

Neural Network Architecture

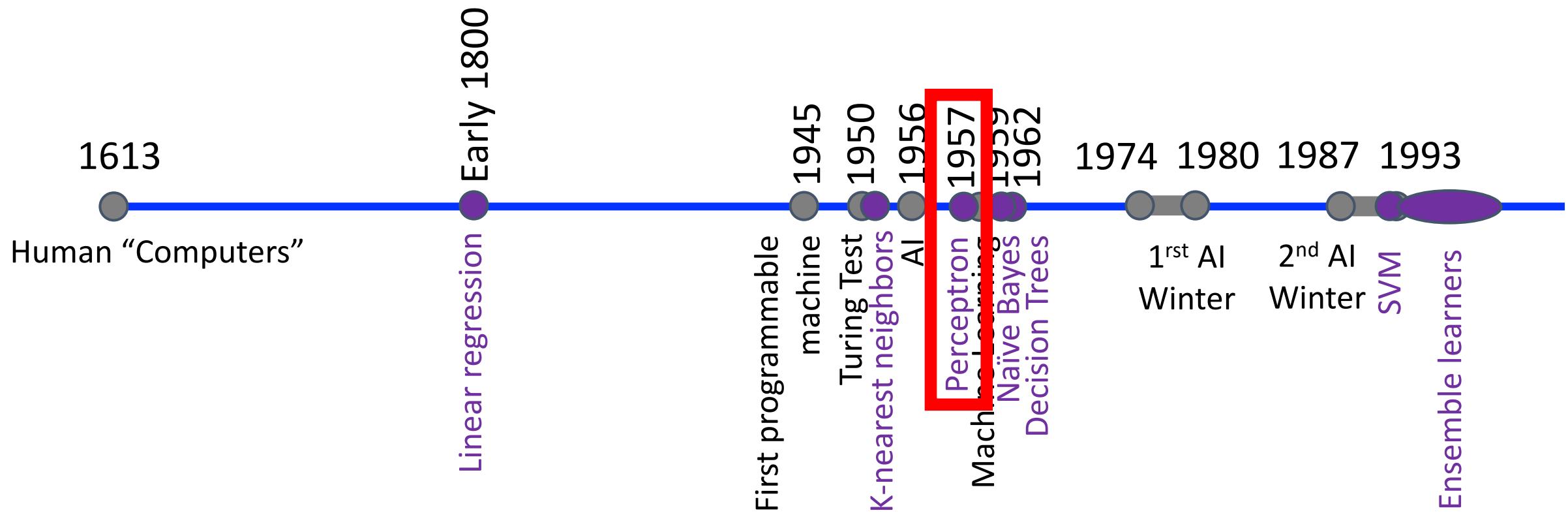
Tanvir Azhar

Lecturer, East Delta University

Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Neural Network Architecture– Learning Mechanism

Recall: Historical Context of ML Models



Recall: Rise & Fall of Perceptron (Artificial Neuron)



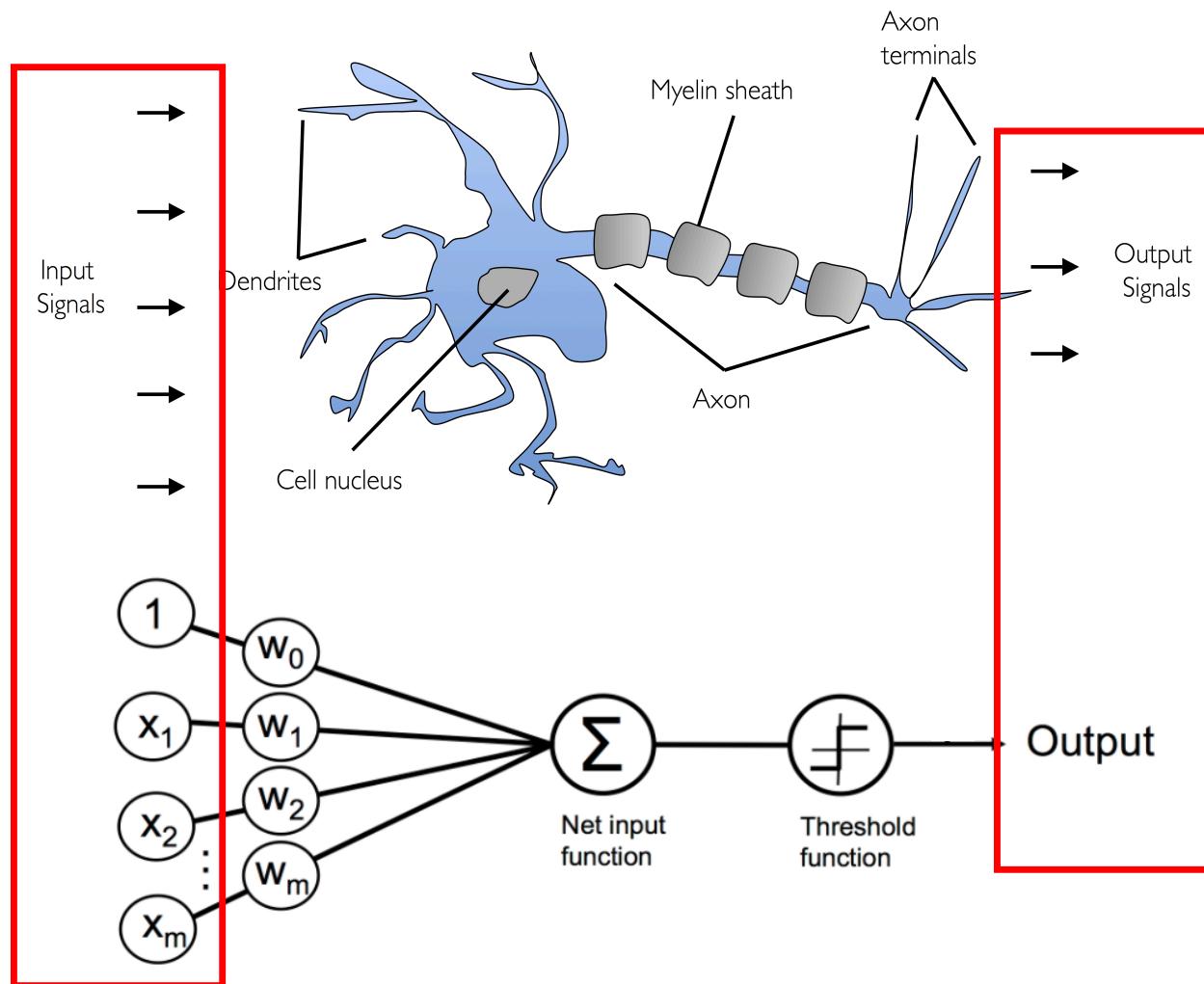
Frank Rosenblatt
(Psychologist)

“[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.... [It] is expected to be finished in about a year at a cost of \$100,000.”

1958 New York Times article: <https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html>

Recall: Rise & Fall of Perceptron (Artificial Neuron)

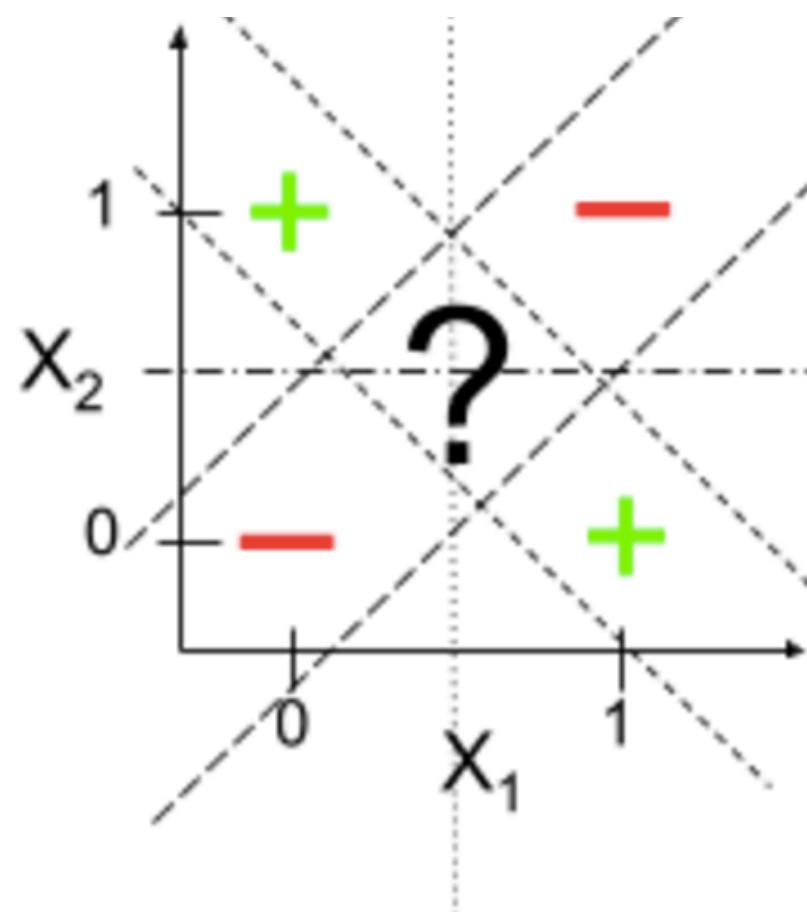
Biological Neuron:



Artificial Neurons
(e.g., Perceptron):

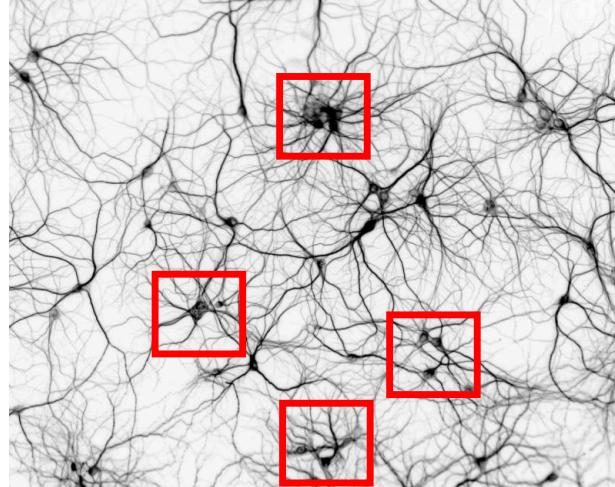
Recall: Rise & Fall of Perceptron (Artificial Neuron)

Cannot solve XOR problem and so separate 1s from 0s with a perceptron (linear function):



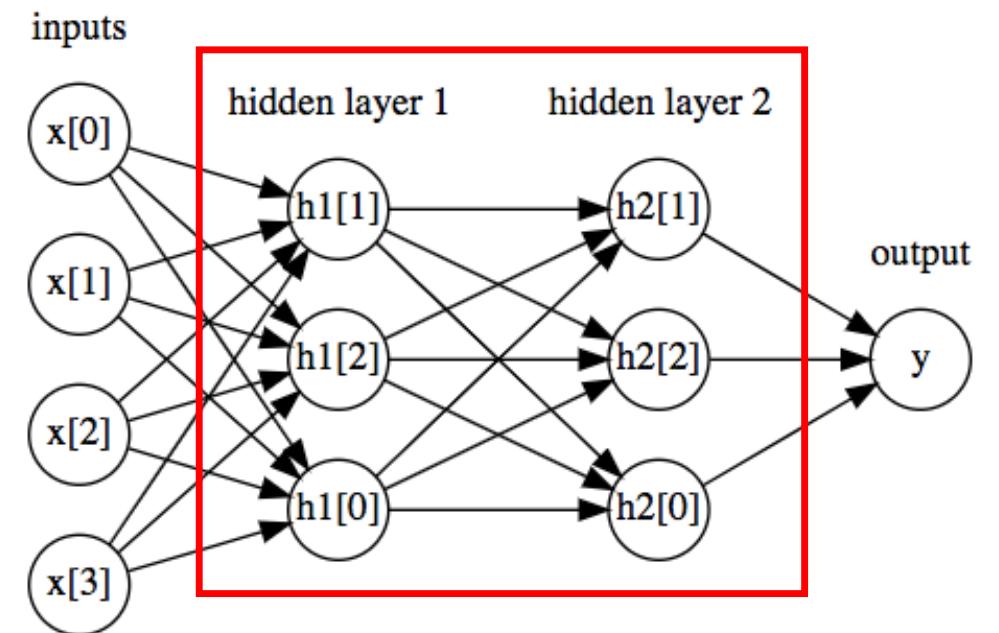
Neural Networks: Connected Neurons

Biological Neural Network:

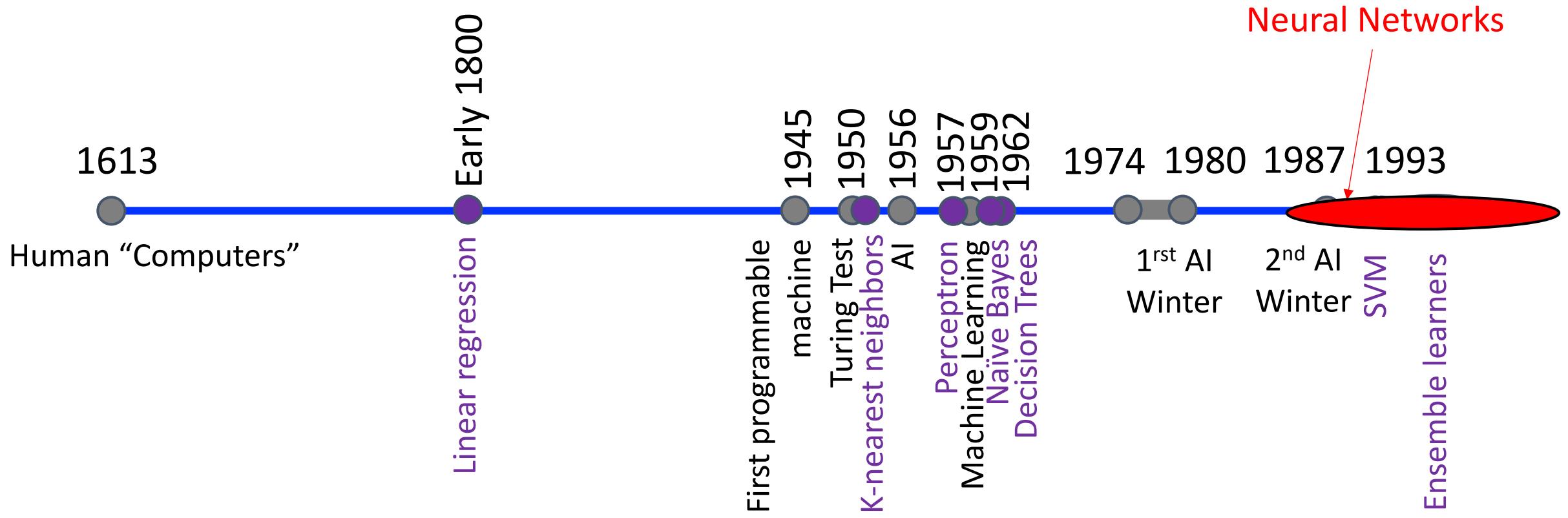


<http://www.rzagabe.com/2014/11/03/an-introduction-to-artificial-neural-networks.html>

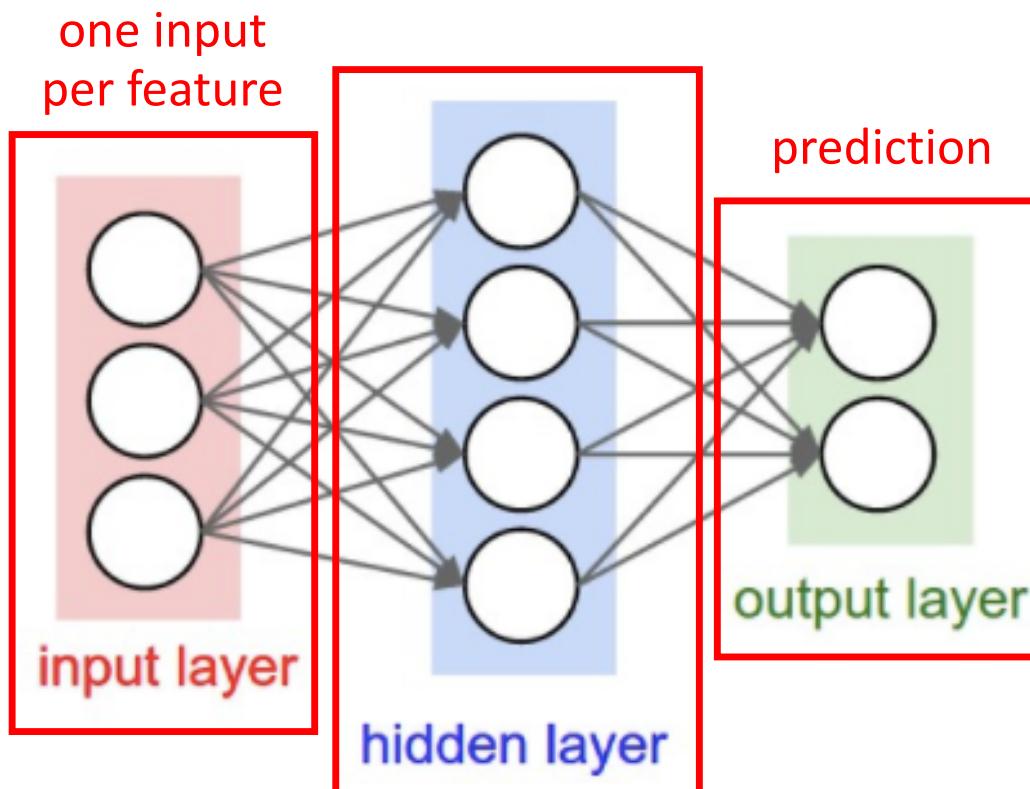
Artificial Neural Network:



Today's Topic: Neural Networks



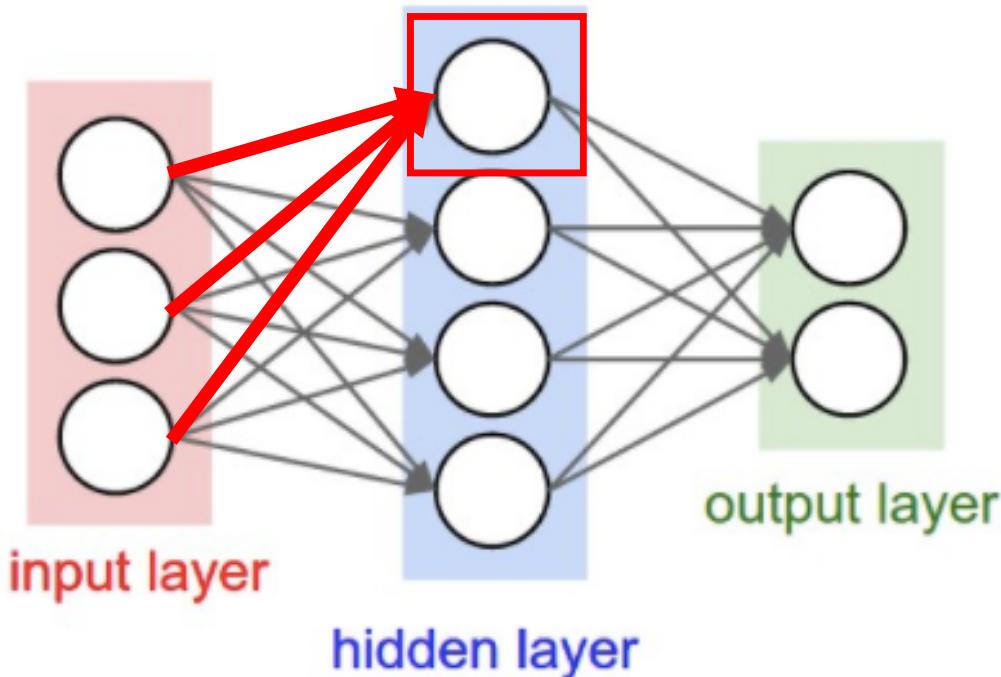
Neural Network



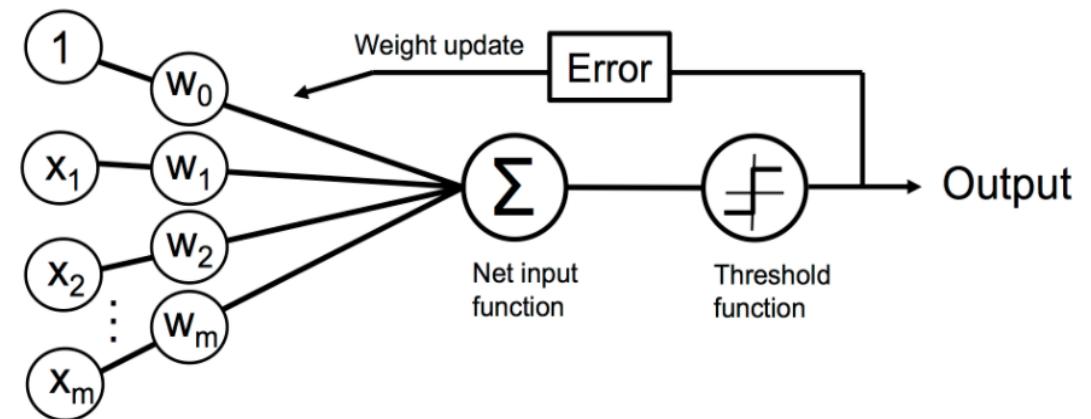
- Also called “multilayer perceptron”
- This is a 2-layer neural network (i.e., count number of hidden layers plus output layer and exclude input layer)

“hidden layer” uses outputs of units (i.e., neurons) and provides them as inputs to other units (i.e., neurons)

Neural Network

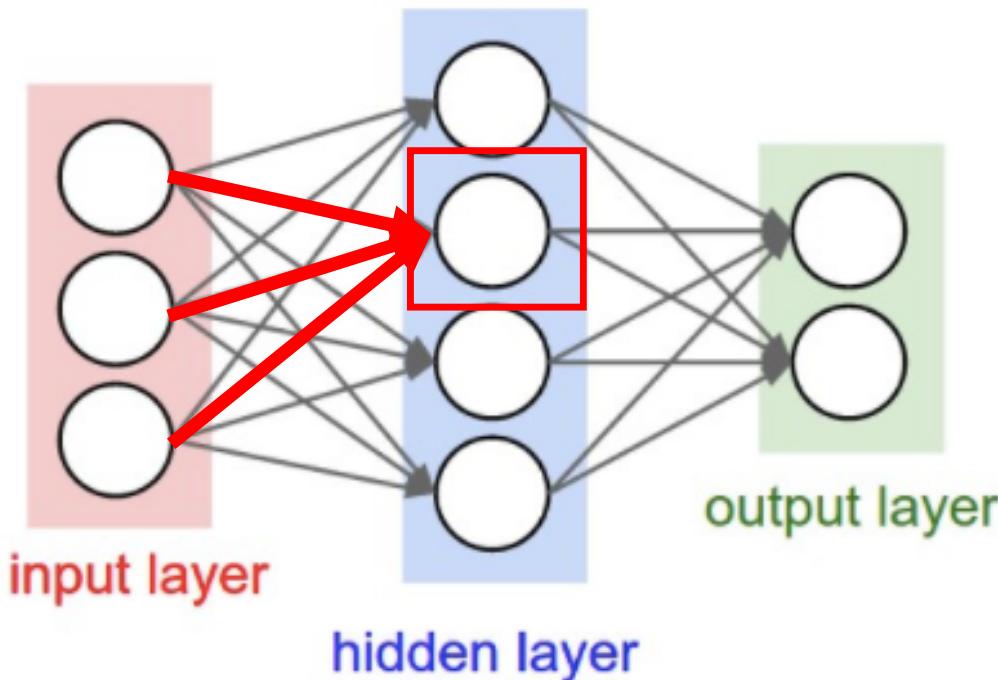


- How does this relate to a perceptron?

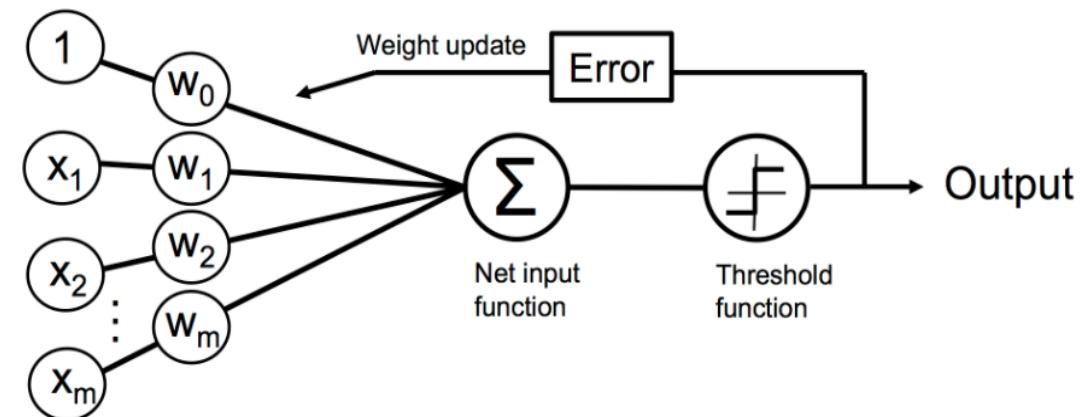


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

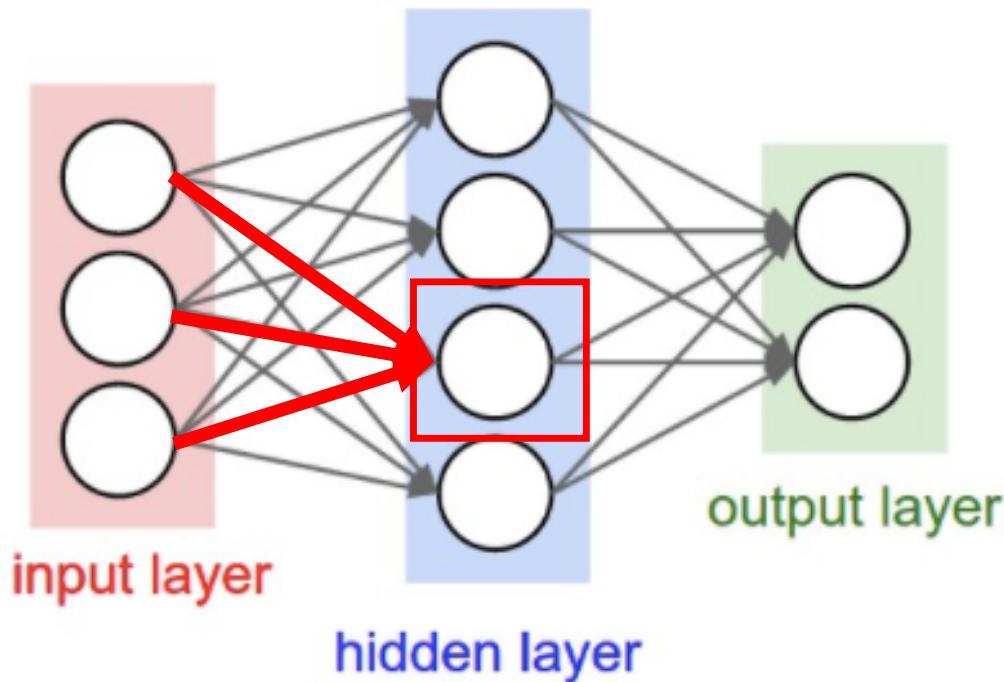


- How does this relate to a perceptron?

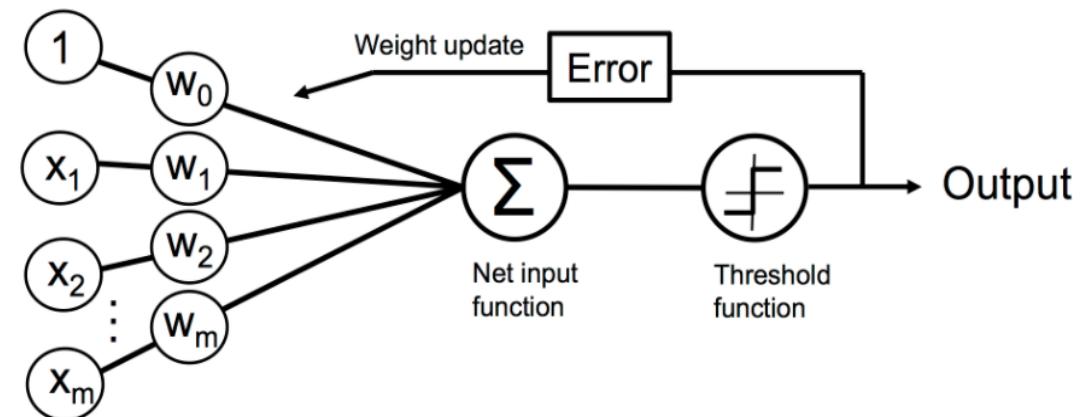


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

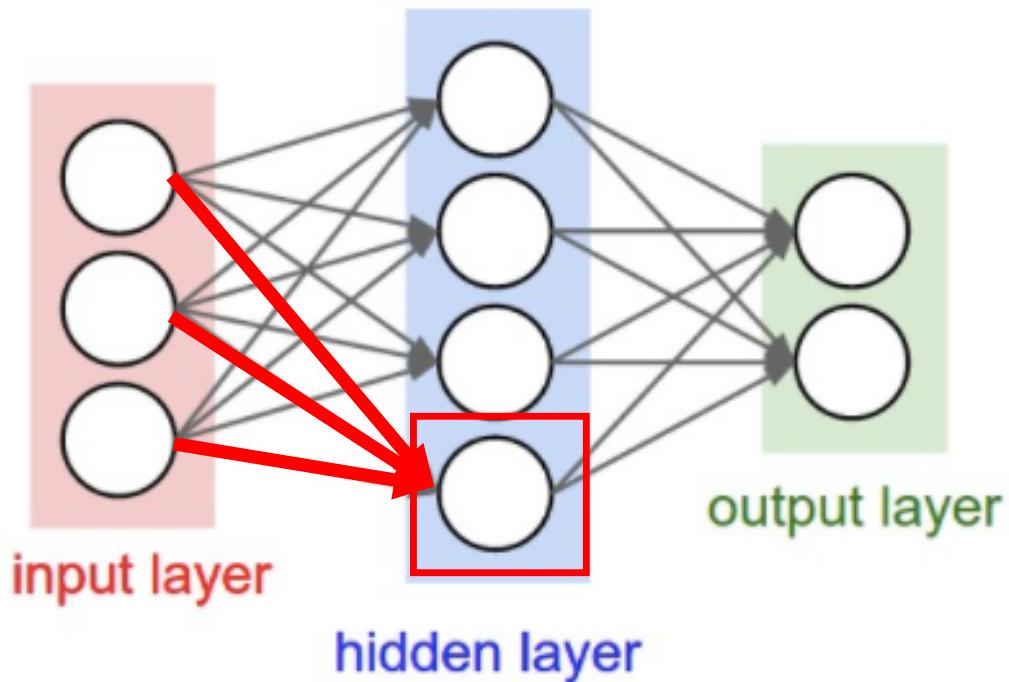


- How does this relate to a perceptron?

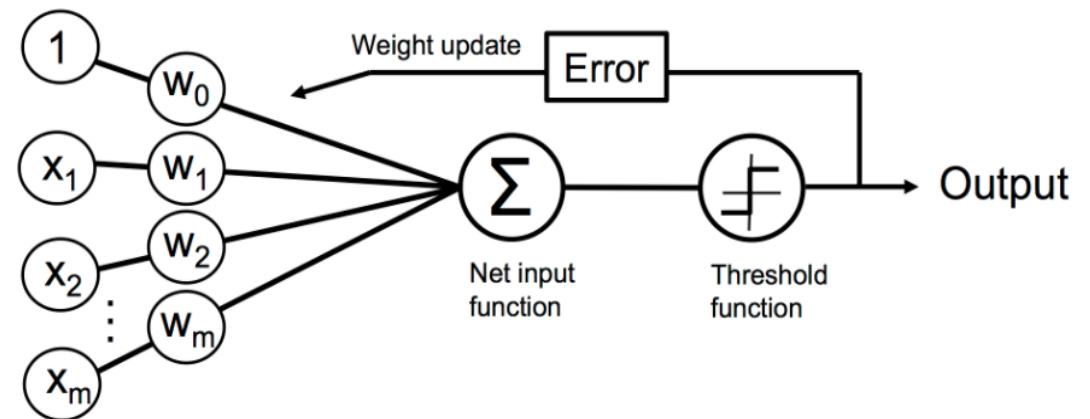


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

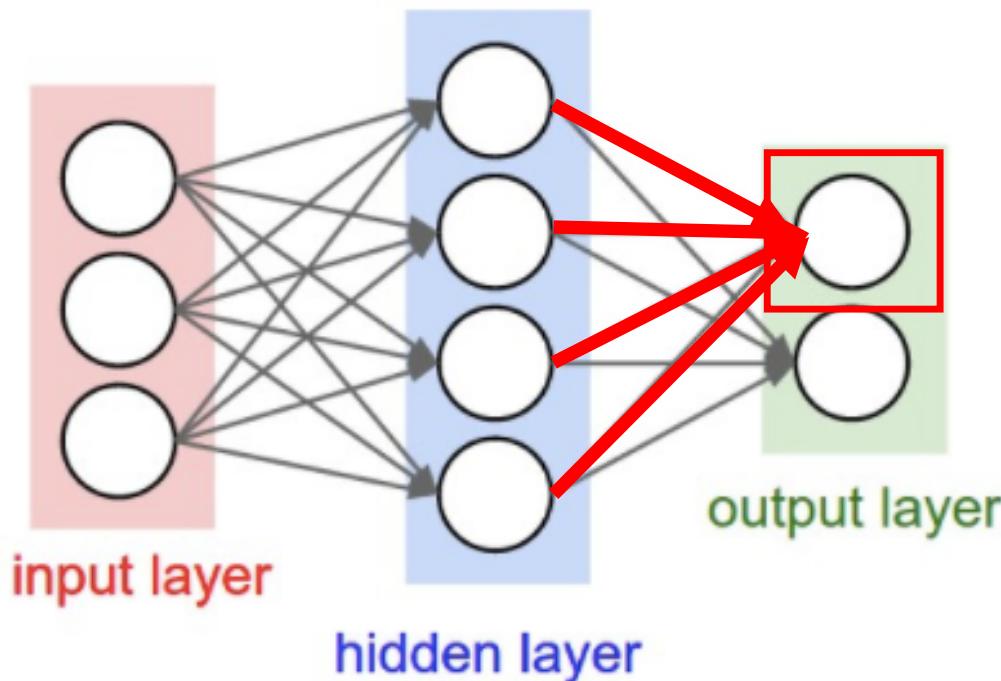


- How does this relate to a perceptron?

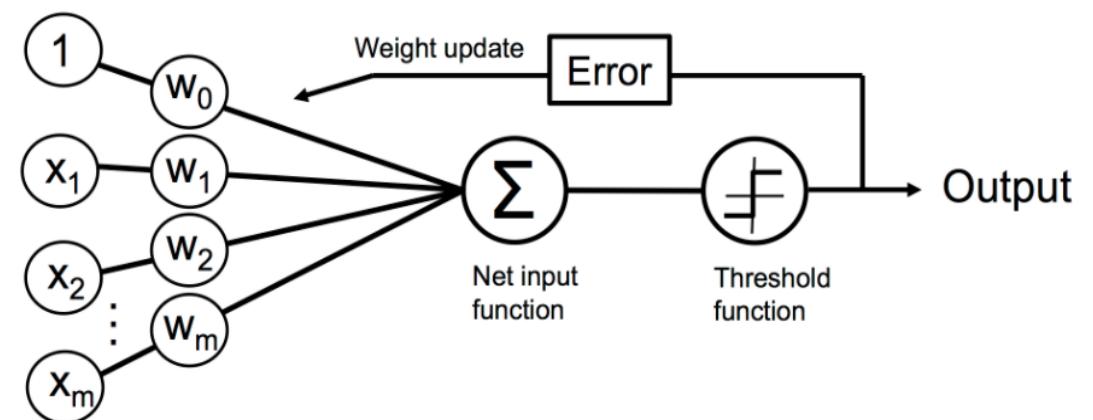


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

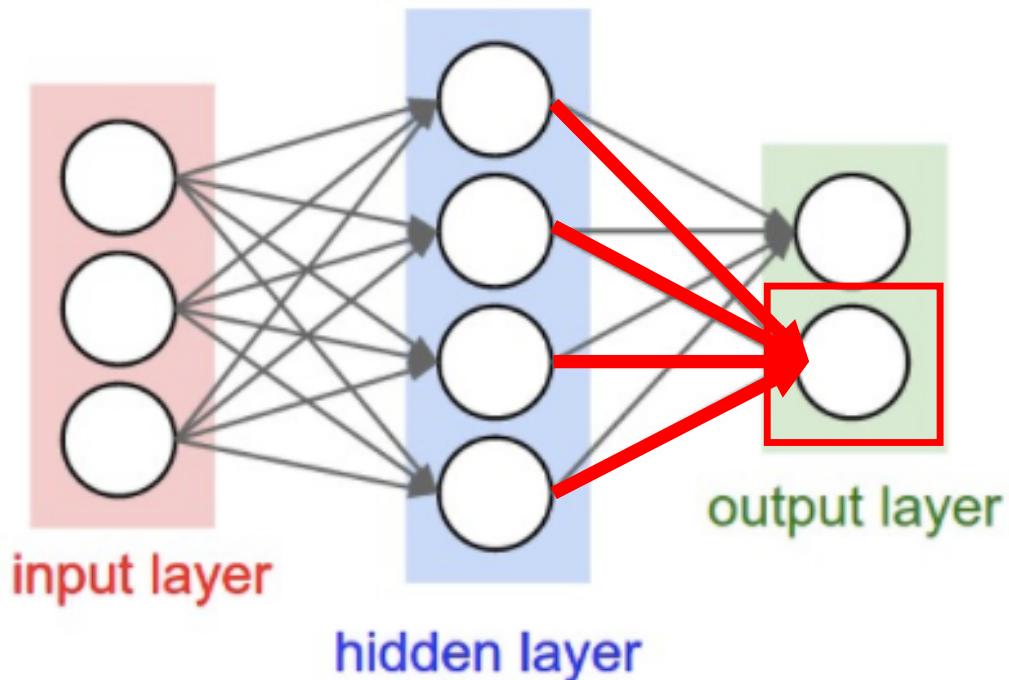


- How does this relate to a perceptron?

