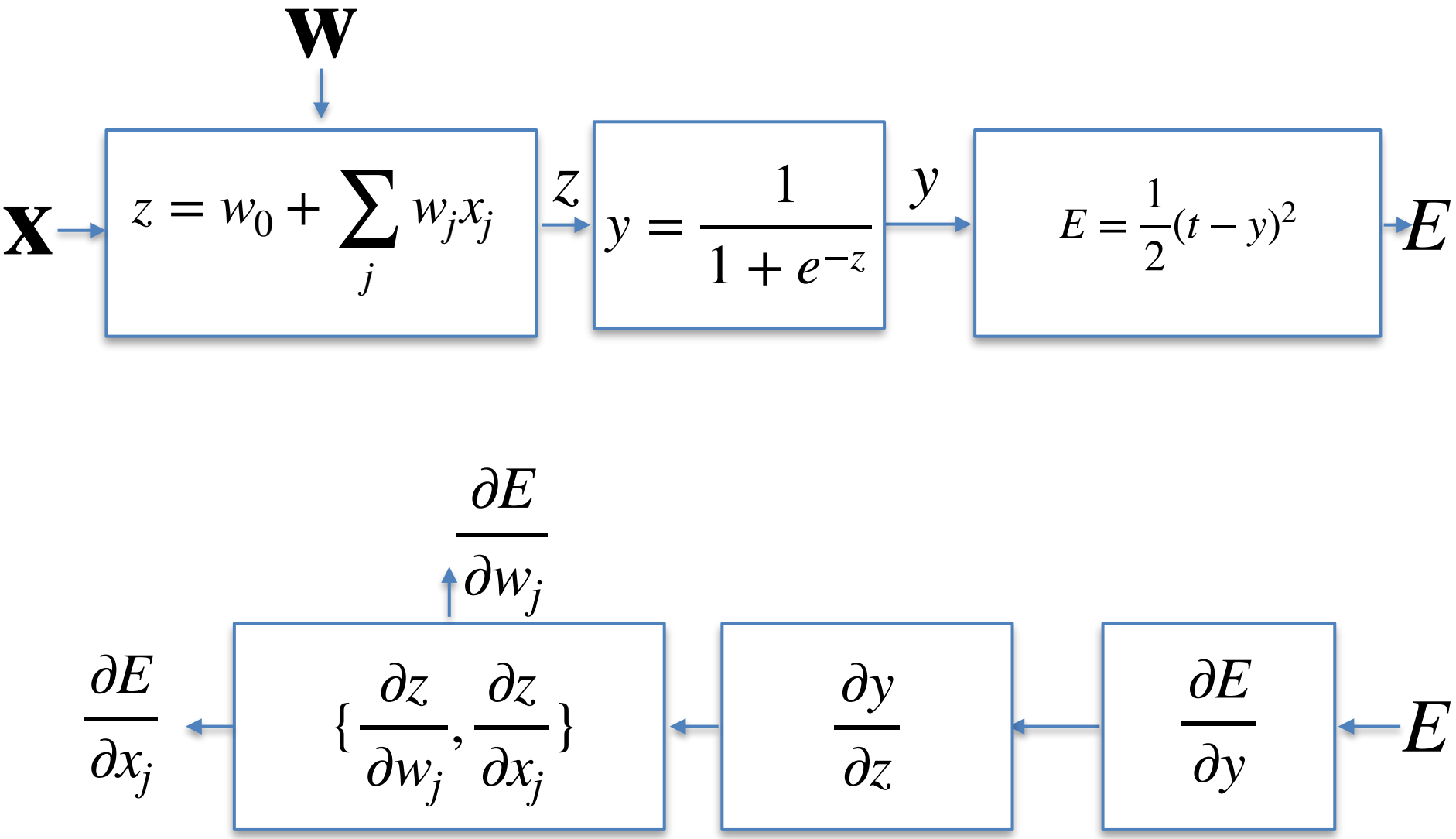
$$y = \frac{1}{1 + e^{-z}}$$



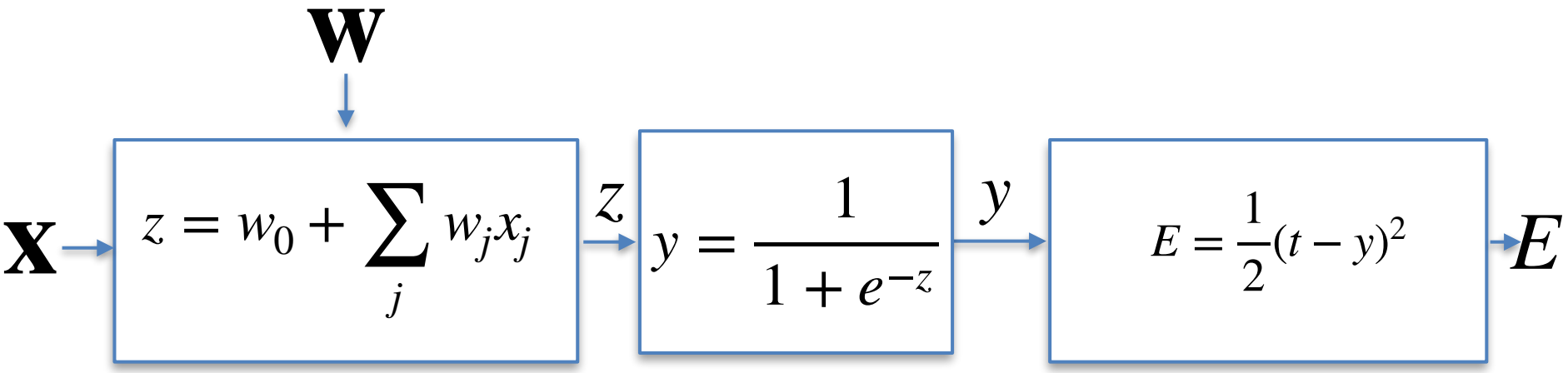
Key Computation: Forward-Prop



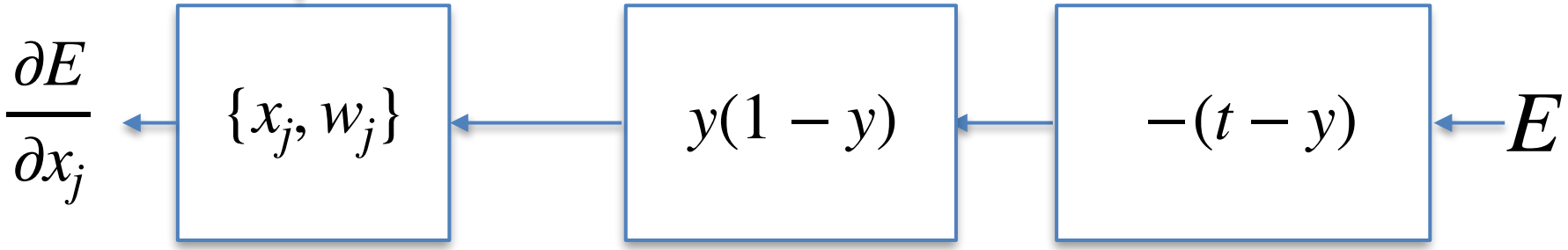
Key Computation: Back-Prop



Key Computation: Back-Prop



$$\frac{\partial E}{\partial w_j} = \frac{\partial z}{\partial w_j}\frac{\partial y}{\partial z}\frac{\partial E}{\partial y} = -x_j y(1-y)(t-y)$$



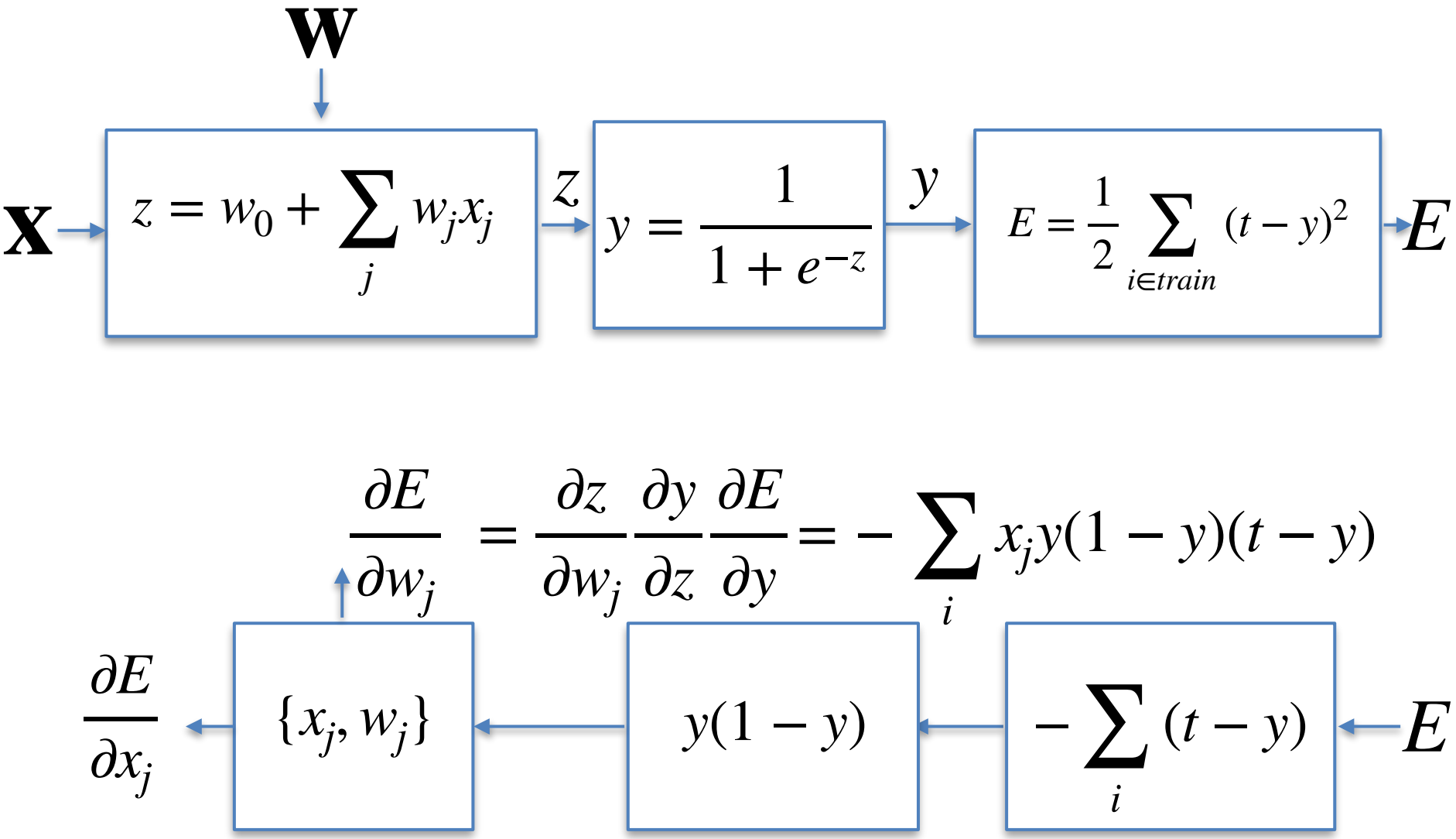
Derivative of a Logistic Neuron

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

$$\frac{dy}{dz} = \frac{-1(-e^{-z})}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

