Neural Network Architecture and Training

Danna Gurari
University of Texas at Austin
Spring 2021

Review

- Last week:
  - Natural Language Processing
  - Computer Vision
  - Feature Representation
  - Dimensionality Reduction

- Assignments (Canvas):
  - Problem set 5 due tonight
  - Project pre-proposal due tonight
  - Lab assignment 3 due in two weeks

- Questions?

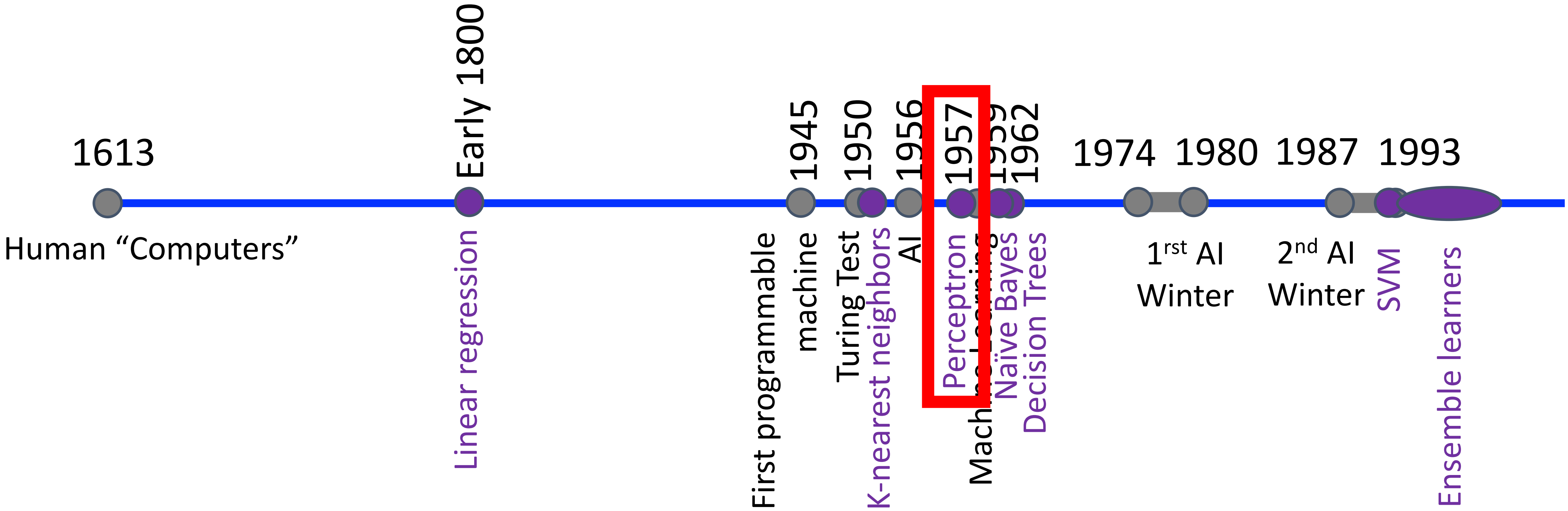
Today’s Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions

Today’s Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions

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."

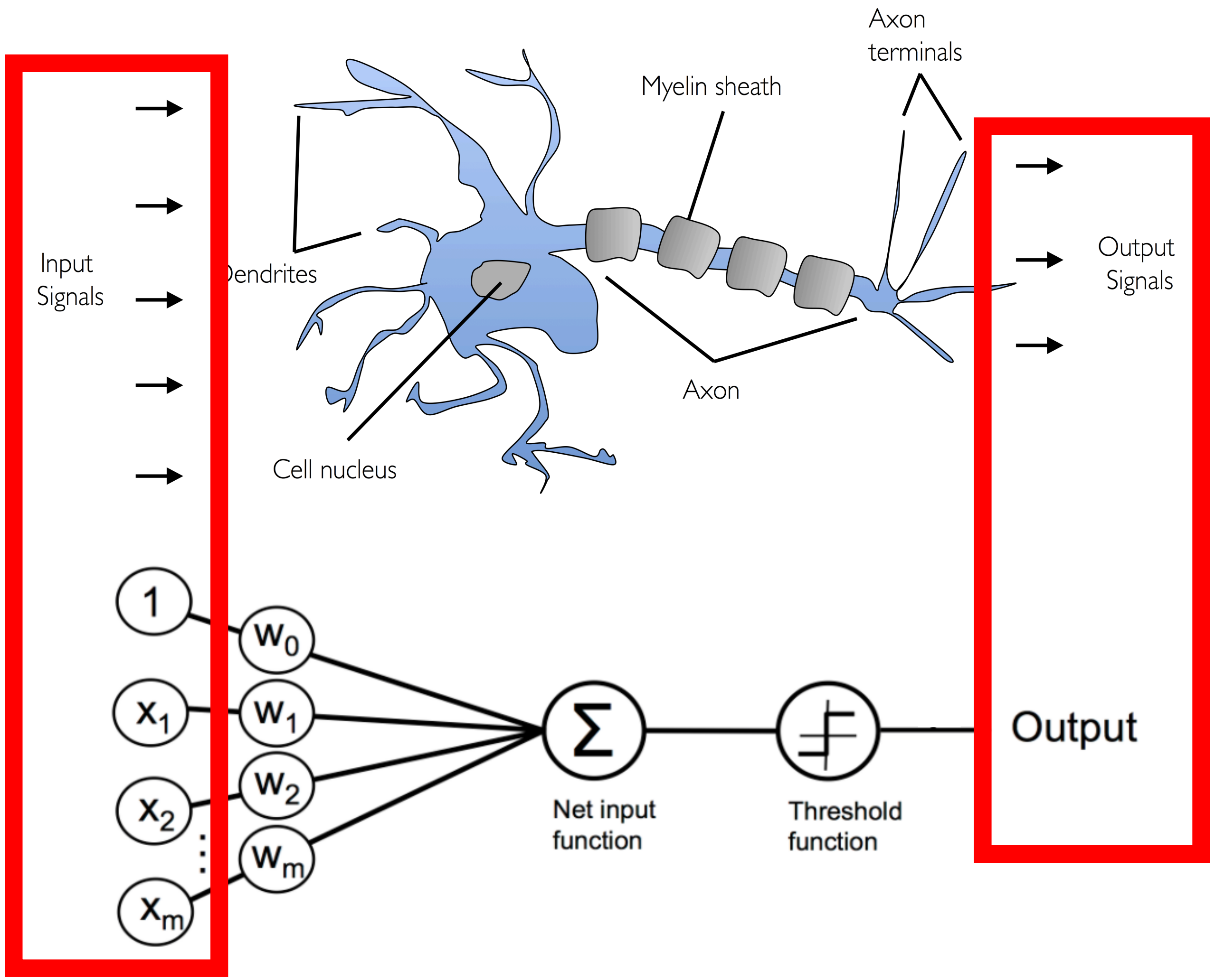
1958 New York Times article: https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html

https://en.wikipedia.org/wiki/Frank_Rosenblatt

Recall: Rise & Fall of Perceptron (Artificial Neuron)

Biological Neuron:

Artificial Neurons (e.g., Perceptron):

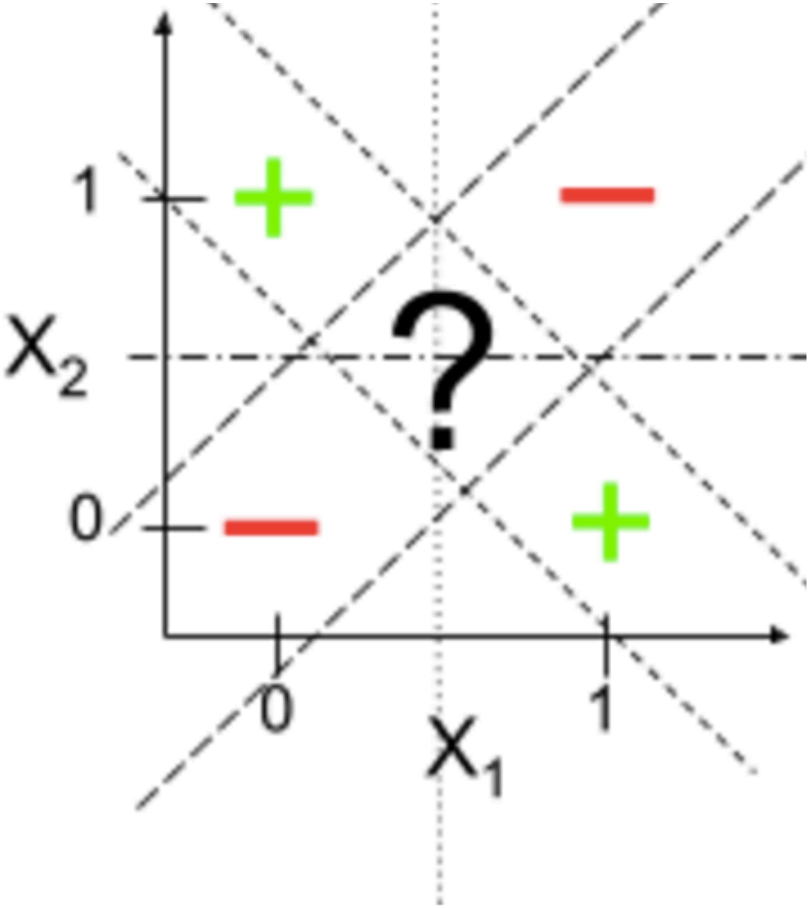


Python Machine Learning; Raschka & Mirjalili

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch02/ch02.ipynb

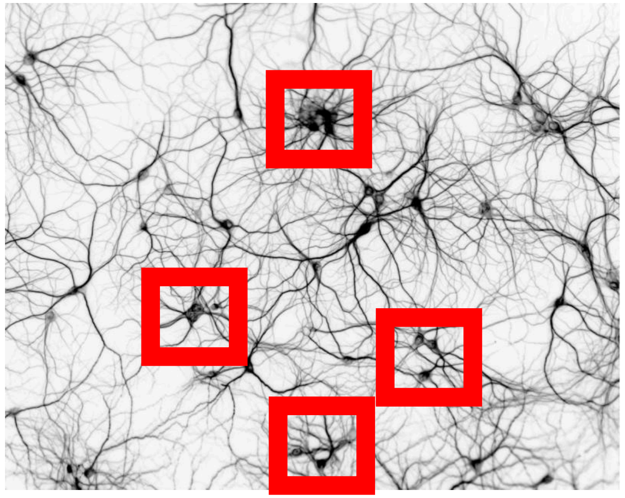
Recall: Rise & Fall of Perceptron (Artificial Neuron)

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



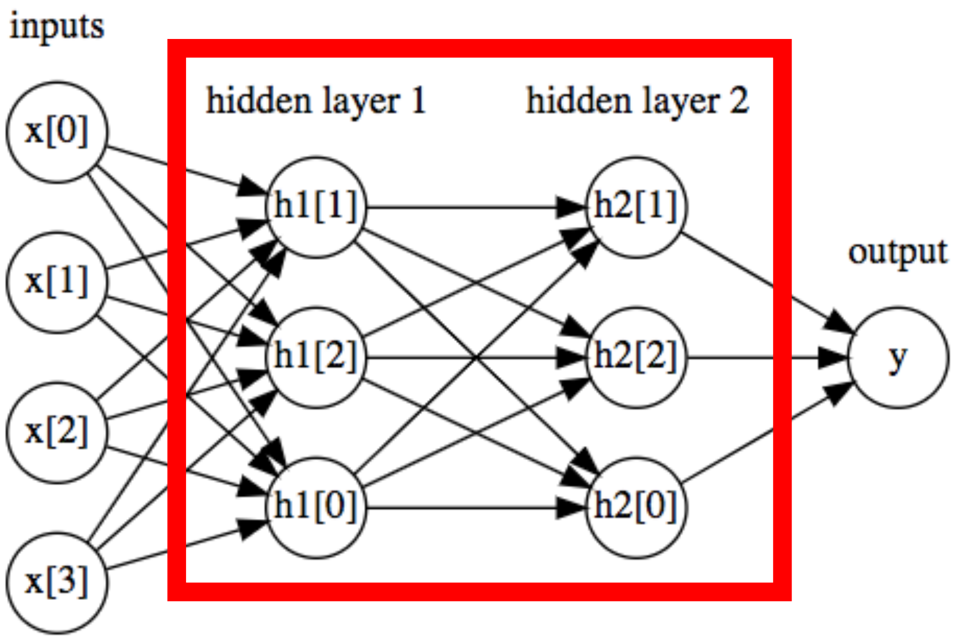
Neural Networks: Connected Neurons

Biological Neural Network:



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

Artificial Neural Network:

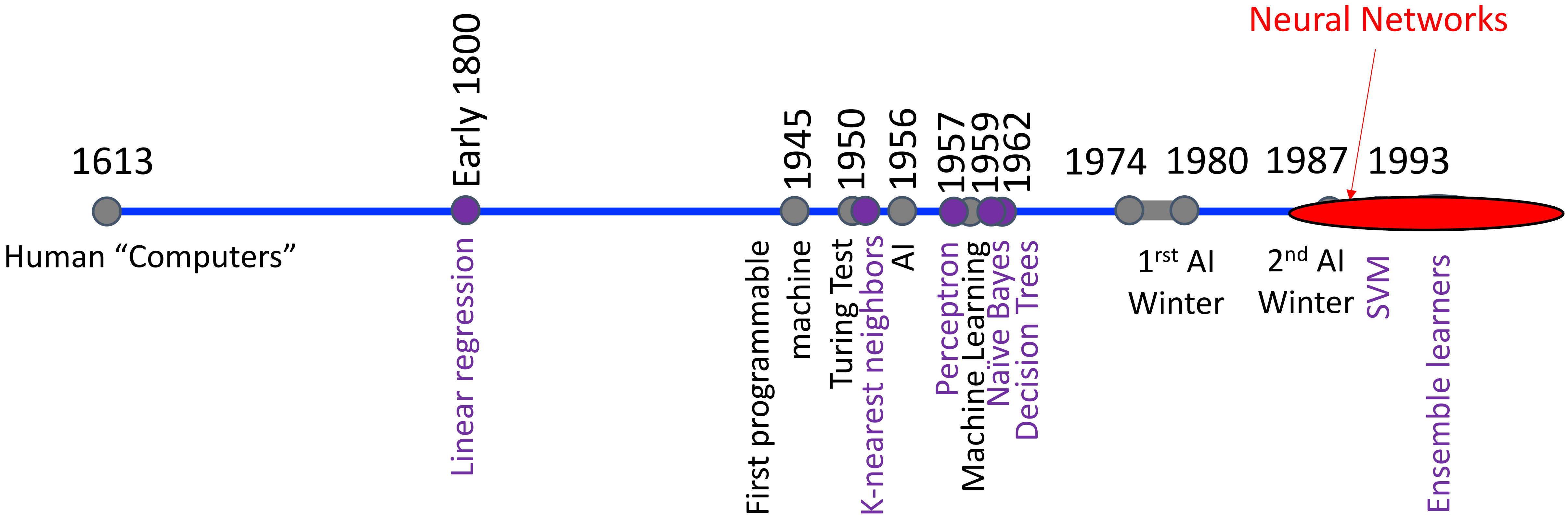


https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb

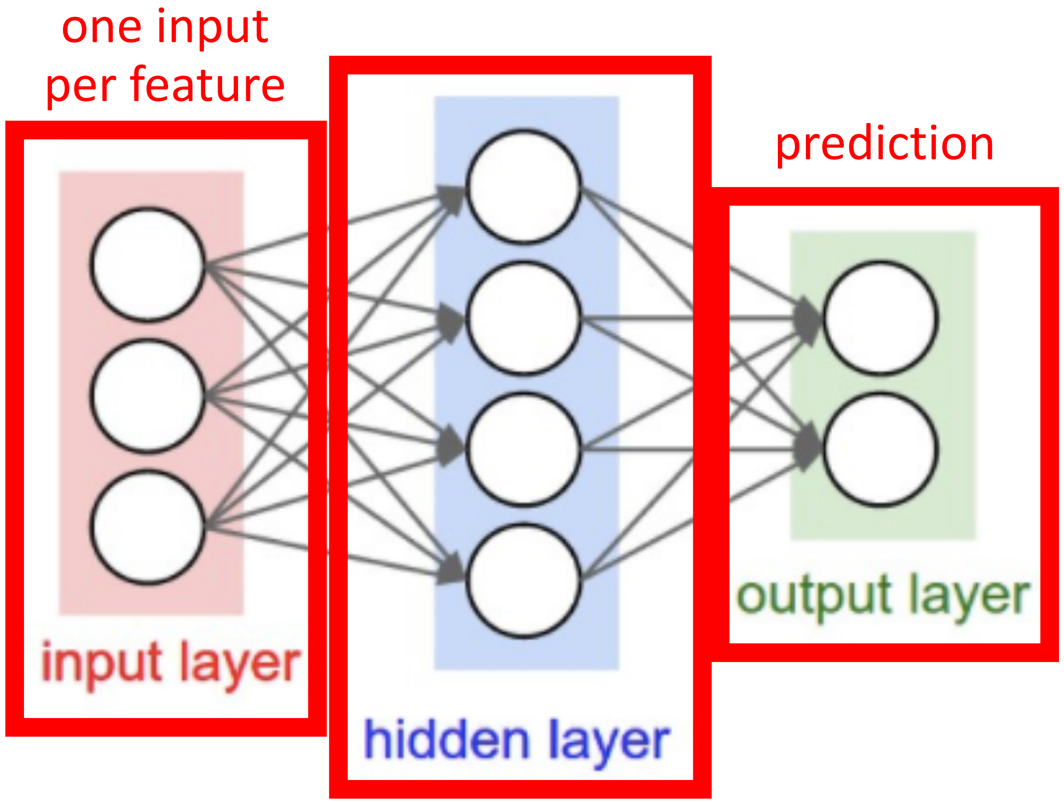
Today’s Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions

Today's Topic: Neural Networks



Neural Network

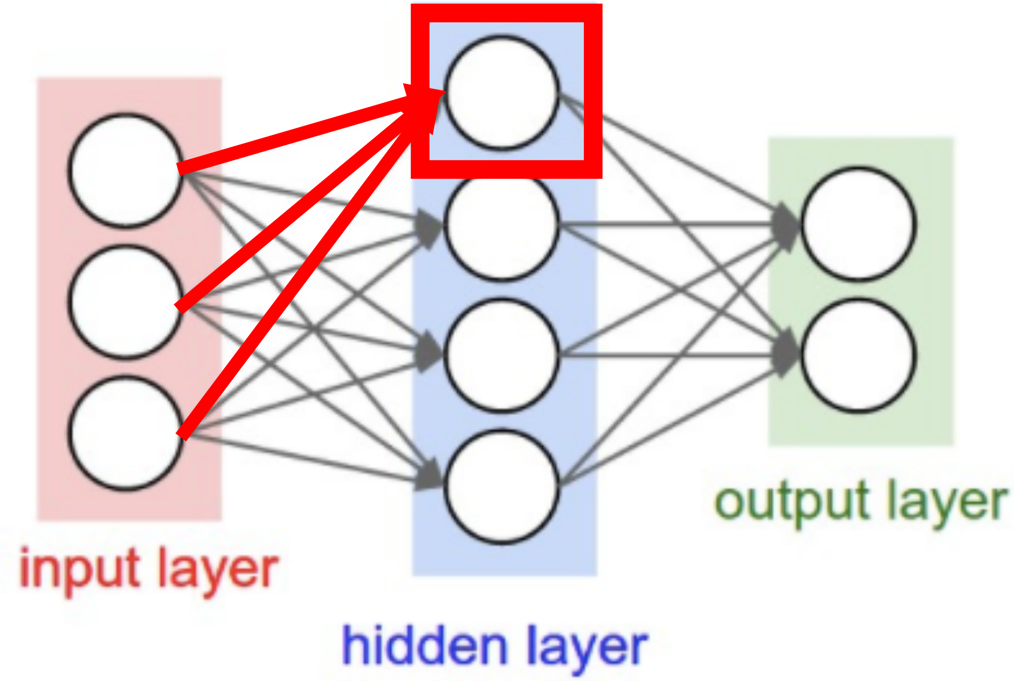


"hidden layer" uses outputs of units (i.e., neurons) and provides them as inputs to other units (i.e., neurons)

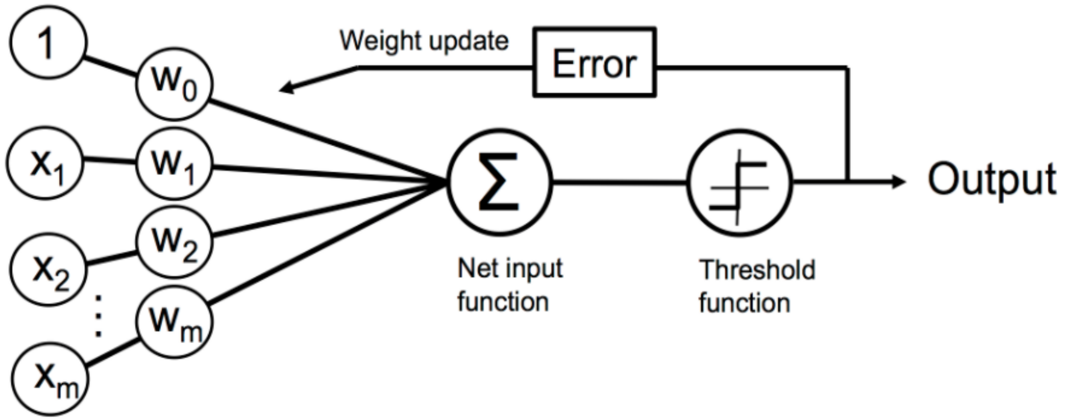
- Also called "multilayer perceptron"
- This is a 2-layer neural network (i.e., count number of hidden layers plus output layer and exclude input layer)