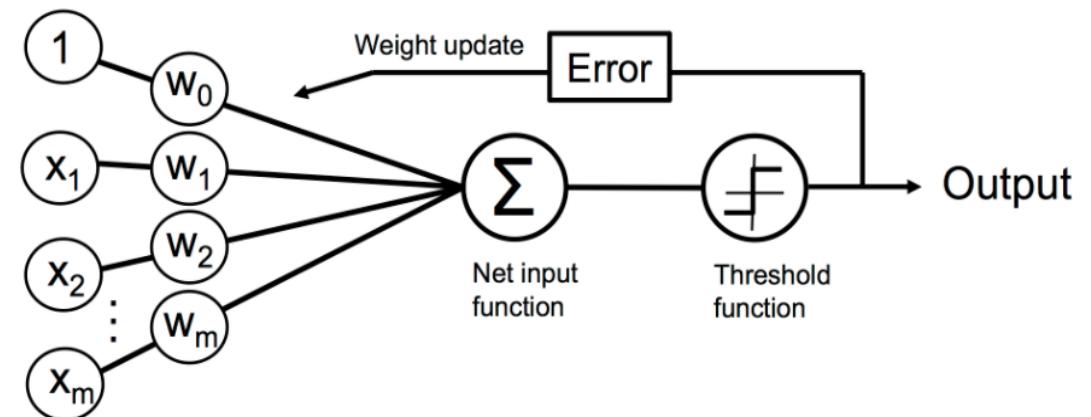


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

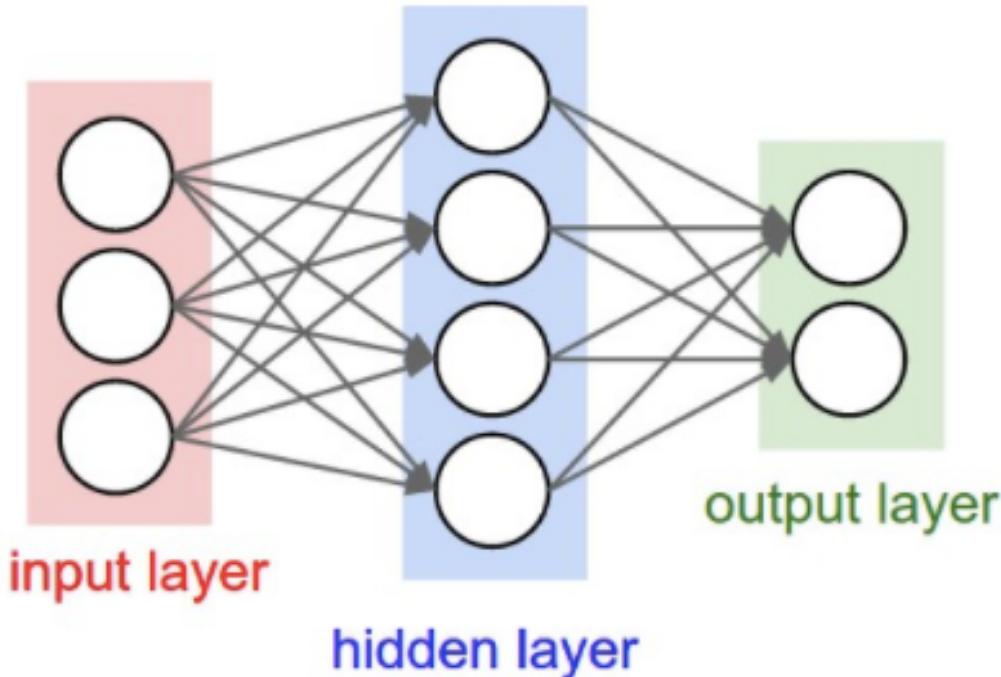


- How does this relate to a perceptron?

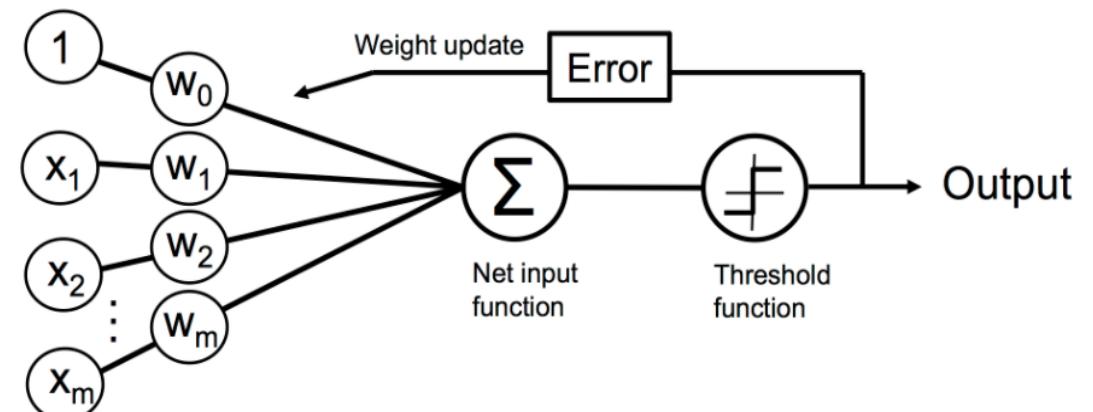


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

Neural Network

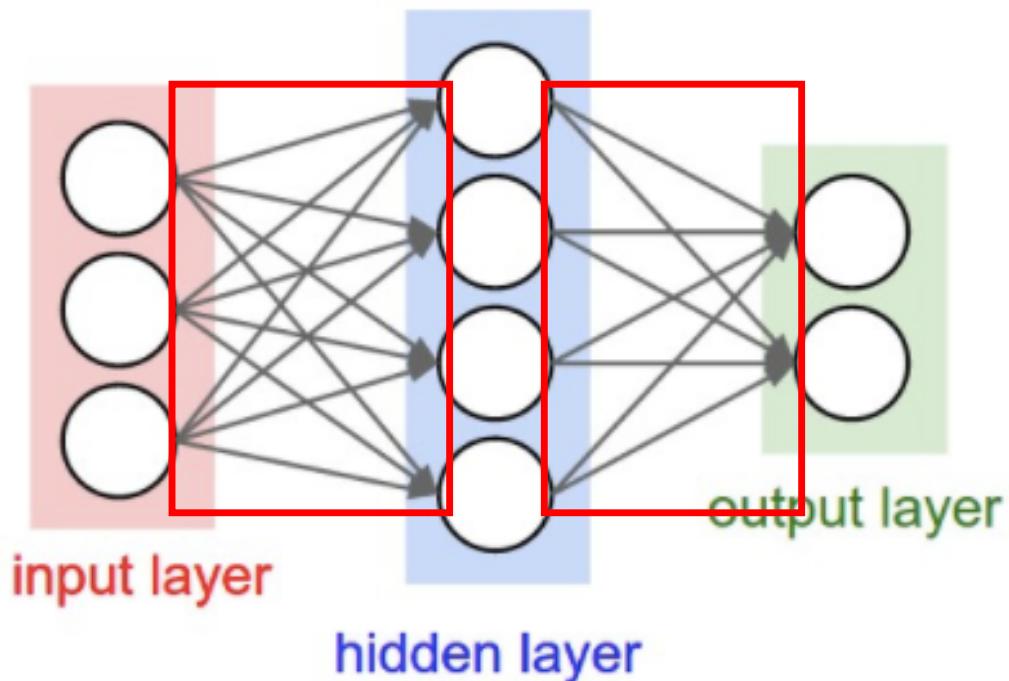


- How does this relate to a perceptron?



- **Training goal: learn model weights**

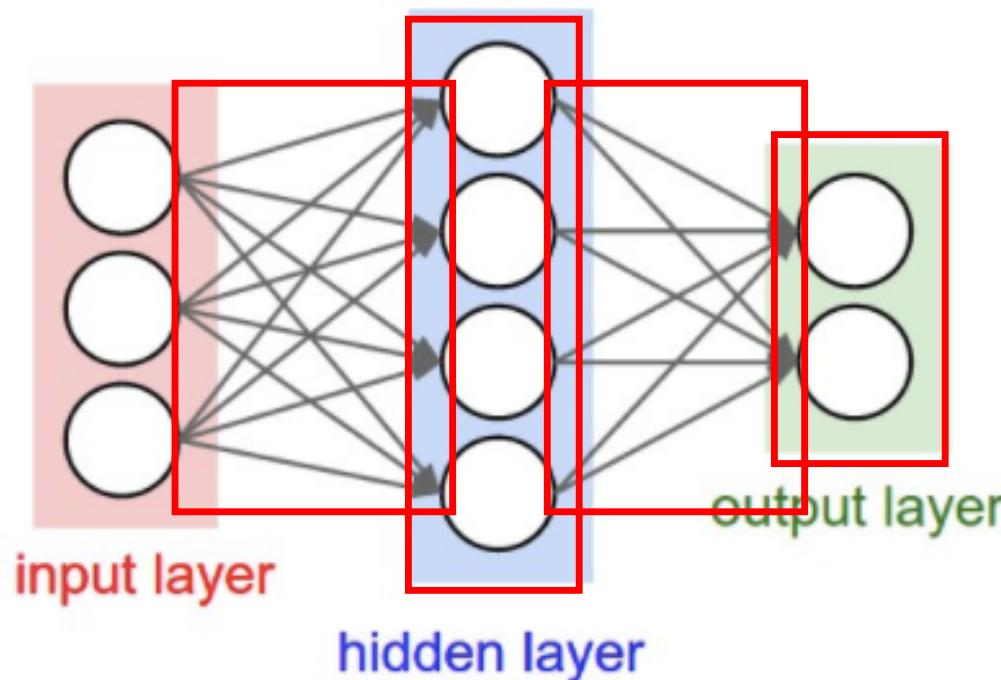
Neural Network



How many **weights** are in this model?

- Input to Hidden Layer:
 - $3 \times 4 = 12$
- Hidden Layer to Output Layer
 - $4 \times 2 = 8$
- Total:
 - $12 + 8 = 20$

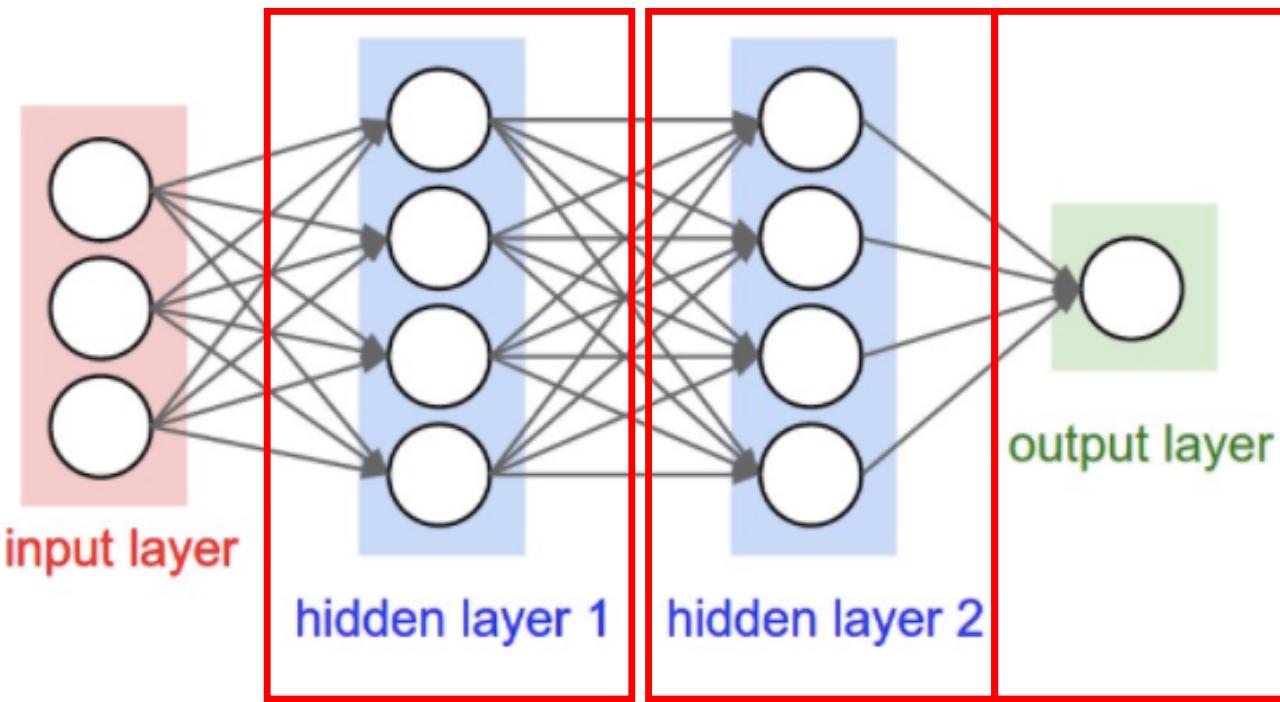
Neural Network



How many **parameters** are there to learn?

- Number of **weights**:
 - 20
- Number of **biases**:
 - $4 + 2 = 6$
- Total:
 - 26

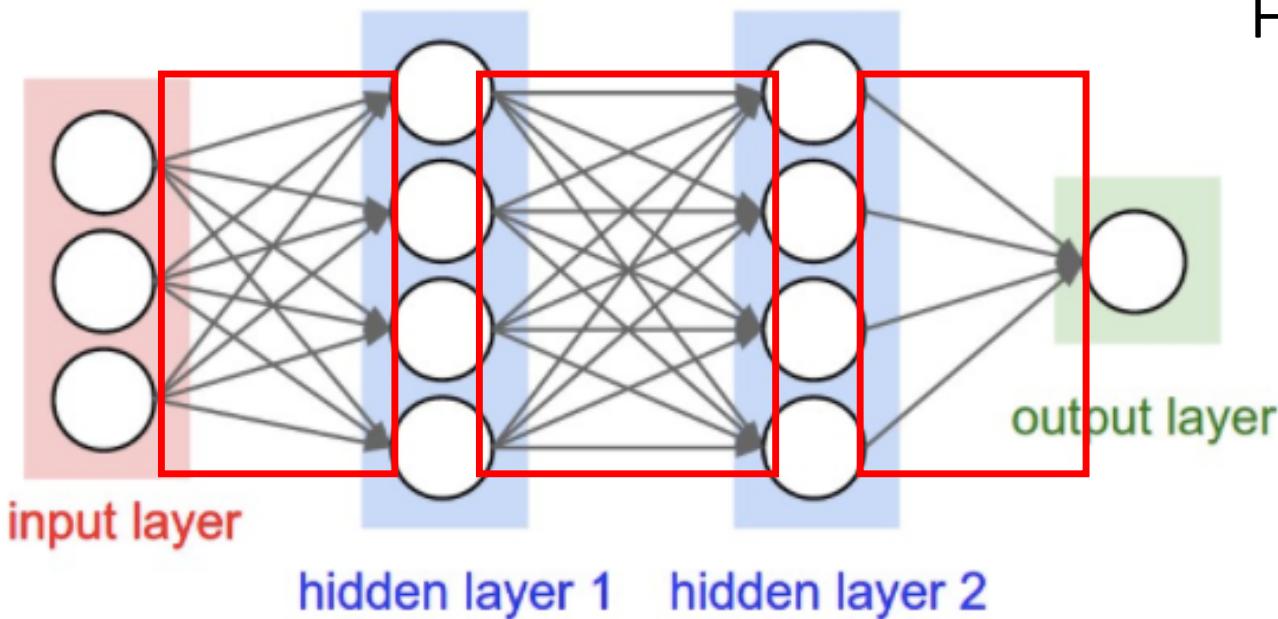
Neural Network



How many layers are in this network?

- 3 (number of hidden layers plus output layer; input layer excluded when counting)

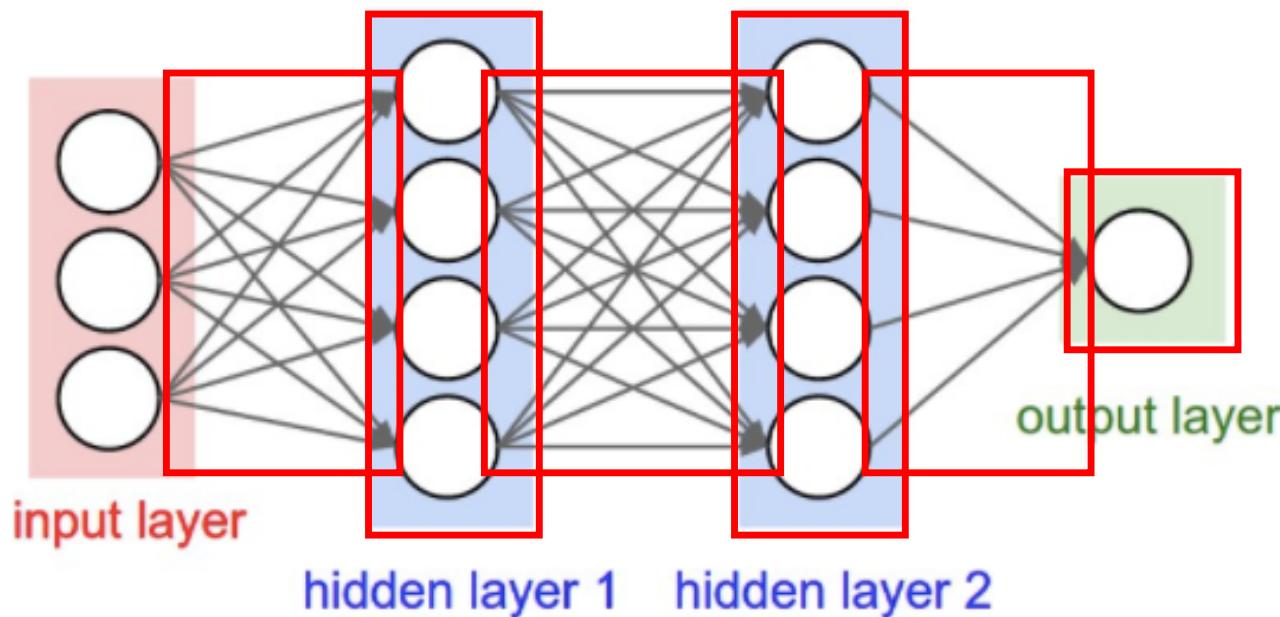
Neural Network



How many weights are in this model?

- Input to Hidden Layer 1:
 - $3 \times 4 = 12$
- Hidden Layer 1 to Hidden Layer 2:
 - $4 \times 4 = 16$
- Hidden Layer 2 to Output Layer
 - $4 \times 1 = 4$
- Total:
 - $12 + 16 + 4 = 32$

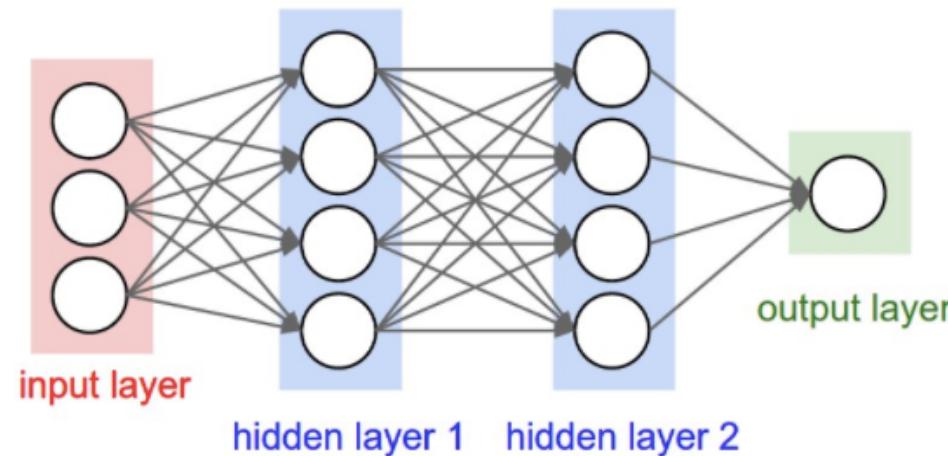
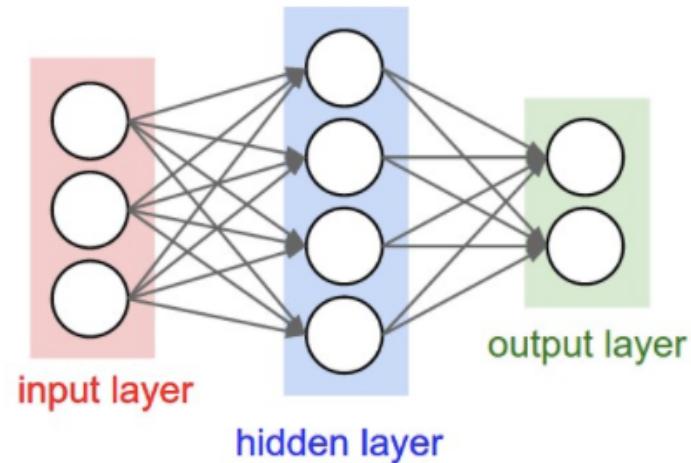
Neural Network



How many parameters are there to learn?

- Number of weights:
 - 32
- Number of biases:
 - $4 + 4 + 1 = 9$
- Total
 - 41

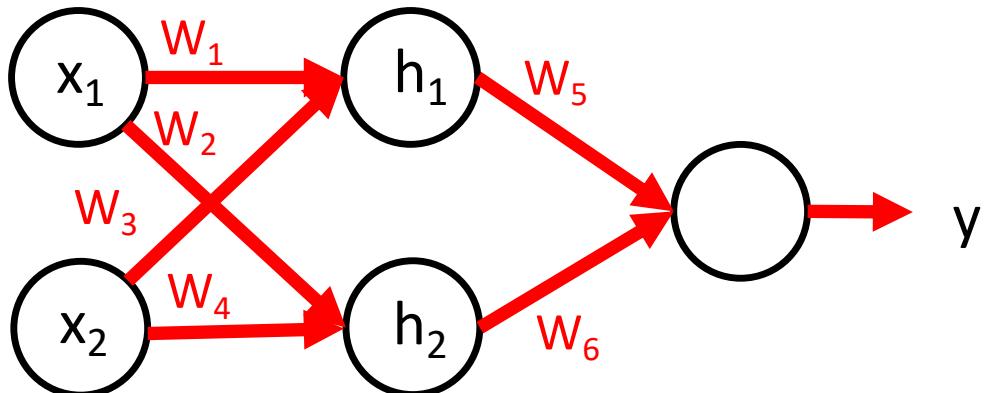
Fully Connected, Feed Forward Neural Networks



- What does it mean for a model to be fully connected?
 - Each unit provides input to each unit in the next layer
- What does it mean for a model to be feed forward?
 - Each layer serves as input to the next layer with no loops

Hidden Layers Alone Are NOT Enough to Model Non-Linear Functions

Key Observation: feedforward networks are just functions chained together
e.g.,

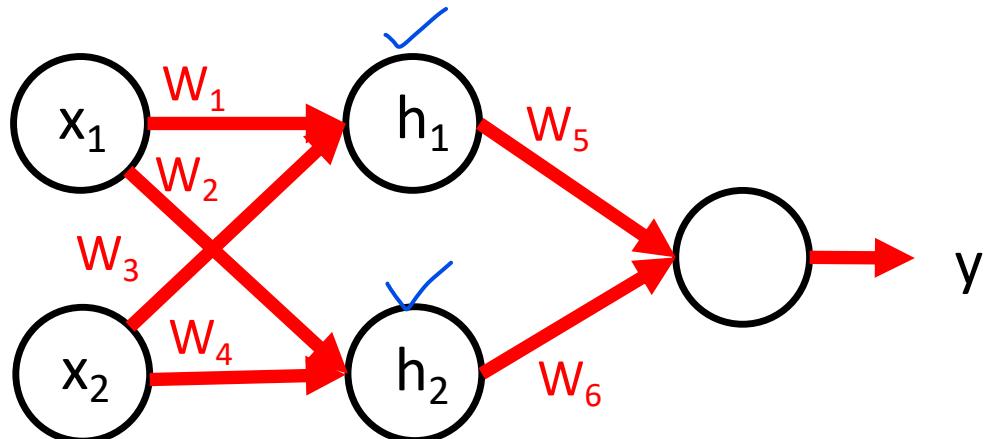


- What is function for h_1 ?
 - $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for h_2 ?
 - $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for y ?
 - $y = h_1w_5 + h_2w_6 + b_3$
 - $y = (w_1x_1 + w_3x_2 + b_1)w_5 + (w_2x_1 + w_4x_2 + b_2)w_6 + b_3$
 - $y = w_1w_5x_1 + w_3w_5x_2 + w_5b_1 + w_2w_6x_1 + w_4w_6x_2 + w_6b_2 + b_3$

A chain of LINEAR functions at any depth is still a LINEAR function!

Hidden Layers Alone Are NOT Enough to Model Non-Linear Functions

Key Observation: feedforward networks are just functions chained together
e.g.,



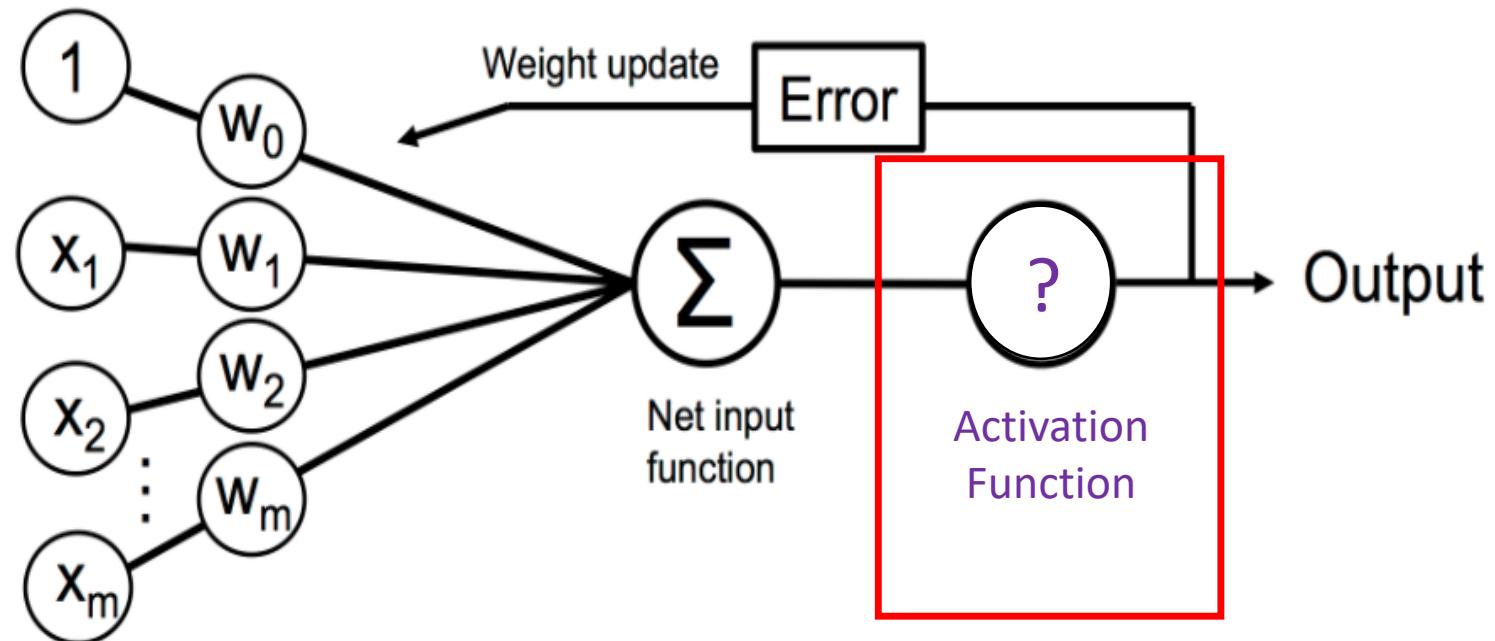
- What is function for h_1 ?
• $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for h_2 ?
• $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for y ?
• $y = h_1w_5 + h_2w_6 + b_3$

Constant \times linear function = linear function

A chain of LINEAR functions at any depth is still a LINEAR function!

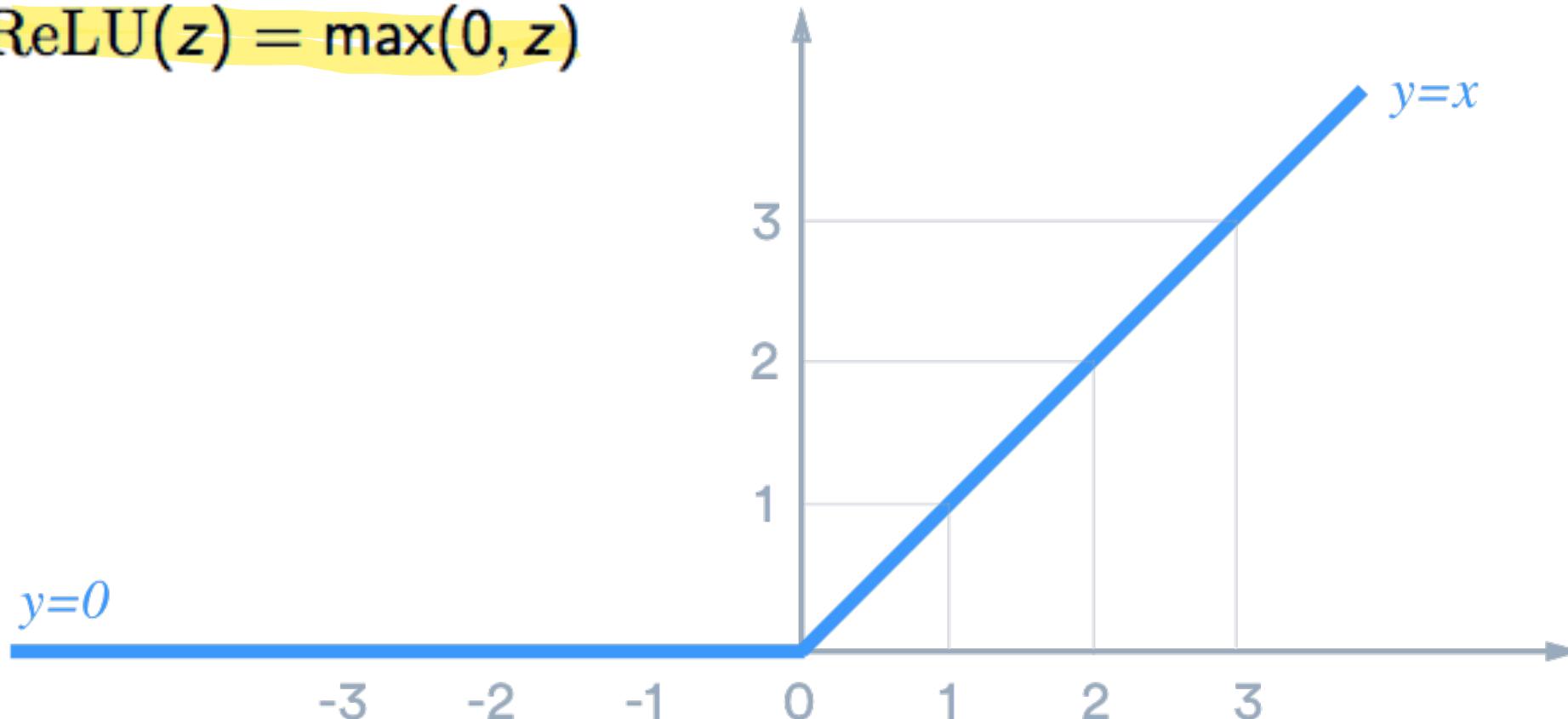
Solution to Model Non-Linear Functions: Non-Linear Activation Functions

- Each unit applies a non-linear “activation” function to the weighted input to mimic a neuron firing



Solution to Model Non-Linear Functions: Non-Linear Activation Functions

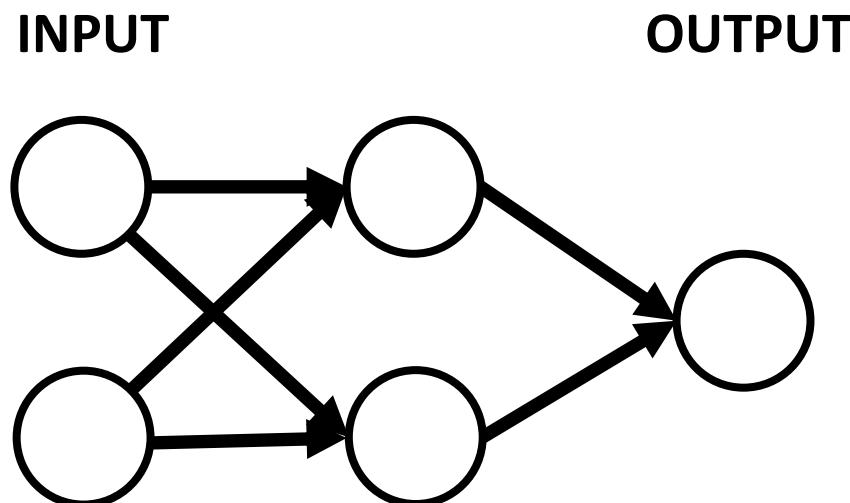
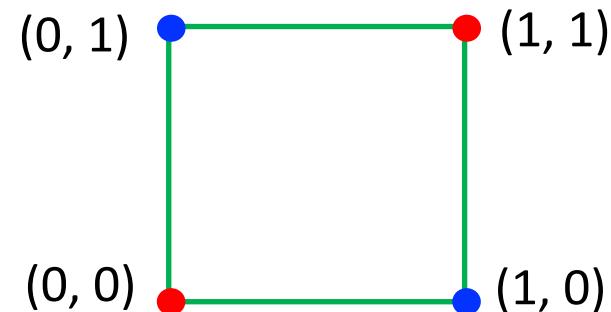
- e.g., $\text{ReLU}(z) = \max(0, z)$





Non-Linear Example: Revisiting XOR problem

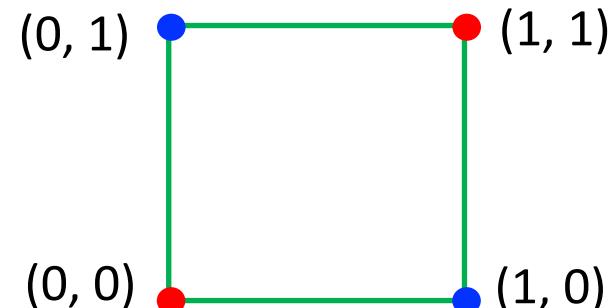
- Non-linear function: separate 1s from 0s:



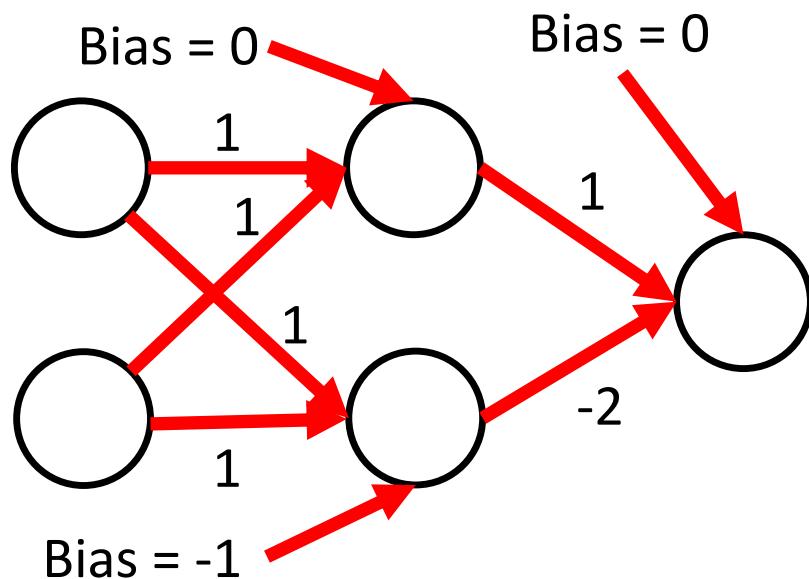
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