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

Key Computation: Back-Prop



Key Computation: Back-Prop

- Iterative Solution

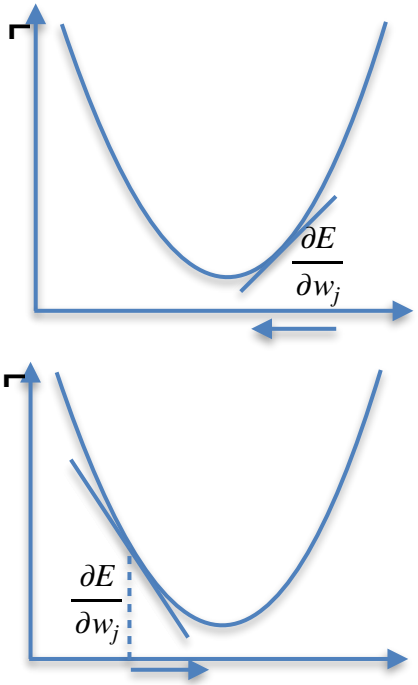
$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j \leftarrow w_j + \Delta w_j$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

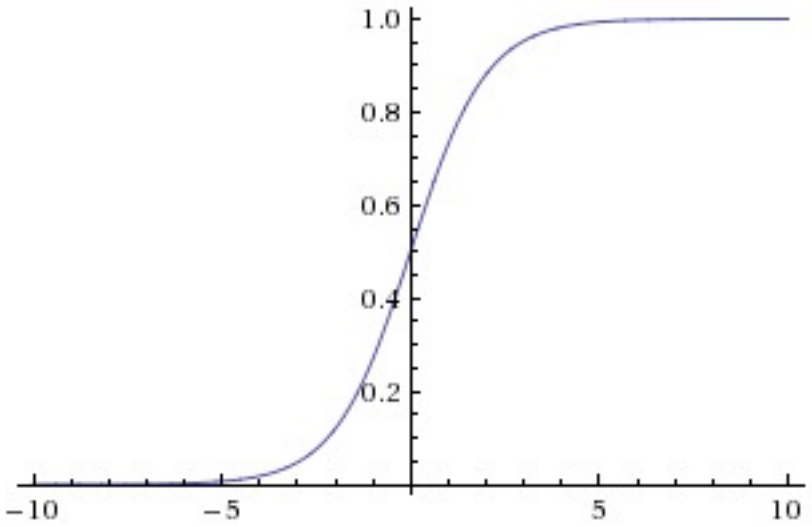
$$\Delta w_j = \eta \sum_i x_j^{(i)} y^{(i)} (1 - y^{(i)}) (t^{(i)} - y^{(i)})$$

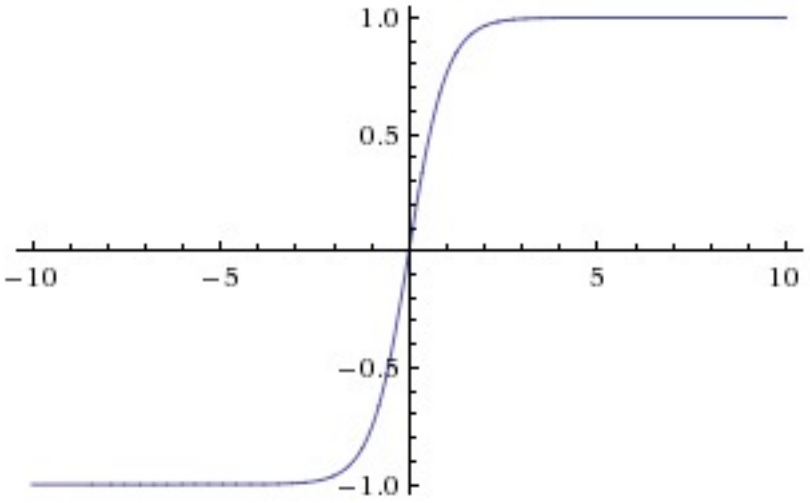
where η is the learning rate



Activation Functions

- sigmoid vs tanh





A quick note

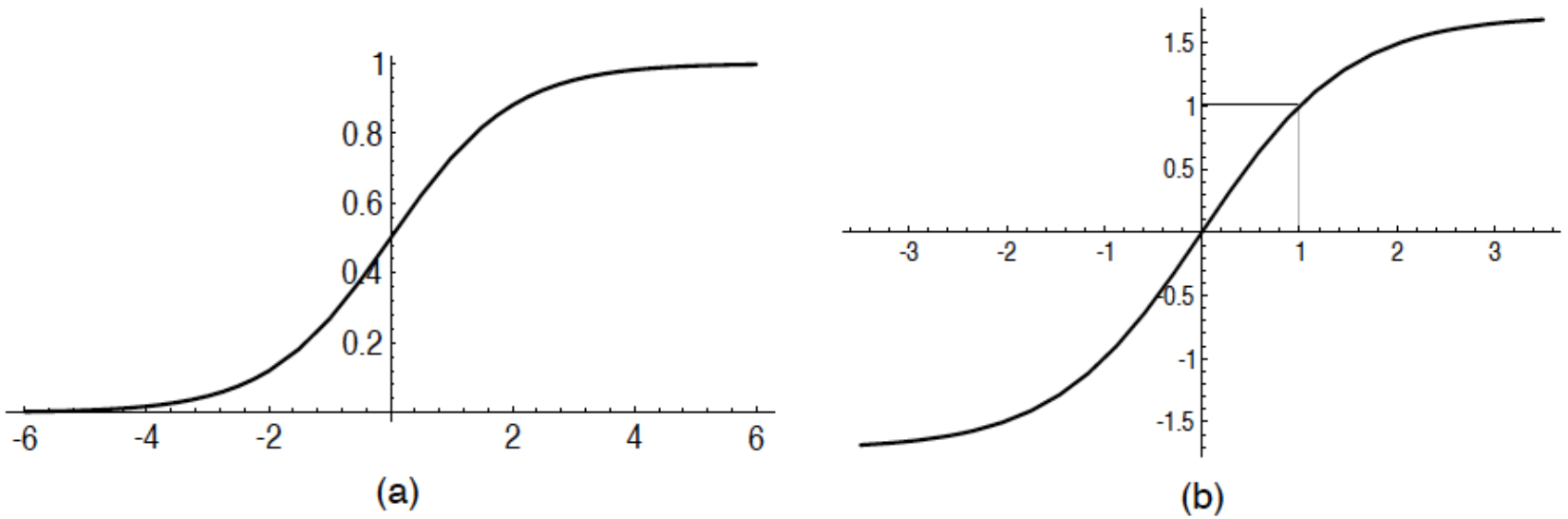
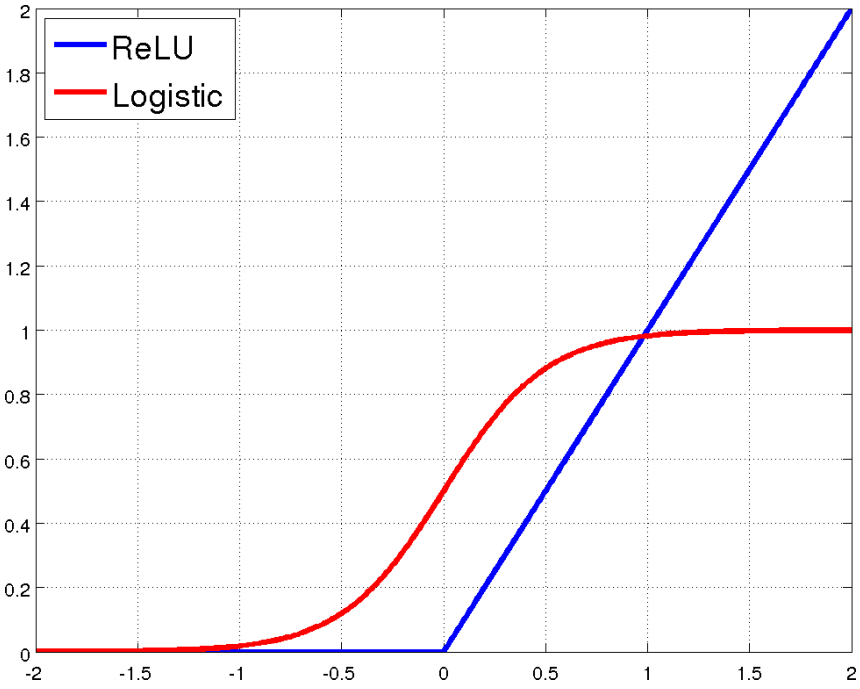
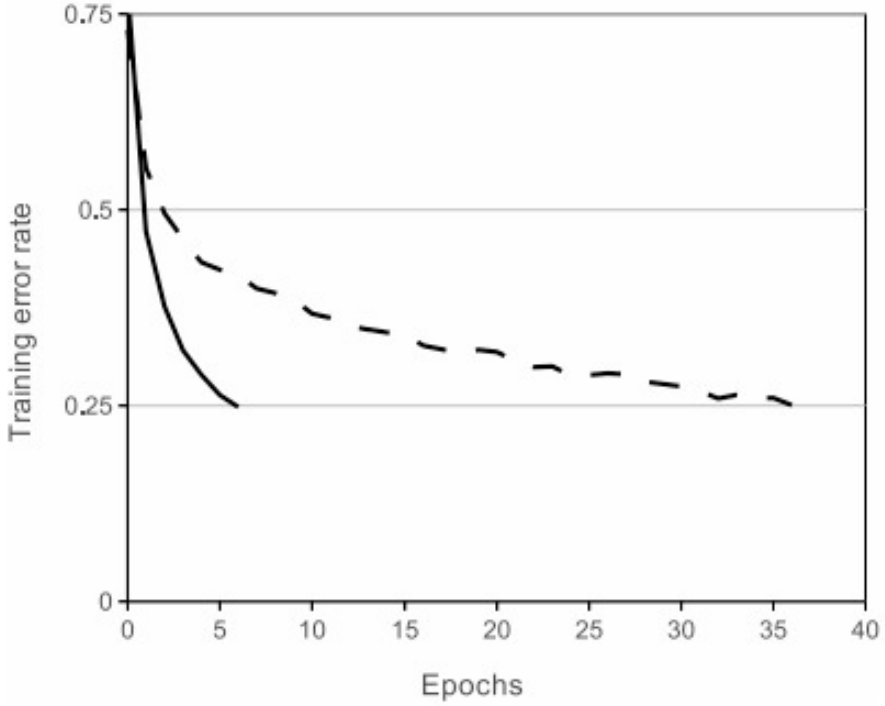