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

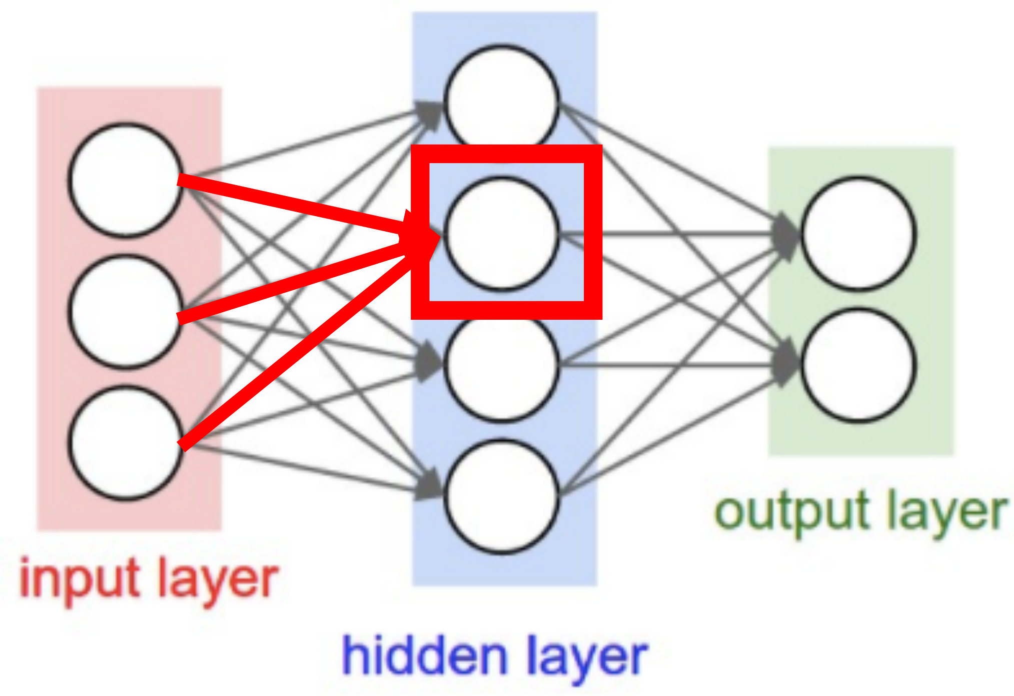


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

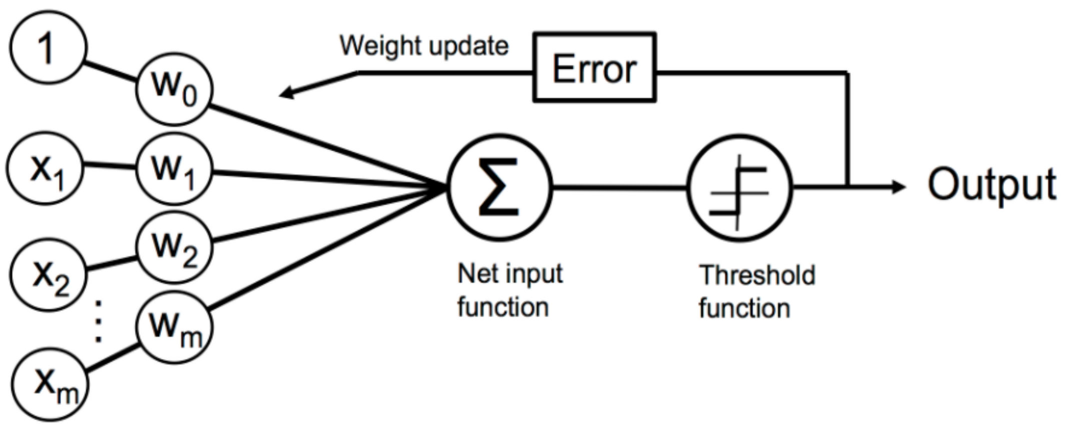
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

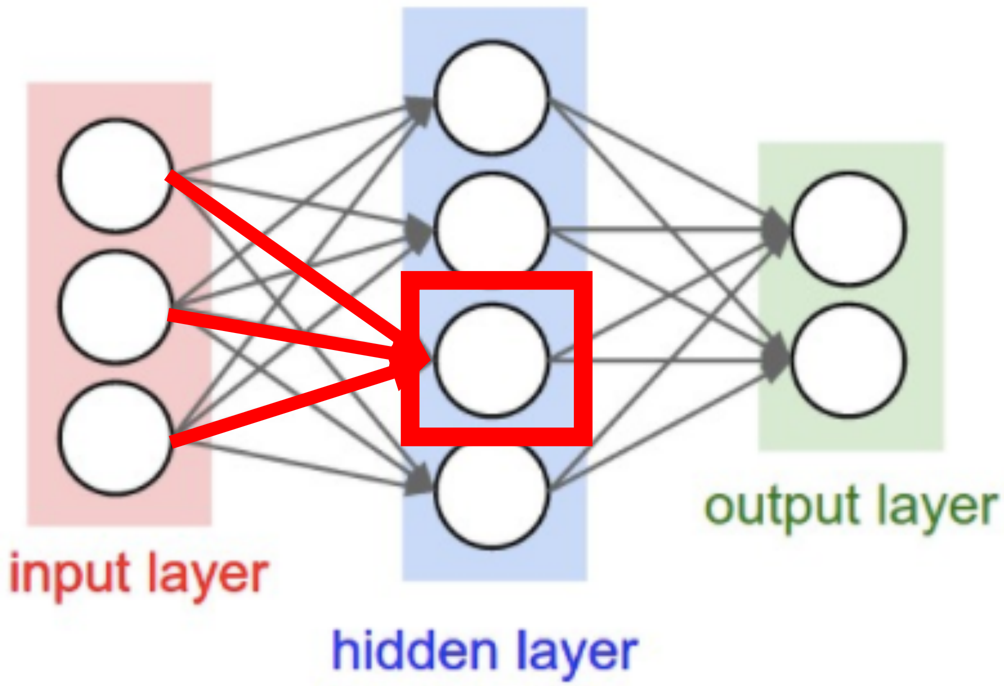


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

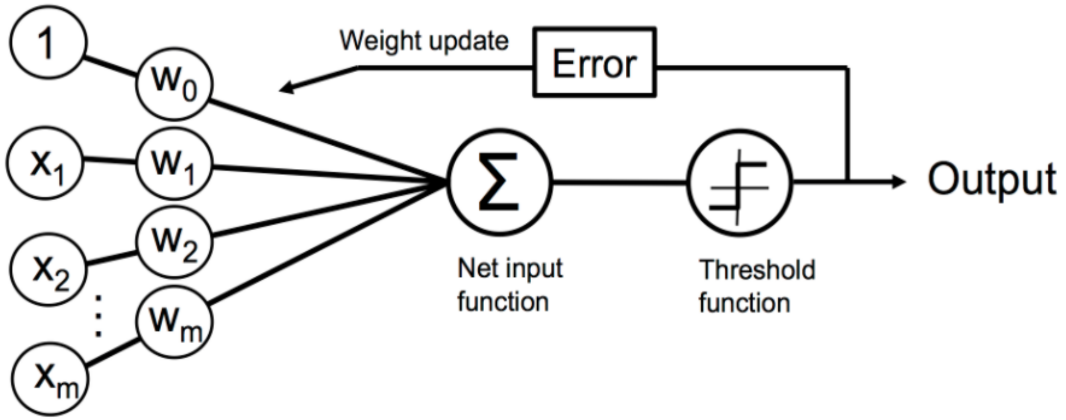
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

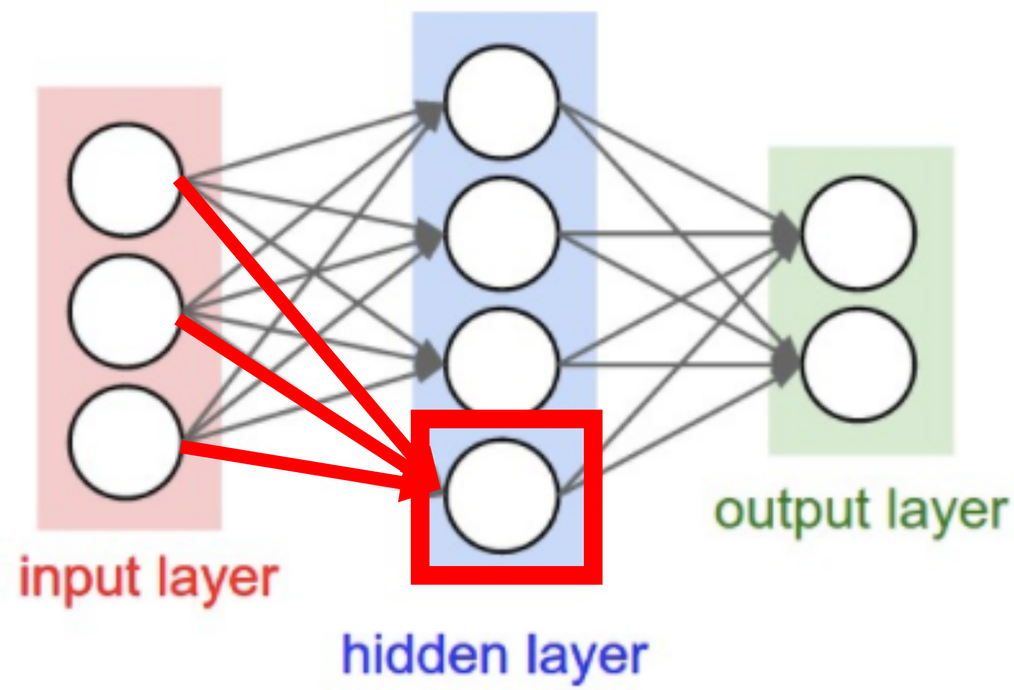


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

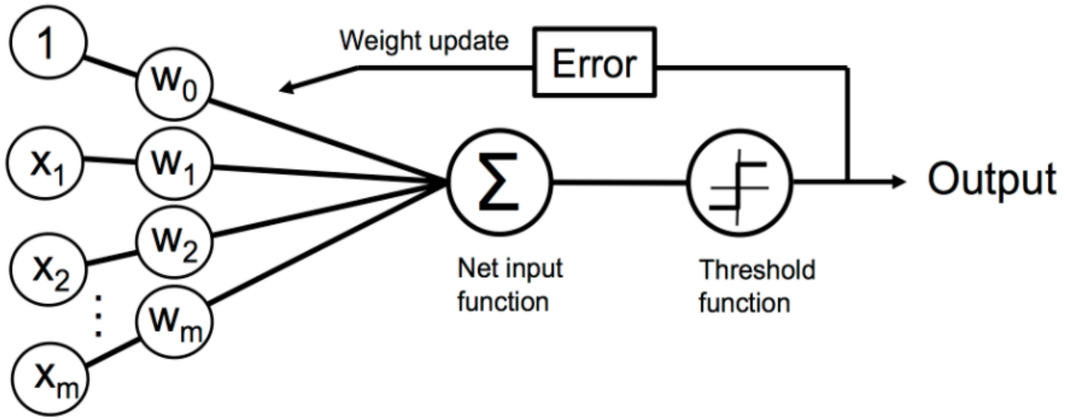
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