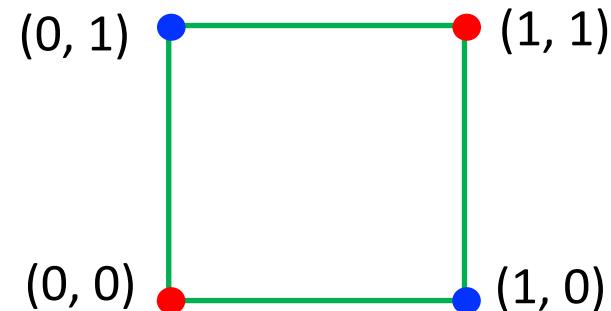
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



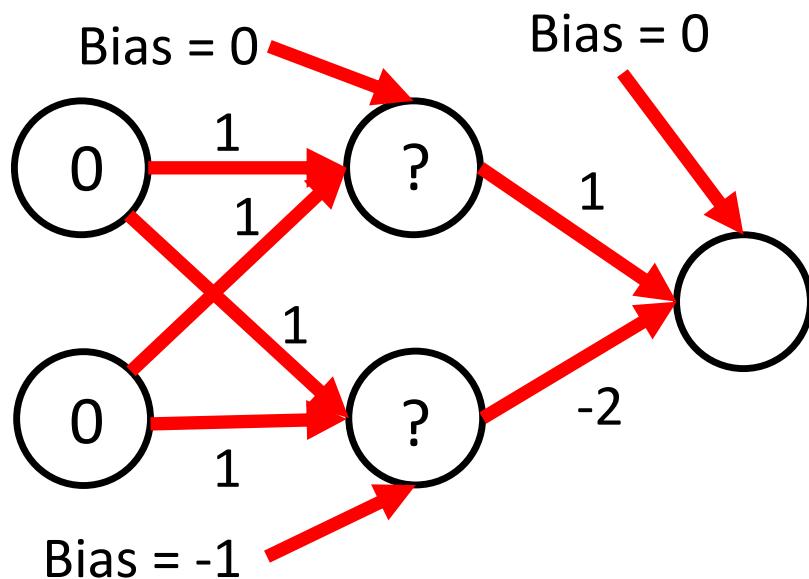
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



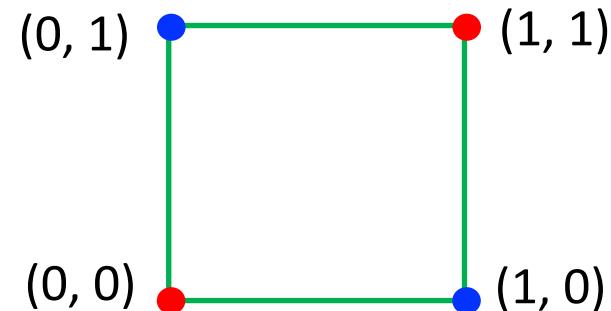
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



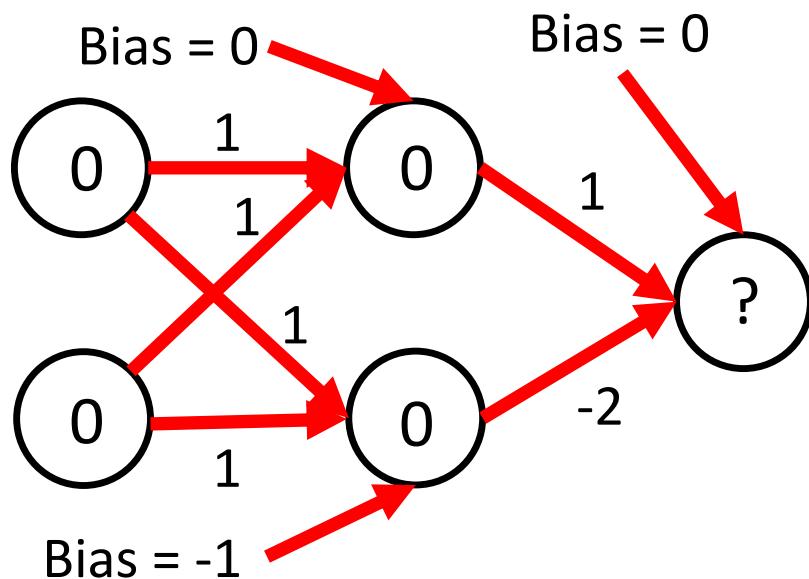
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



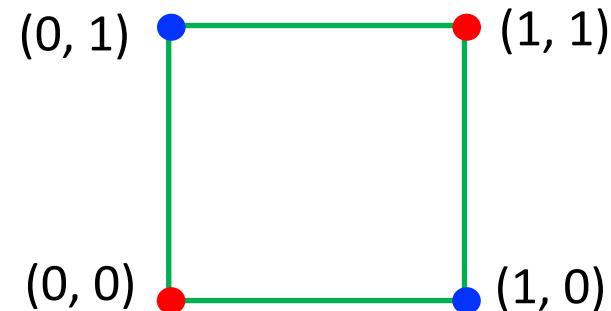
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



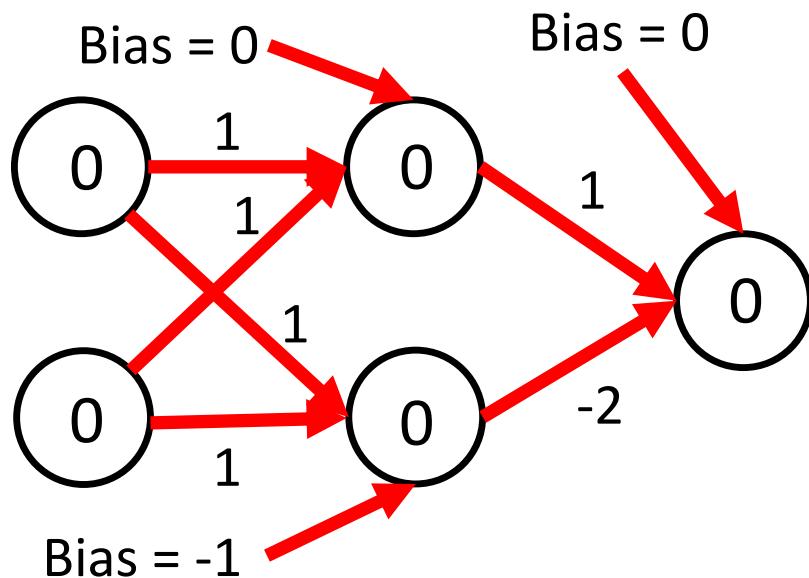
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



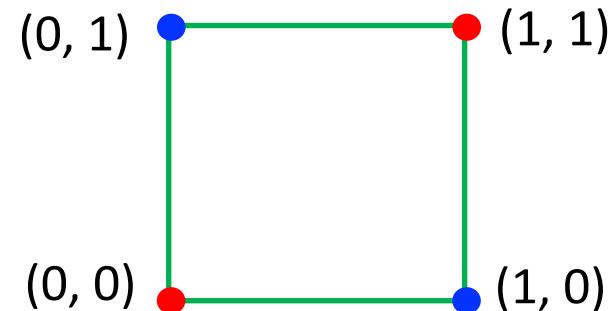
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



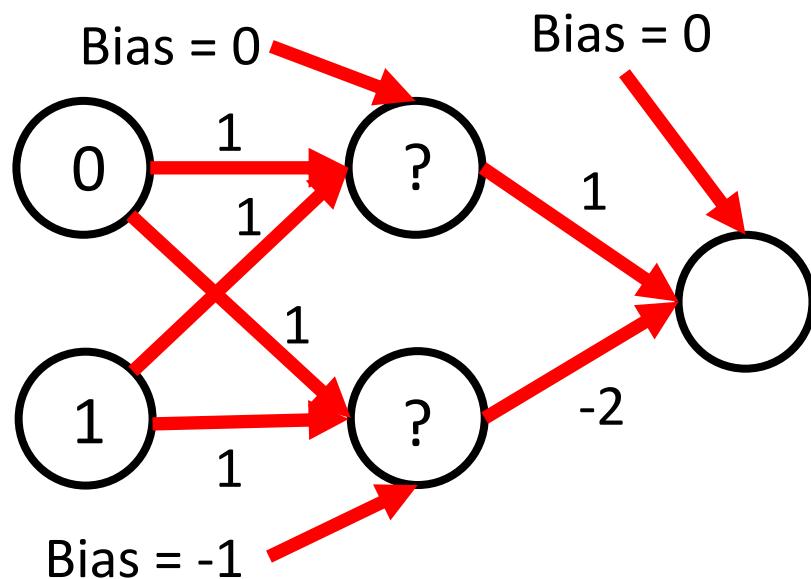
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



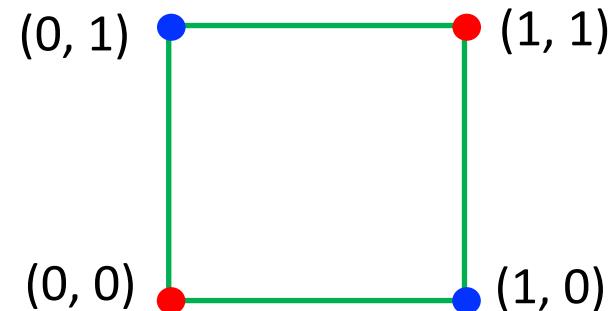
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



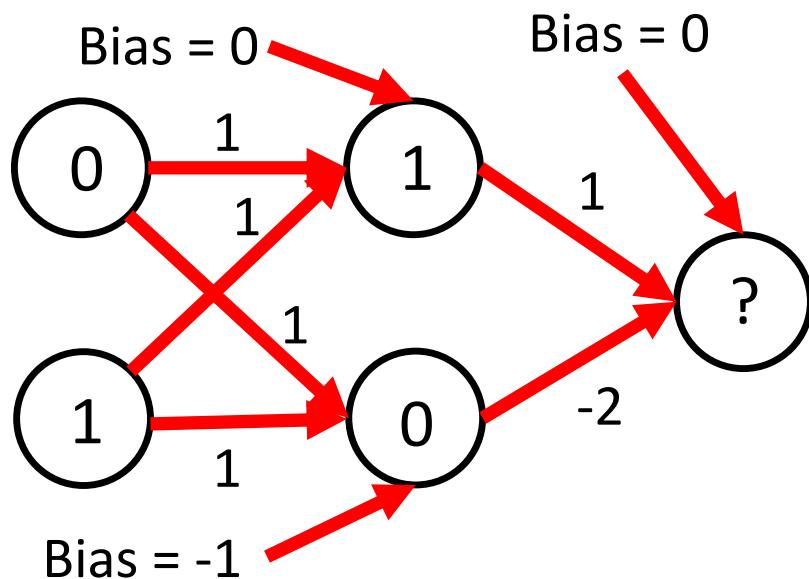
INPUT		OUTPUT	
A	B	A XOR B	
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



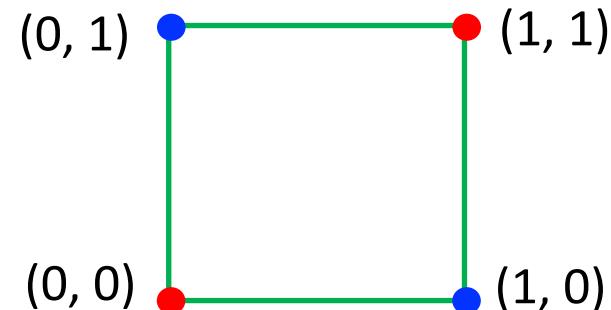
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



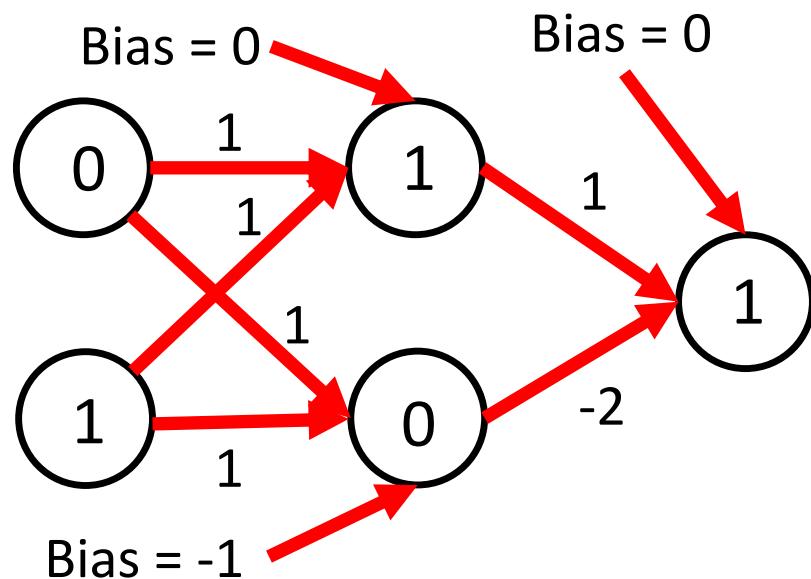
INPUT	OUTPUT	
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



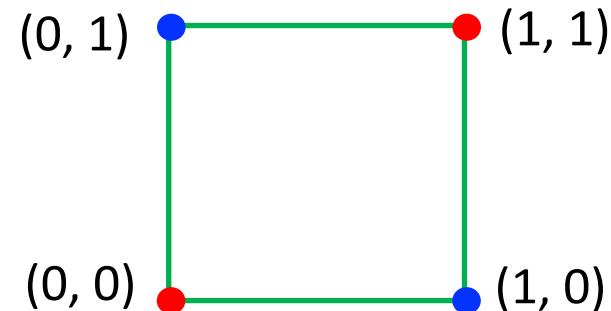
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



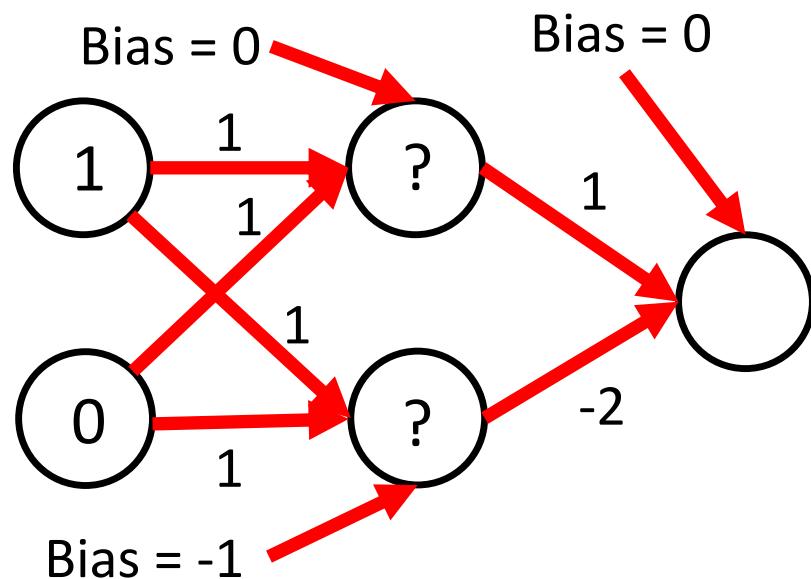
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



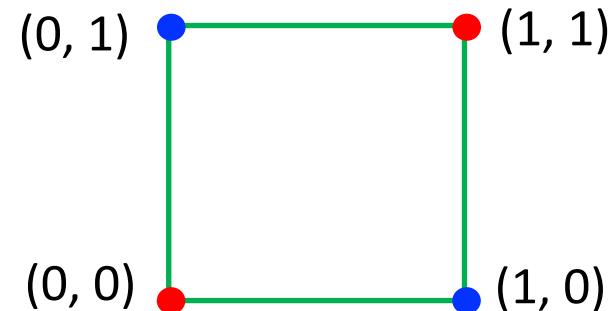
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



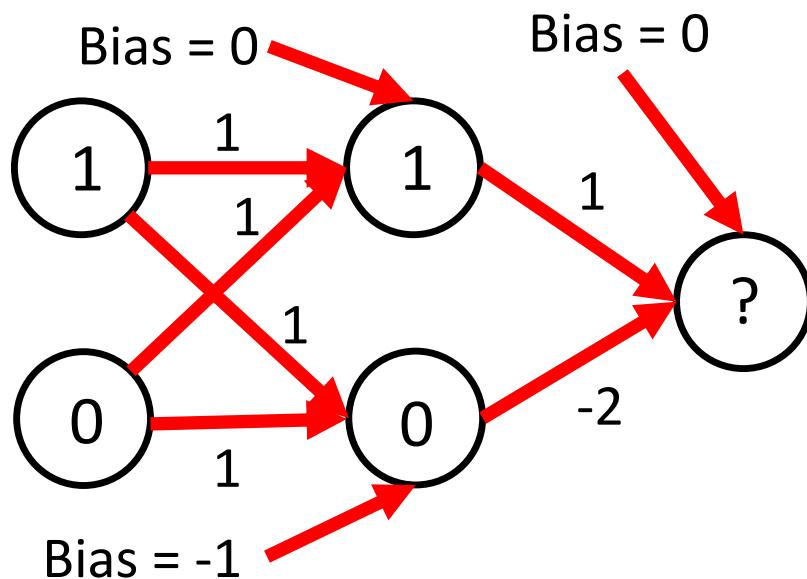
INPUT	OUTPUT	
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



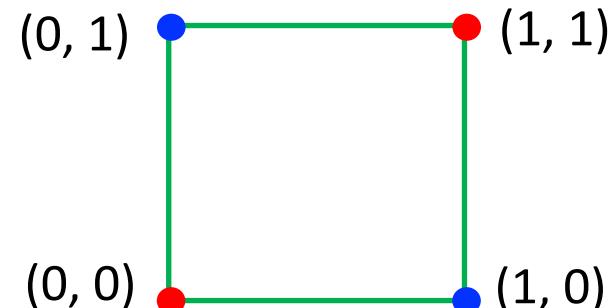
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



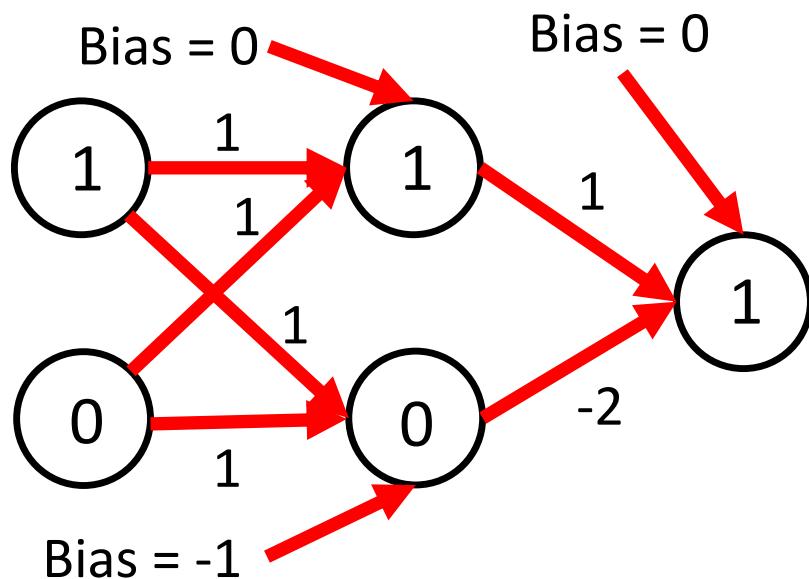
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



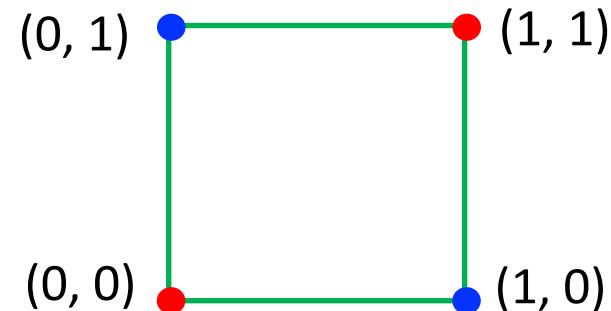
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



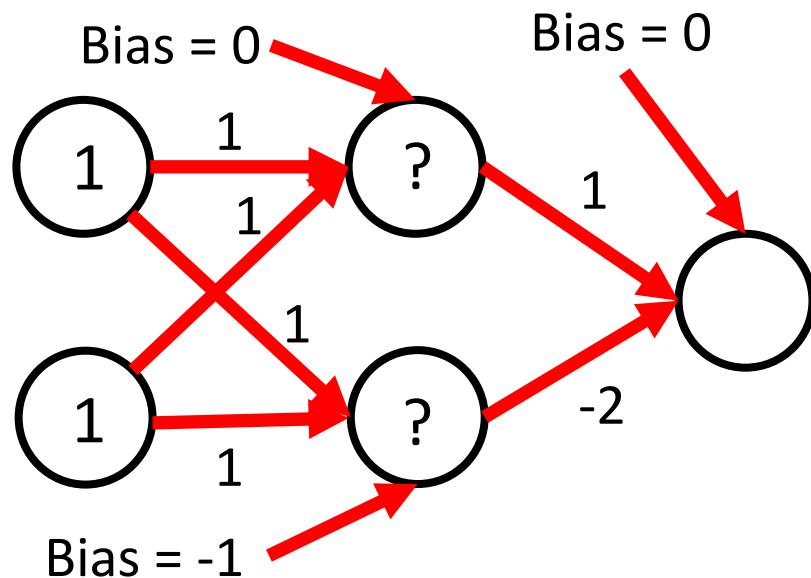
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



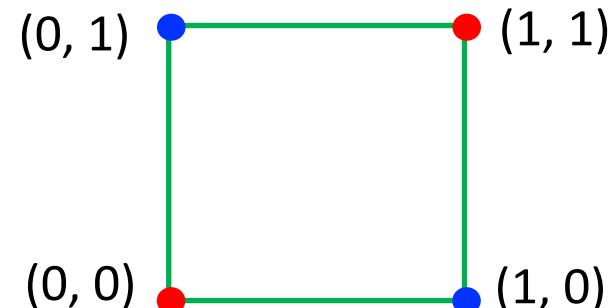
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



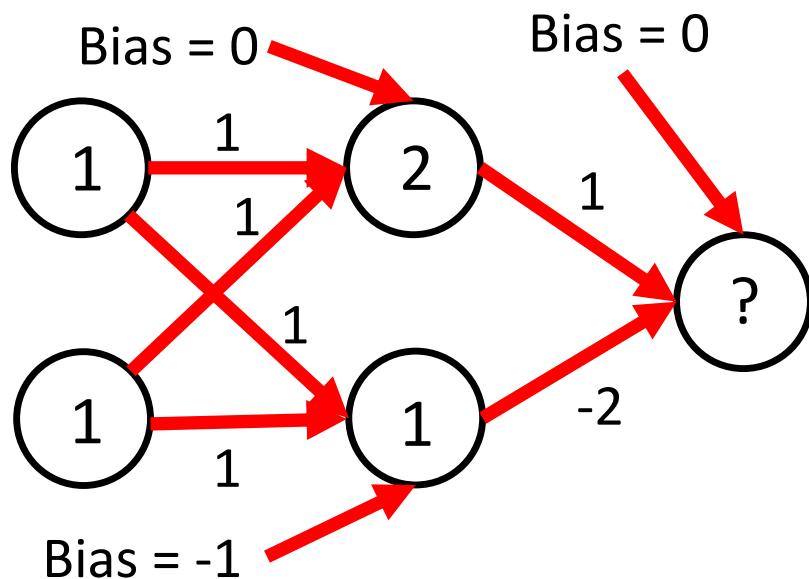
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



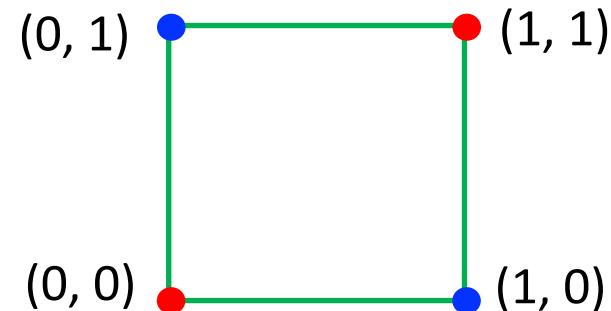
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



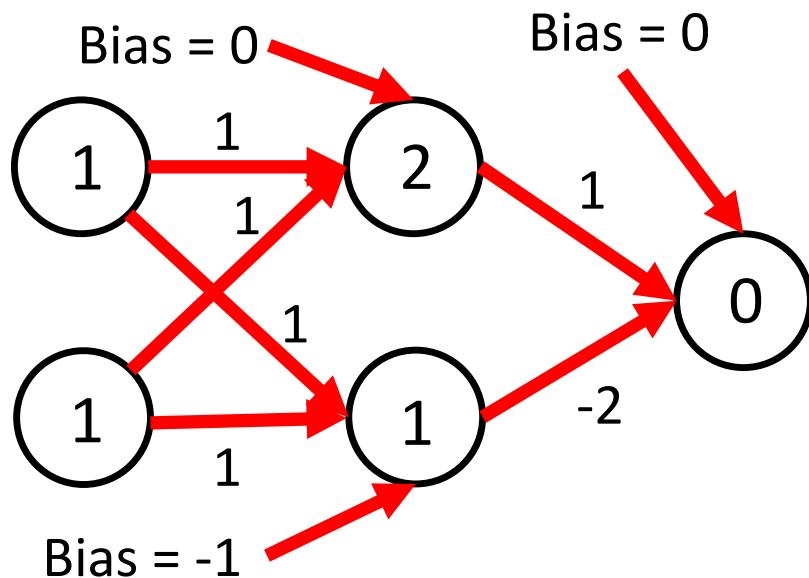
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



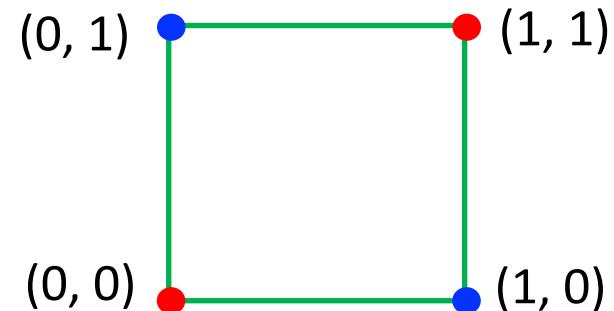
- Approach: ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with these parameters:



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

- Non-linear function: separate 1s from 0s:

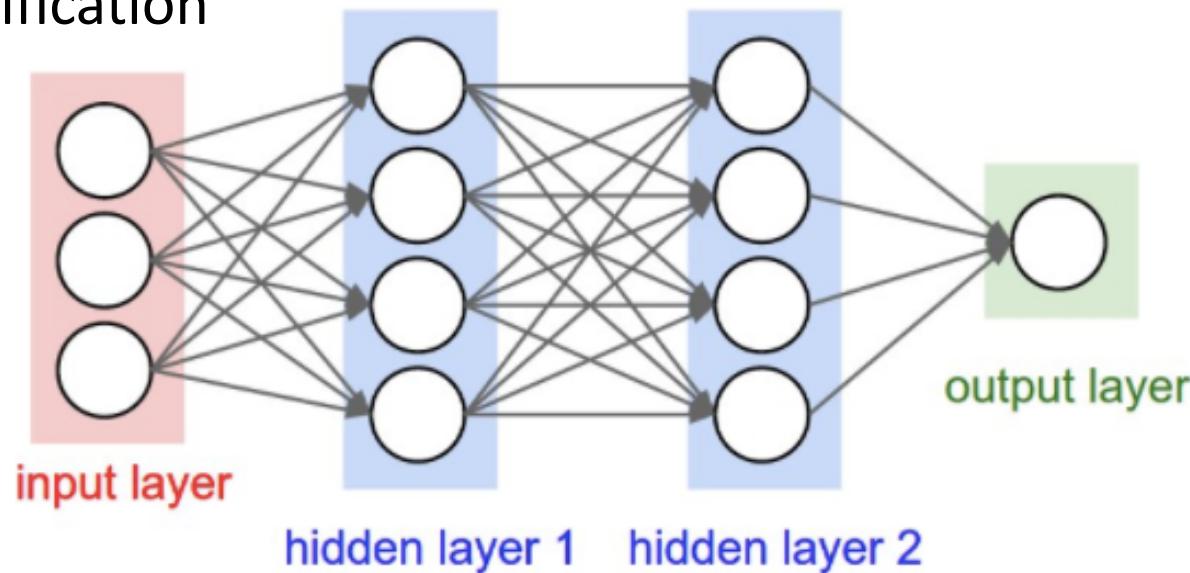


- Approach: Use ReLU activation function ($\text{ReLU}(z) = \max(0, z)$) with this model:

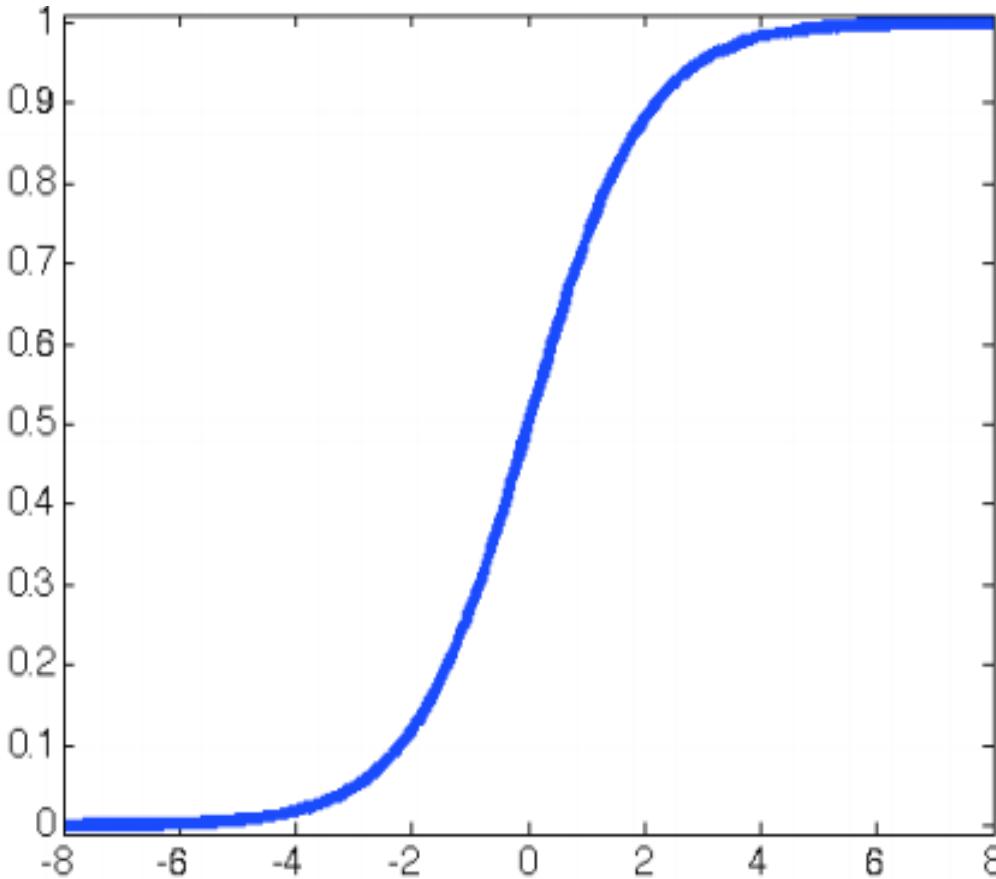
Neural networks can solve XOR problem...
and so model non-linear functions!