Fig. 4. (a) Not recommended: the standard logistic function, $f(x) = 1/(1 + e^{-x})$. (b) Hyperbolic tangent, $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$.

Image Credit: LeCun et al. '98

Rectified Linear Units (ReLU)



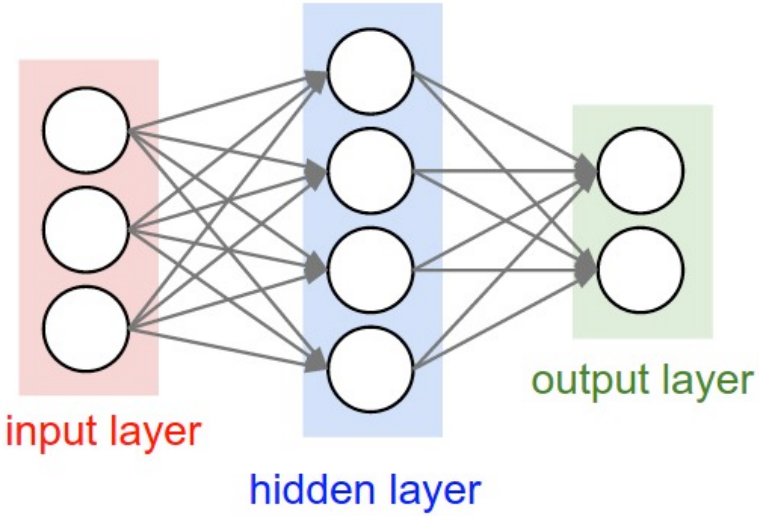


[Krizhevsky et al., NIPS12]

- ReLU $\max(0,z)$
- Leaky ReLU $\max(0.1z, z)$

Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights



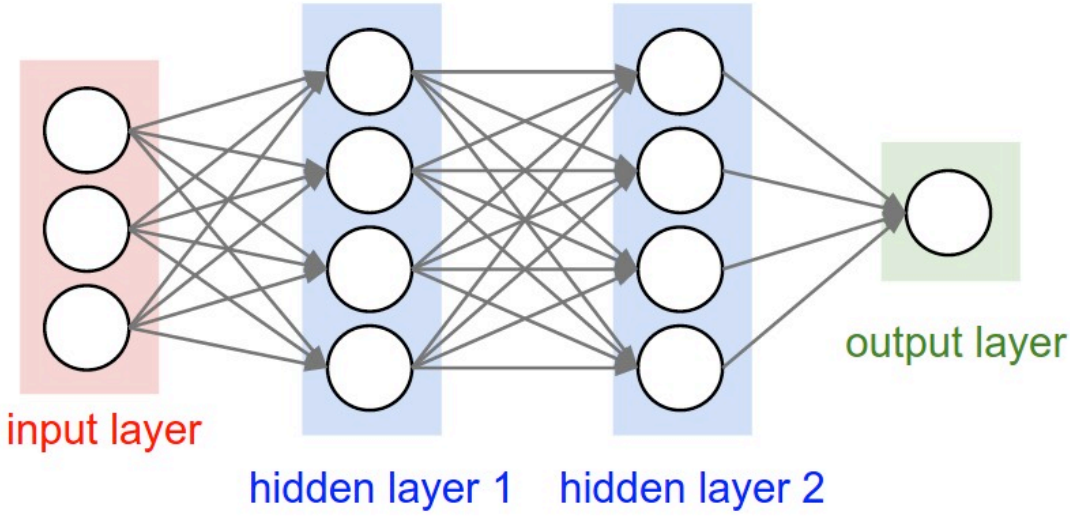
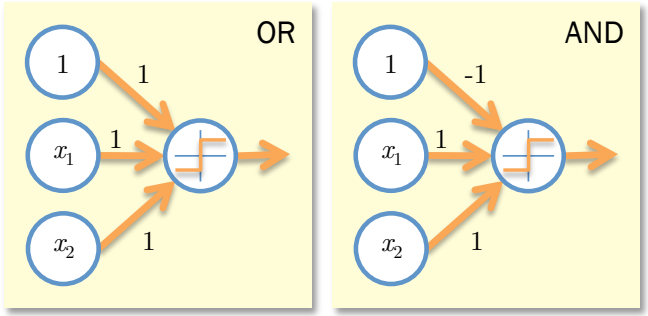
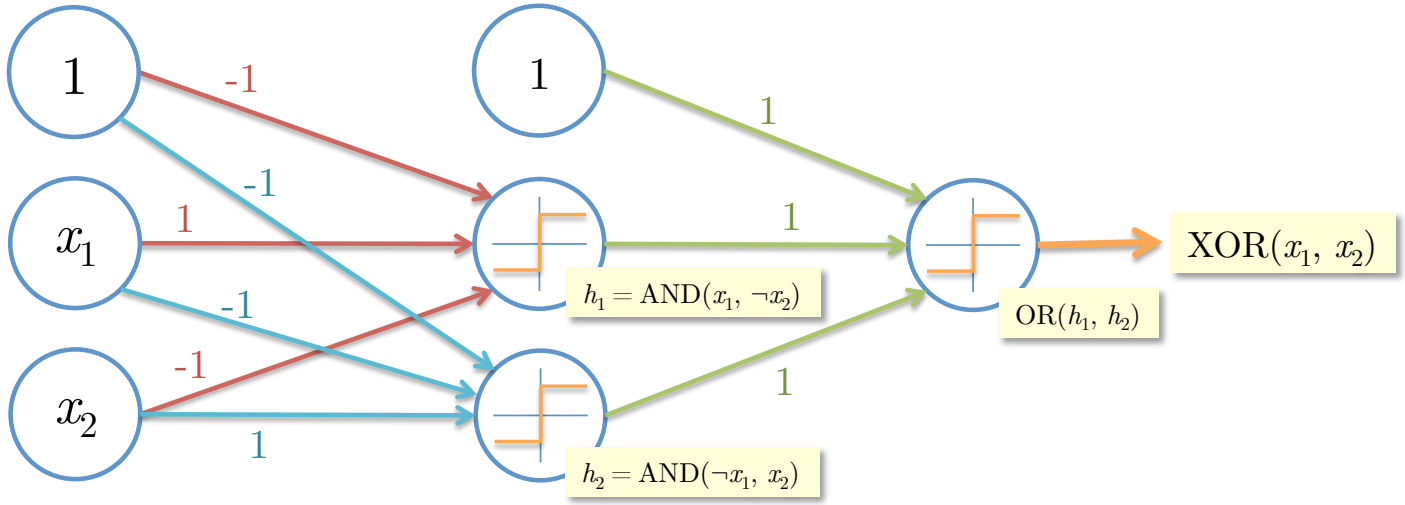