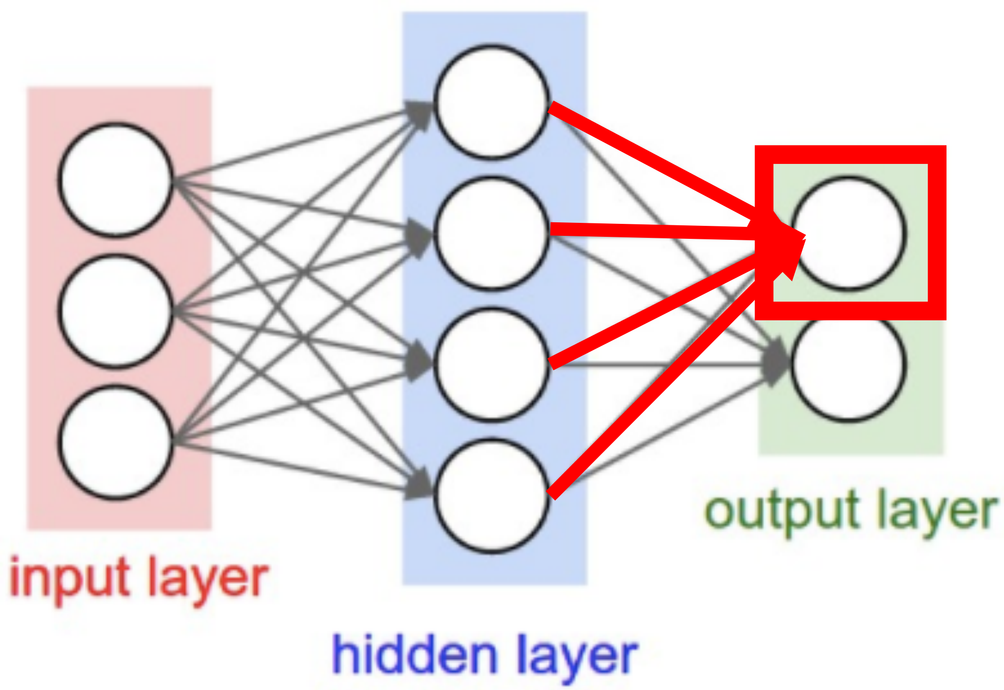


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

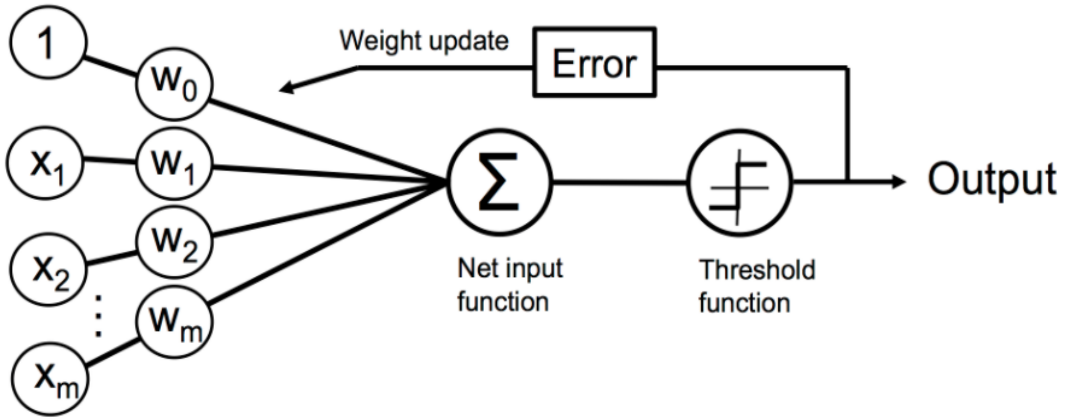
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

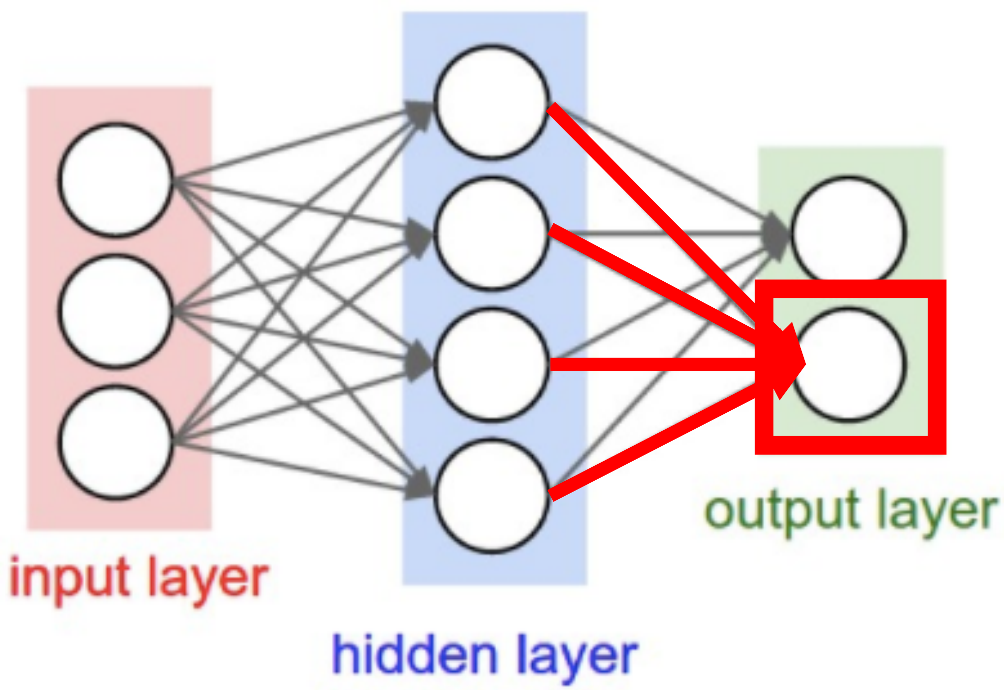


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

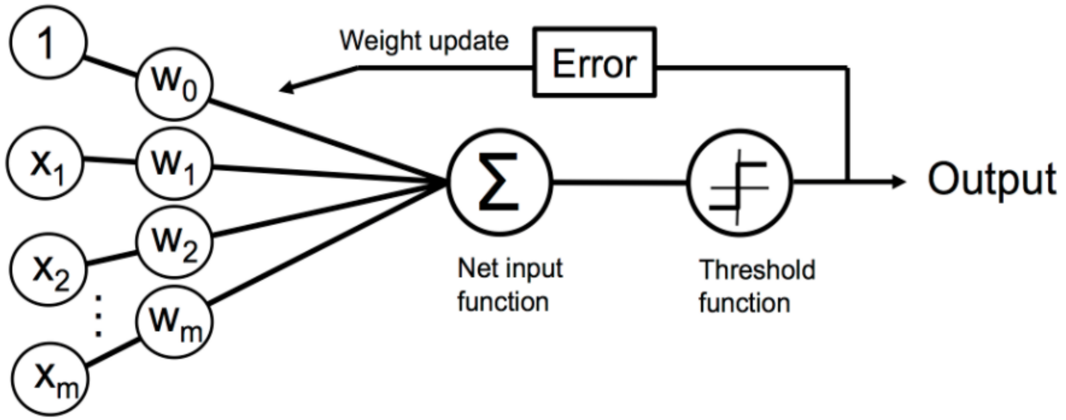
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

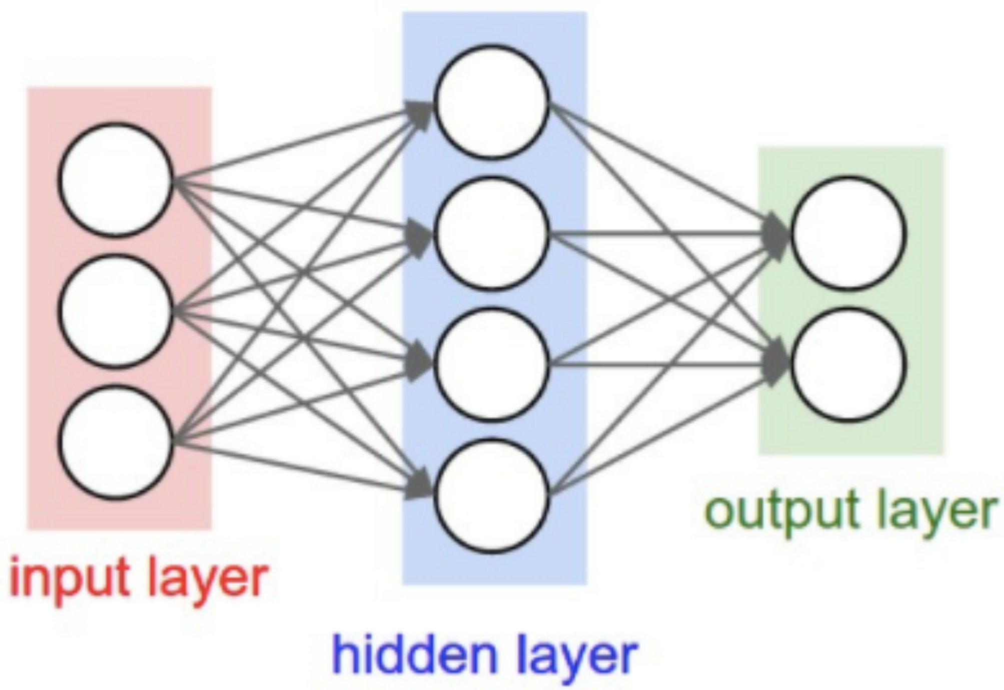


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

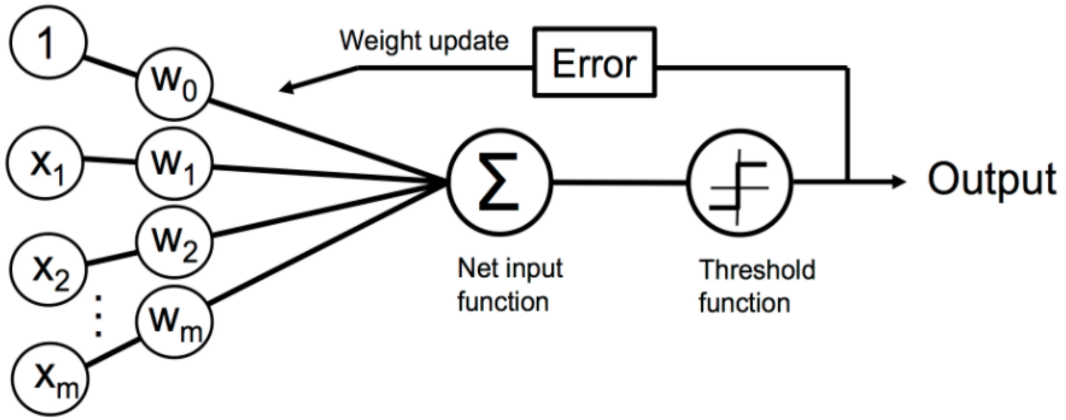
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