Output Units

- Matches the neural network to the task it must perform; e.g.,
 - Linear regression
 - Binary classification
 - Multi-class classification
 - Multi-label classification



Sigmoid (for Binary Classification)



$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

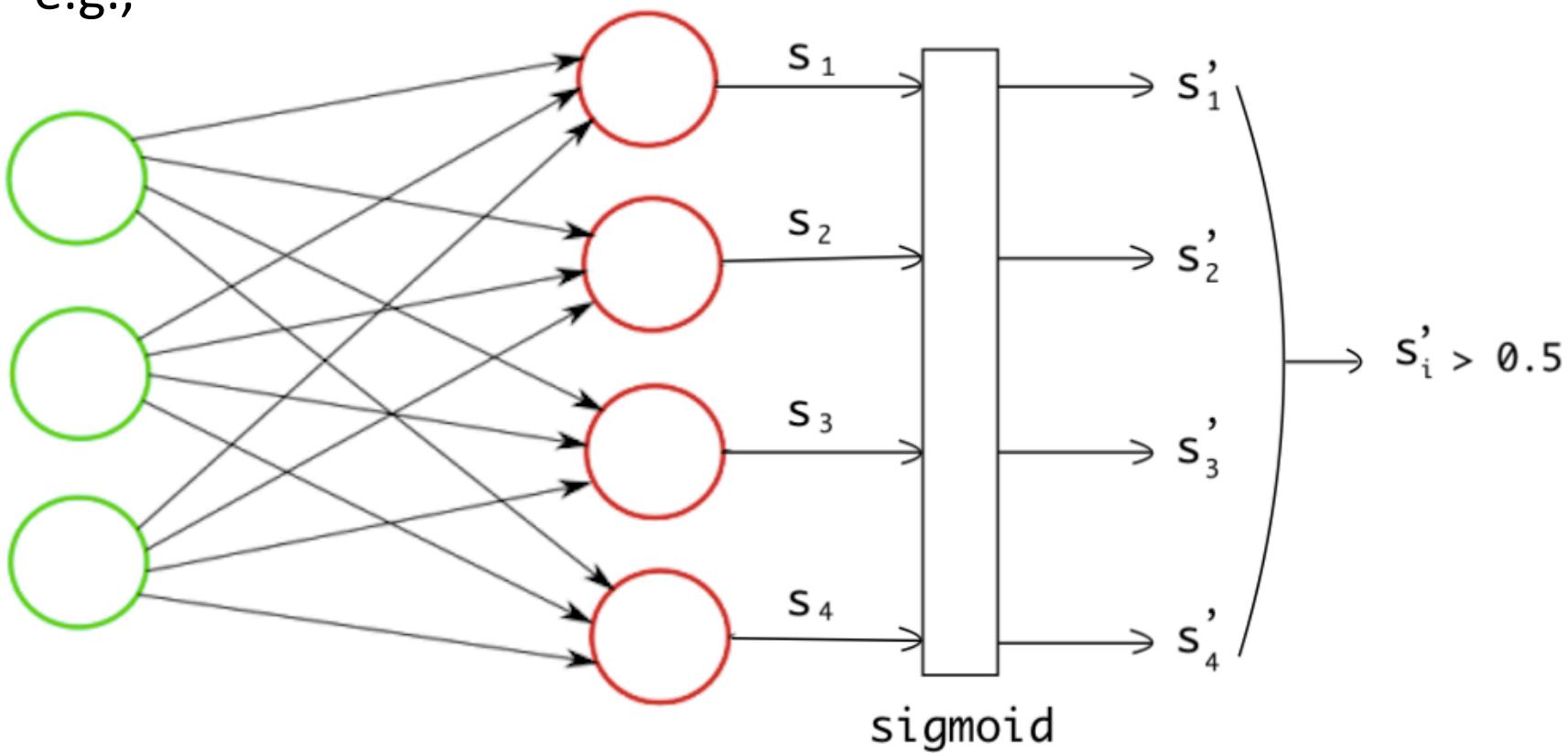
If ≥ 0.5 , output 1;

Else, outputs 0

Sigmoid (for Multilabel Classification)

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

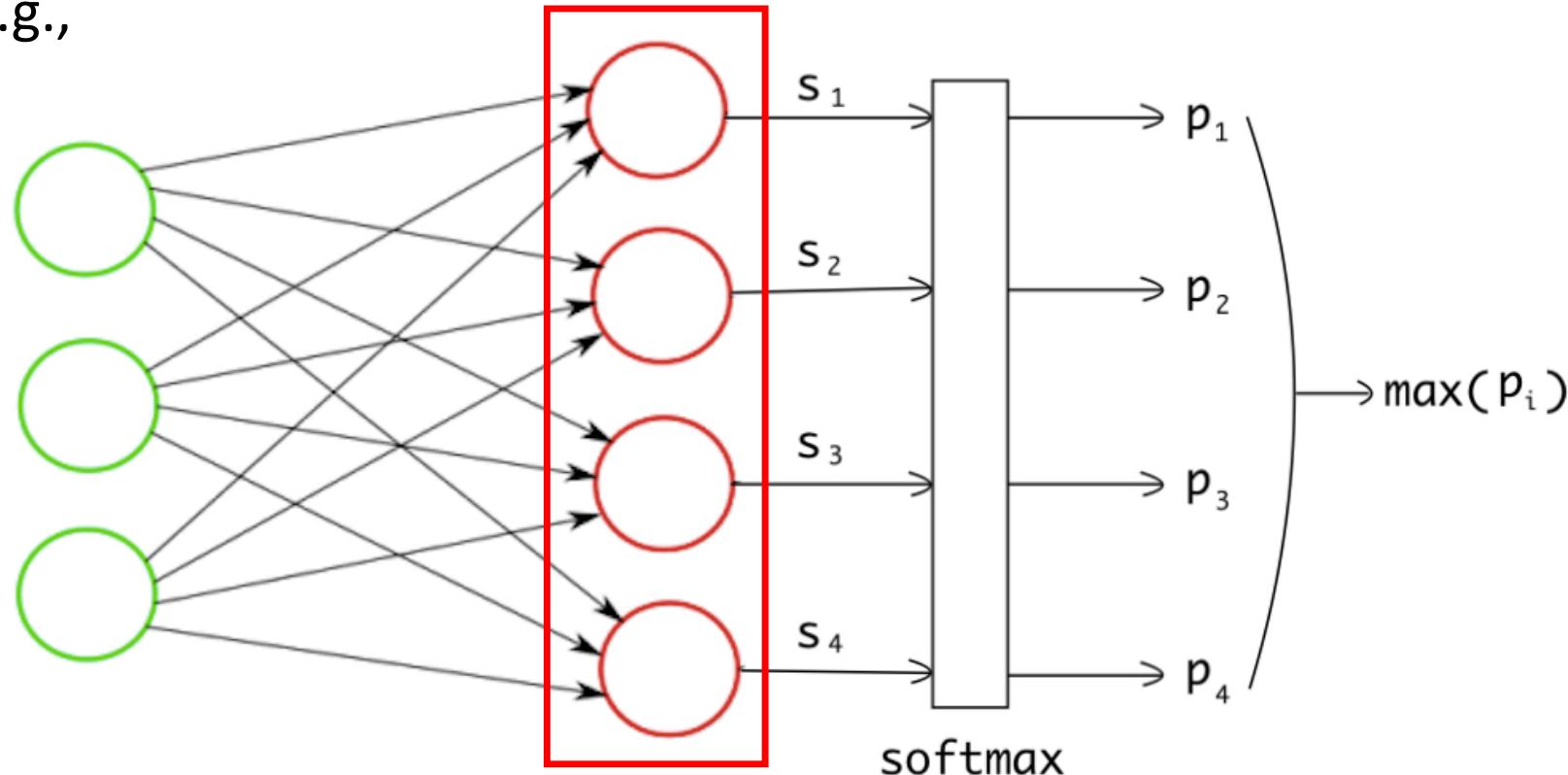
e.g.,



Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
 - e.g.,

$$\phi_{\text{softmax}}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$



Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
 - e.g.,

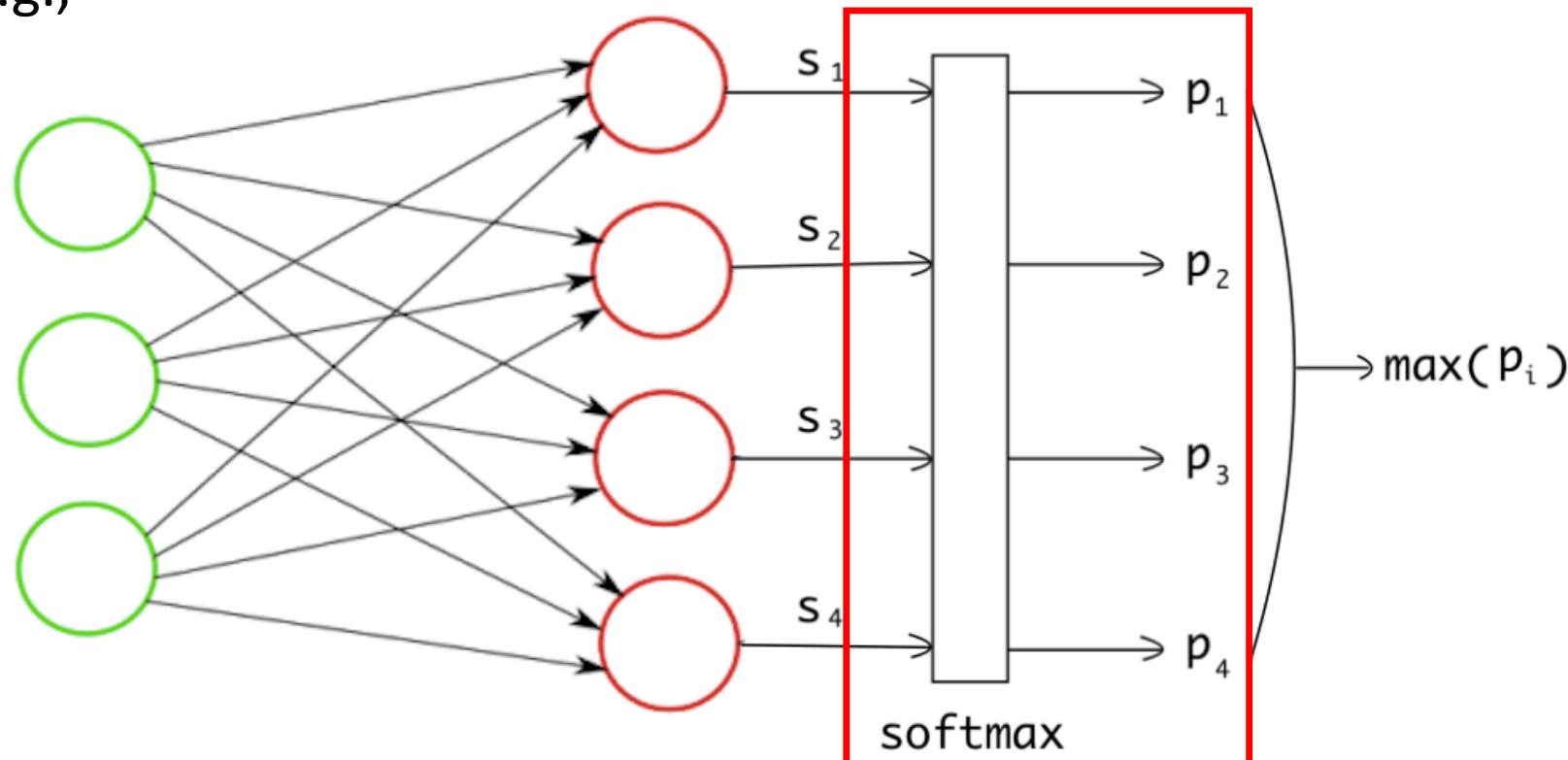
$$\phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$

	Scoring Function
Dog	-3.44
Cat	1.16
Boat	-0.81
Airplane	3.91

Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
 - e.g.,

$$\phi_{\text{softmax}}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$



Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that **sums to 1**:
 - e.g.,

$$\phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$

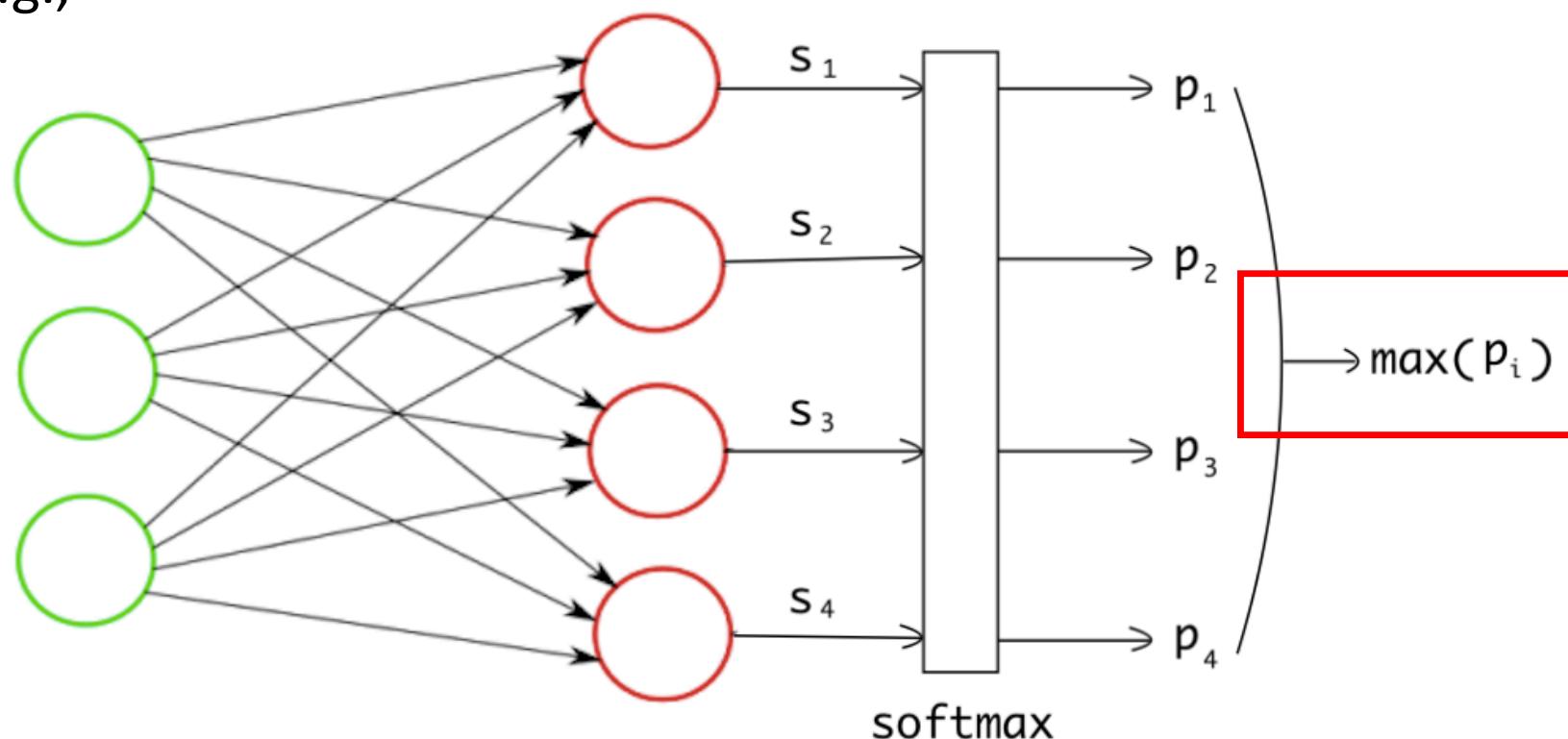
Normalization

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315

Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
 - e.g.,

$$\phi_{\text{softmax}}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$



Softmax (for Multiclass Classification)

- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
 - e.g.,

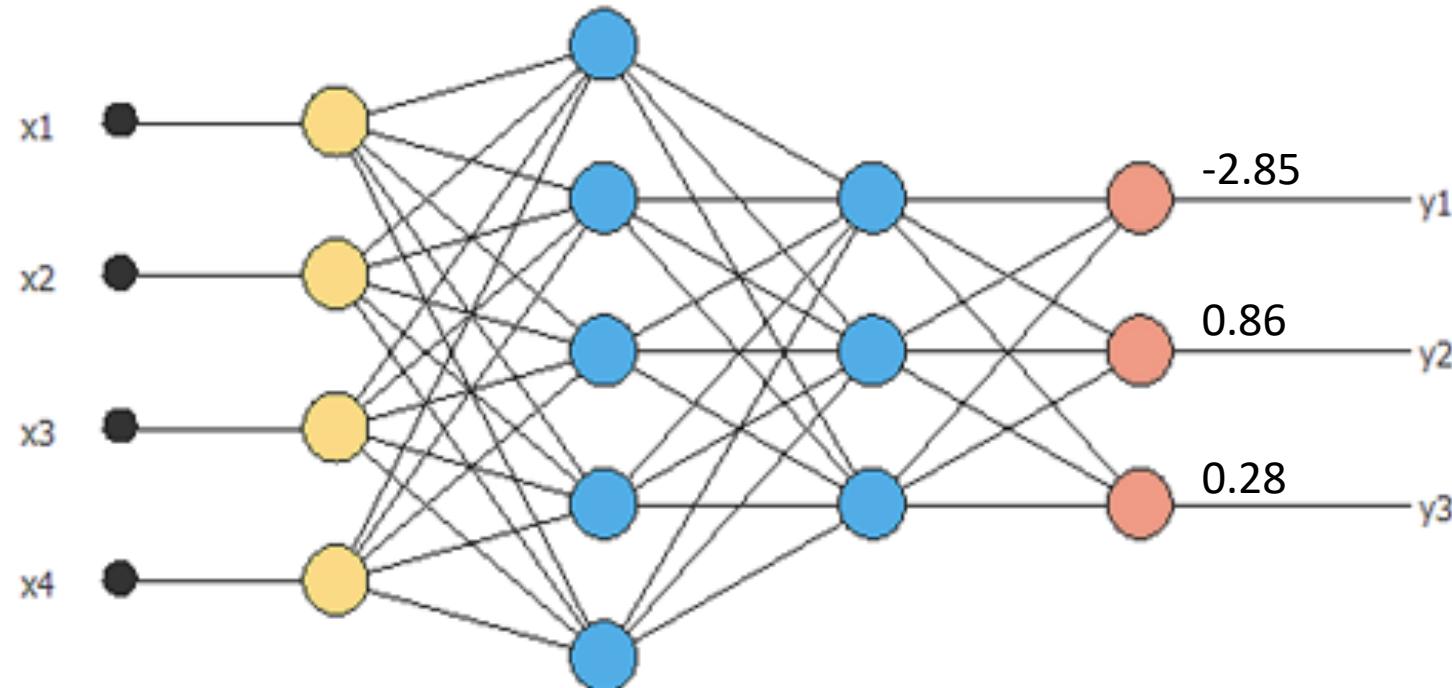
$$\phi_{\text{softmax}}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}$$

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315



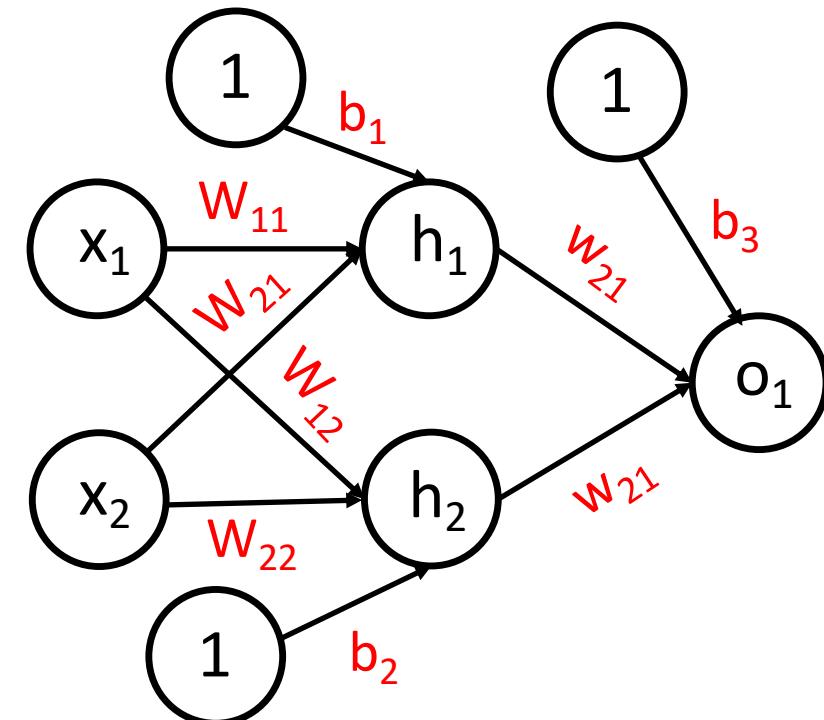
Group Discussion Questions

- How many model parameters must be learned for the network below?
- Assuming you apply a sigmoid function at the final layer with the output values specified below, which label(s) will be classified as present versus not?
- What label will be classified if you instead apply a softmax function to the output values?



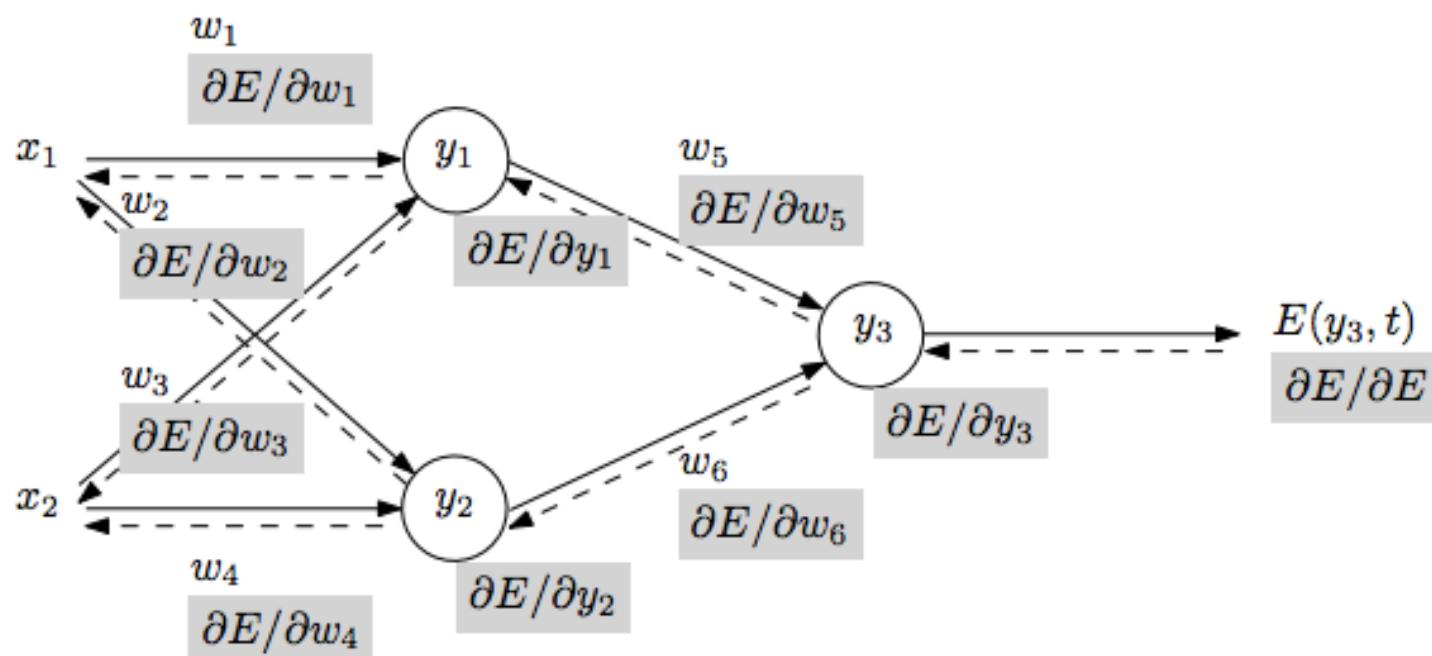
Recall: What to Learn in Neural Network?

- Learn:
 - weights connecting units
 - bias for each unit
- e.g., 2 layer neural network:
- Algorithm decides how to use each layer to produce the output; for this reason, layers are called “hidden”



✓ Neural Network: How to Learn?

(a) Forward pass



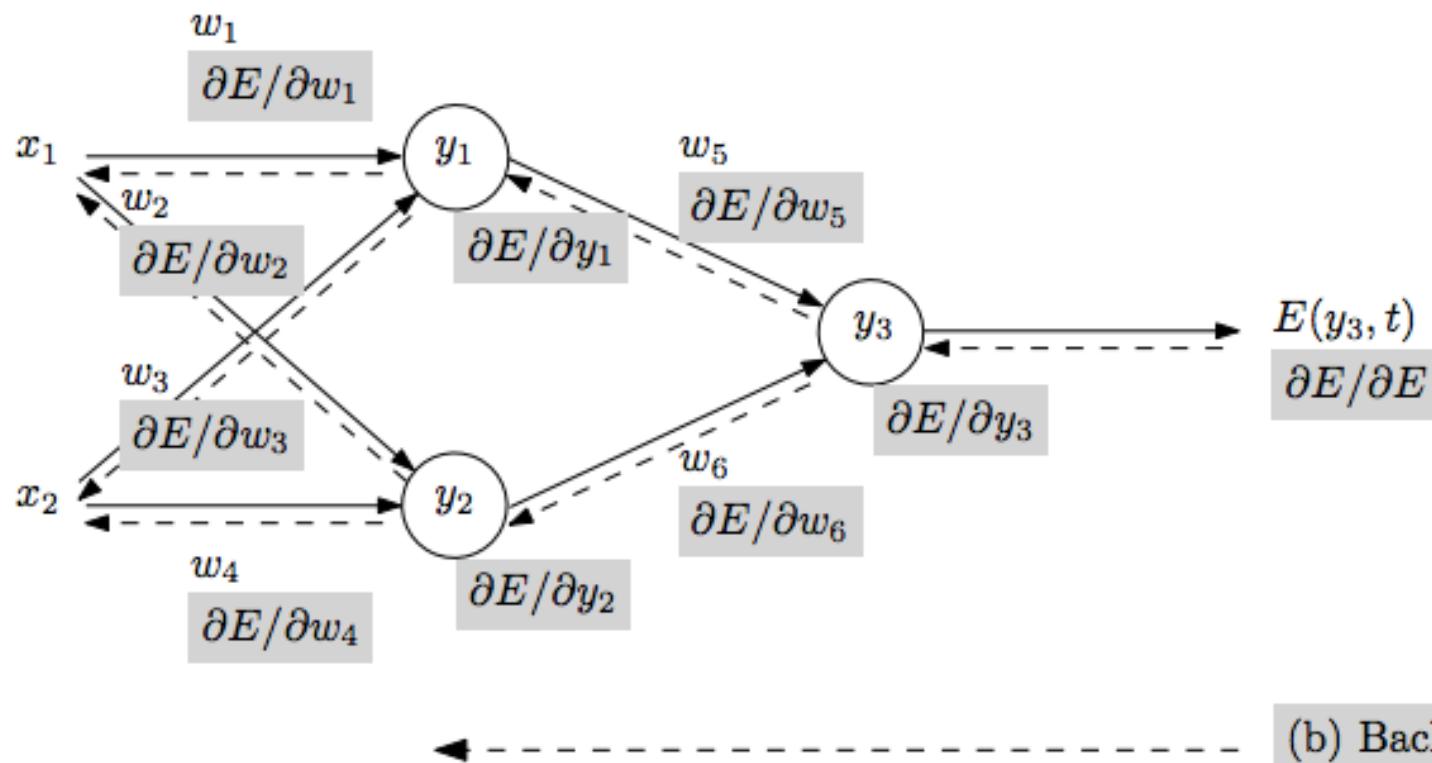
- Repeat until stopping criterion met:

(b) Backward pass

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass



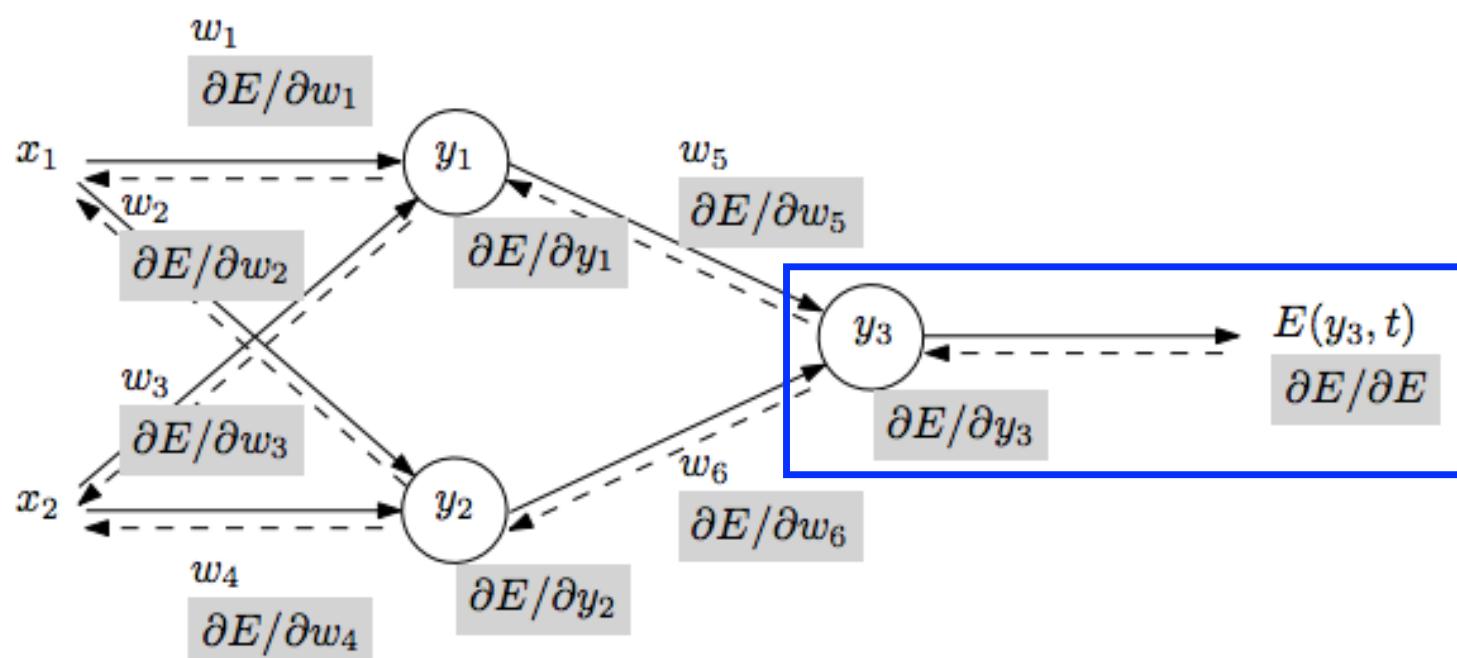
- Repeat until stopping criterion met:

1. **Forward pass:** propagate training data through network to make prediction

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass



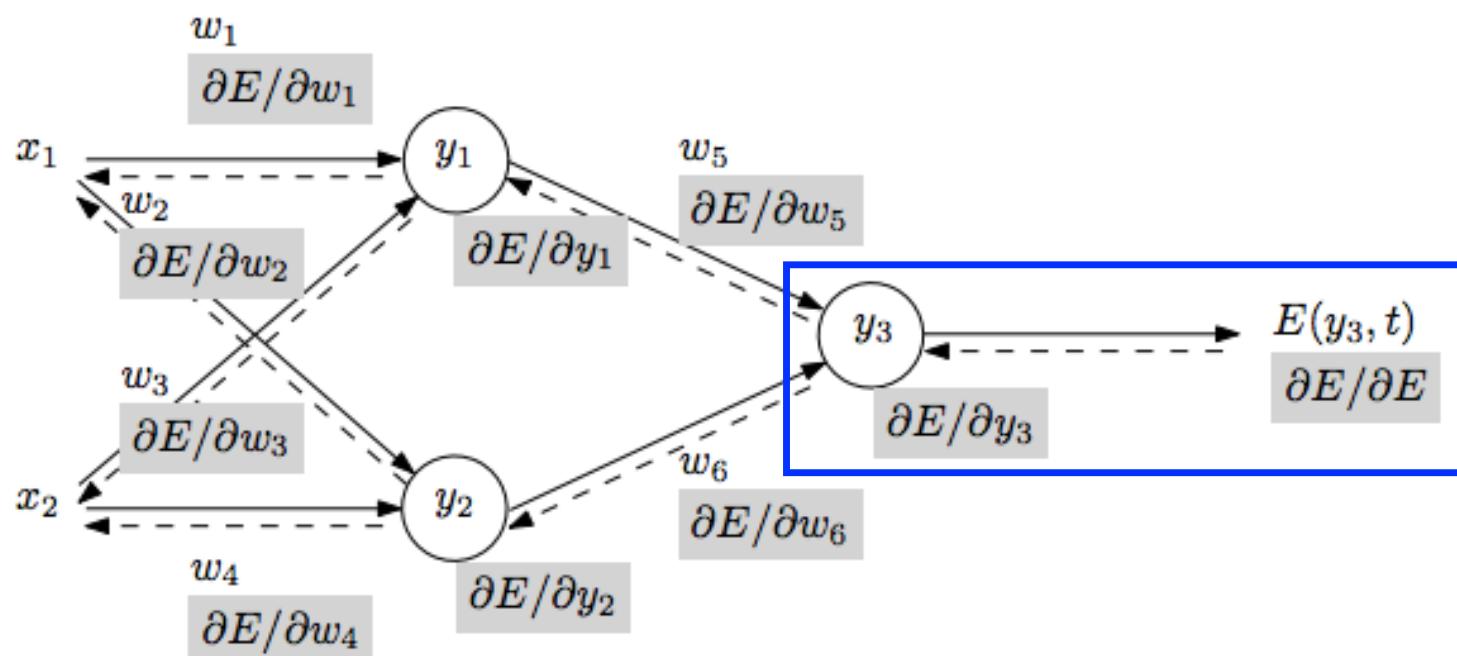
(b) Backward pass

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass



(b) Backward pass

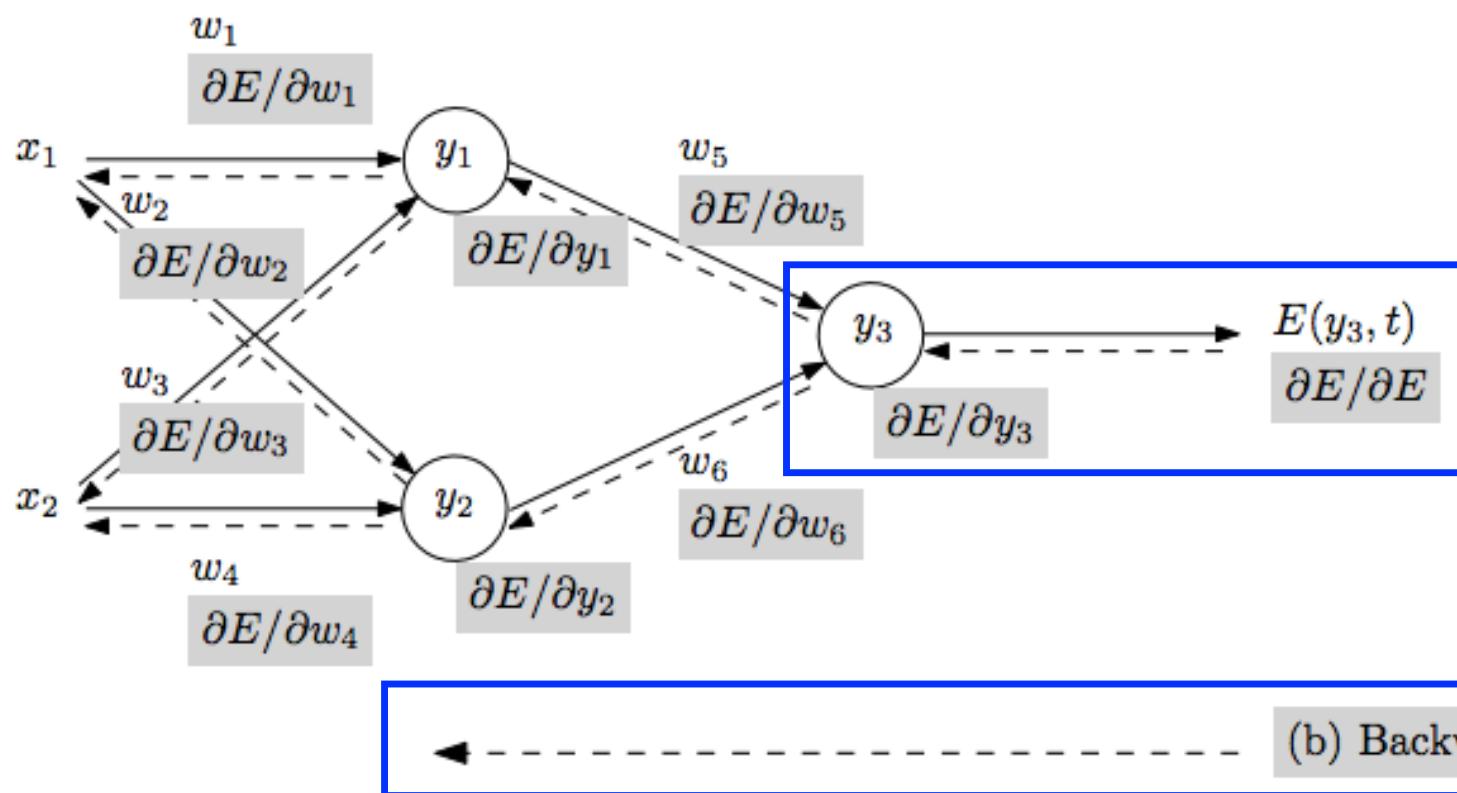
1. Define “loss” function, which quantifies the model’s errors on training data

e.g., $(\text{Predicted} - \text{Actual})^2$

- Note it is a function of the **weights** in the network

Neural Network: How to Learn?

(a) Forward pass

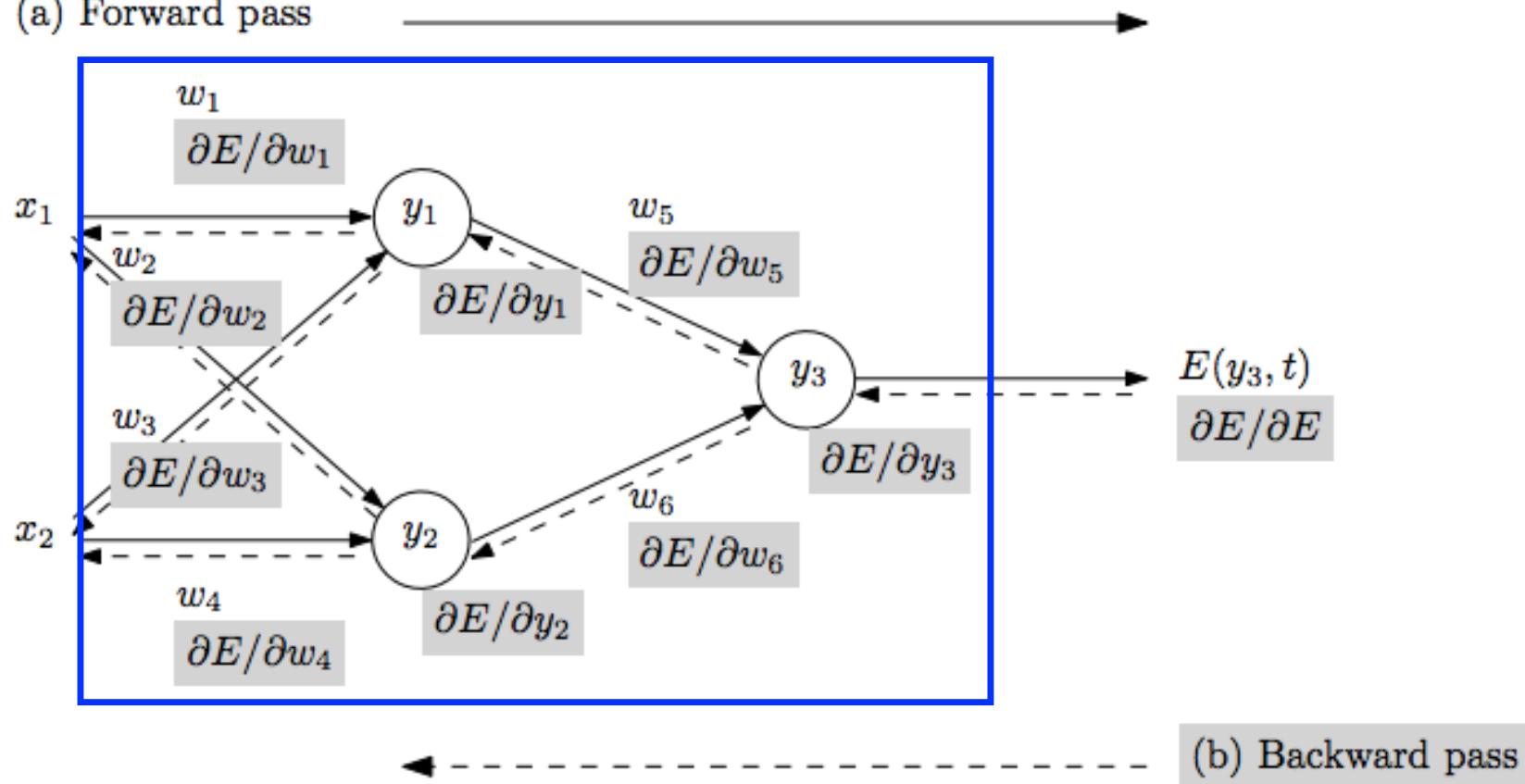


2. Backpropagation calculates gradient of the loss function with respect to the neural network's **weights** to propagate error backwards

- From output of network to input, it measures the error contribution of each connection

Neural Network: How to Learn?