Image Credit: Andrej Karpathy, CS231n

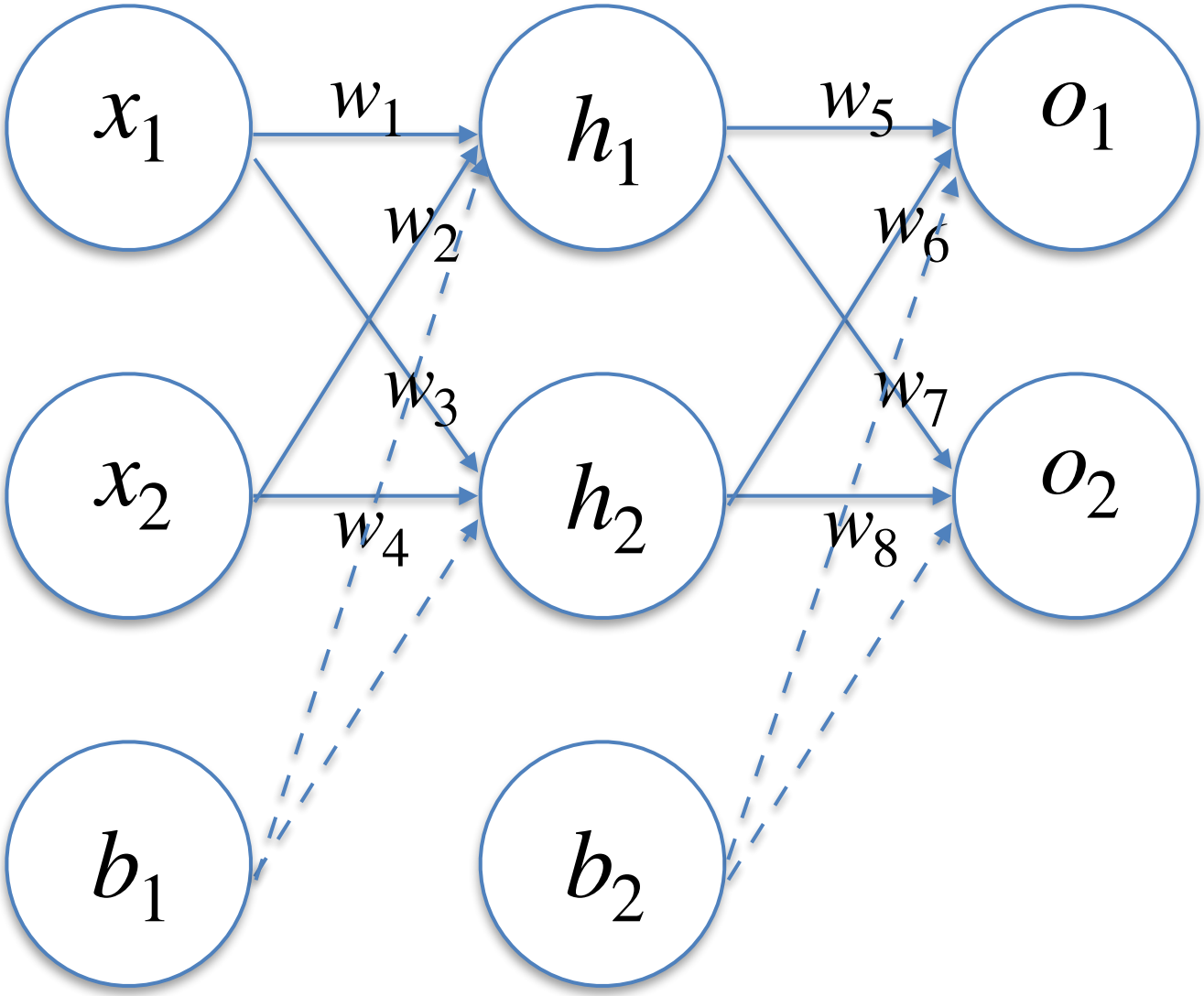
XOR - A Multilayer Network

$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(x_1, \neg x_2), \mathrm{AND}(\neg x_1, x_2))$$