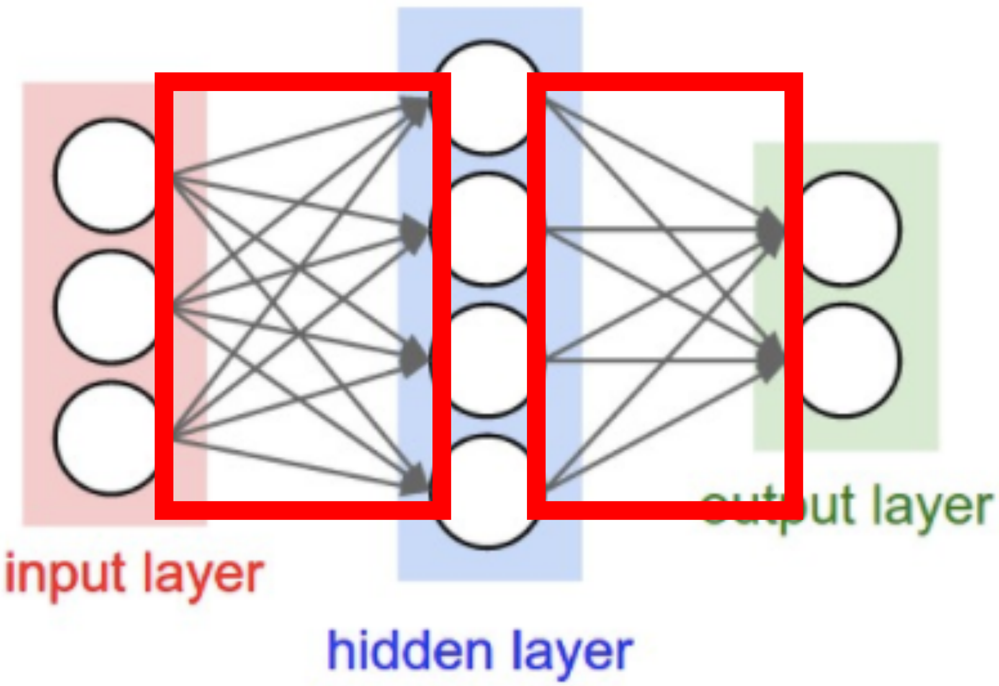


- **Training goal: learn model weights**

Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network

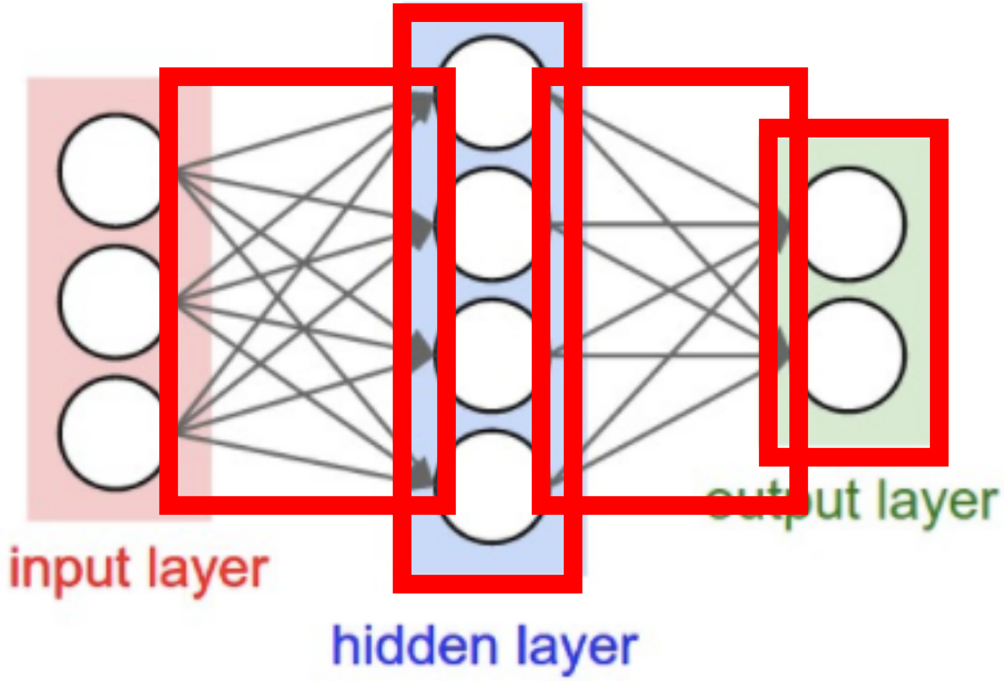


How many weights are in this model?

- Input to Hidden Layer:
  - 3x4 = 12
- Hidden Layer to Output Layer
  - 4x2 = 8
- Total:
  - 12 + 8 = 20

http://cs231n.github.io/neural-networks-1/

Neural Network

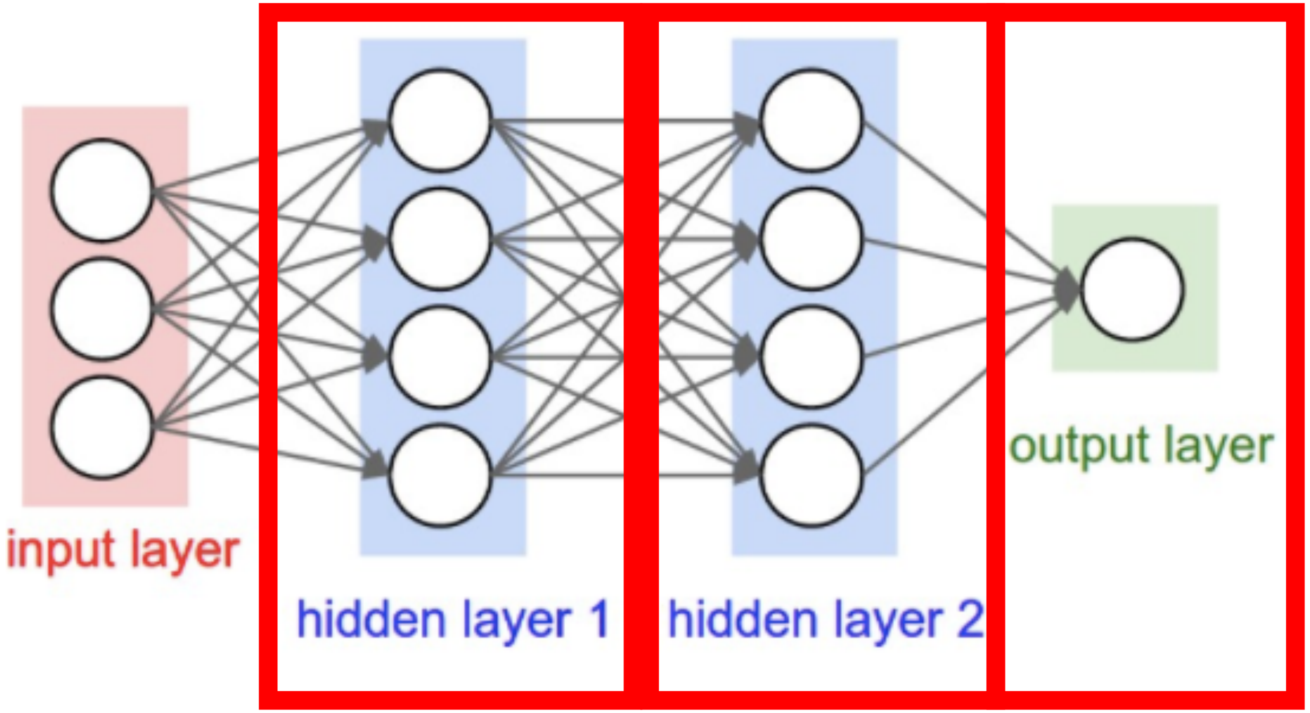


How many parameters are there to learn?

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

http://cs231n.github.io/neural-networks-1/

Neural Network

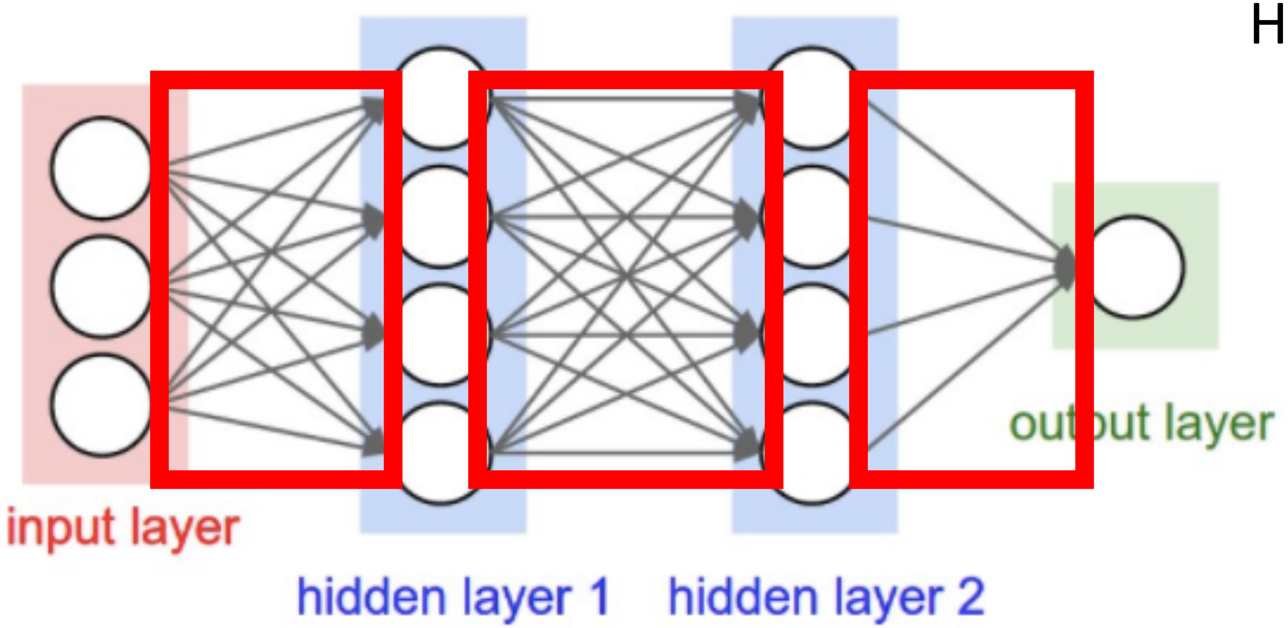


How many layers are in this network?

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

http://cs231n.github.io/neural-networks-1/

Neural Network

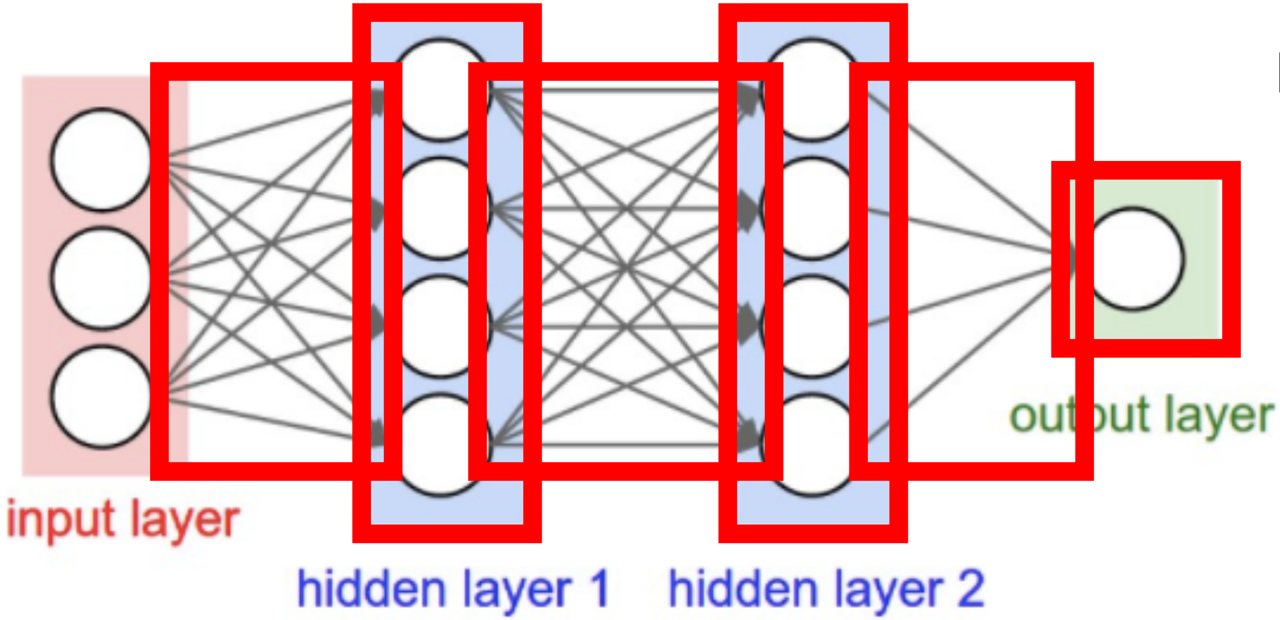


How many weights are in this model?