(a) Forward pass

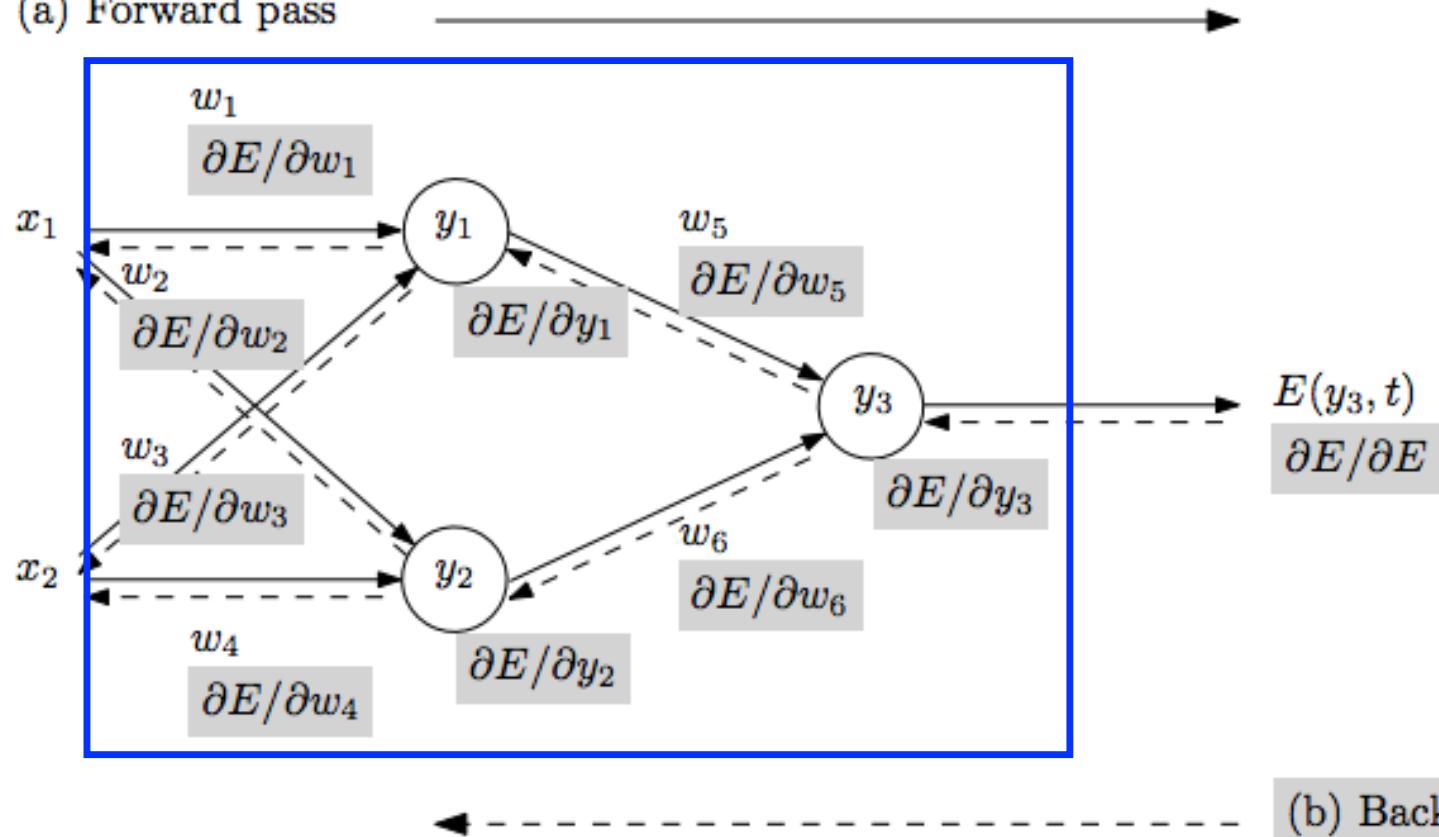


- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward
 3. **Update each weight** using calculated gradients

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass



- Update weights by taking an opposite step towards the gradient for each layer

$$W^{(l)} = W^{(l)} - \eta \Delta^{(l)}$$

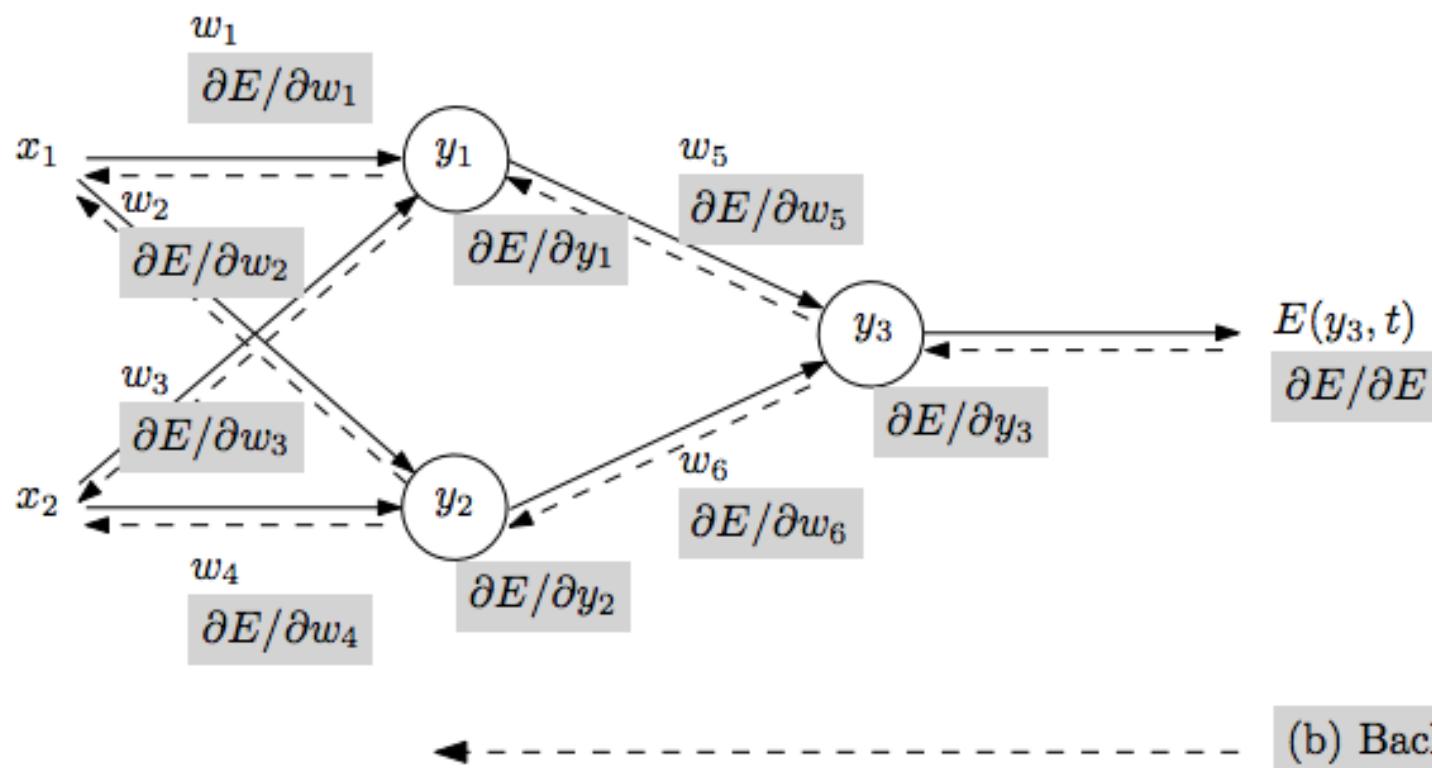
learning rate

(b) Backward pass

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass



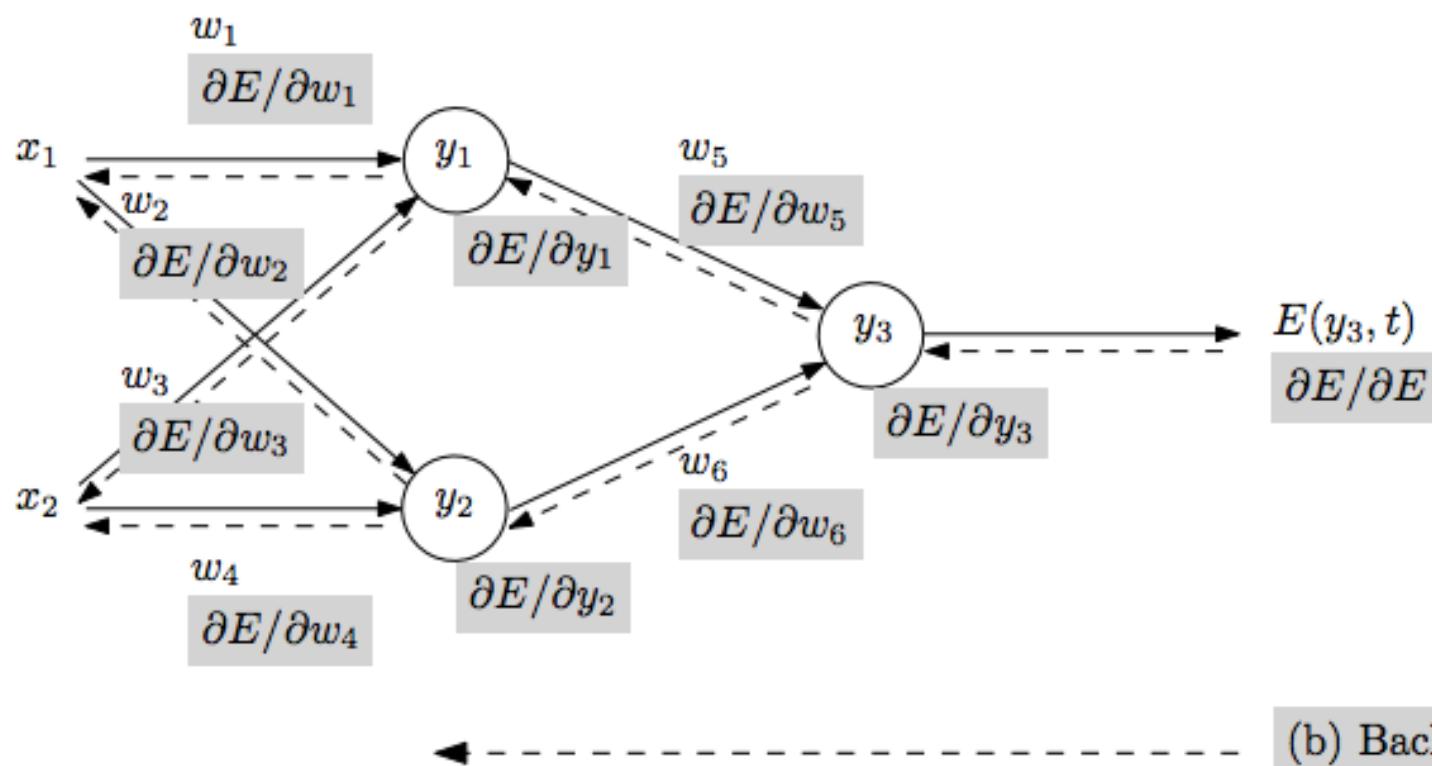
(b) Backward pass

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward
 3. **Update each weight** using calculated gradients

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Neural Network: How to Learn?

(a) Forward pass

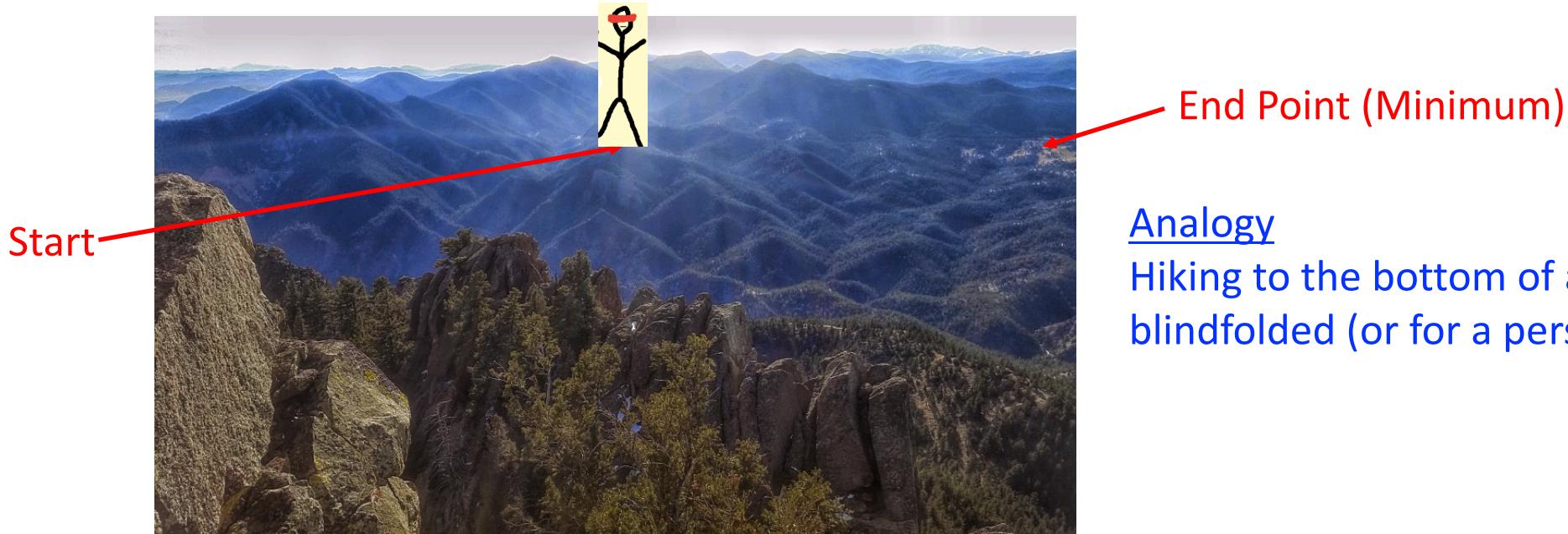


- What stopping criterion to use when training?
 - Weight changes are incredibly small
 - Percentage of misclassified example is below some threshold
 - Finished a pre-specified number of epochs
 - ...

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Gradient Descent Learning Approach

- Recall: solves mathematical problems by updating estimates of the solution via an iterative process to “optimize” a function
 - e.g., minimize or maximize an objective function $f(x)$ by altering x



End Point (Minimum)

Start

Analogy

Hiking to the bottom of a mountain range... blindfolded (or for a person who is blind)!

- When minimizing the objective function, it also is often called interchangeably the cost function, loss function, or error function.

Key Challenge: How to Compute Gradient?

Equation for calculating gradients depends on:

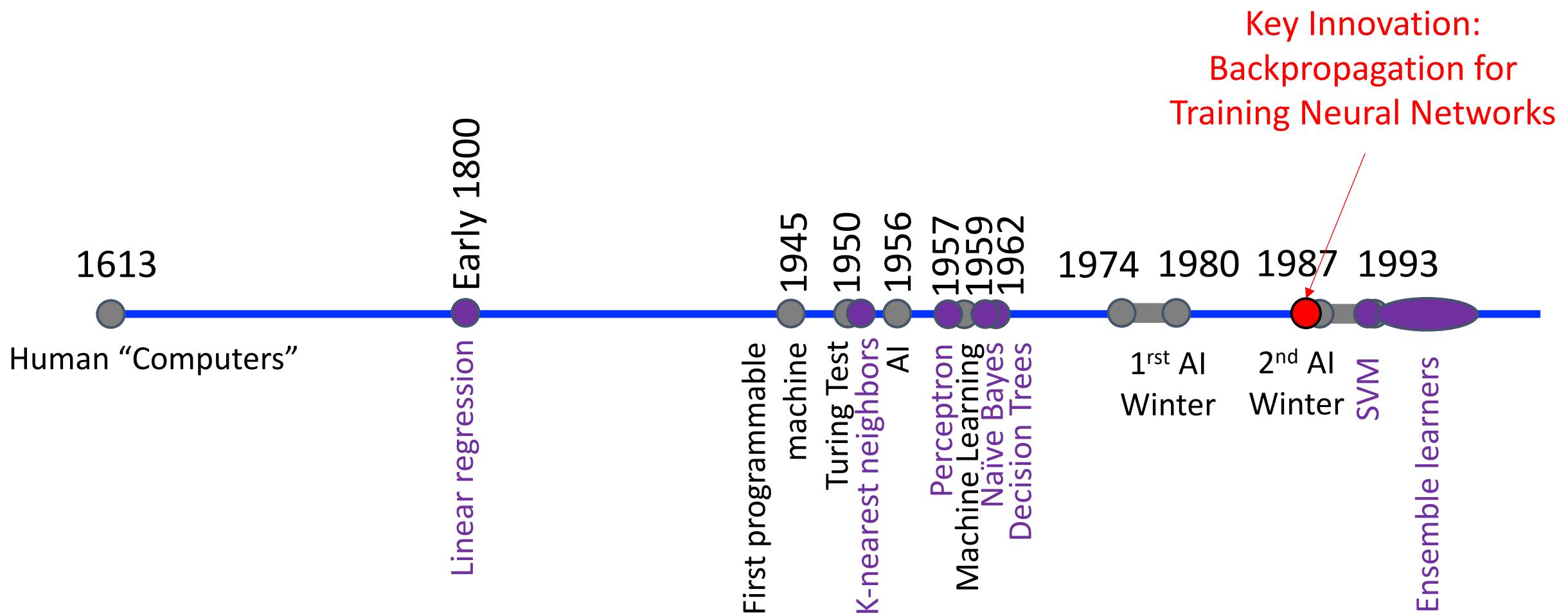
1) Loss function

2) Network activation function

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward
 3. Update each weight using calculated gradients

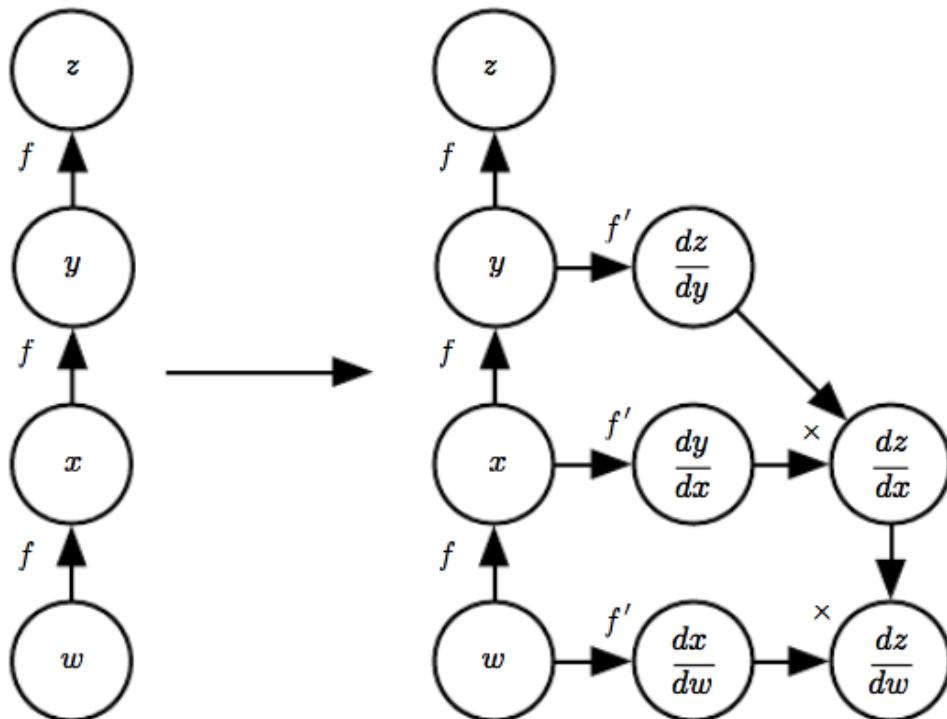
(b) Backward pass

Neural Network Training: Backpropagation



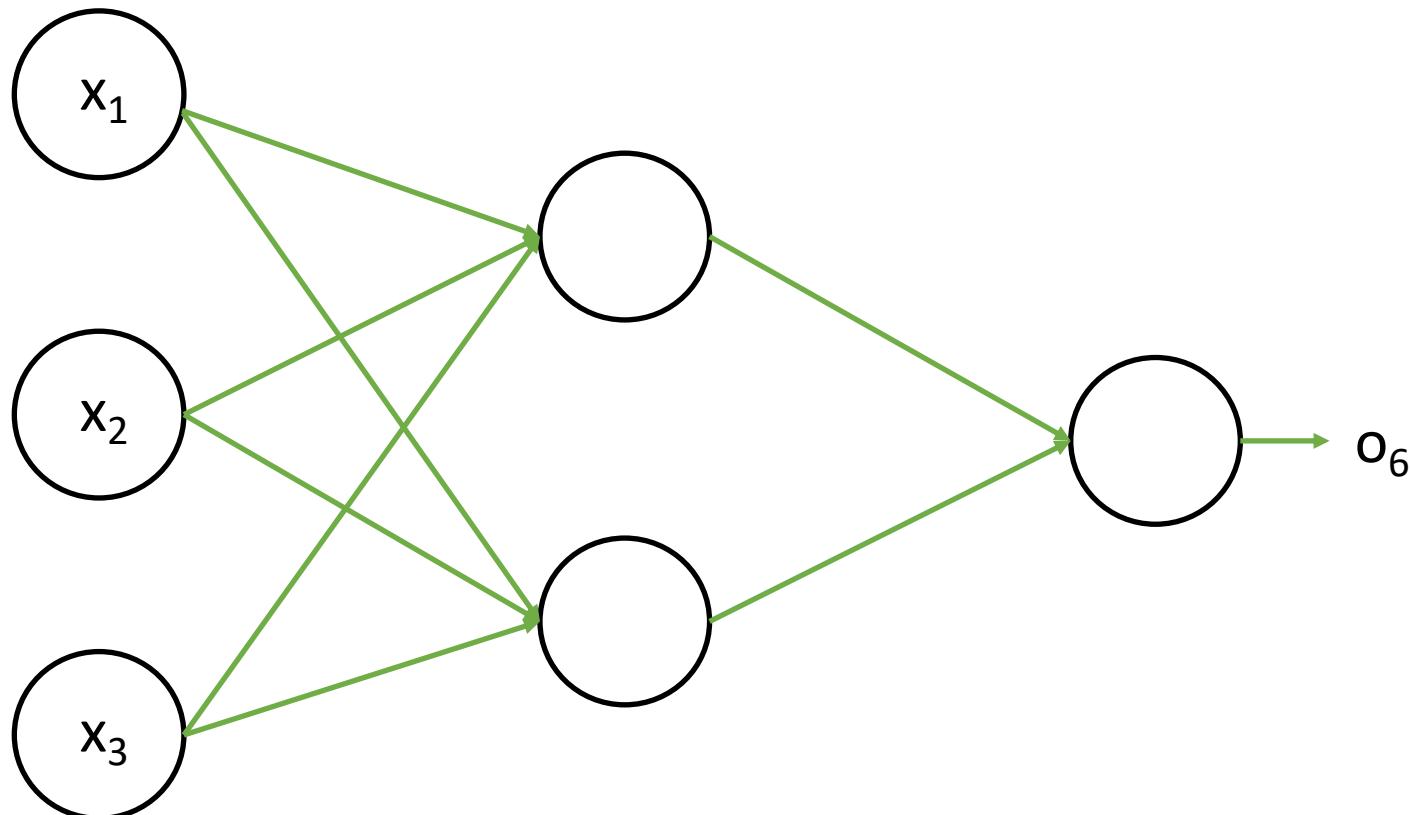
Neural Network Training: Backpropagation

- Backpropagation idea: chain: $x = f(w)$, $y = f(x)$, $z = f(y)$

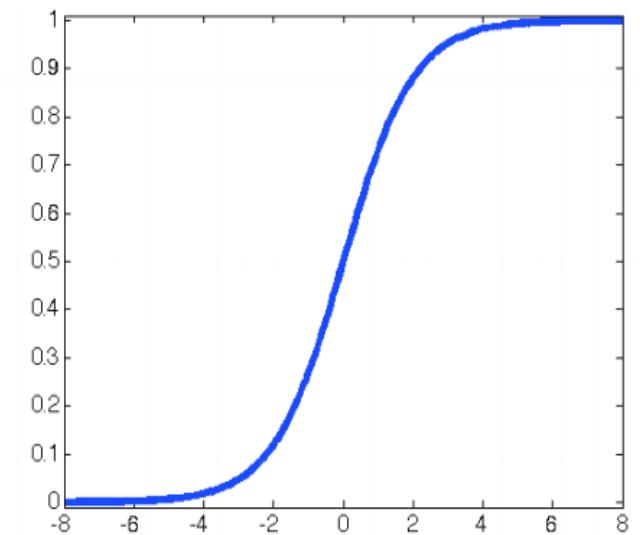


$$\begin{aligned}\frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ &= f'(y) f'(x) f'(w) \\ &= f'(f(f(w))) f'(f(w)) f'(w).\end{aligned}$$

Example: Choose Neural Network Architecture

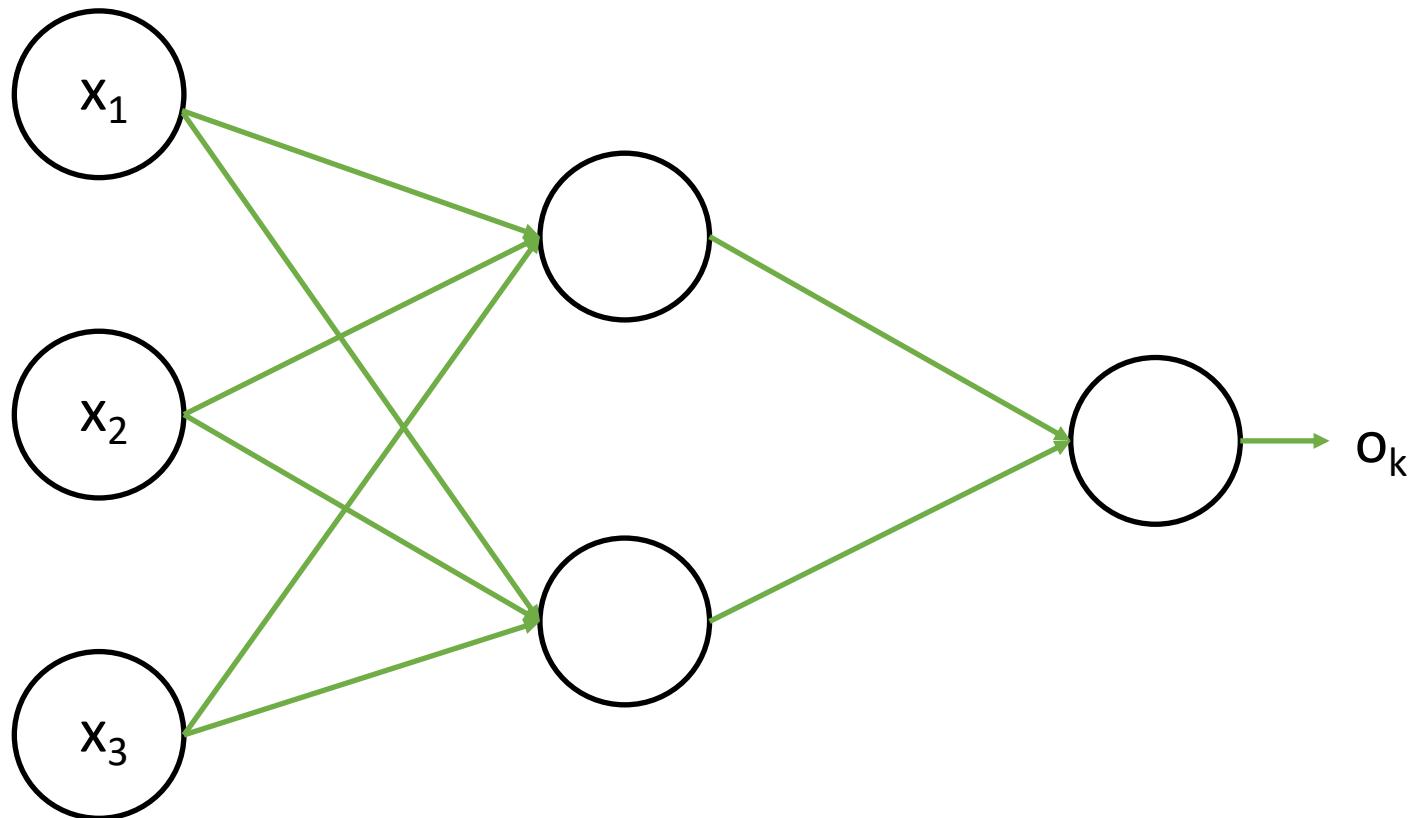


Sigmoid Activation Function



$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

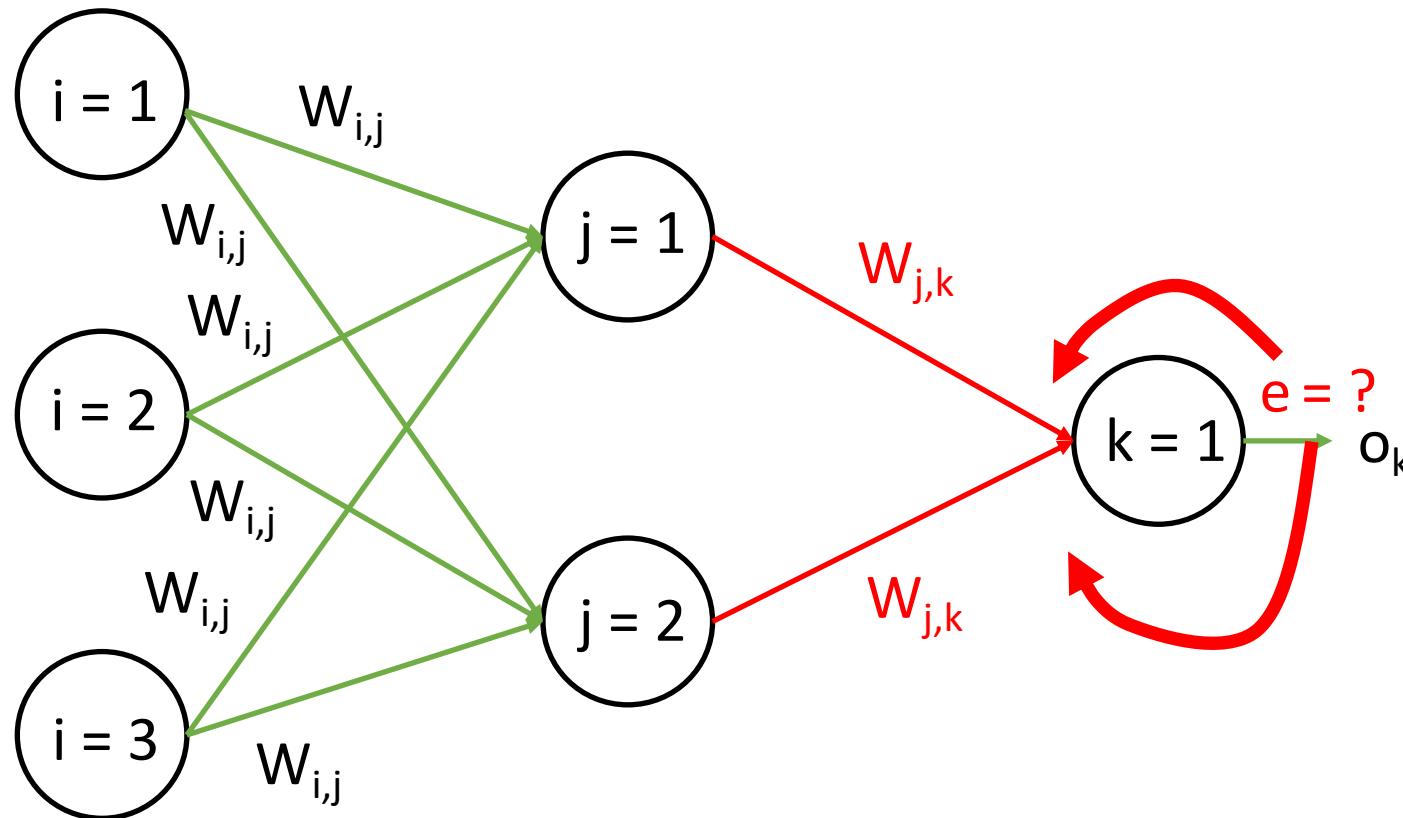
Example: Choose Loss Function for Training



Squared Error Function

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

Example: Resolve How to Compute Gradient? (Output Layer)



t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

Example: Resolve How to Compute Gradient? (Output Layer)

t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

Using the following chain rule :

$$\frac{\partial E}{\partial w_{jk}} = \boxed{\frac{\partial E}{\partial o_k}} * \boxed{\frac{\partial o_k}{\partial w_{jk}}}$$

$$\frac{\partial E}{\partial o_k} = -2(t_k - o_k)$$

Sigmoid activation function: $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$\frac{d\sigma(x)}{dx} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right)$$

$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x)) \sigma(x)$$

Example: Resolve How to Compute Gradient? (Output Layer)

t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

Using the following chain rule :

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial w_{jk}}$$

We can rewrite our function as follows:

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k)$$

$$\frac{\partial E}{\partial o_k} = -2(t_k - o_k)$$

Sigmoid activation function: $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x)) \sigma(x)$$

For efficiency, compute last

$$sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) \cdot o_j$$

Example: Resolve How to Compute Gradient? (Output Layer)

t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

Using the following chain rule :

$$\frac{\partial E}{\partial w_{jk}} = \boxed{\frac{\partial E}{\partial o_k}} * \boxed{\frac{\partial o_k}{\partial w_{jk}}}$$

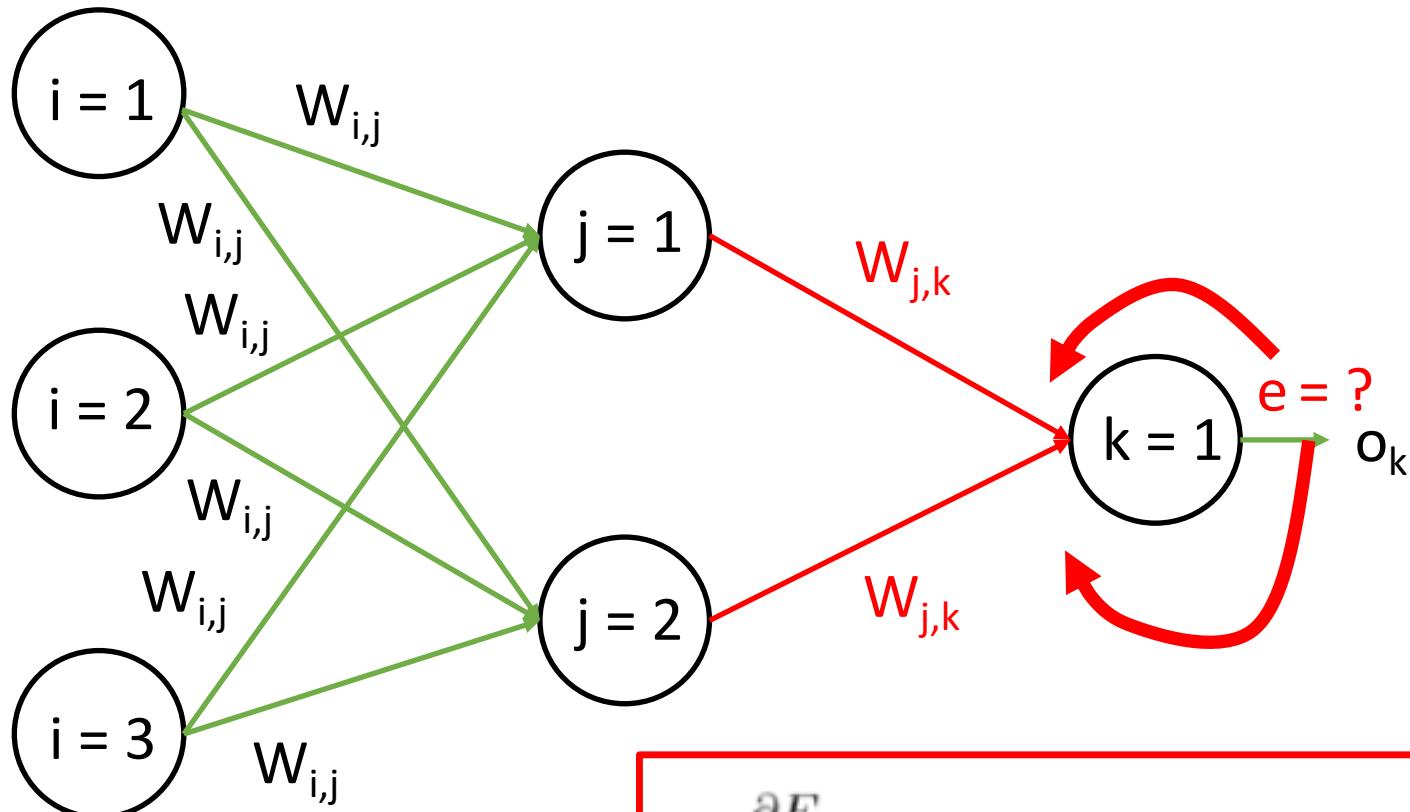
$$\frac{\partial E}{\partial o_k} = -2(t_k - o_k)$$