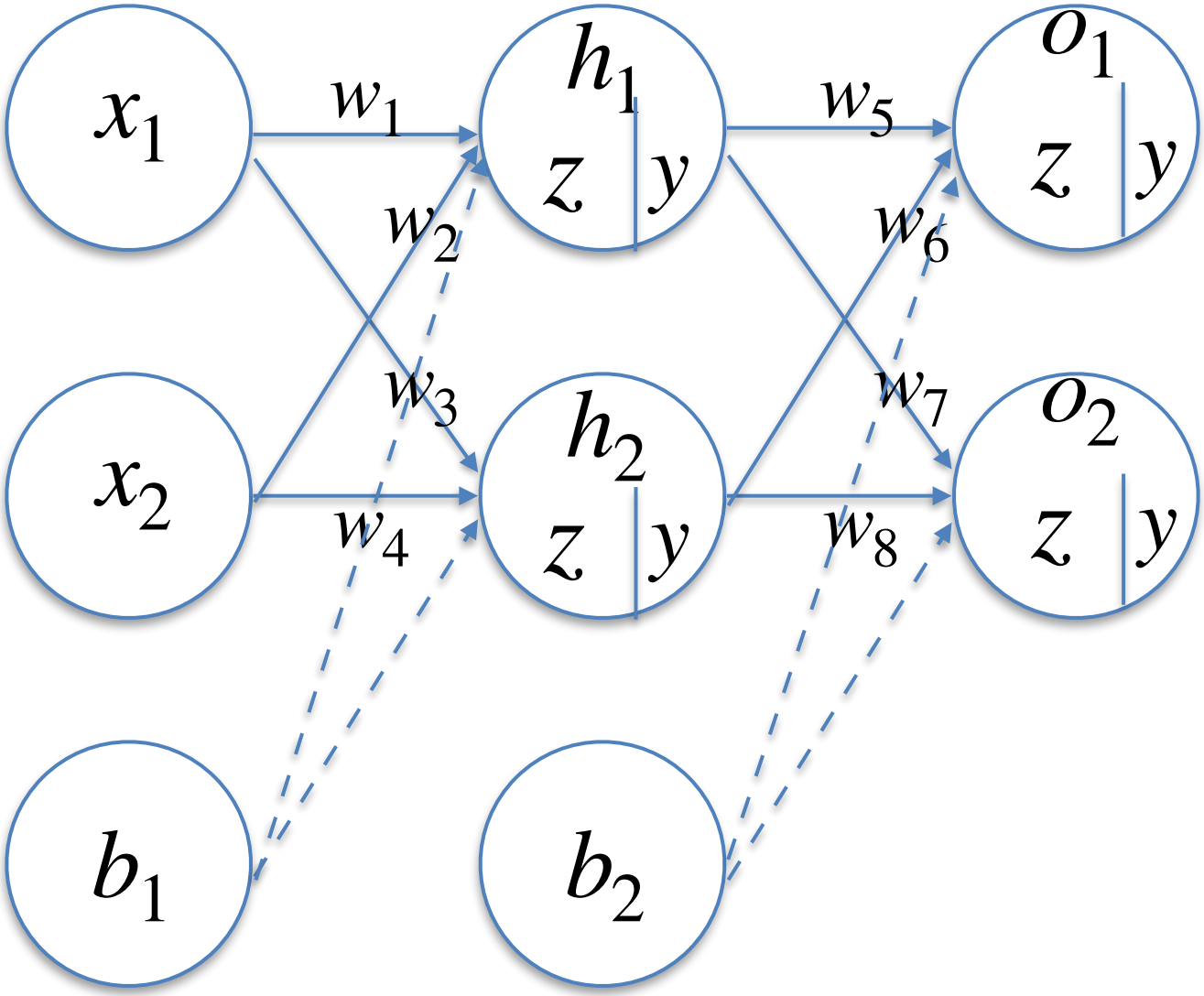




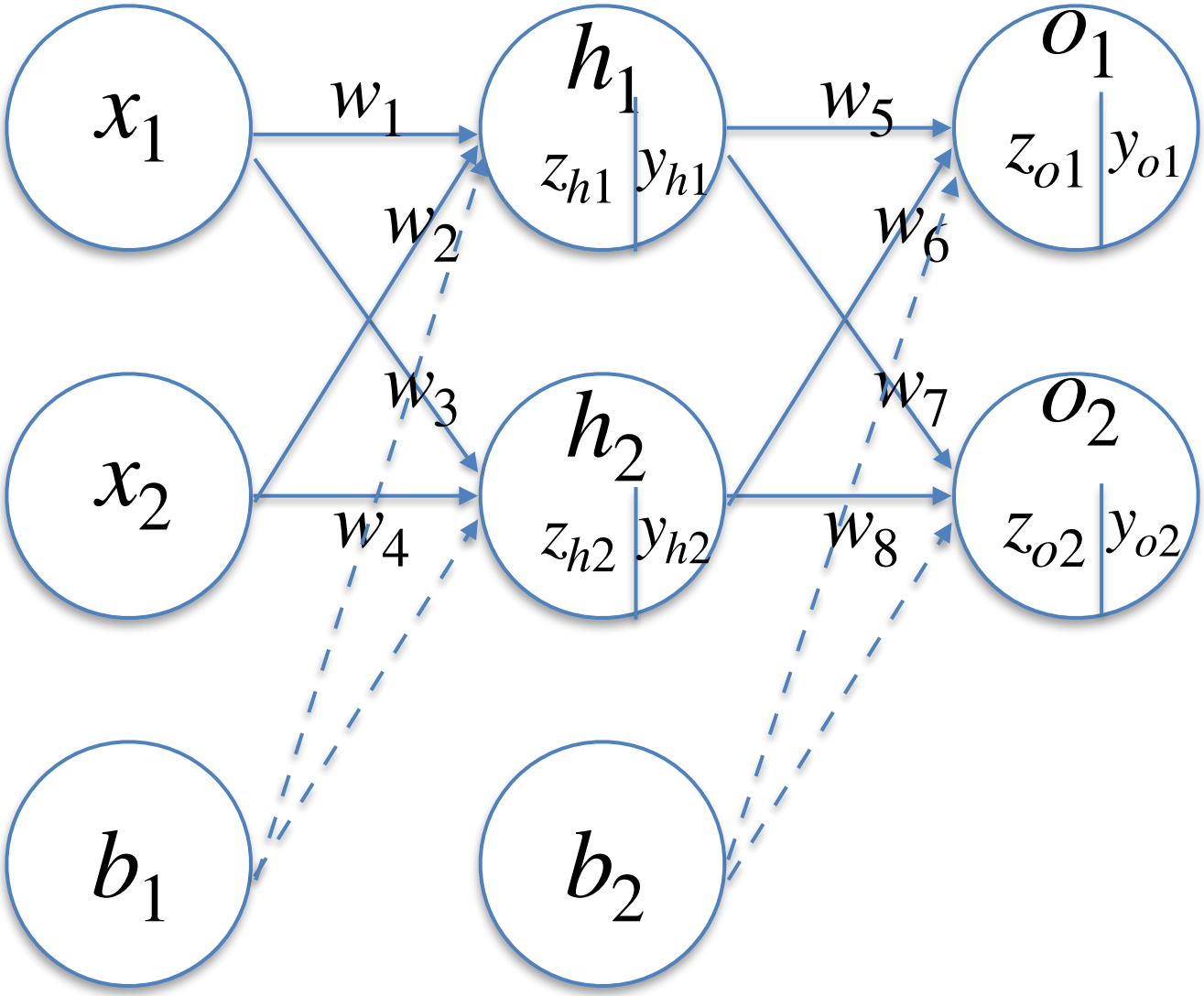
Multilayer Networks



Multilayer Networks

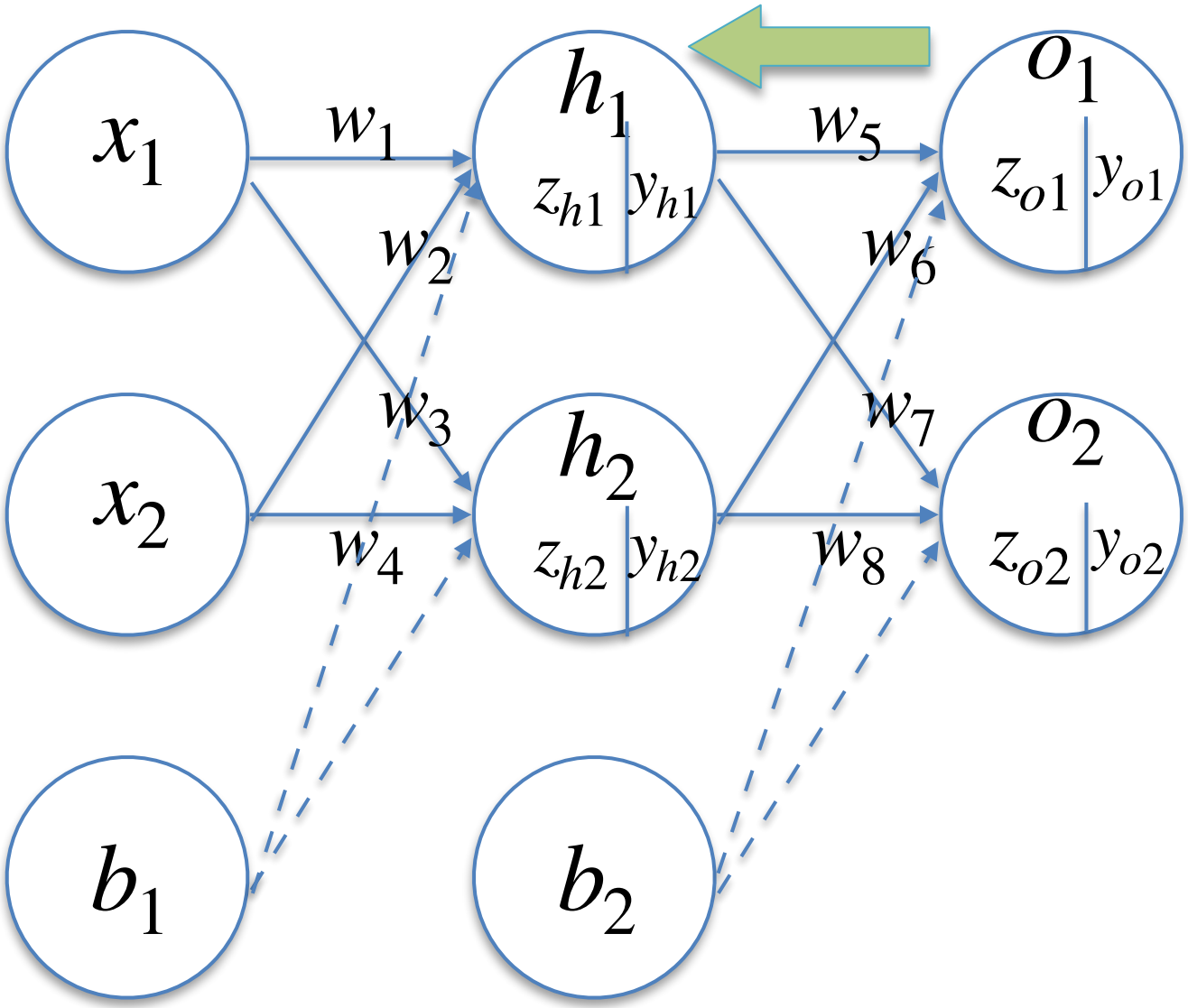


Multilayer Networks

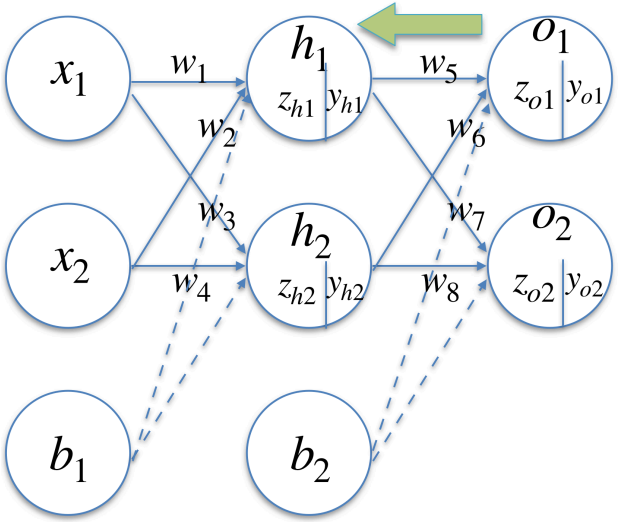


Key Computation: Back-Prop

$$\frac{\partial E}{\partial w_5} = ?$$



Key Computation: Back-Prop

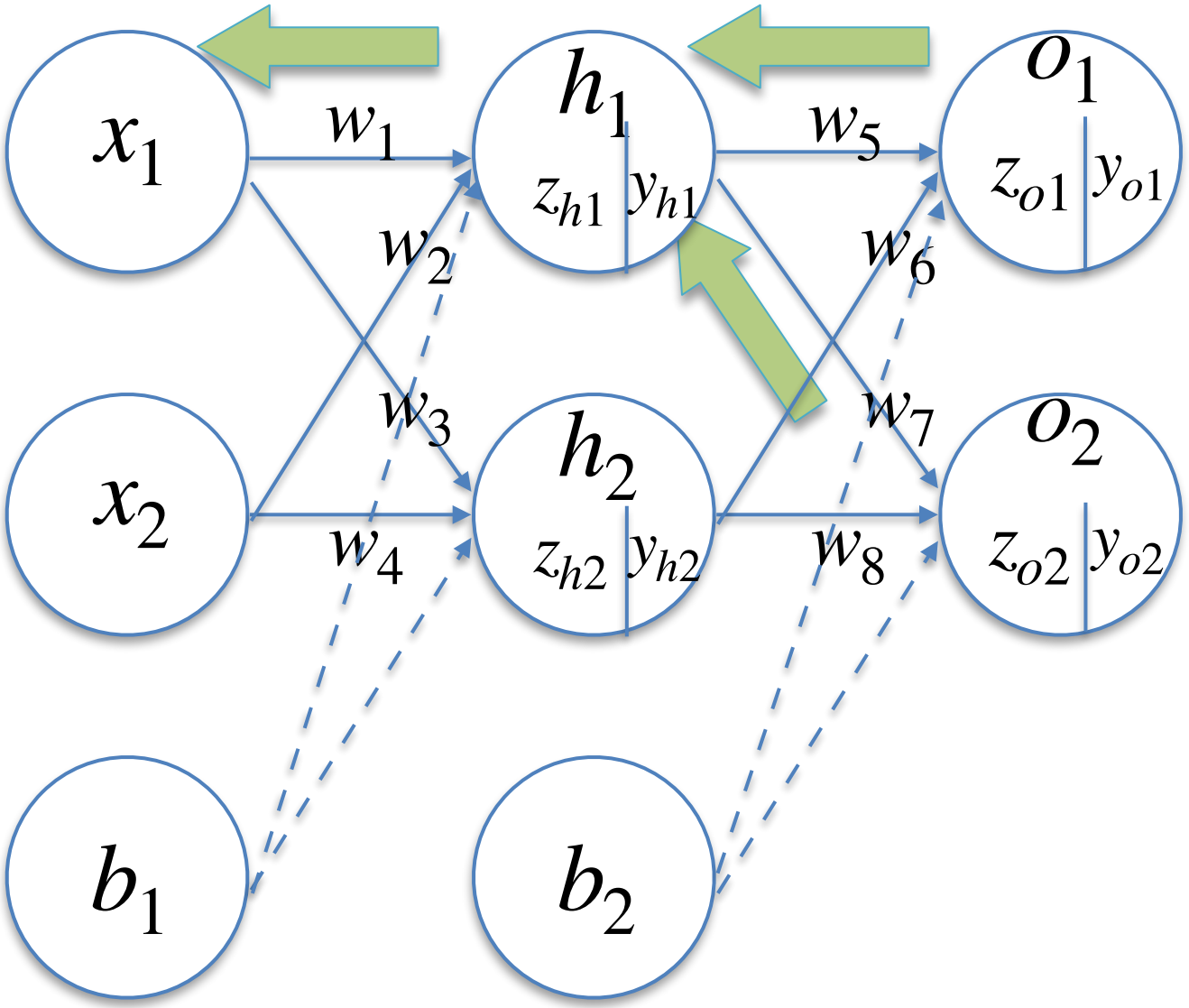


$$\frac{\partial E}{\partial w_5} \leftarrow \frac{\partial z_{o1}}{\partial w_5} \leftarrow \frac{\partial y_{o1}}{\partial z_{o1}} \leftarrow \frac{\partial E_{o1}}{\partial y_{o1}} + \frac{\partial E_{o2}}{\partial y_{o1}} \leftarrow E_T$$

$$\frac{\partial E}{\partial w_5} \leftarrow y_{h1} \leftarrow y_{o1}(1 - y_{o1}) \leftarrow -(t_{o1} - y_{o1}) \leftarrow E_T$$