- Input to Hidden Layer 1:
  - 3x4 = 12
- Hidden Layer 1 to Hidden Layer 2:
  - 4x4 = 16
- Hidden Layer 2 to Output Layer
  - 4x1 = 4
- Total:
  - 12 + 16 + 4 = 32

http://cs231n.github.io/neural-networks-1/

Neural Network

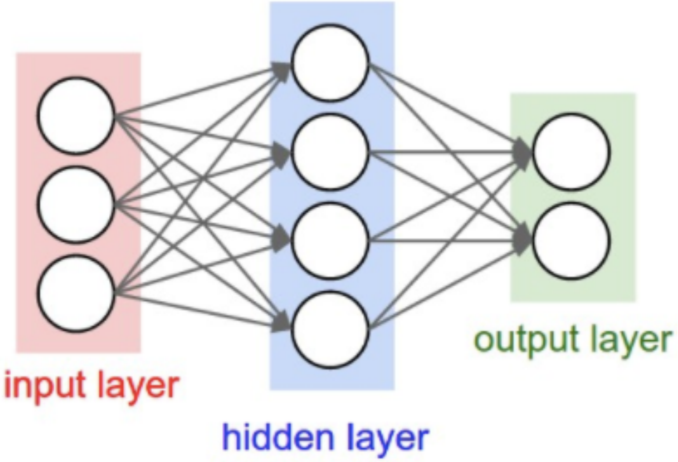


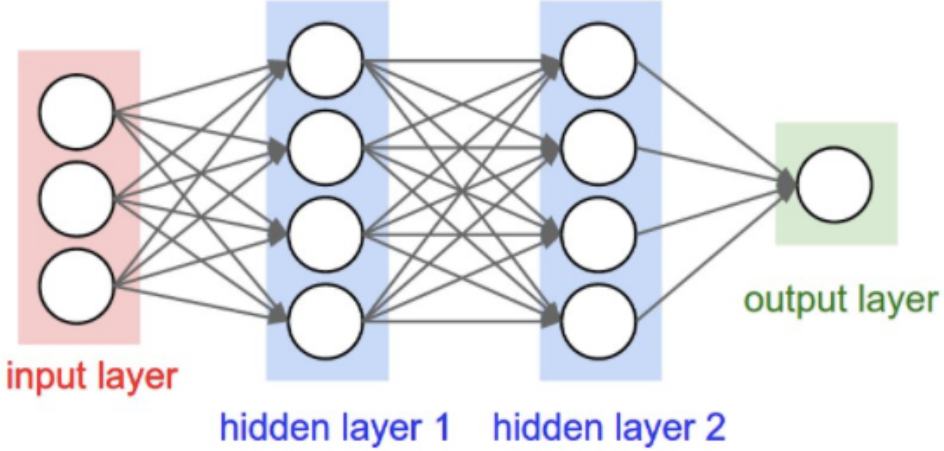
How many parameters are there to learn?

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

http://cs231n.github.io/neural-networks-1/

Fully Connected, Feed Forward Neural Networks





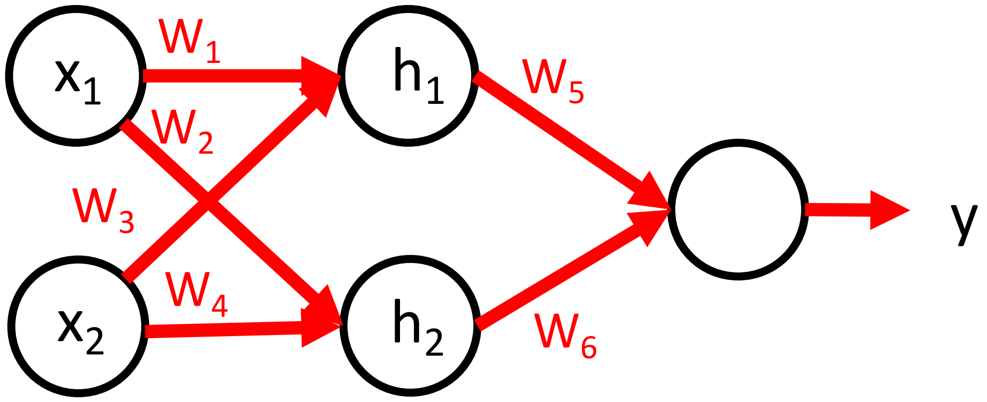
- What does it mean for a model to be fully connected?
  - Each unit provides input to each unit in the next layer
- What does it mean for a model to be feed forward?
  - Each layer serves as input to the next layer with no loops

http://cs231n.github.io/neural-networks-1/

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

Key Observation: feedforward networks are just functions chained together

e.g.,



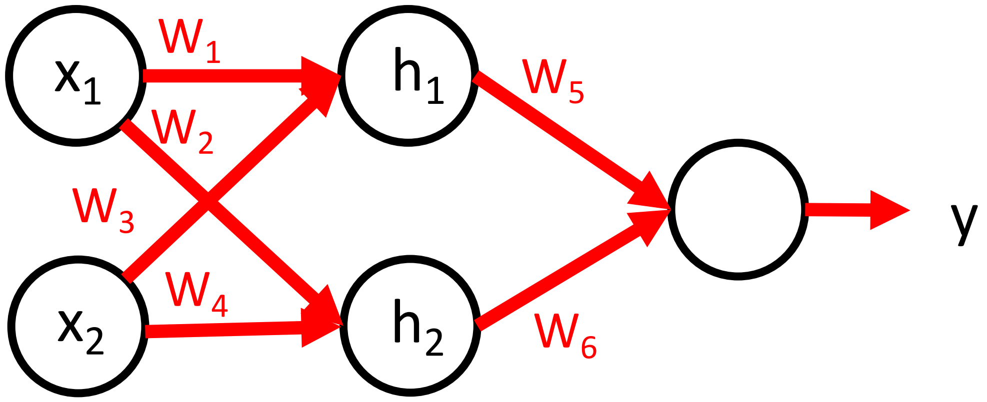
- What is function for h_1?
  - $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for h_2?
  - $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for y?
  - $y = h_1w_5 + h_2w_6 + b_3$
  - $y = (w_1x_1 + w_3x_2 + b_1)w_5 + (w_2x_1 + w_4x_2 + b_2)w_6 + b_3$
  - $\boxed{y = w_1w_5x_1 + w_3w_5x_2 + w_5b_1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

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.,



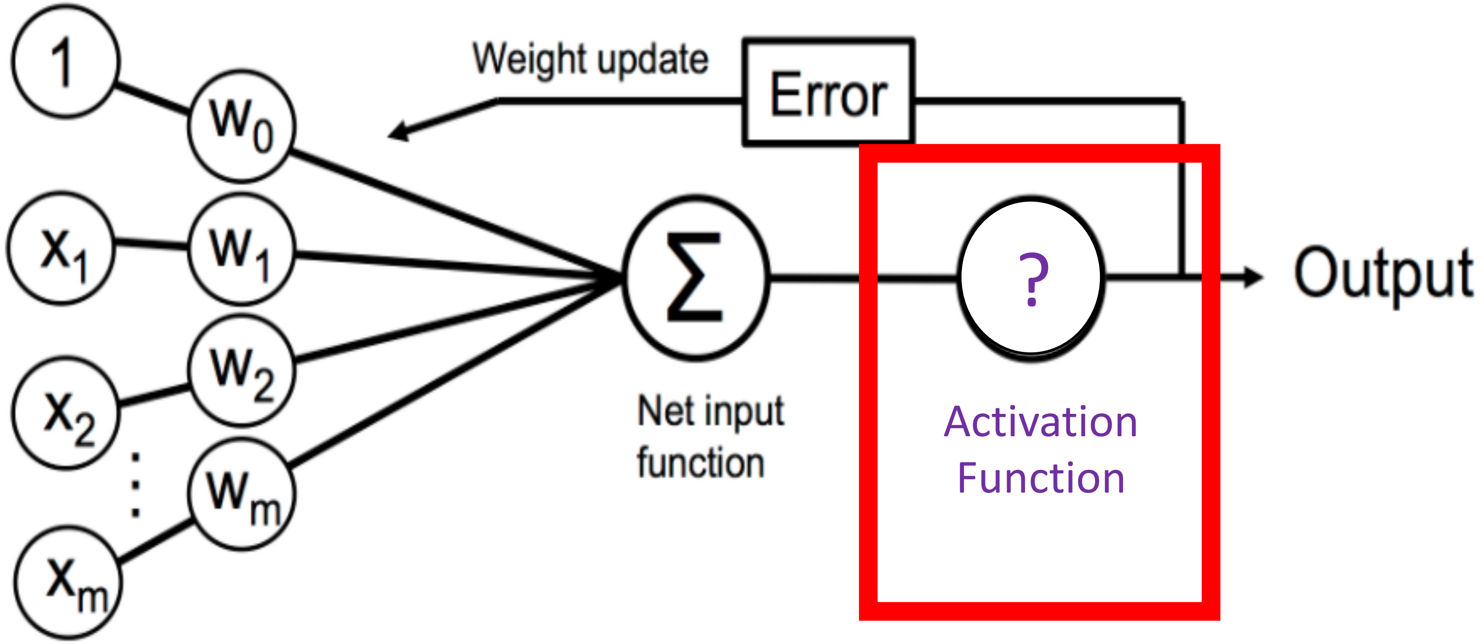
- What is function for h_1?
  - $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for h_2?
  - $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for y?
  - $y = \underbrace{h_1w_5} + \underbrace{h_2w_6} + b_3$

Constant x 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



Python Machine Learning; Raschka & Mirjalili

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

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

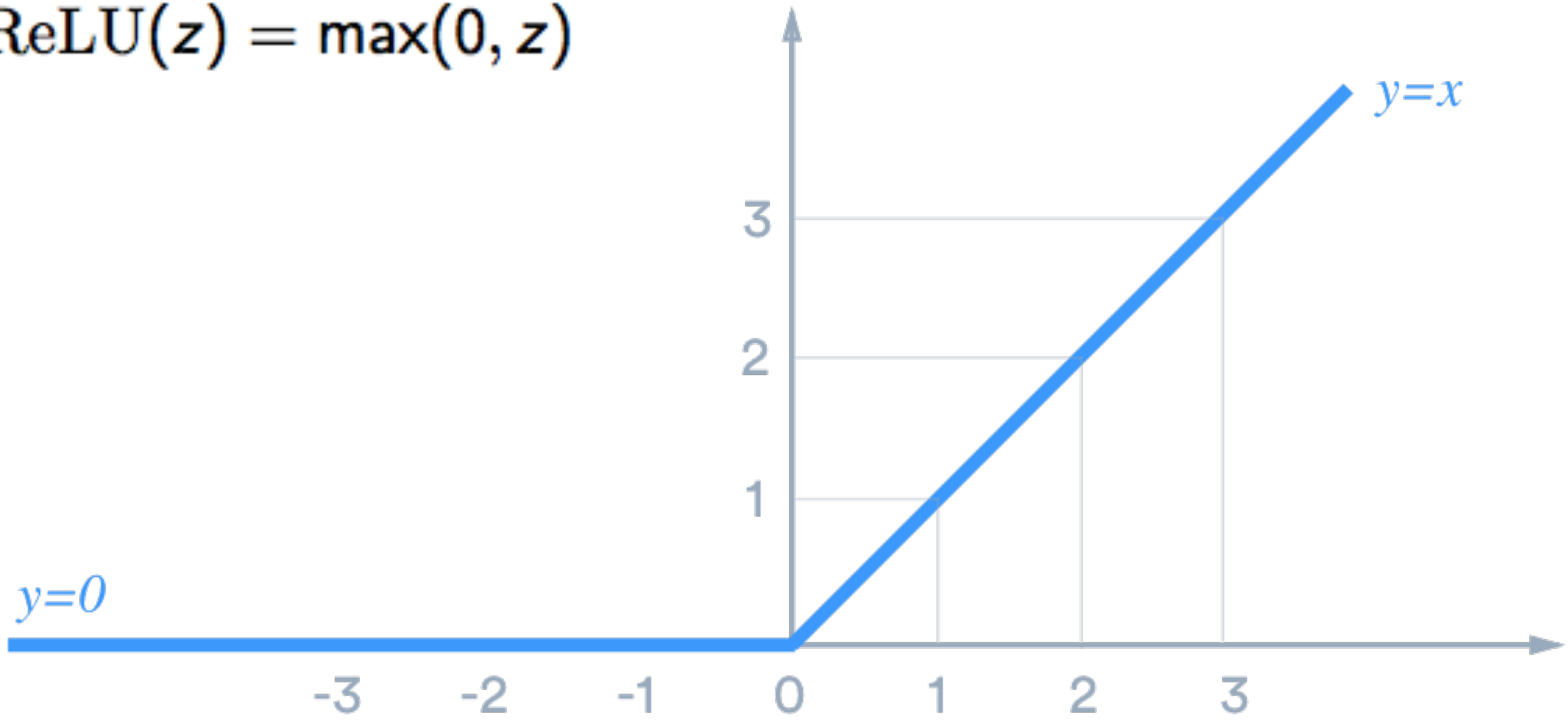
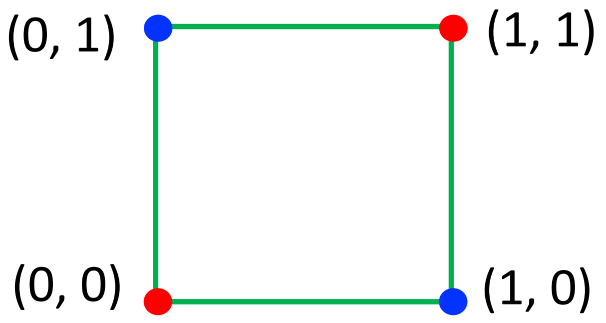
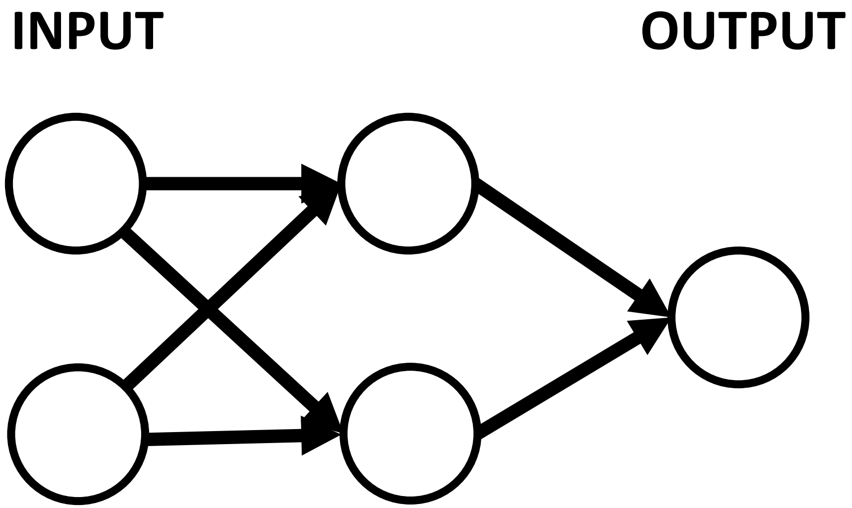