Sigmoid activation function: $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x)) \sigma(x)$$

Key Observation: Possible because activation function
and loss function are **differentiable!!!**

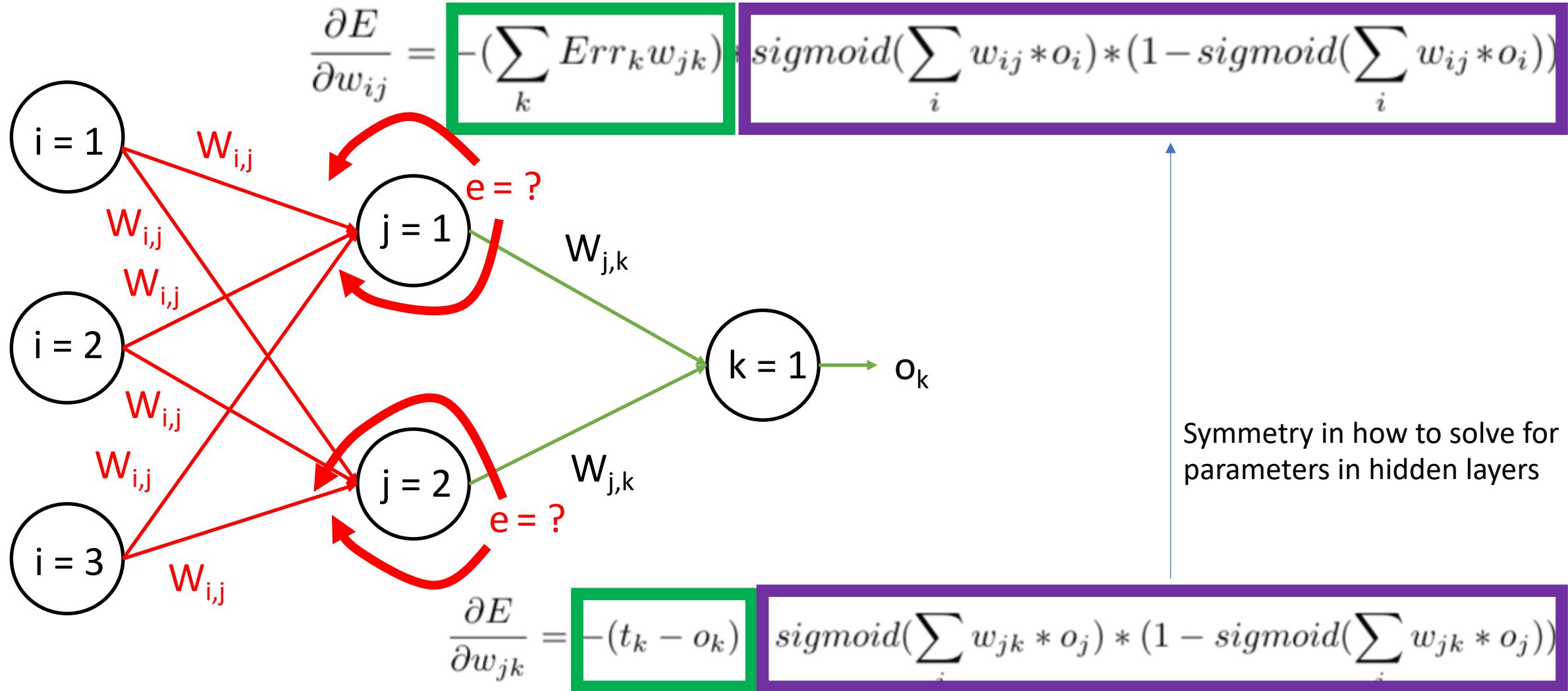
Example: Resolve How to Compute Gradient? (Output Layer)



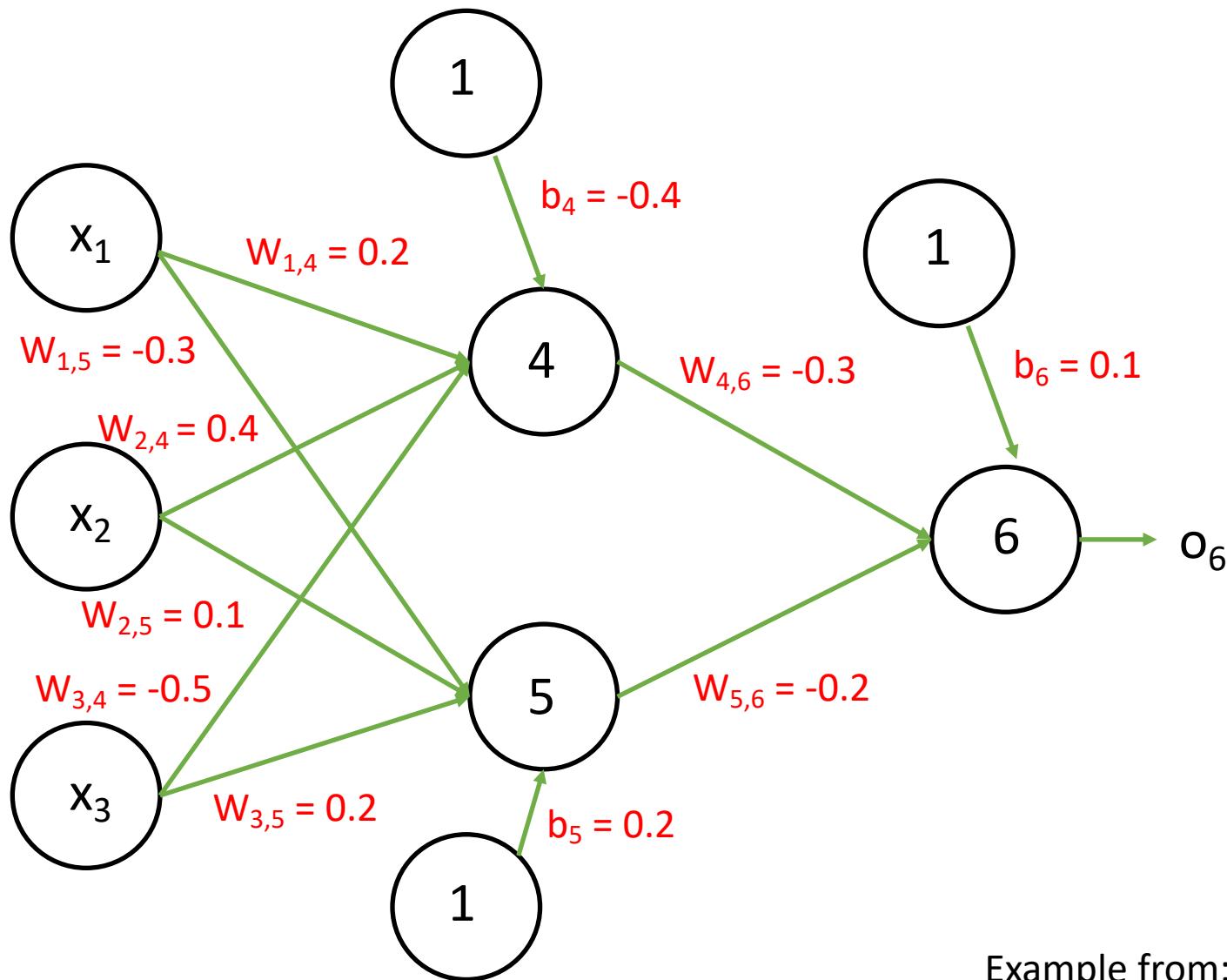
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

$$\frac{\partial E}{\partial w_{jk}} = (t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j))$$

Example: How to Compute Gradient? (Hidden Layer)

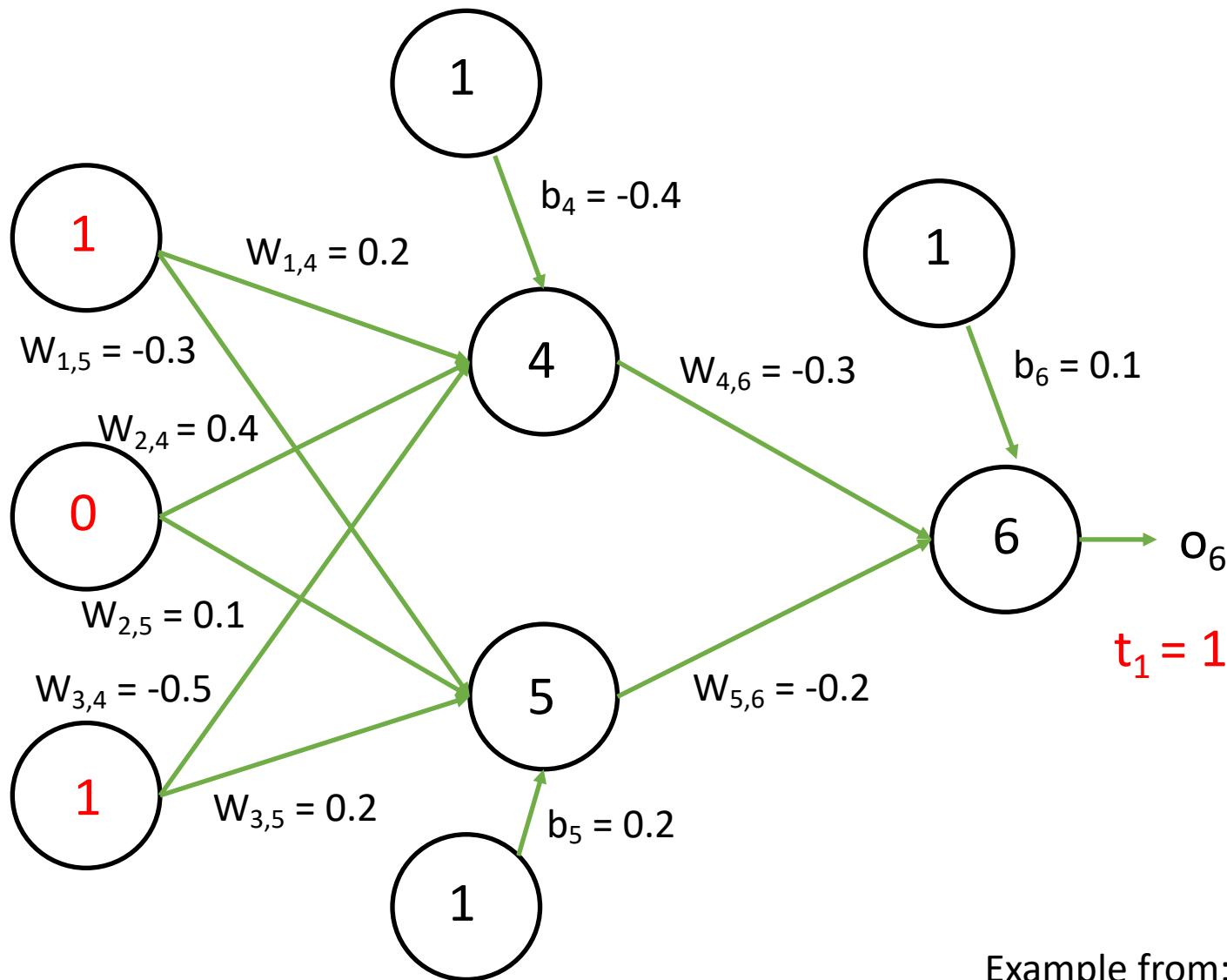


Example: Initialize Values (Weights, Biases)



Example from: Jiawei Han and Micheline Kamber; Data Mining.

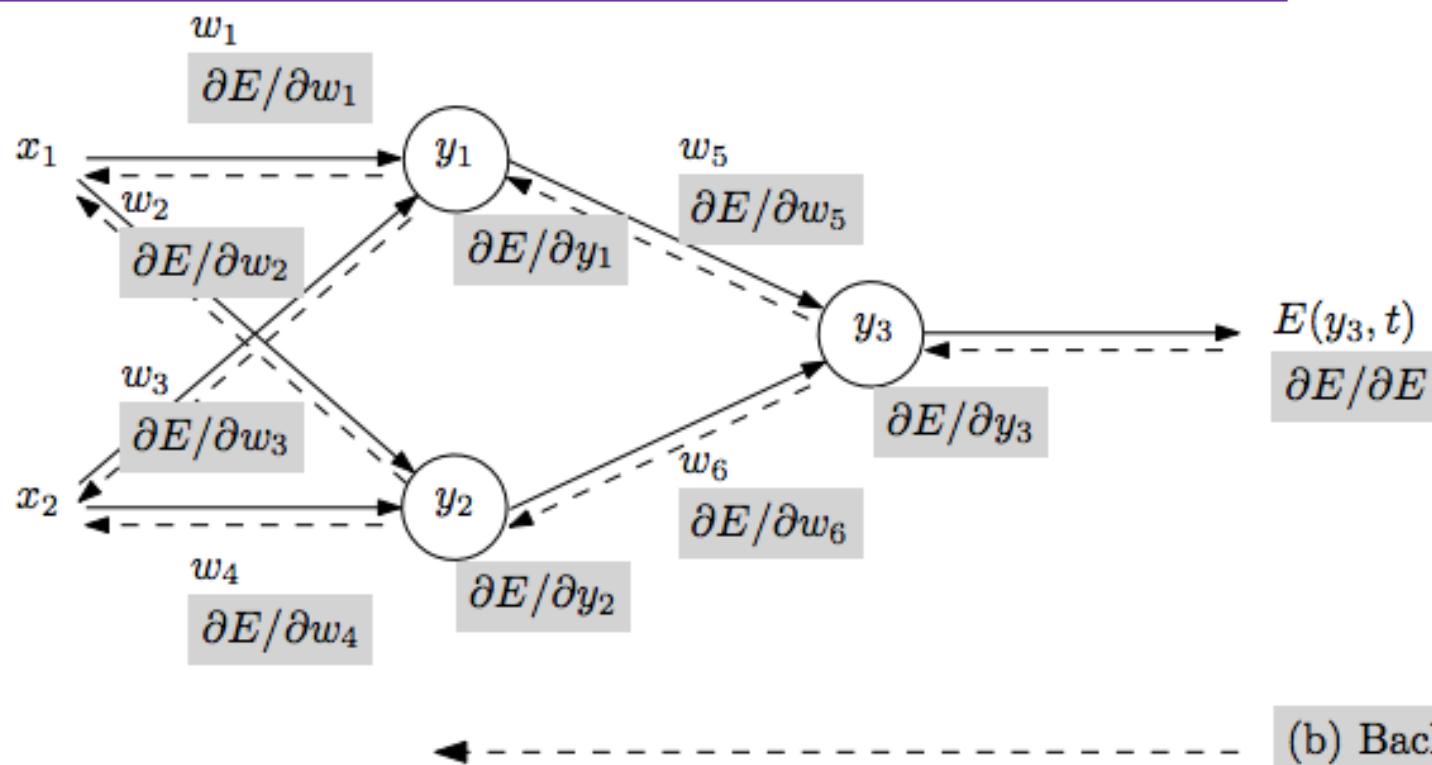
Example: Input Training Example



Example from: Jiawei Han and Micheline Kamber; Data Mining.

Example: Step 1 – Forward Pass

(a) Forward pass



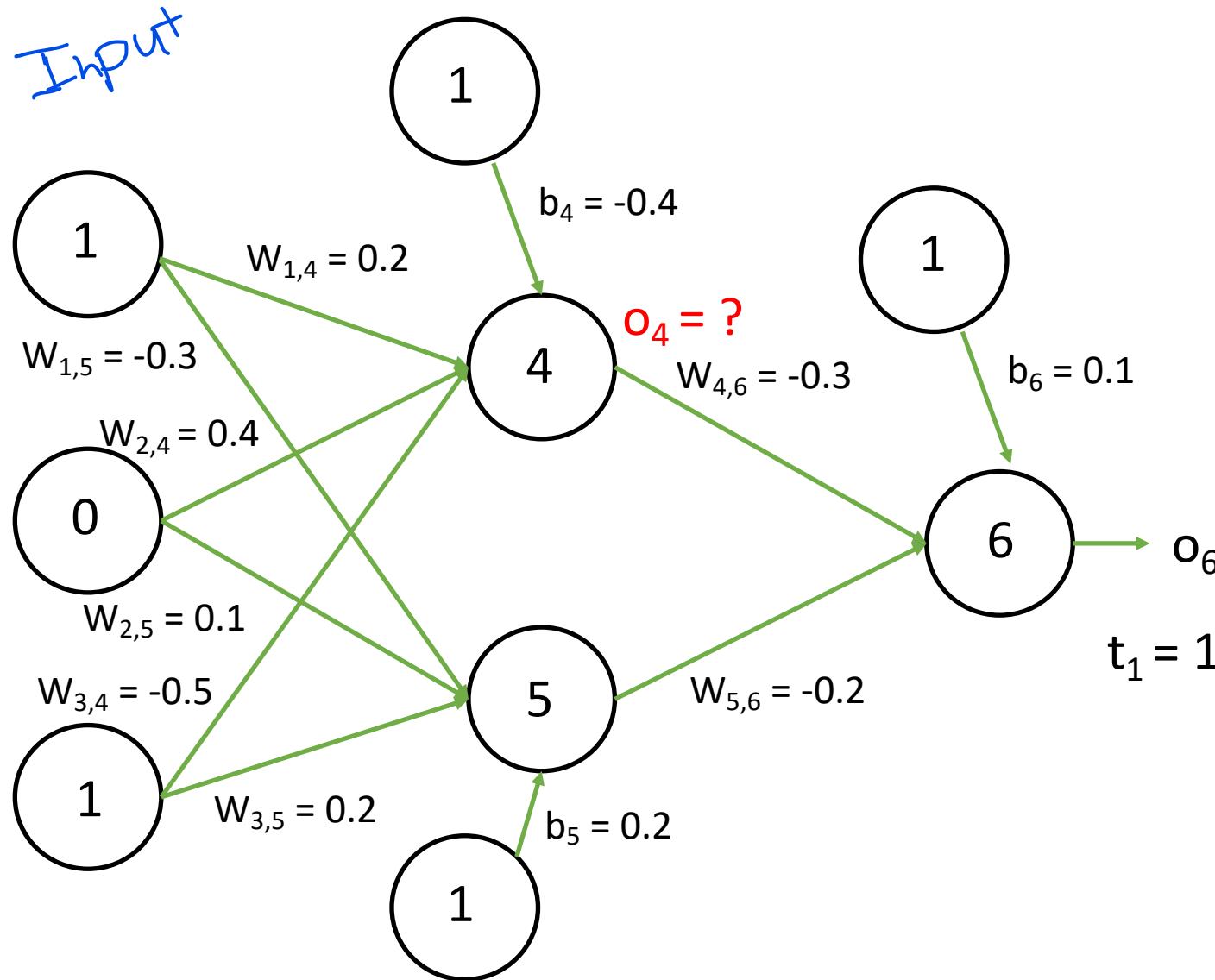
- Repeat until stopping criterion met:

1. **Forward pass:** propagate training data through network to make prediction

(b) Backward pass

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Example: Step 1 – Forward Pass



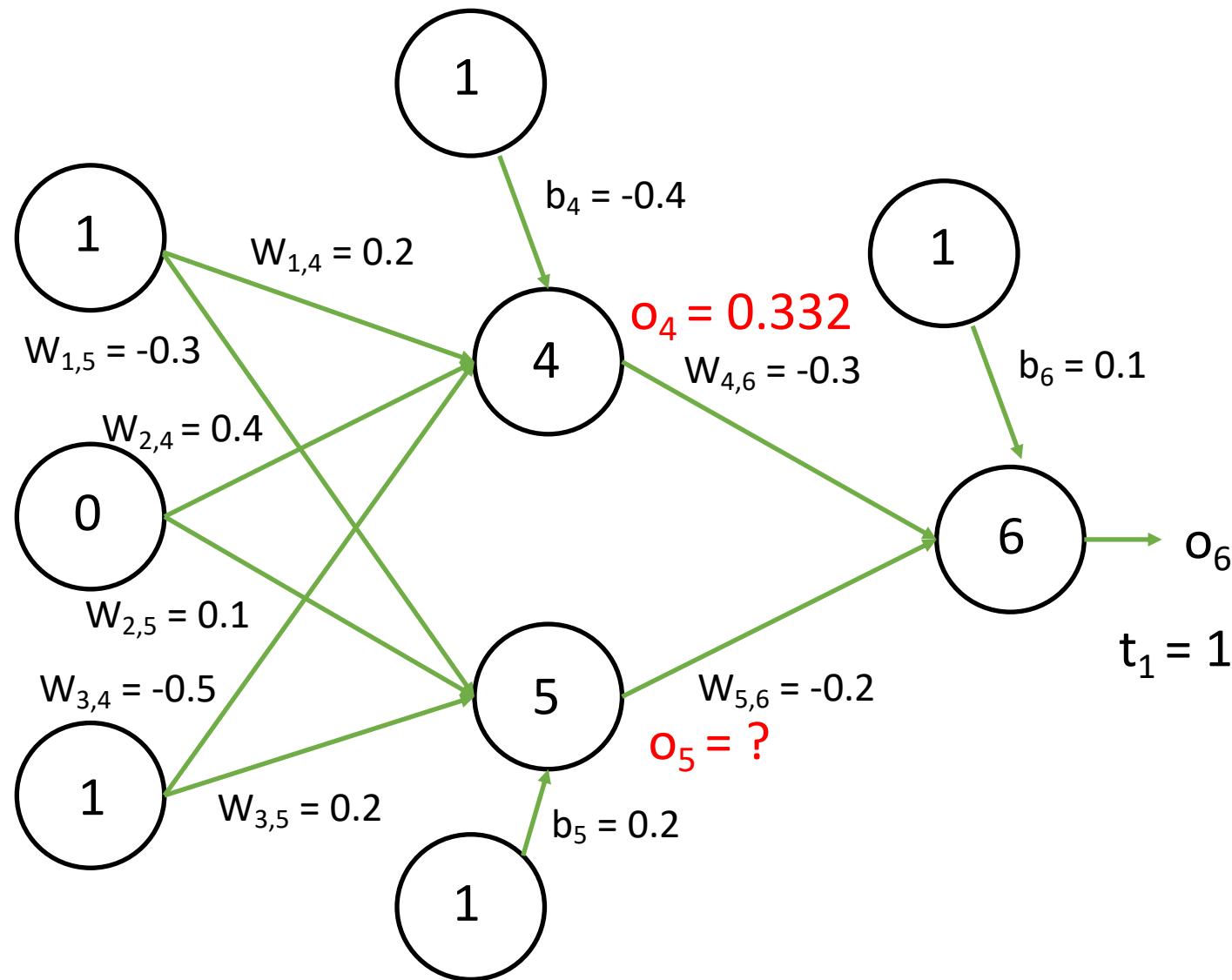
Input to node 4:

$$i_4 = (1 \times \underline{0.2} + 0 \times \underline{0.4} + 1 \times \underline{-0.5}) - 0.4$$
$$i_4 = -0.7$$

Output of node 4 (sigmoid function):

$$o_4 = \text{sigmoid}(-0.7)$$
$$o_4 = 1/(1+e^{-(-0.7)})$$
$$o_4 = 0.332$$

Example: Step 1 – Forward Pass



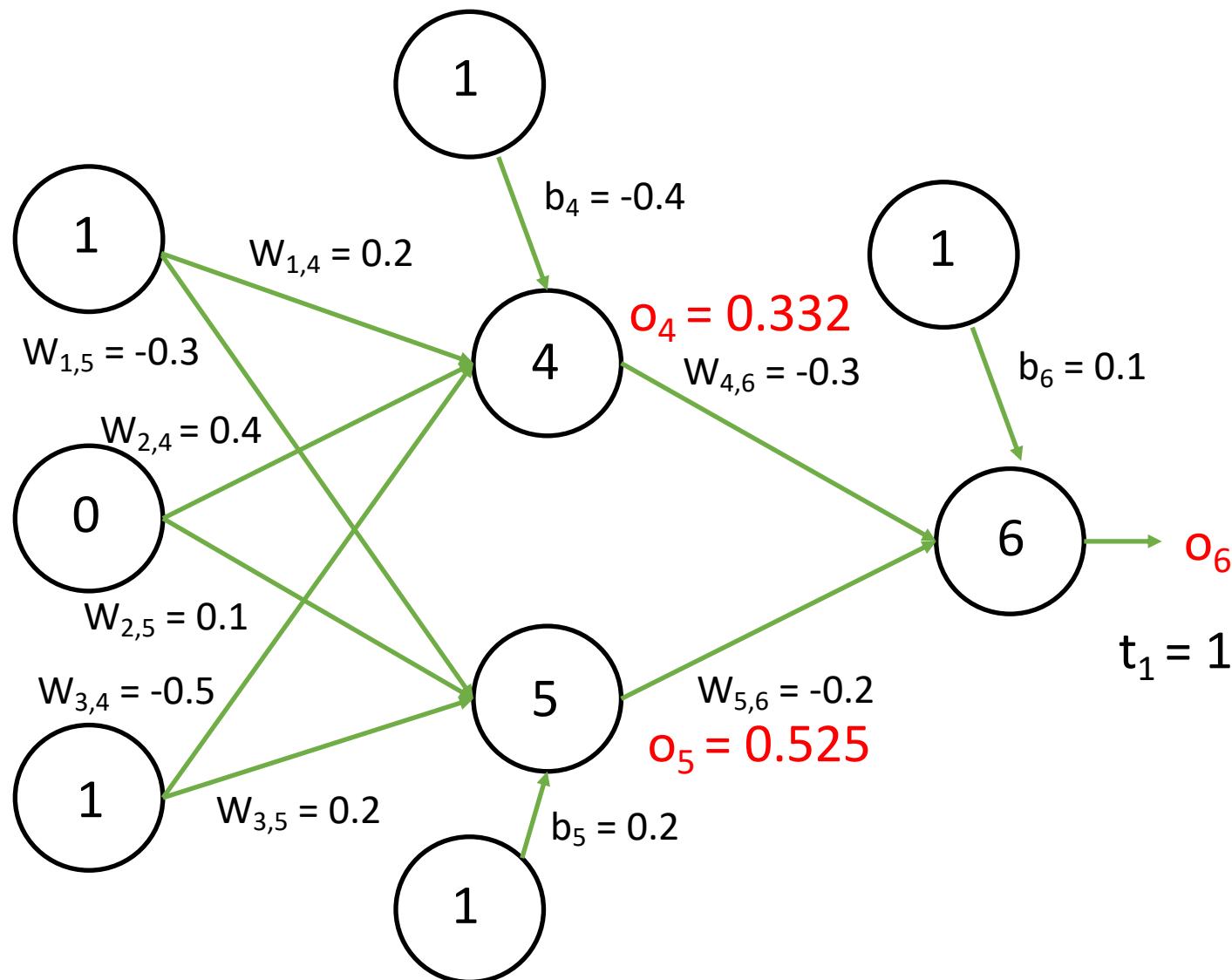
Input to node 5:

$$i_5 = (1 \times -0.3 + 0 \times 0.1 + 1 \times 0.2) + 0.2$$
$$i_5 = 0.1$$

Output of node 5 (sigmoid function):

$$o_5 = \text{sigmoid}(0.1)$$
$$o_5 = 1/(1+e^{-0.1})$$
$$o_5 = 0.525$$

Example: Step 1 – Forward Pass



Input to node 6:

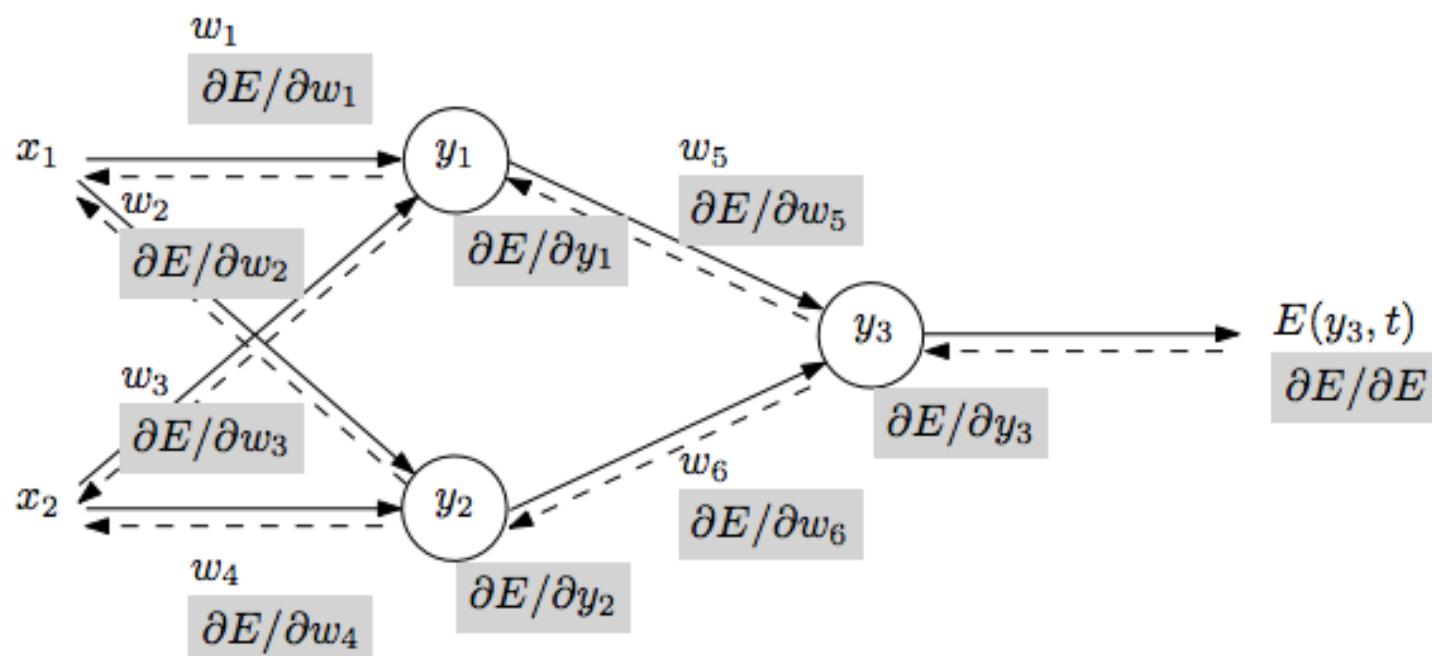
$$i_6 = (0.332 \times -0.3 + 0.1 \times -0.2) + 0.1$$
$$i_6 = -0.105$$

Output of node 6 (sigmoid function):