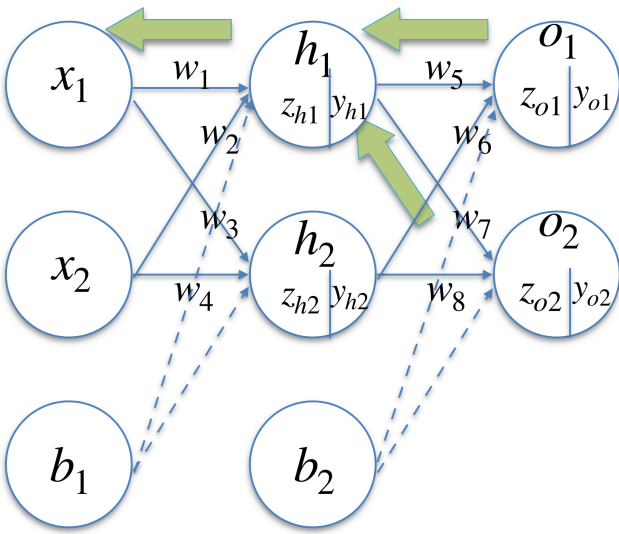
Key Computation: Back-Prop

$$\frac{\partial E}{\partial w_1} = ?$$



Key Computation: Back-Prop

$$\frac{\partial E}{\partial w_1} = ?$$



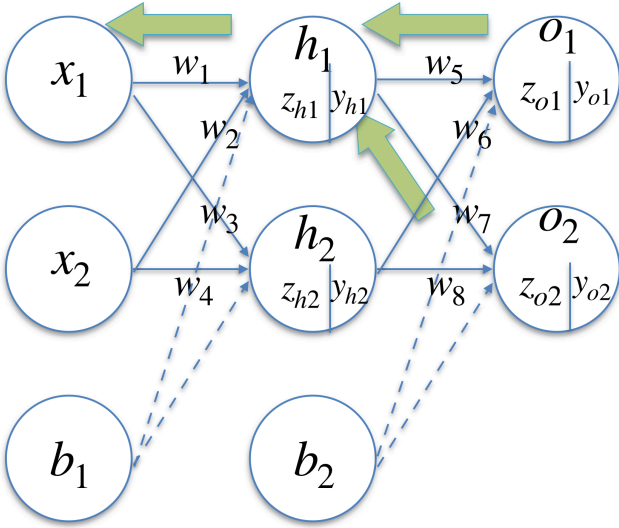
$$\frac{\partial z_{h1}}{\partial w_1} \leftarrow \frac{\partial y_{h1}}{\partial z_{h1}} \leftarrow \frac{\partial E}{\partial y_{h1}} \leftarrow E$$

$$\frac{\partial z_{h1}}{\partial w_1} \leftarrow \frac{\partial y_{h1}}{\partial z_{h1}} \leftarrow \frac{\partial E_{o1}}{\partial y_{h1}} + \frac{\partial E_{o2}}{\partial y_{h1}} \leftarrow E$$

$$\frac{\partial z_{h1}}{\partial w_1} \leftarrow \frac{\partial y_{h1}}{\partial z_{h1}} \leftarrow \frac{\partial z_{o1}}{\partial y_{h1}}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_{o1}}{\partial y_{o1}} + \frac{\partial z_{o2}}{\partial y_{h1}}\frac{\partial y_{o2}}{\partial z_{o2}}\frac{\partial E_{o2}}{\partial y_{o2}} \leftarrow E$$

Key Computation: Back-Prop

$$\frac{\partial E}{\partial w_1} = ?$$



$$\frac{\partial z_{h1}}{\partial w_1} \leftarrow \frac{\partial y_{h1}}{\partial z_{h1}} \leftarrow \frac{\partial E}{\partial y_{h1}} \leftarrow E$$

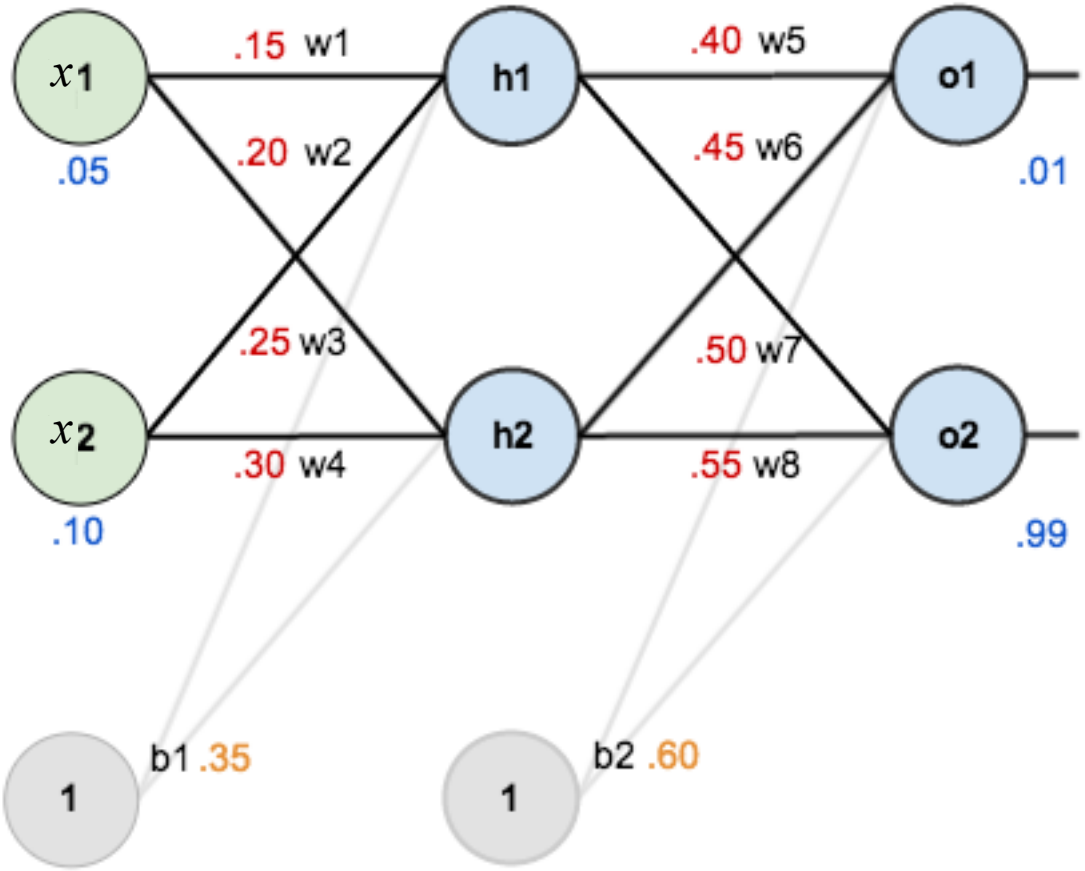
$$\frac{\partial z_{h1}}{\partial w_1} \leftarrow \frac{\partial y_{h1}}{\partial z_{h1}} \leftarrow \frac{\partial z_{o1}}{\partial y_{h1}}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_{o1}}{\partial y_{o1}} + \frac{\partial z_{o2}}{\partial y_{h1}}\frac{\partial y_{o2}}{\partial z_{o2}}\frac{\partial E_{o2}}{\partial y_{o2}} \leftarrow E$$

$$x_1 \leftarrow y_{h1}(1-y_{h1}) \leftarrow -w_5 y_{o1}(1-y_{o1})(t_{o1}-y_{o1}) + -w_7 y_{o2}(1-y_{o2})(t_{o2}-y_{o2}) \leftarrow E$$

Numerical Example

A Step by Step Backpropagation Example

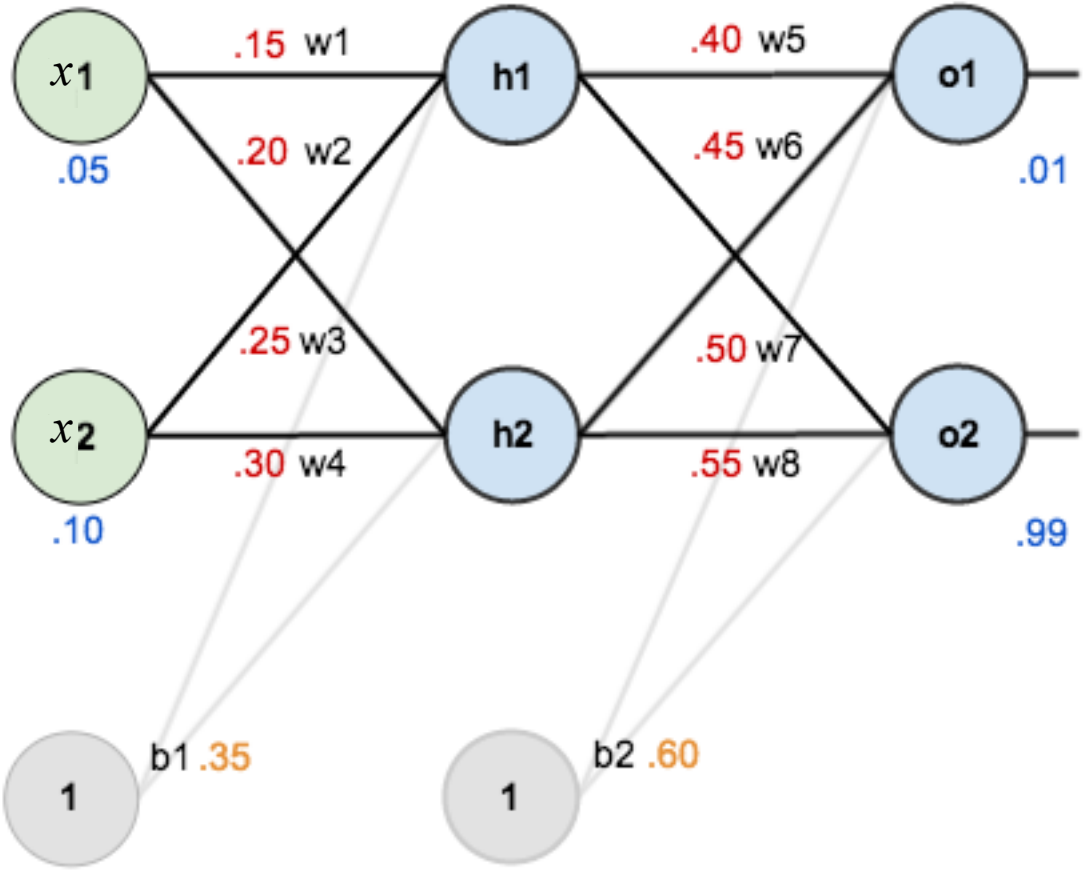
- Here's the basic structure



https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

A Step by Step Backpropagation Example

- Here are the initial weights, the biases, and training inputs/outputs:



A Step by Step Backpropagation Example

- The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

- For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

A Step by Step Backpropagation Example

- The Forward Pass
  - Total net input for h_1:

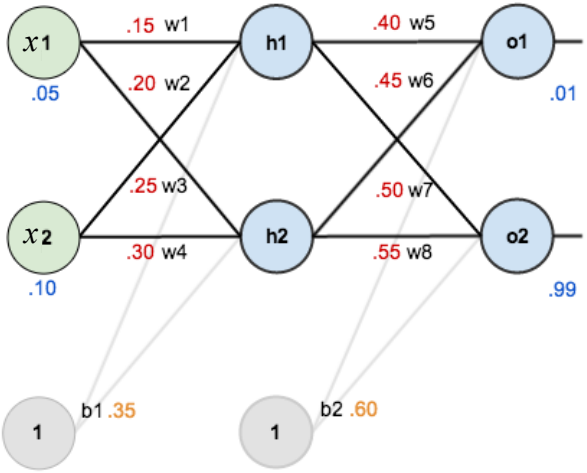
$$z_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$
$$z_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

  - We then squash it using the logistic function to get the output of :

$$y_{h1} = \frac{1}{1 + e^{-z_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

  - Carrying out the same process for h_2 we get:

$$y_{h2} = 0.596884378$$



A Step by Step Backpropagation Example

- The Forward Pass
  - Output for o_1:

$$z_{o1} = w_5 * y_{h1} + w_6 * y_{h2} + b_2 * 1$$

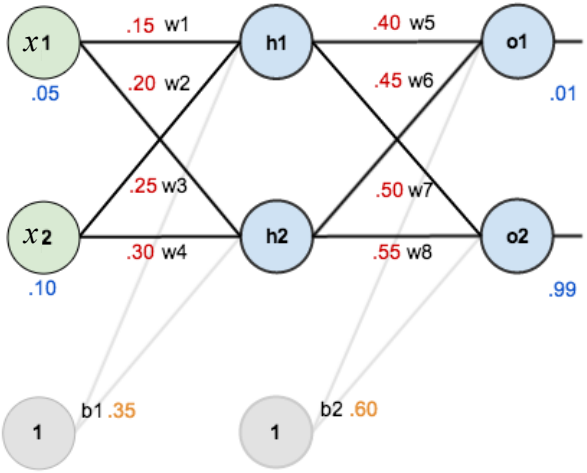
$$z_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1$$

$$= 1.105905967$$

$$y_{o1} = \frac{1}{1 + e^{-z_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

  - For o_2 we get:

$$y_{o2} = 0.772928465$$



A Step by Step Backpropagation Example

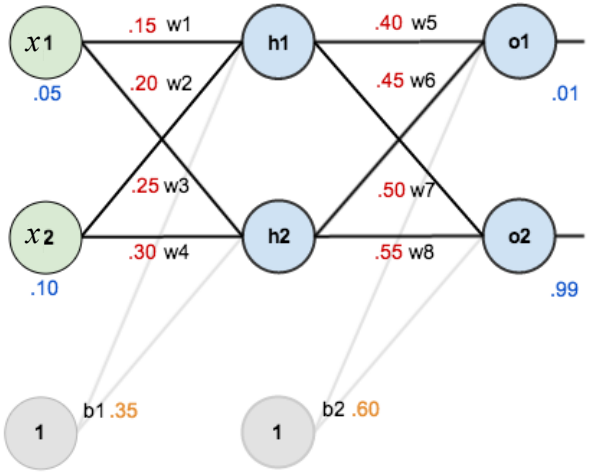
- Calculating the Error

$$E_T = \sum \tfrac{1}{2}(target - output)^2$$

$$E_{o2} = 0.023560026$$

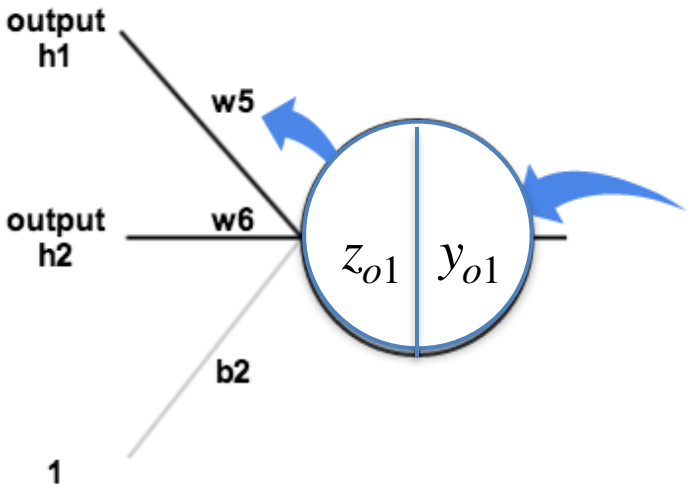
$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2 = \tfrac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_T = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

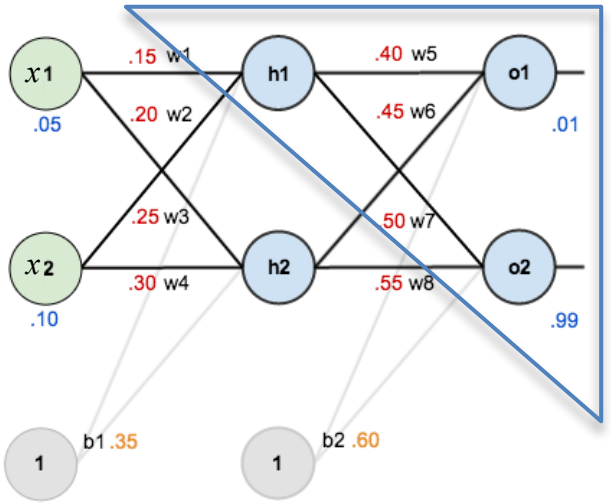


A Step by Step Backpropagation Example

- The Backward Pass
  - Output Layer
    - Consider w_5. We want to know how much a change in affects the total error, aka $\frac{\partial E_T}{\partial w_5}$
    - By applying the chain rule we know that: $$\frac{\partial E_T}{\partial w_5} = \frac{\partial z_{o1}}{\partial w_5}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_T}{\partial y_{o1}}$$
    - Visually, here's what we're doing:



$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2$$
$$E_T = E_{o1} + E_{o2}$$



A Step by Step Backpropagation Example

- The Backward Pass

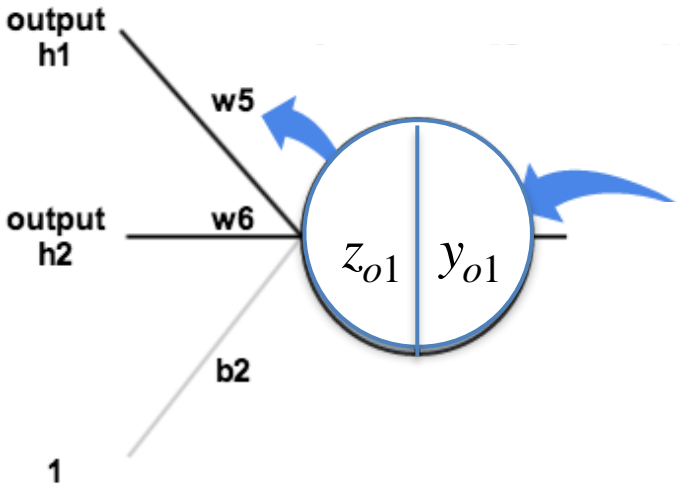
$$\frac{\partial E_T}{\partial w_5} = \frac{\partial z_{o1}}{\partial w_5}\frac{\partial y_{o1}}{\partial z_{o1}}\boxed{\frac{\partial E_T}{\partial y_{o1}}}$$

  - how much does the total error change w.r.t the output?

$$E_T = \frac{1}{2}(t_{o1} - y_{o1})^2 + \frac{1}{2}(t_{o2} - y_{o2})^2$$

$$\frac{\partial E_T}{\partial y_{o1}} = 2 * \frac{1}{2}(t_{o1} - y_{o1})^{2-1} * (-1) + (-1) * 0$$

$$= -(0.01 - 0.75136507) = 0.74136507$$



$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2$$

$$E_T = E_{o1} + E_{o2}$$

A Step by Step Backpropagation Example

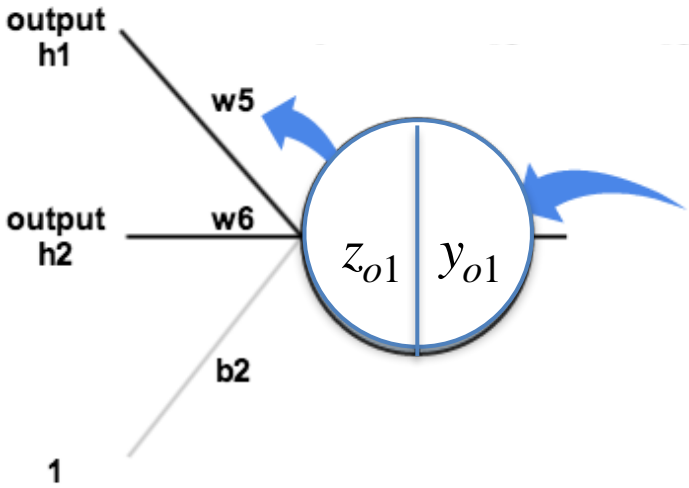
- The Backward Pass

$$\frac{\partial E_T}{\partial w_5} = \frac{\partial z_{o1}}{\partial w_5}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_T}{\partial y_{o1}}$$

  - how much does the output o_1 change w.r.t its total net input?

$$y_{o1} = \frac{1}{1 + e^{-z_{o1}}}$$

$$\frac{\partial y_{o1}}{\partial z_{o1}} = y_{o1}(1 - y_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$