Image Source: https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda

Non-Linear Example: Revisiting XOR problem

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

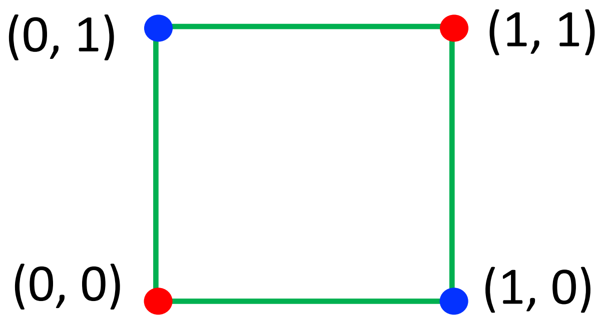




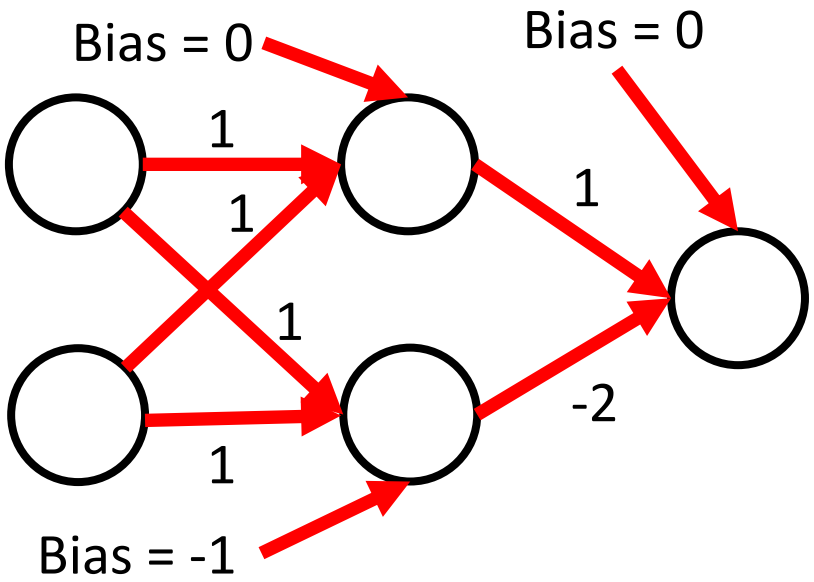
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



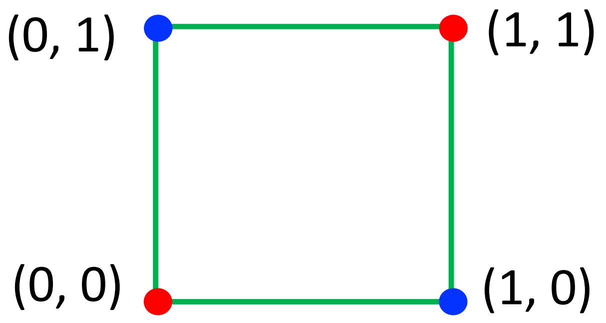
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



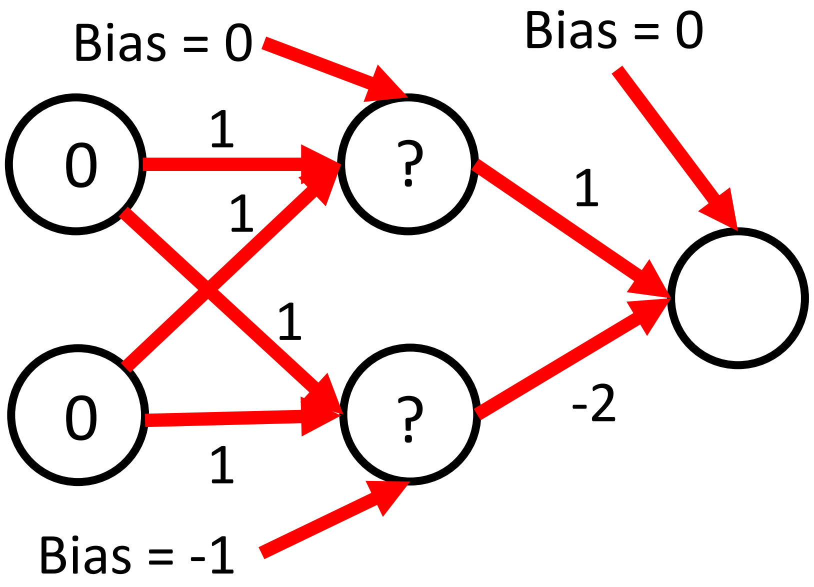
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



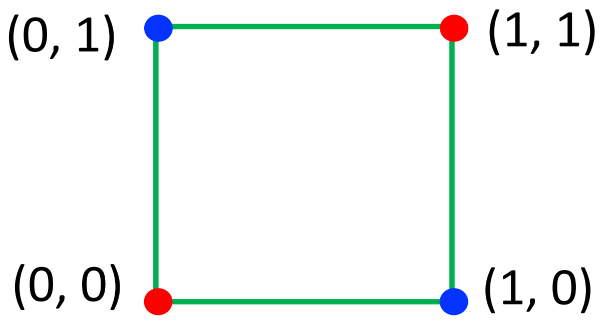
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



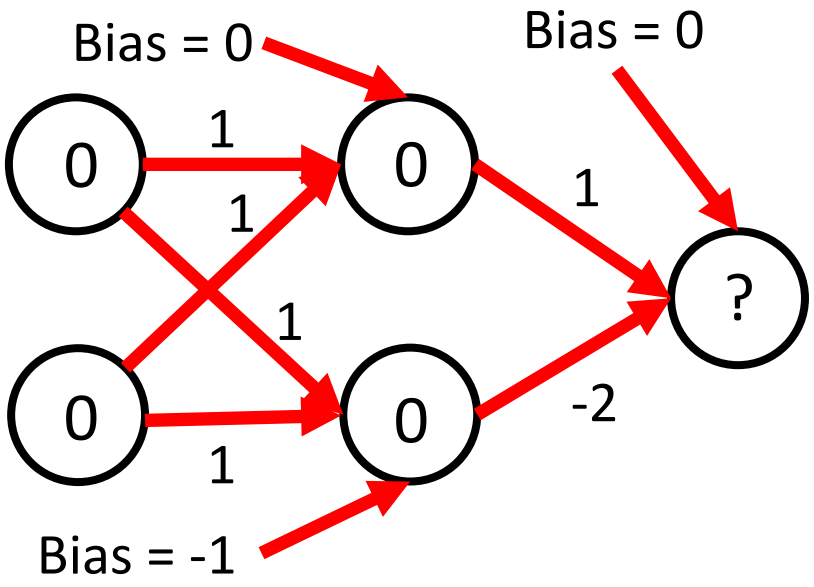
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



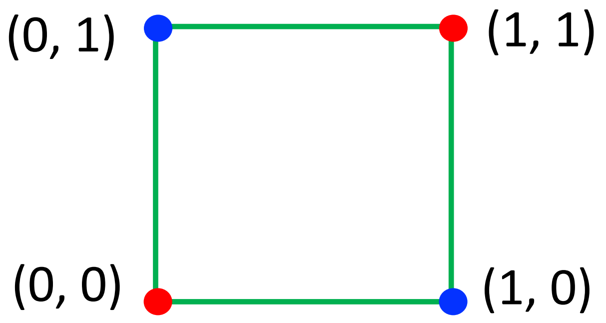
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



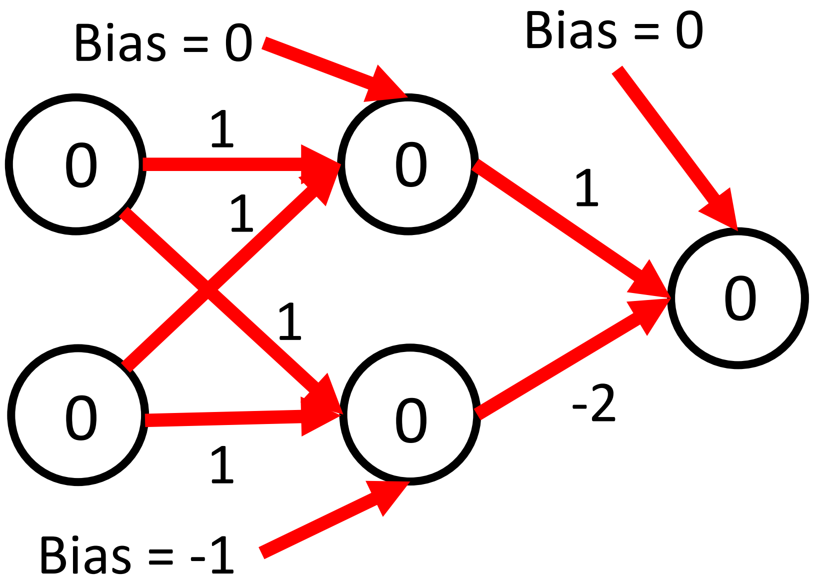
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