$$o_6 = \text{sigmoid}(-0.105)$$
$$o_6 = 1/(1+e^{-(-0.105)})$$
$$o_6 = 0.474$$

Example: Step 2 – Backward Pass

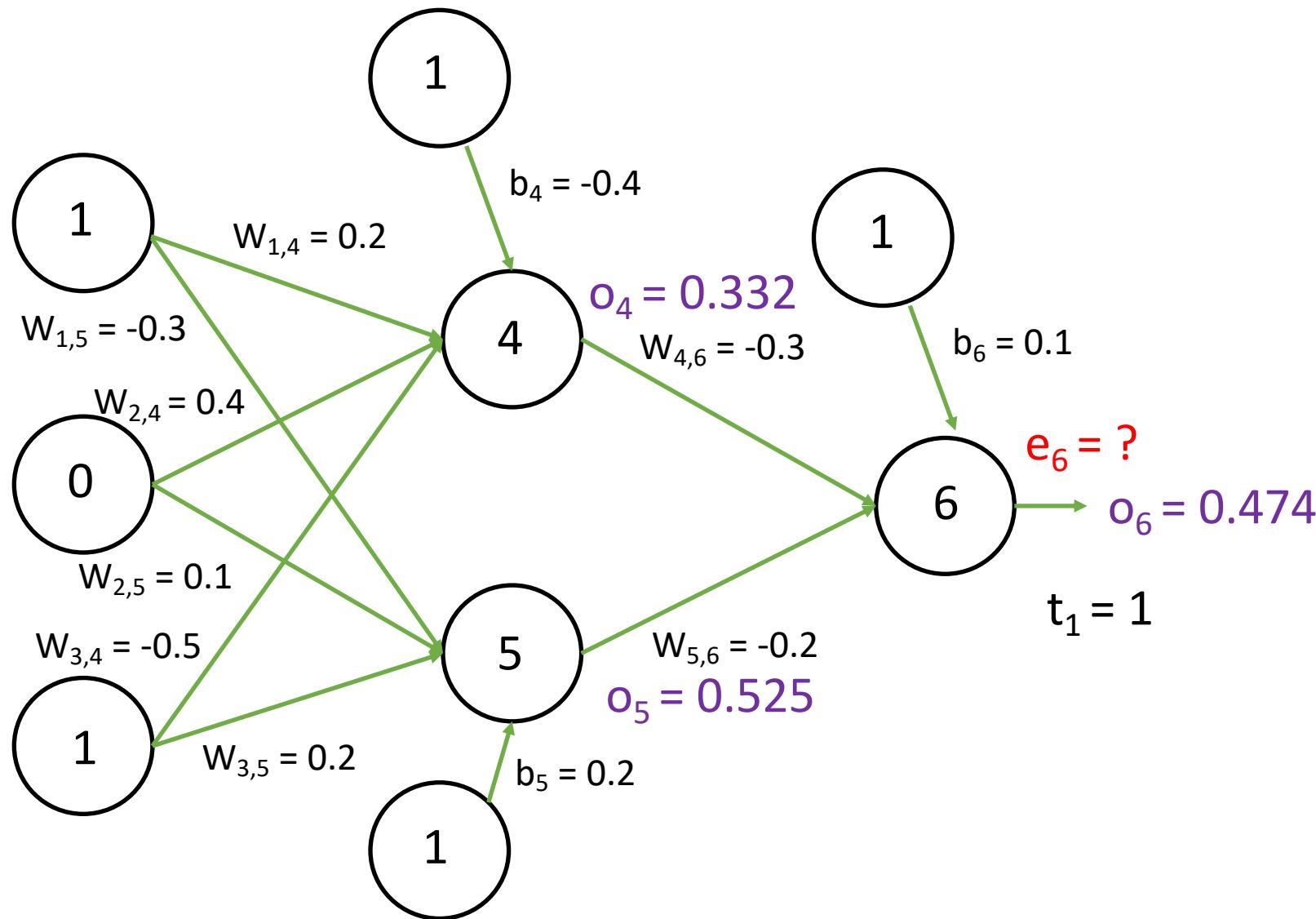
(a) Forward pass



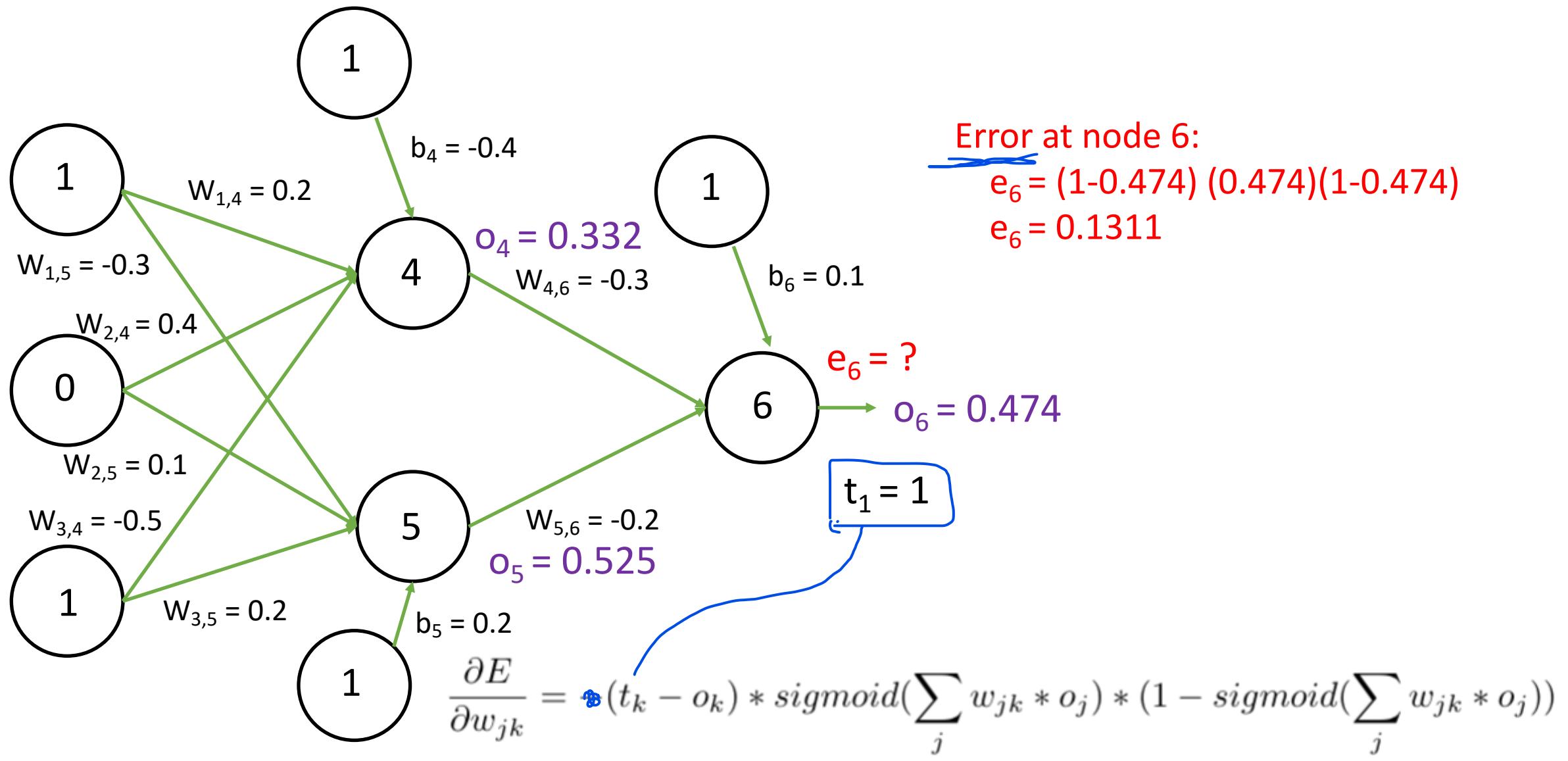
(b) Backward pass

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward

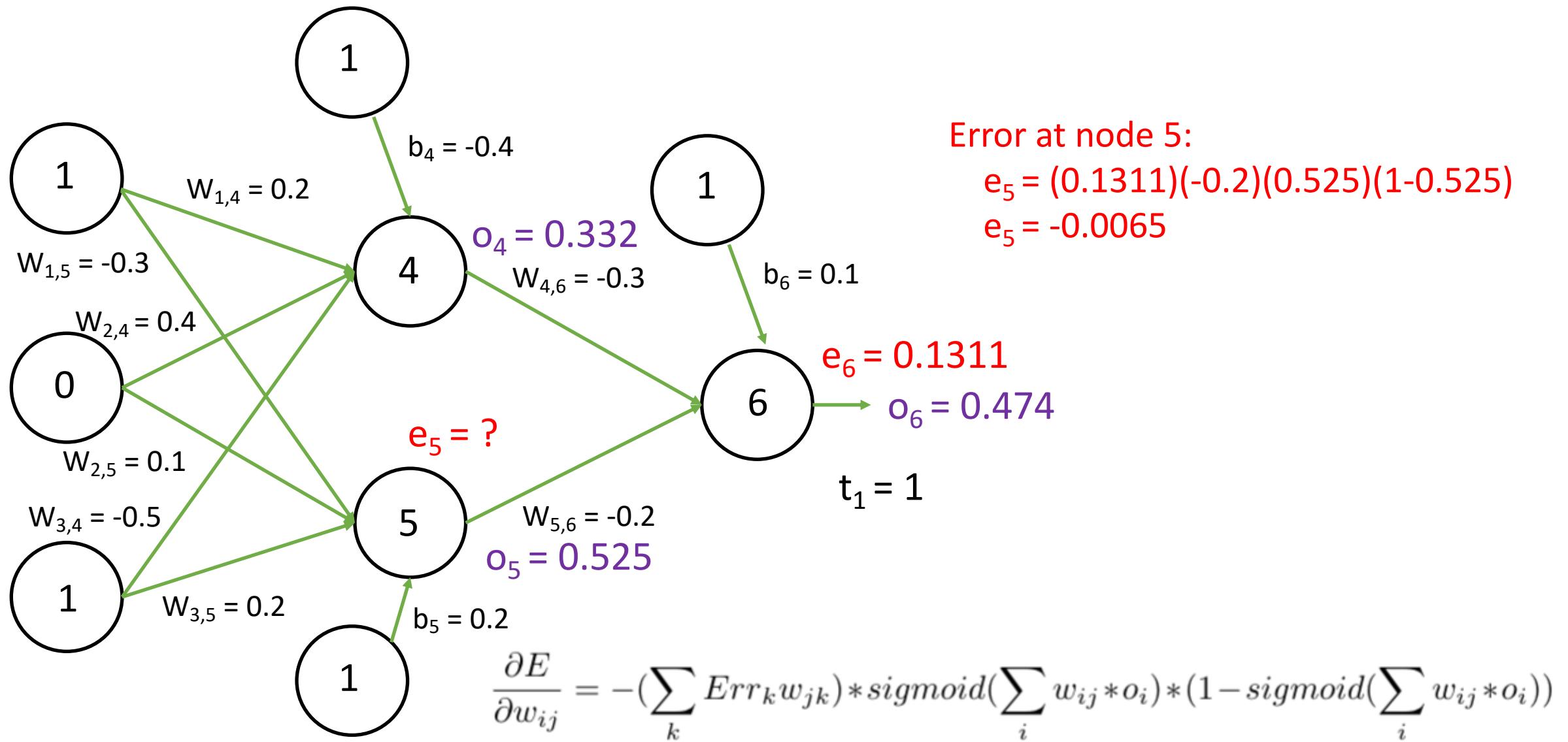
Example: Step 2 – Backward Pass



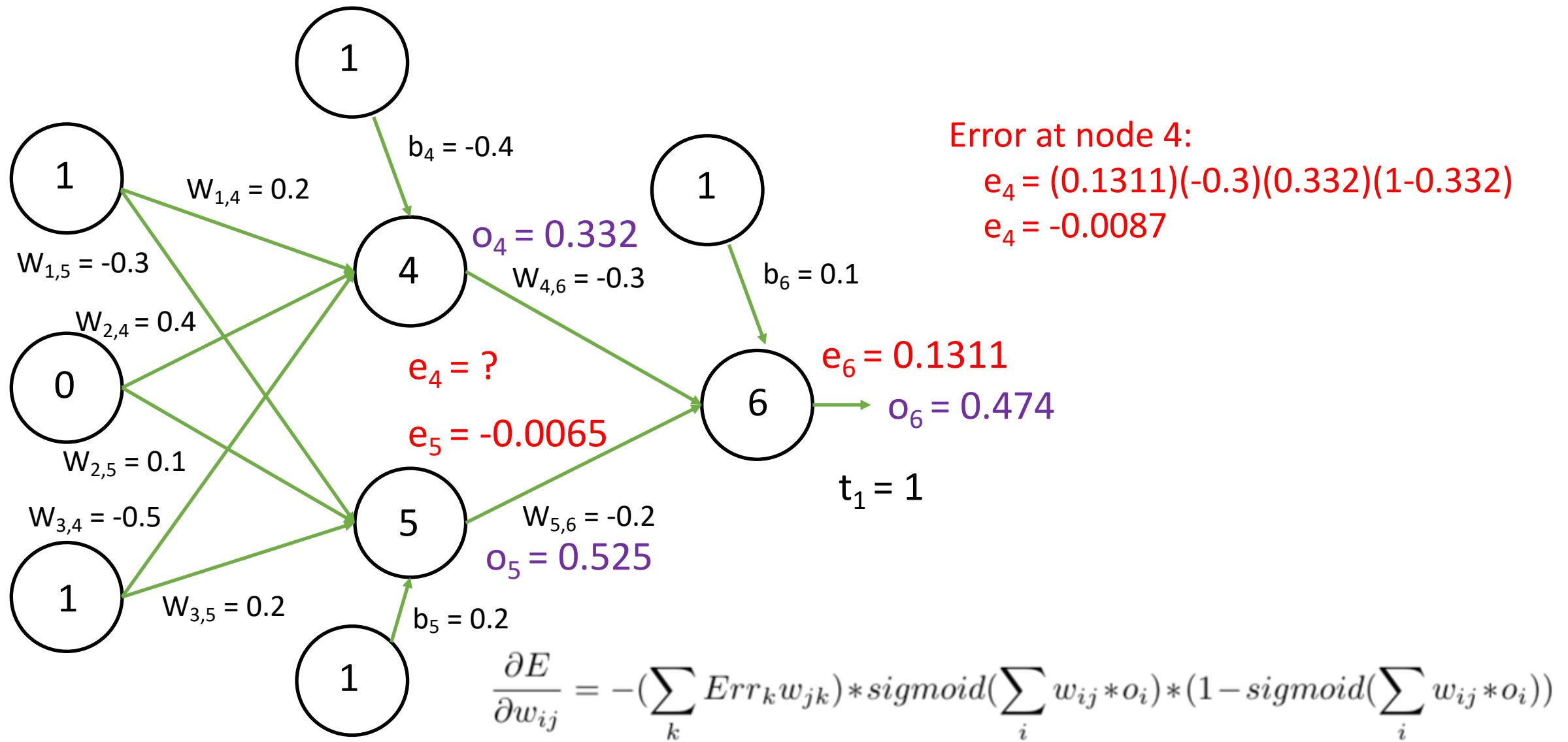
Example: Step 2 – Backward Pass



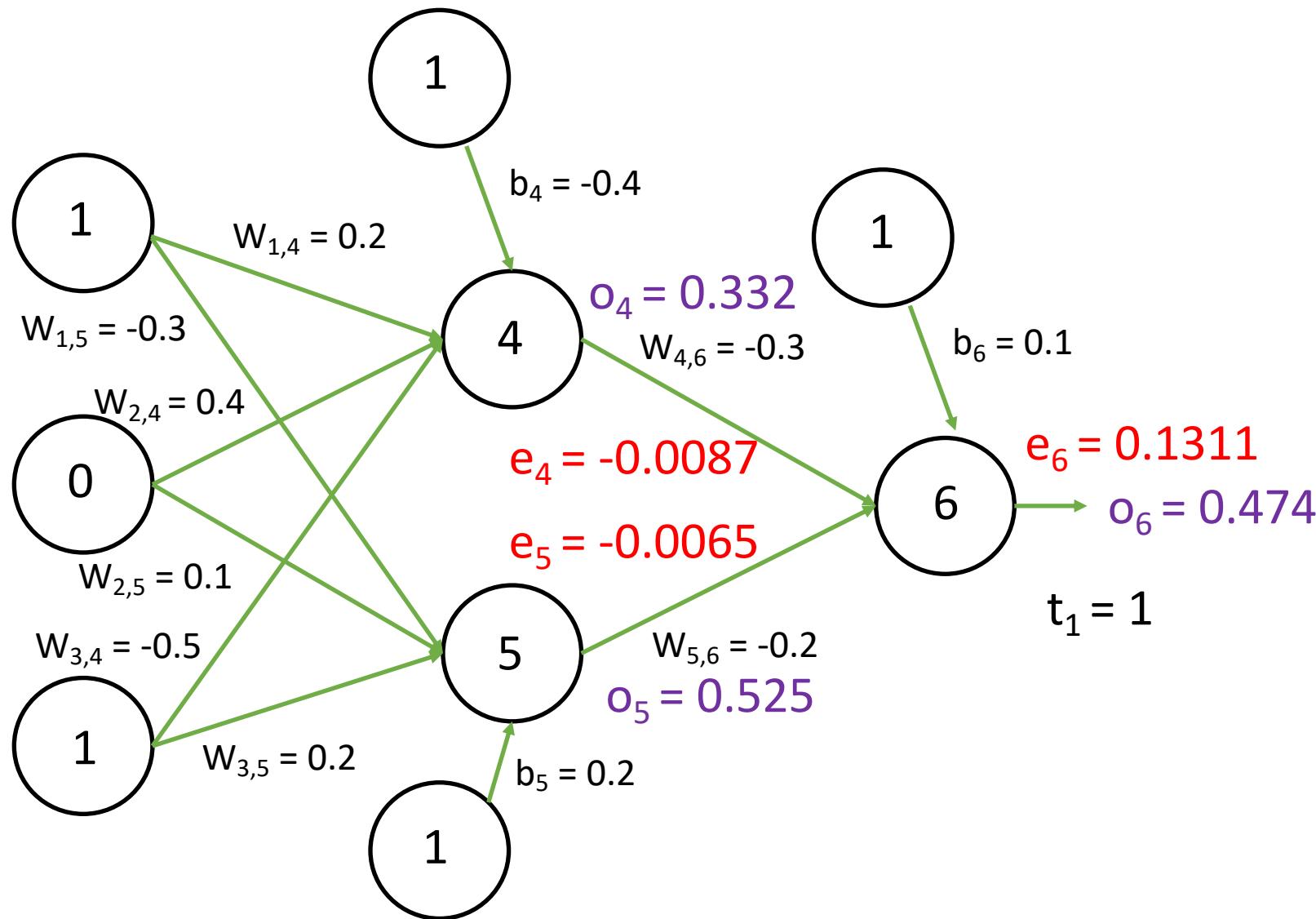
Example: Step 2 – Backward Pass



Example: Step 2 – Backward Pass

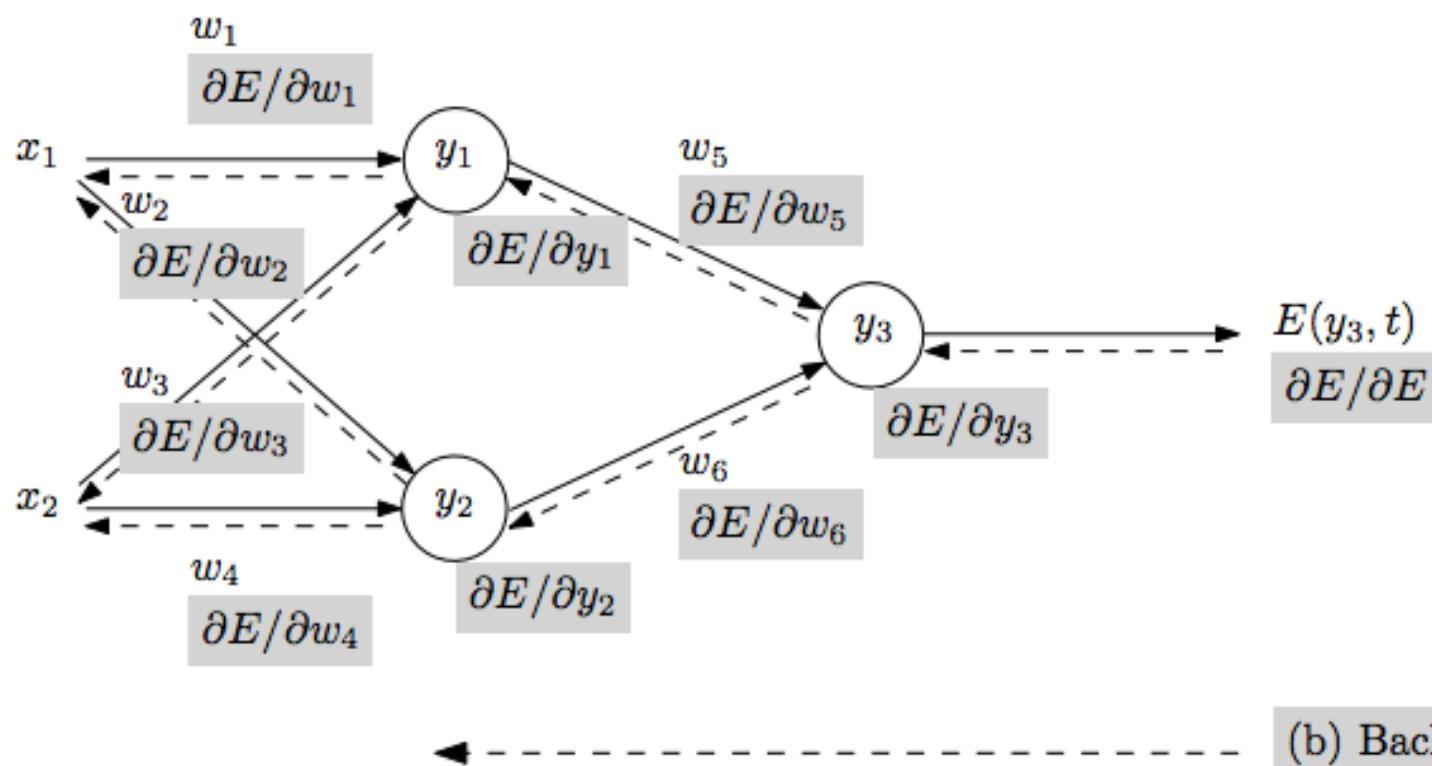


Example: Step 2 – Backward Pass



Example: Step 3 – Update Weights

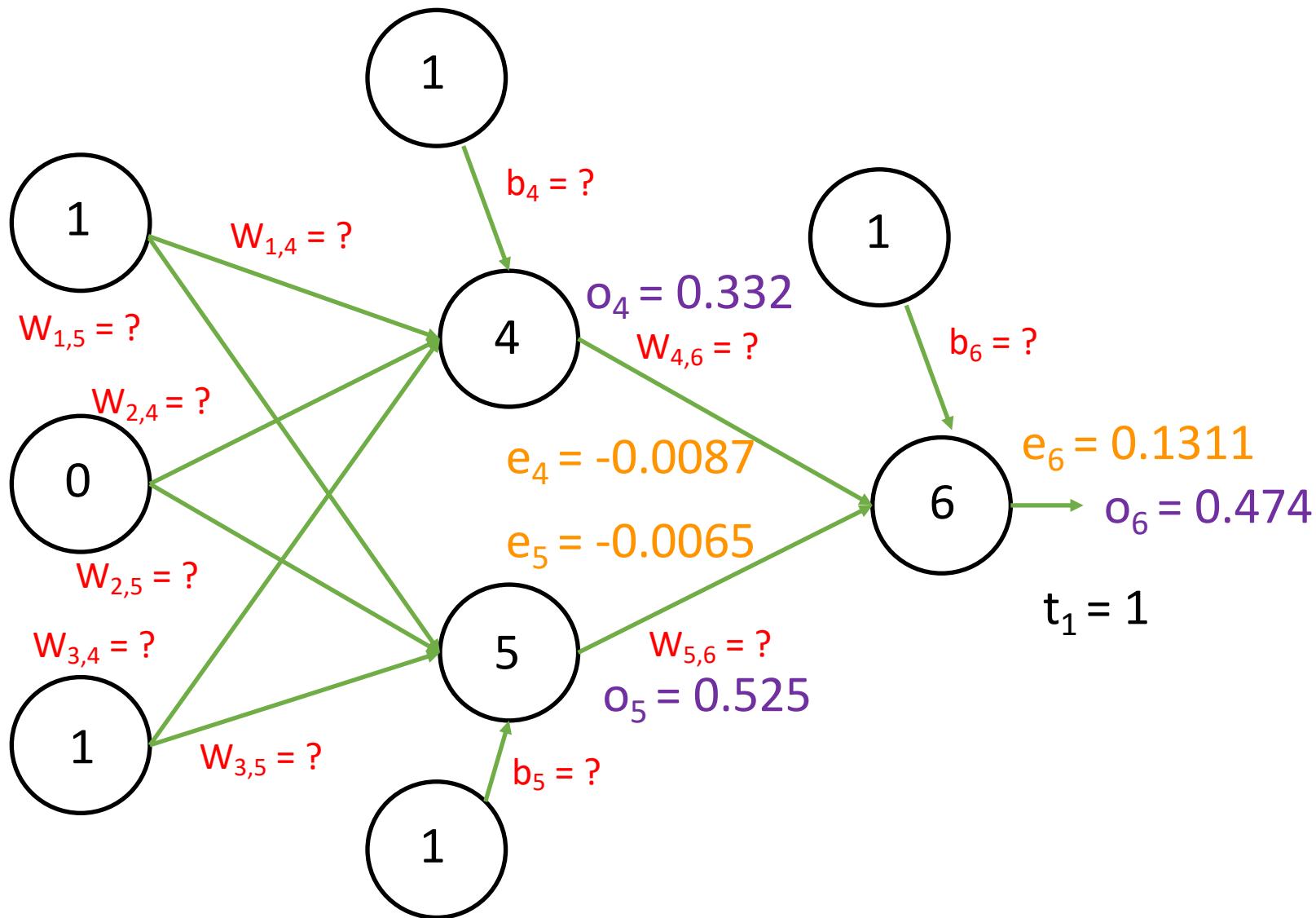
(a) Forward pass



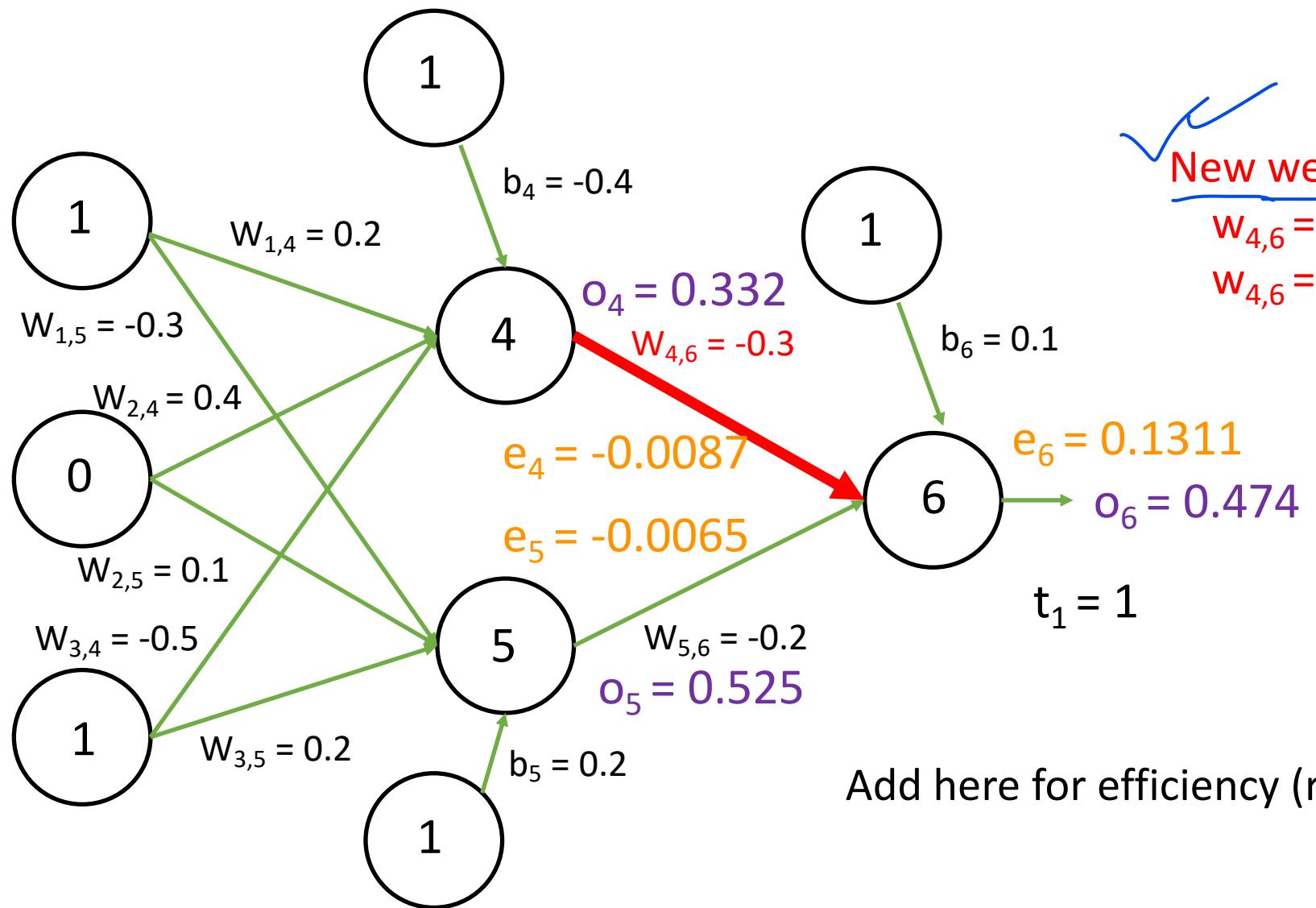
(b) Backward pass

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward
 3. **Update each weight using calculated gradients**

Example: Step 3 – Update Weights



Example: Step 3 – Update Weights



New weights (learning rate = 0.9):

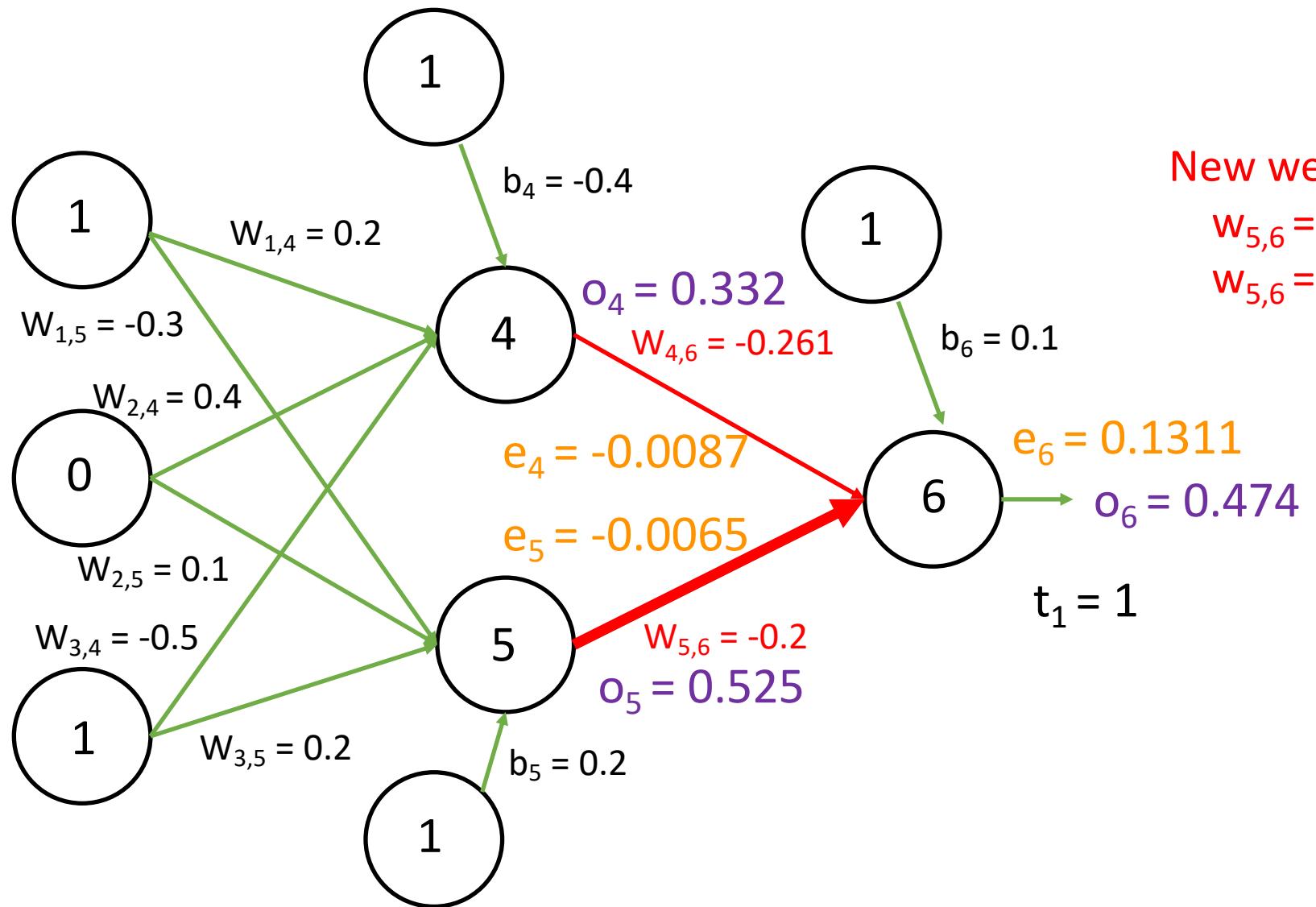
$$w_{4,6} = -0.3 + (0.9)(0.1311)(0.332)$$

$$w_{4,6} = -0.261$$

Add here for efficiency (removed from earlier equation)

$$w_{jk} = w_{jk} + \eta Err_{obj}$$

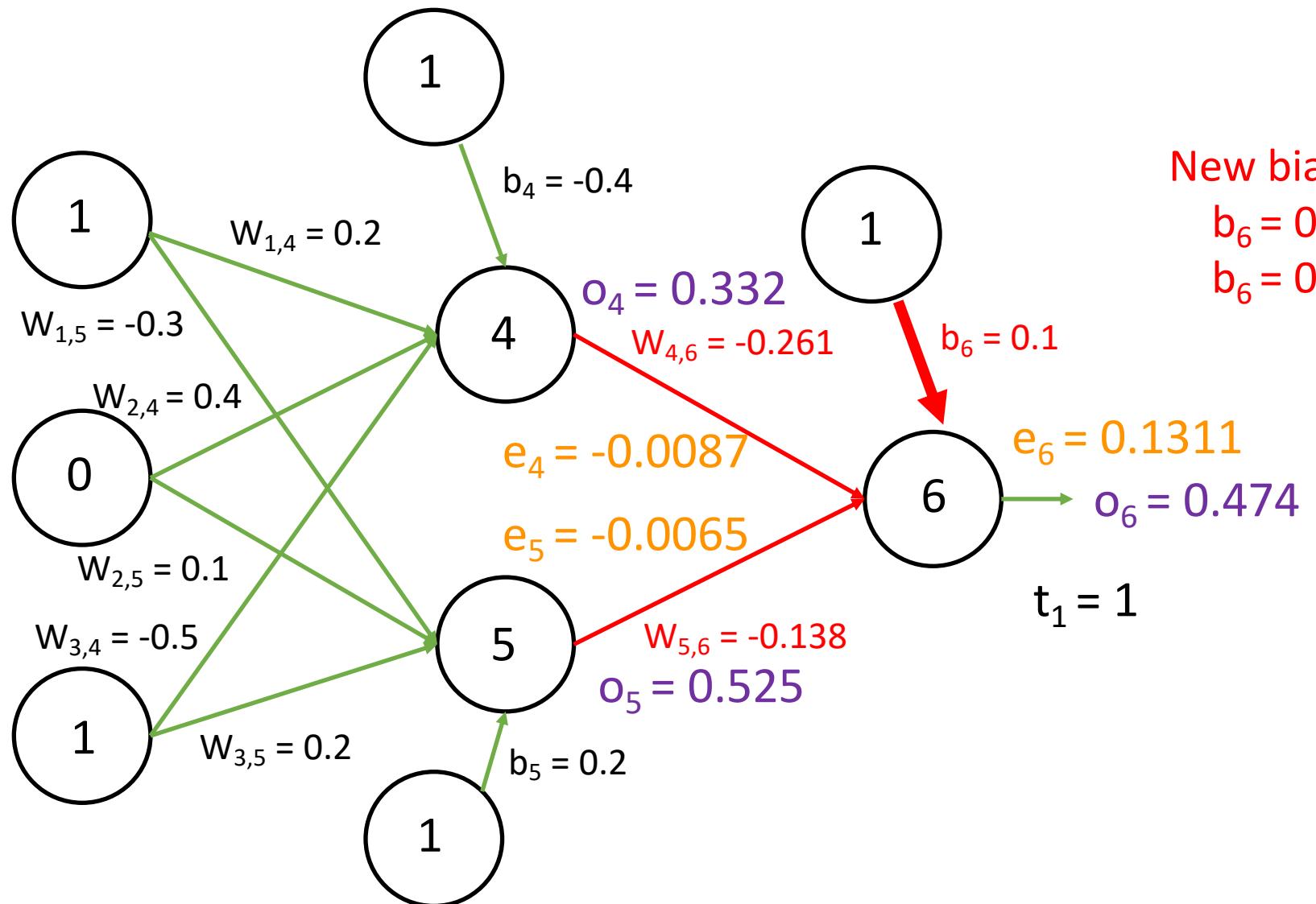
Example: Step 3 – Update Weights



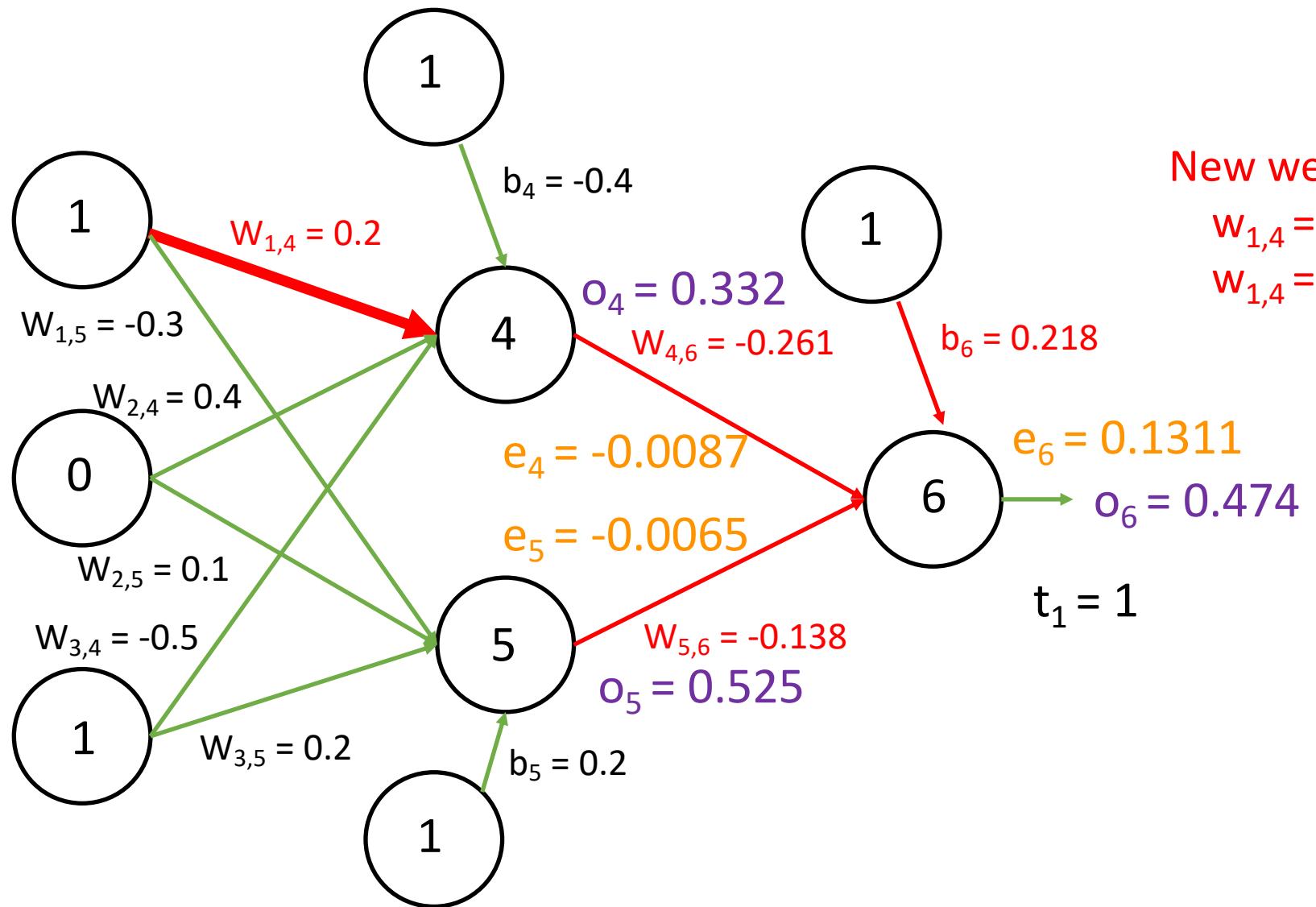
New weights (learning rate = 0.9):
 $w_{5,6} = -0.2 + (0.9)(0.1311)(0.525)$
 $w_{5,6} = -0.138$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights

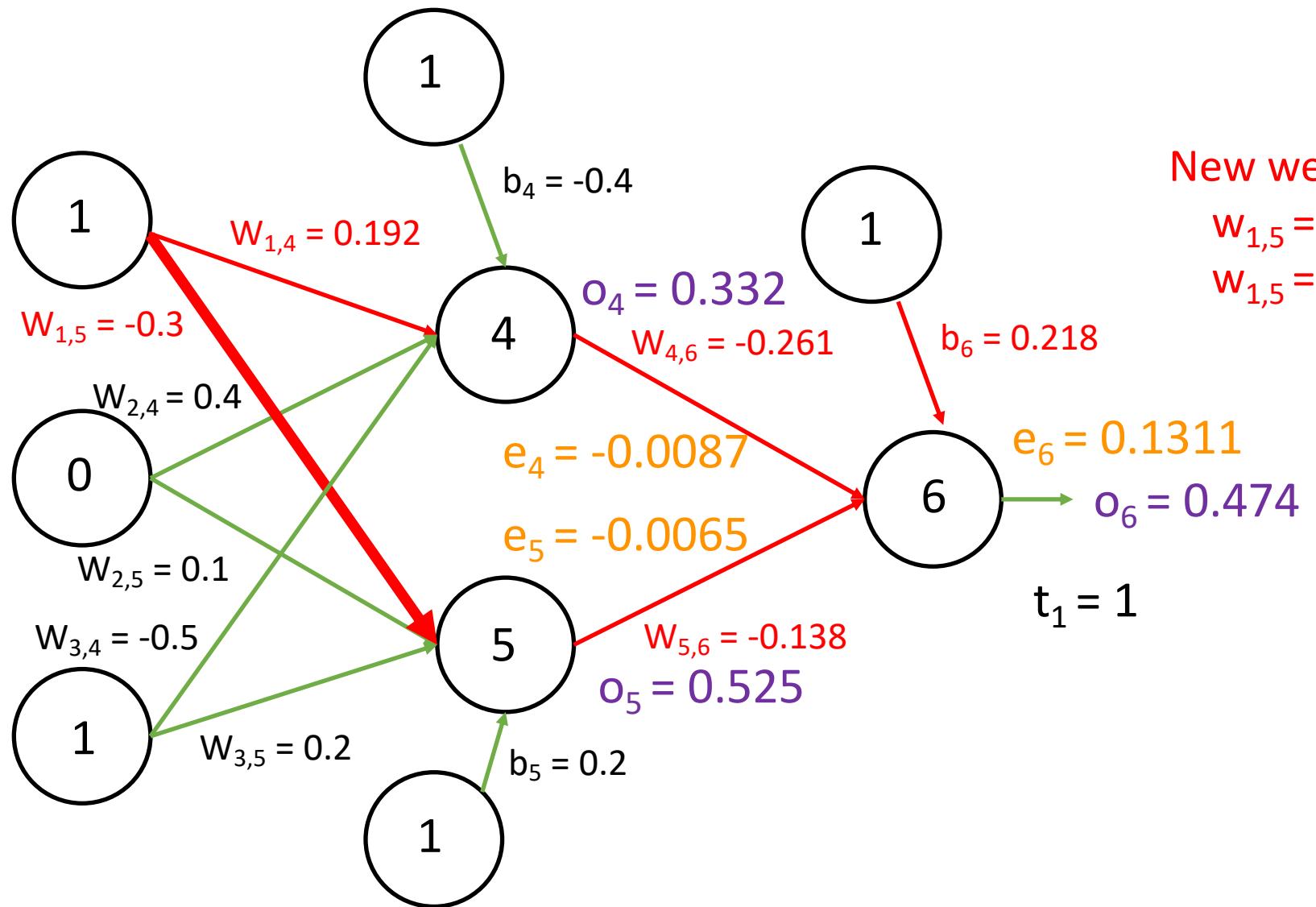


Example: Step 3 – Update Weights



$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



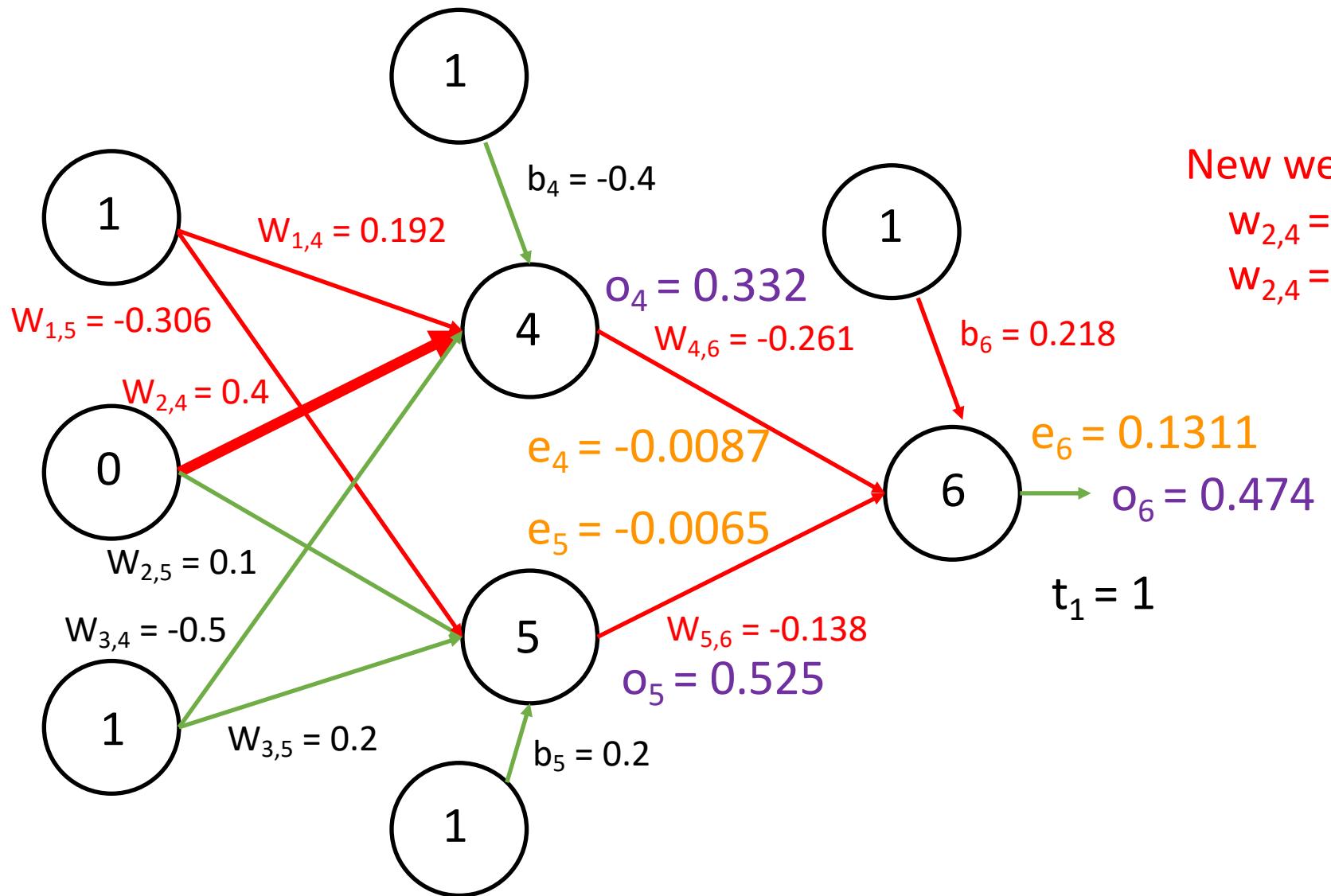
New weights (learning rate = 0.9):

$$w_{1,5} = -0.3 + (0.9)(-0.0065)(1)$$

$$w_{1,5} = -0.306$$

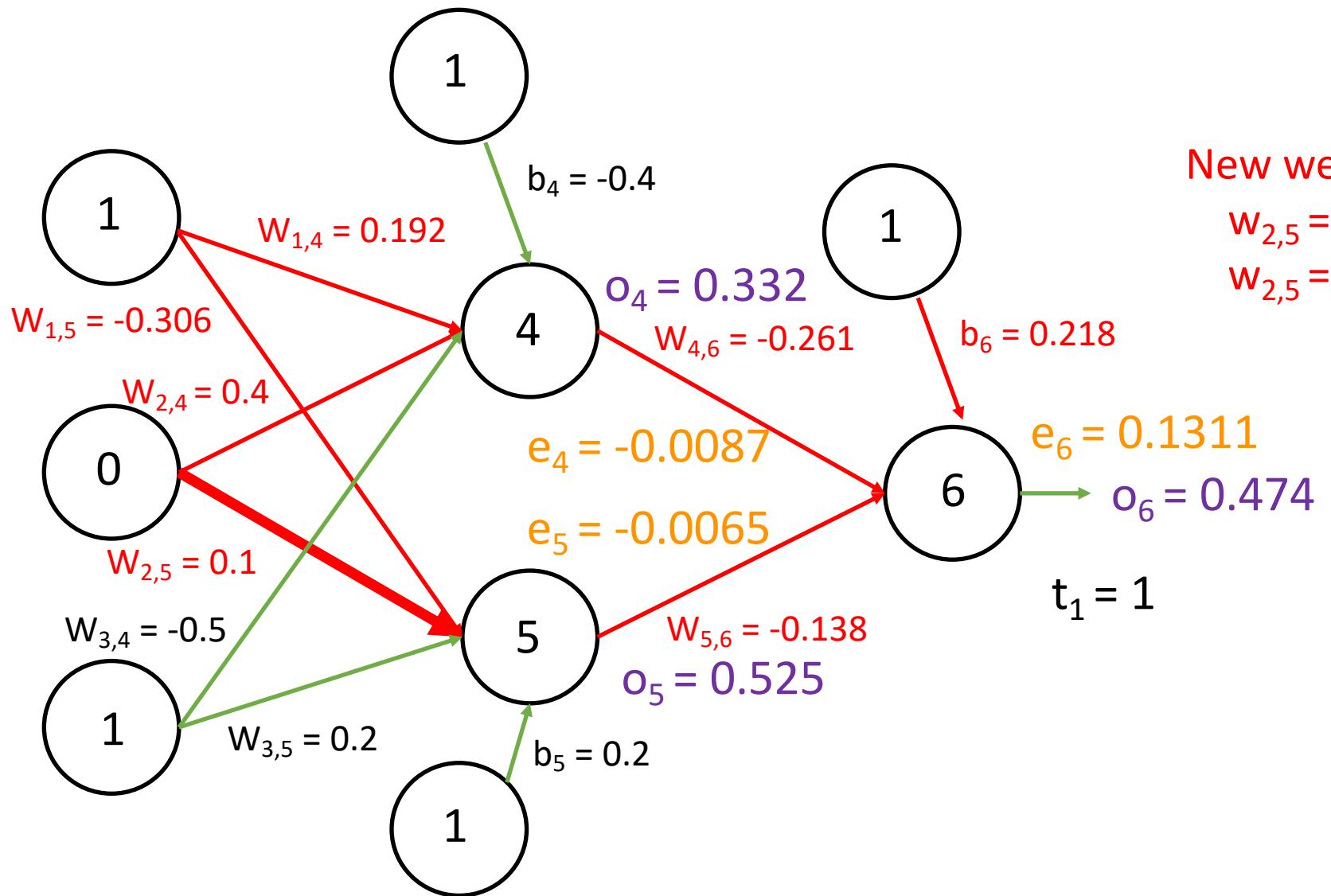
$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



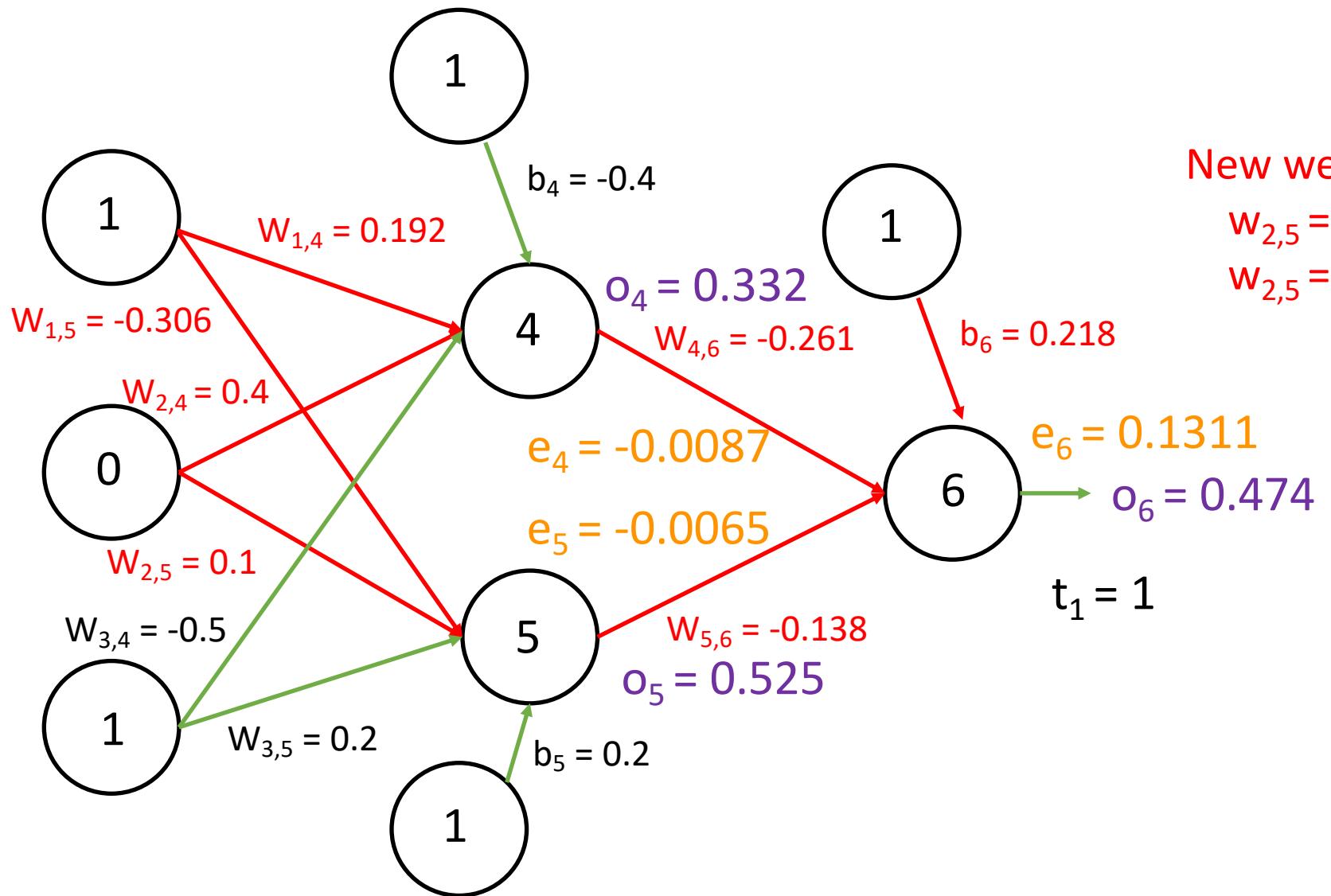
New weights (learning rate = 0.9):

$$w_{2,5} = 0.1 + (0.9)(-0.0065)(0)$$

$$w_{2,5} = 0.1$$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



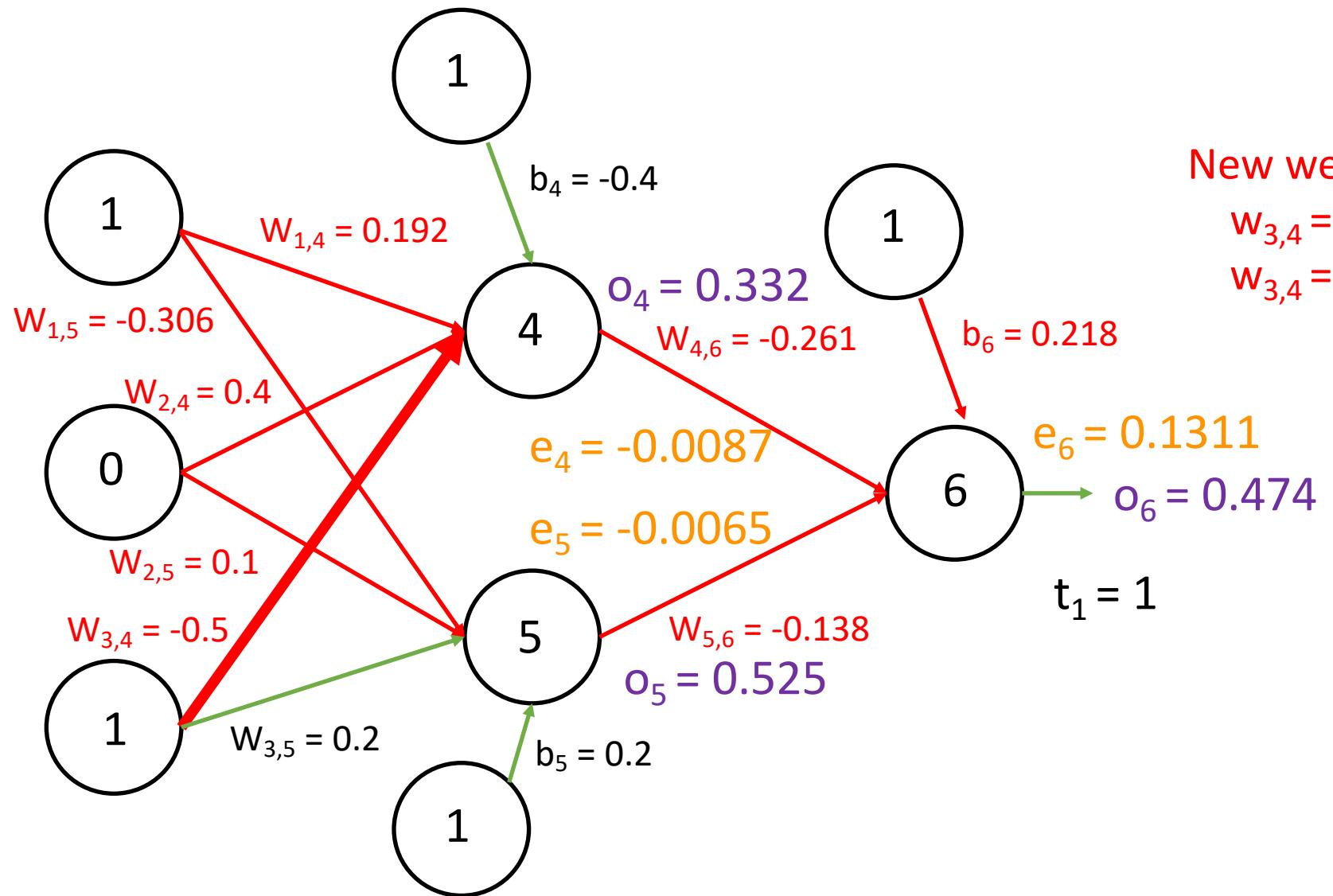
New weights (learning rate = 0.9):

$$w_{2,5} = 0.1 + (0.9)(-0.0065)(0)$$

$$w_{2,5} = 0.1$$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



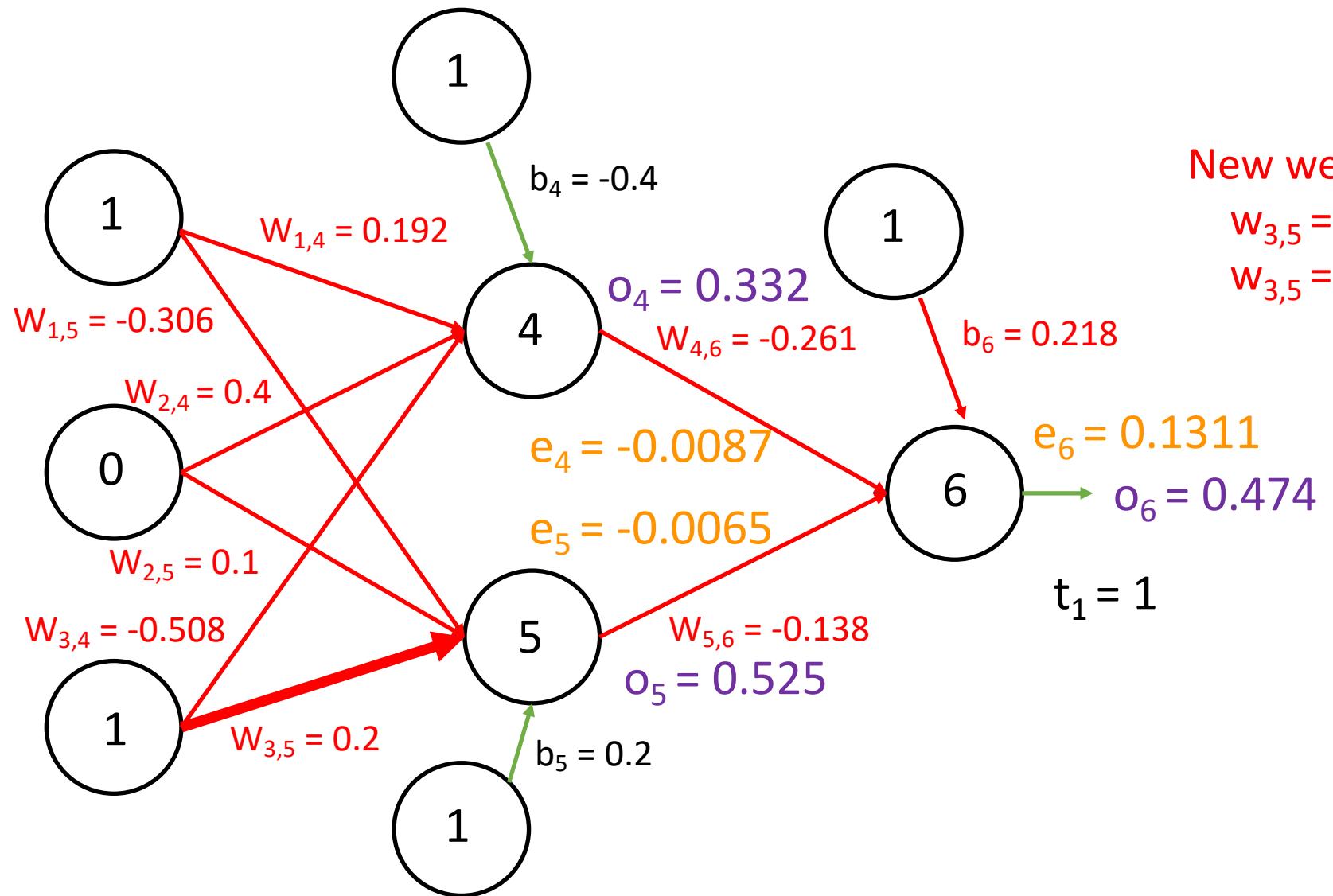
New weights (learning rate = 0.9):

$$w_{3,4} = -0.5 + (0.9)(-0.0087)(1)$$

$$w_{3,4} = -0.508$$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

Example: Step 3 – Update Weights



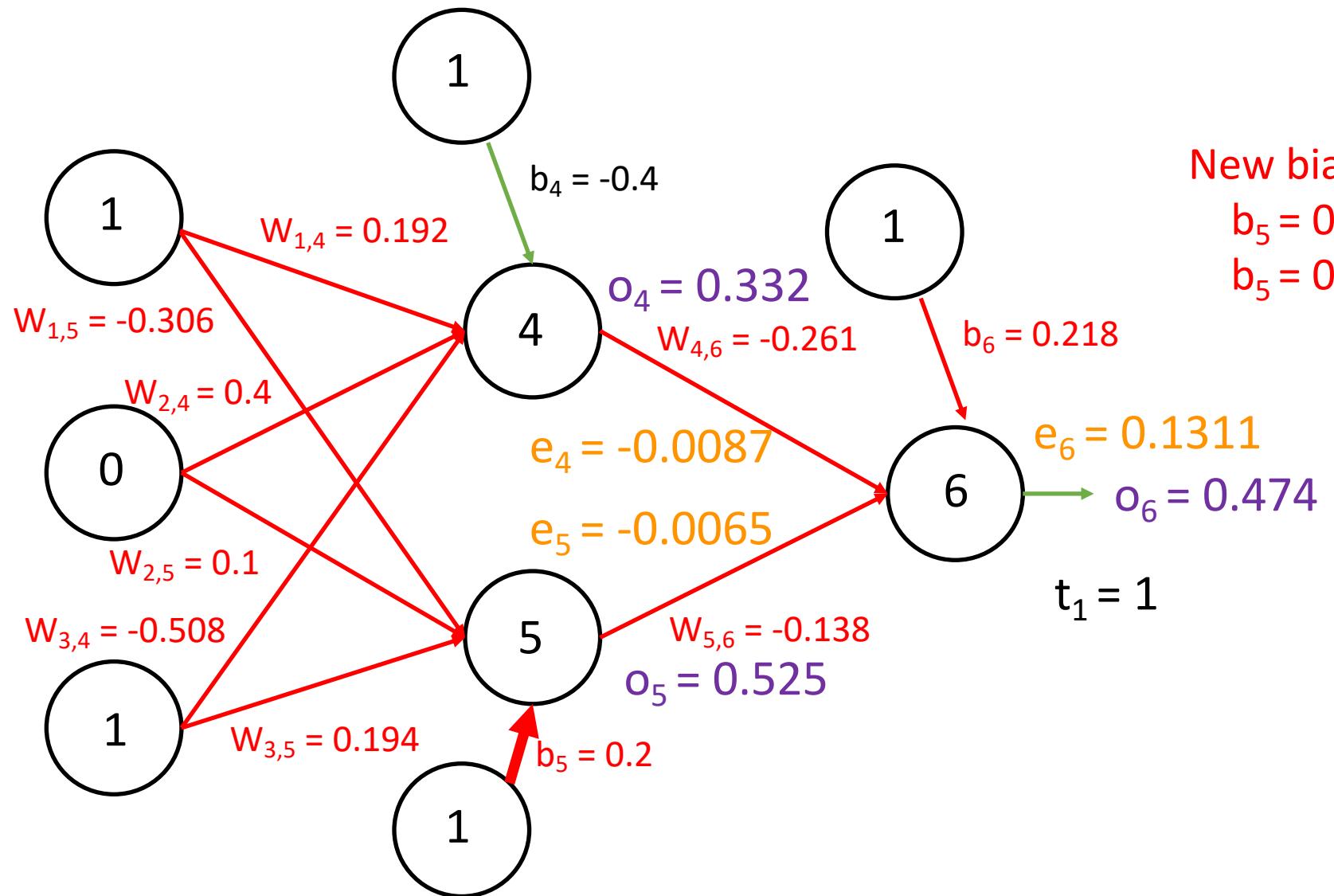
New weights (learning rate = 0.9):

$$w_{3,5} = 0.2 + (0.9)(-0.0065)(1)$$

$$w_{3,5} = 0.194$$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

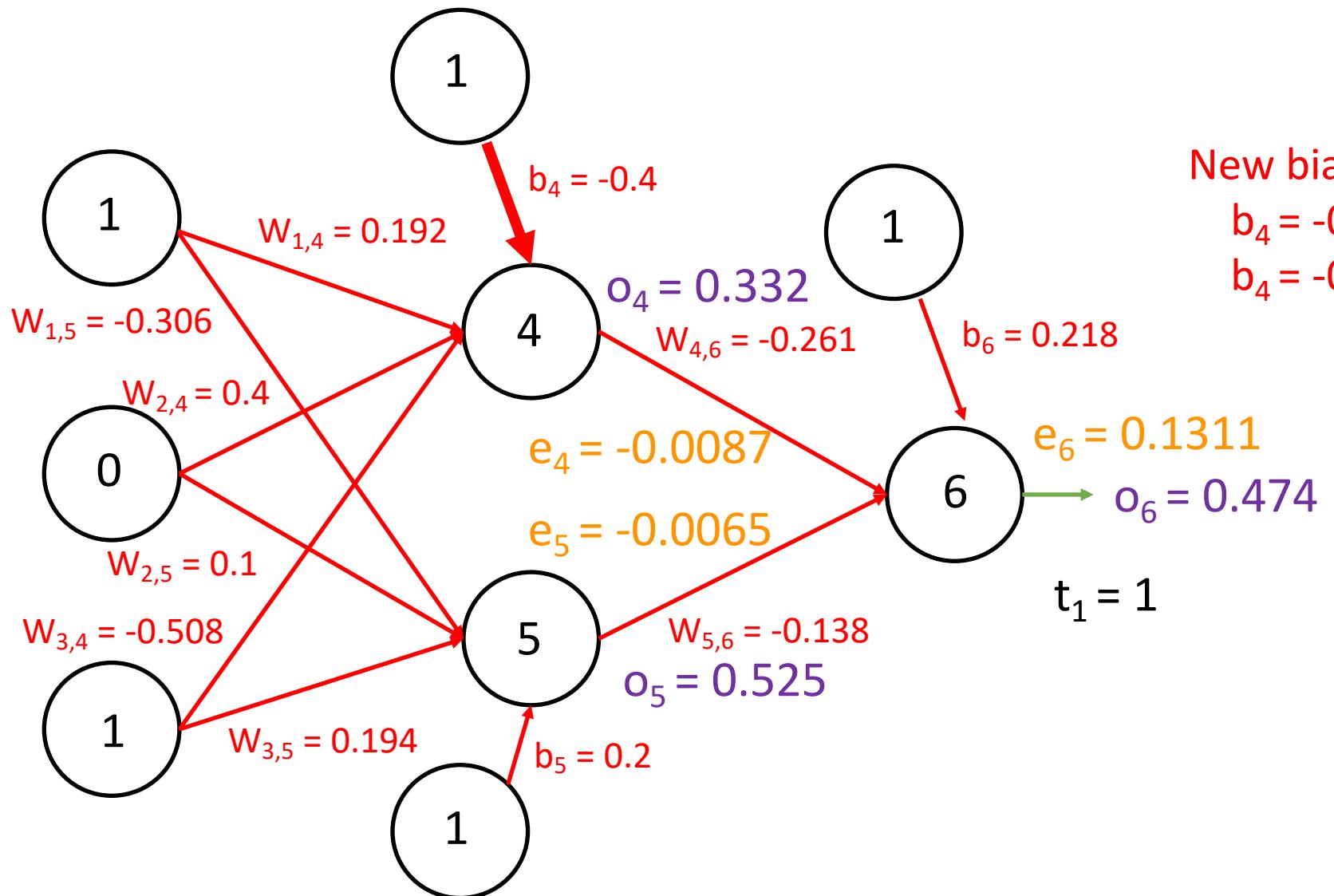
Example: Step 3 – Update Weights



New bias (learning rate = 0.9):
 $b_5 = 0.2 + (0.9)(-0.0065)$
 $b_5 = 0.194$

$$b_k = b_k + \eta Err_k$$

Example: Step 3 – Update Weights



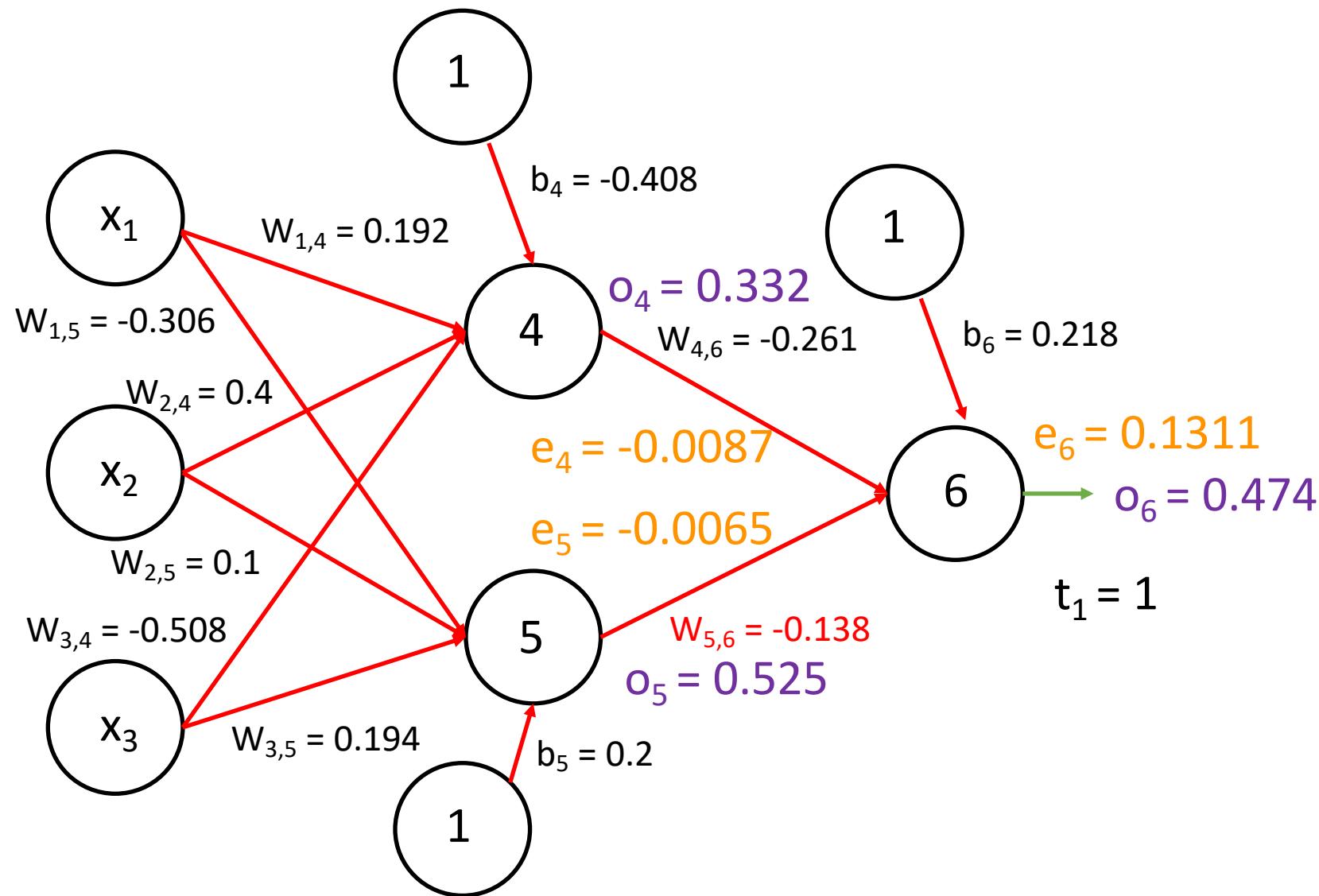
New bias (learning rate = 0.9):

$$b_4 = -0.4 + (0.9)(-0.0087)$$

$$b_4 = -0.408$$

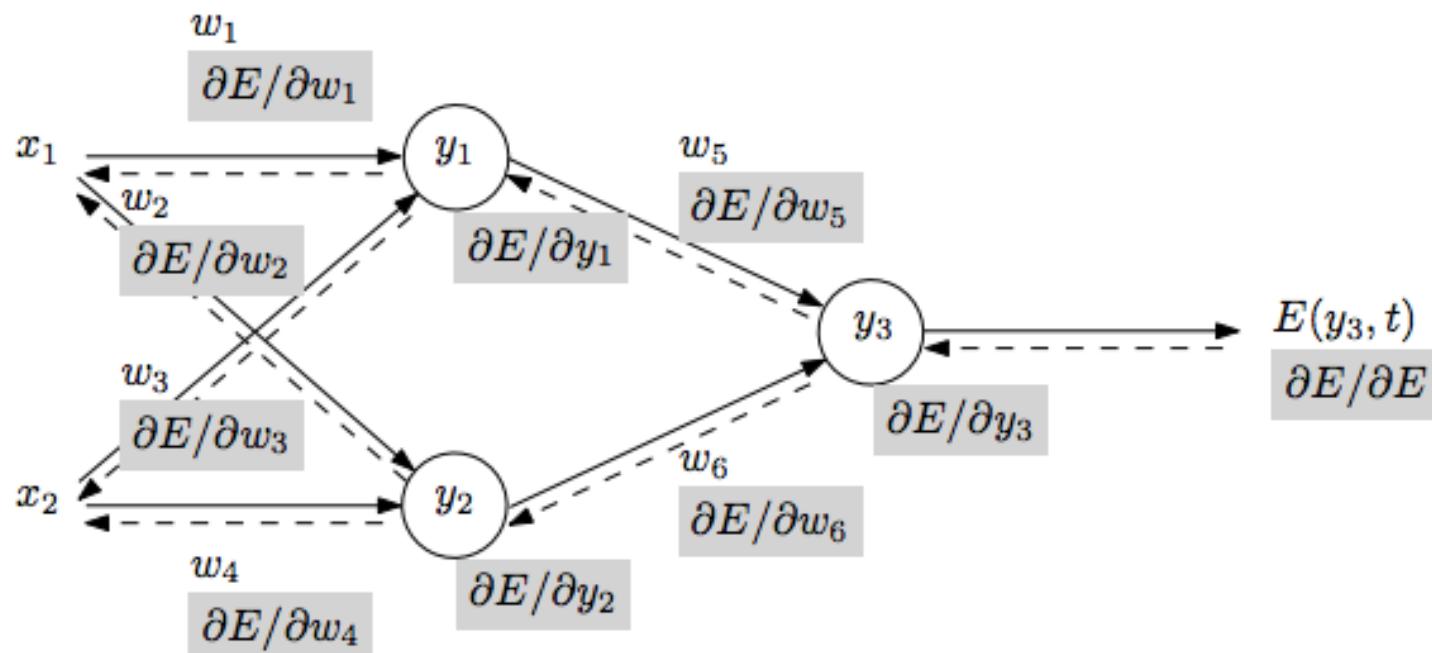
$$b_k = b_k + \eta E r r_k$$

Repeat Steps 1-3 With New Examples



Repeat Steps 1-3 With New Examples

(a) Forward pass



(b) Backward pass

- Repeat until stopping criterion met:
 1. **Forward pass:** propagate training data through network to make prediction
 2. **Backward pass:** using predicted output, calculate gradients backward
 3. Update each weight using calculated gradients