$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2$$

$$E_T = E_{o1} + E_{o2}$$

A Step by Step Backpropagation Example

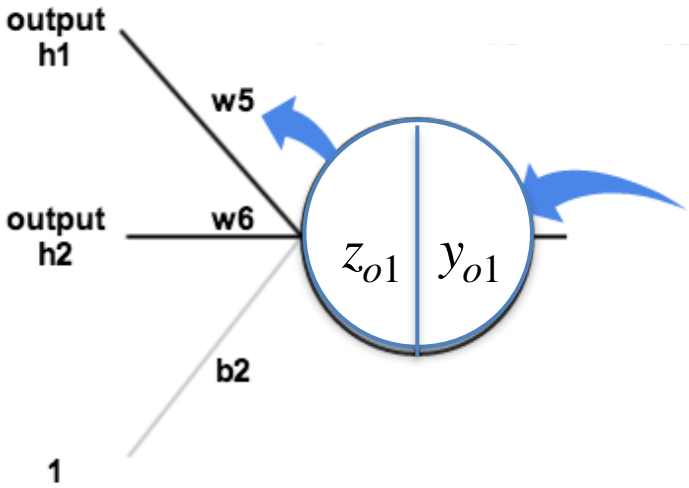
- The Backward Pass

$$\frac{\partial E_T}{\partial w_5} = \frac{\partial z_{o1}}{\partial w_5}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_T}{\partial y_{o1}}$$

  - how much does the total net input of o_1 change with respect to w_5?

$$z_{o1} = w_5 * y_{h1} + w_6 * y_{h2} + b_2 * 1$$

$$\frac{\partial z_{o1}}{\partial w_5} = 1 * y_{h1} = 0.593269992$$



$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2$$

$$E_T = E_{o1} + E_{o2}$$

A Step by Step Backpropagation Example

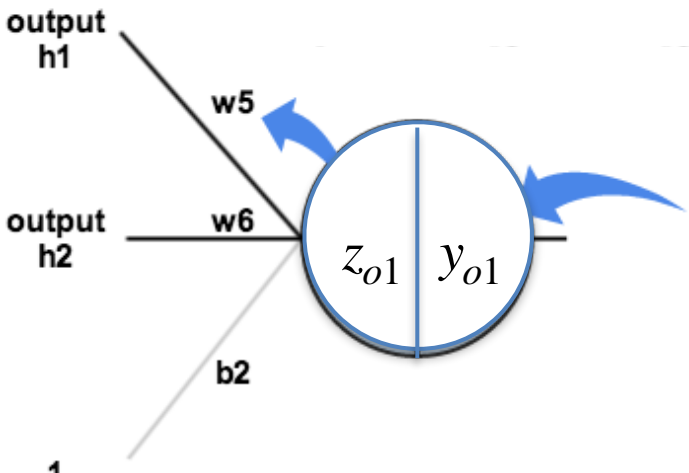
- The Backward Pass
  - Putting it all together:

$$\frac{\partial E_T}{\partial w_5} = \frac{\partial z_{o1}}{\partial w_5}\frac{\partial y_{o1}}{\partial z_{o1}}\frac{\partial E_T}{\partial y_{o1}}$$

$$\frac{\partial E_T}{\partial w_5} = -(t_{o1} - y_{o1}) y_{o1} * (1 - y_{o1}) * y_{h1}$$

$$= 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$w_5^+ = w_5 - \eta \frac{\partial E_T}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$



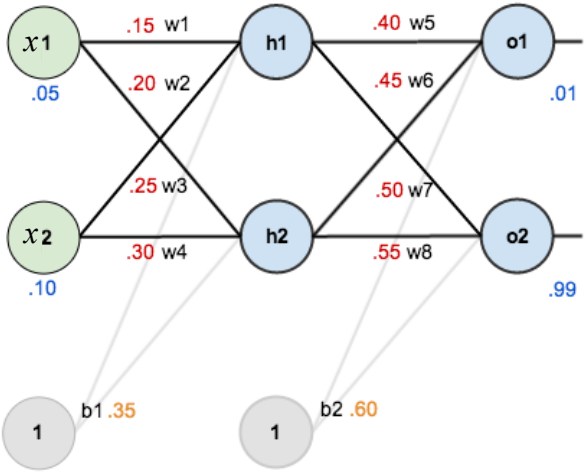
$$E_{o1} = \frac{1}{2}(t_{o1} - y_{o1})^2$$

$$E_T = E_{o1} + E_{o2}$$

A Step by Step Backpropagation Example

- The Backward Pass
  - We can repeat this process to get the new weights w_6, w_7, and w_8

$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$

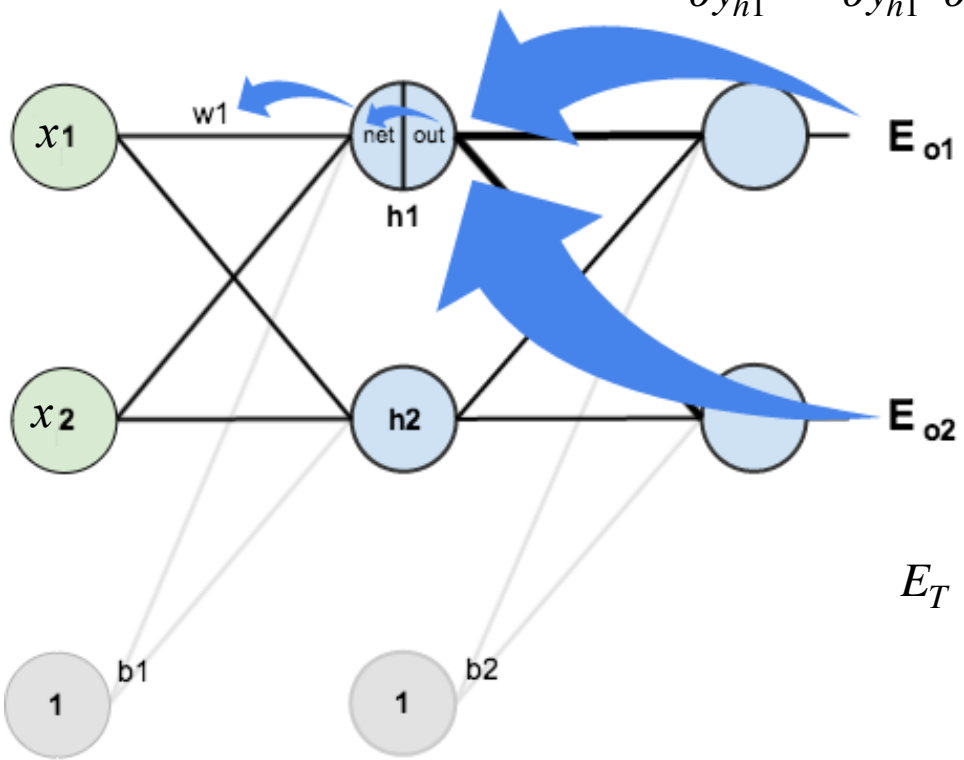


A Step by Step Backpropagation Example

- The Backward Pass - Hidden Layer
  - Continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4

$$\frac{\partial E_T}{\partial w_1} = \frac{\partial z_{h1}}{\partial w_1}\frac{\partial y_{h1}}{\partial z_{h1}}\frac{\partial E_T}{\partial y_{h1}}$$

$$\frac{\partial E_T}{\partial y_{h1}} = \frac{\partial E_{o1}}{\partial y_{h1}}\frac{\partial E_{o2}}{\partial y_{h1}}$$

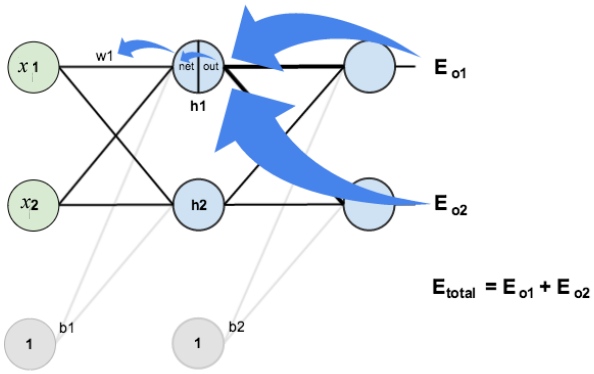


$$E_T = E_{o1} + E_{02}$$

A Step by Step Backpropagation Example

- The Backward Pass - Hidden Layer
  - Following the same process for $\frac{\partial E_{o2}}{\partial y_{h1}}$,
  - we get:

$$\frac{\partial E_{o2}}{\partial y_{h1}} = -0.019049119$$



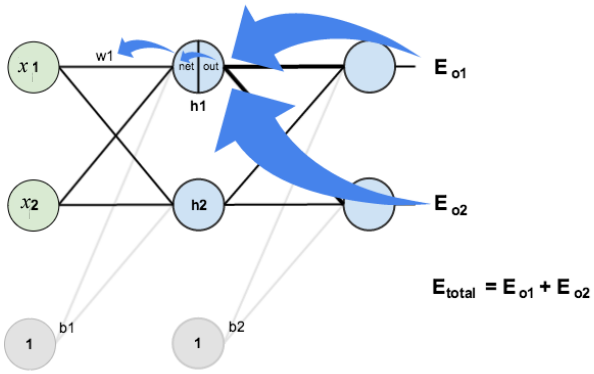
  - Therefore:

$$\frac{\partial E_T}{\partial y_{h1}} = \frac{\partial E_{o1}}{\partial y_{h1}} + \frac{\partial E_{o2}}{\partial y_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

A Step by Step Backpropagation Example

- The Backward Pass - Hidden Layer

$$\frac{\partial E_T}{\partial w_1} = \frac{\partial z_{h1}}{\partial w_1}\frac{\partial y_{h1}}{\partial z_{h1}}\frac{\partial E_T}{\partial y_{h1}}$$



$$z_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$

$$\frac{\partial z_{h1}}{\partial w_1} = i_1 = 0.05$$

$$y_{h1} = \frac{1}{1 + e^{-z_{h1}}}$$

$$\frac{\partial y_{h1}}{\partial z_{h1}} = y_{h1}(1 - y_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$= 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_T}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

- Similarly,

$$w_2^+ = 0.19956143 \qquad w_3^+ = 0.24975114 \qquad w_4^+ = 0.29950229$$

Next…

- Convolutional Neural Networks