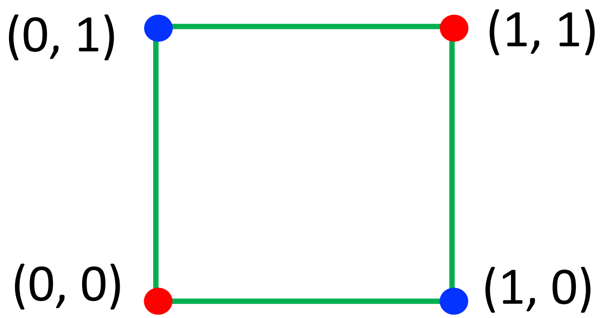
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



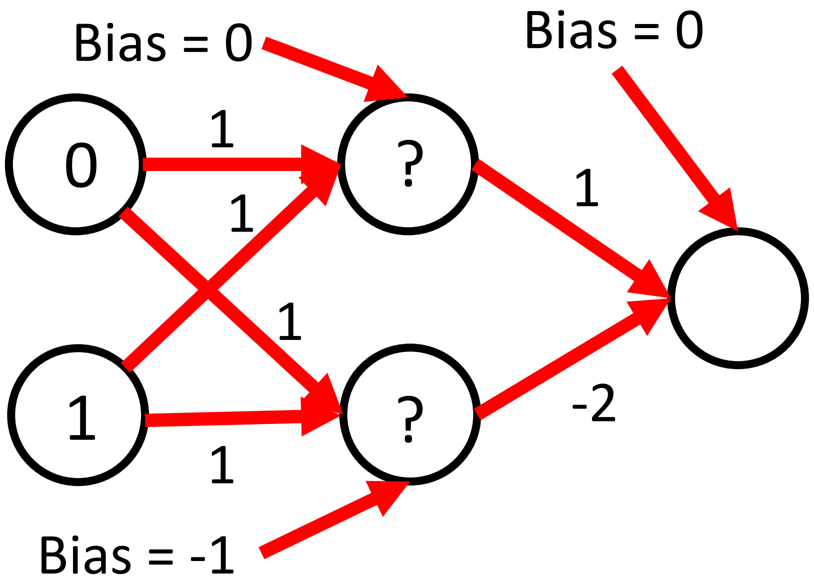
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



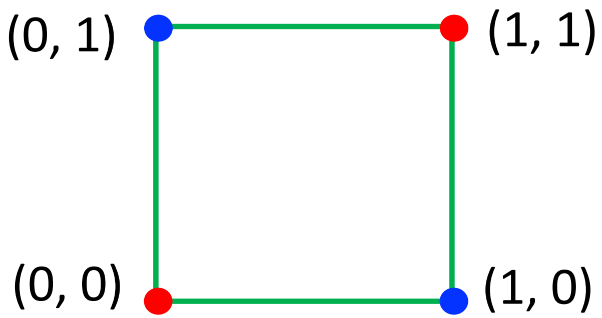
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



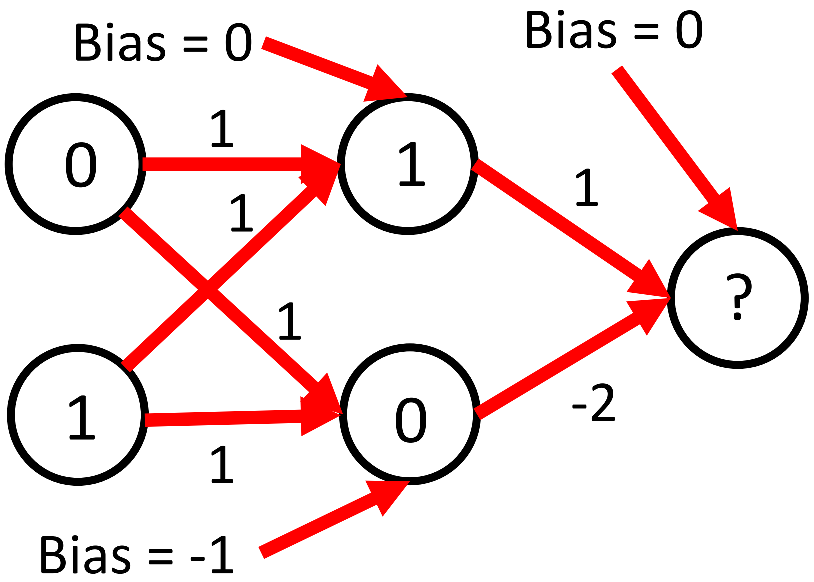
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



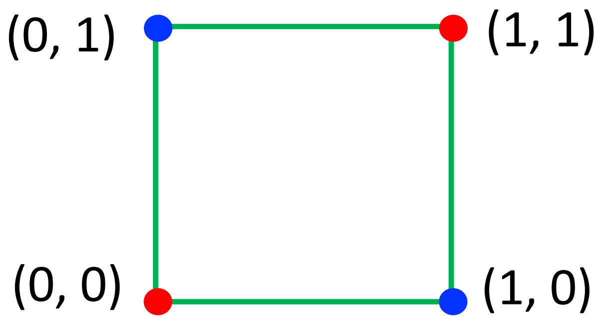
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



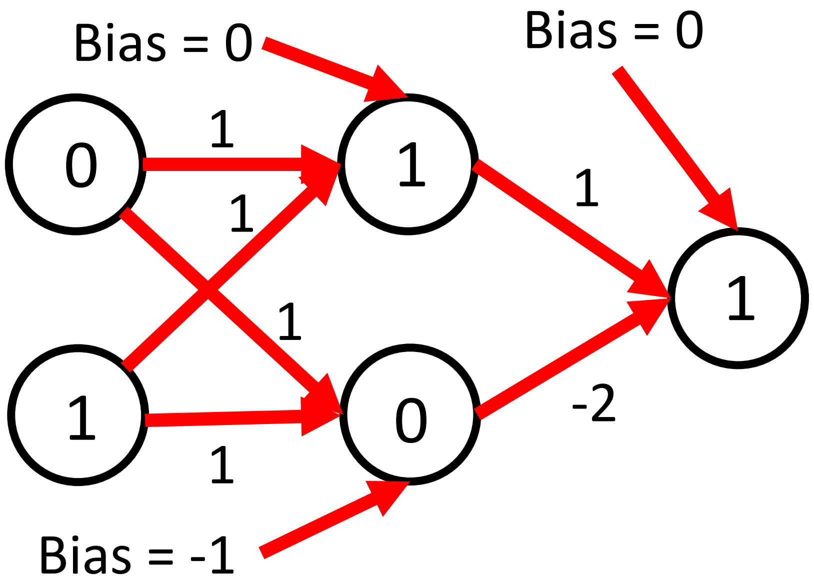
INPUT		OUTPUT
A	B	A XOR B
0	0	0
$\boxed{0}$	$\boxed{1}$	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



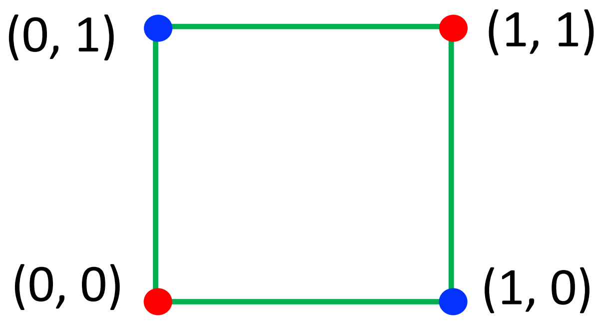
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



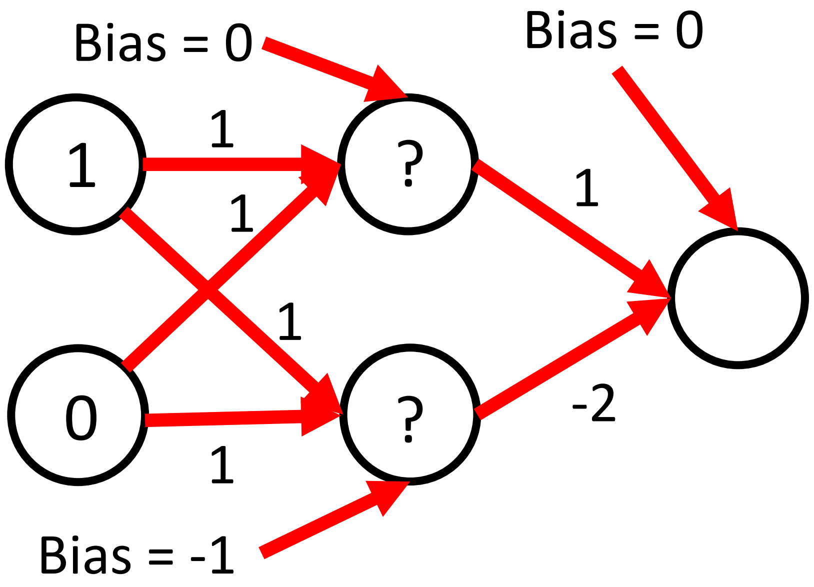
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



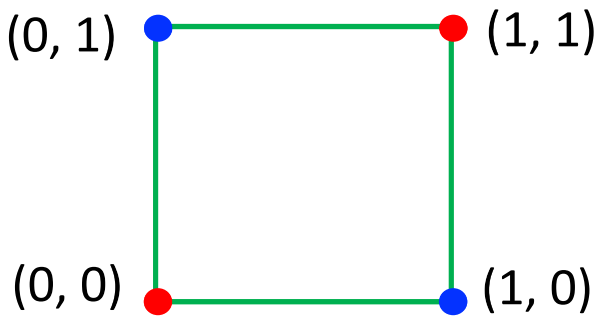
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



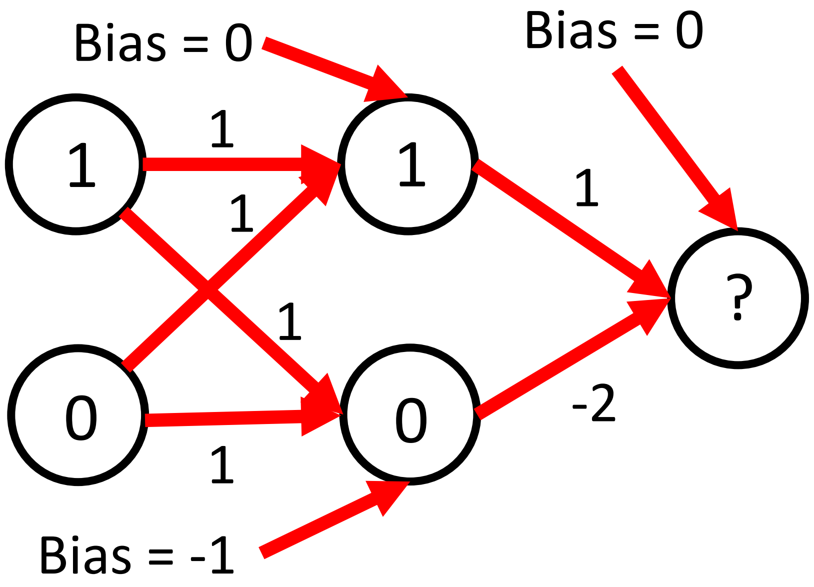
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



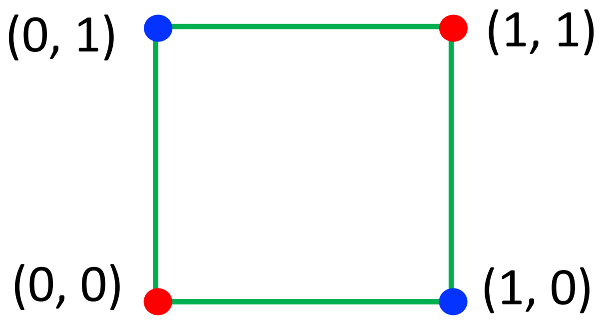
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



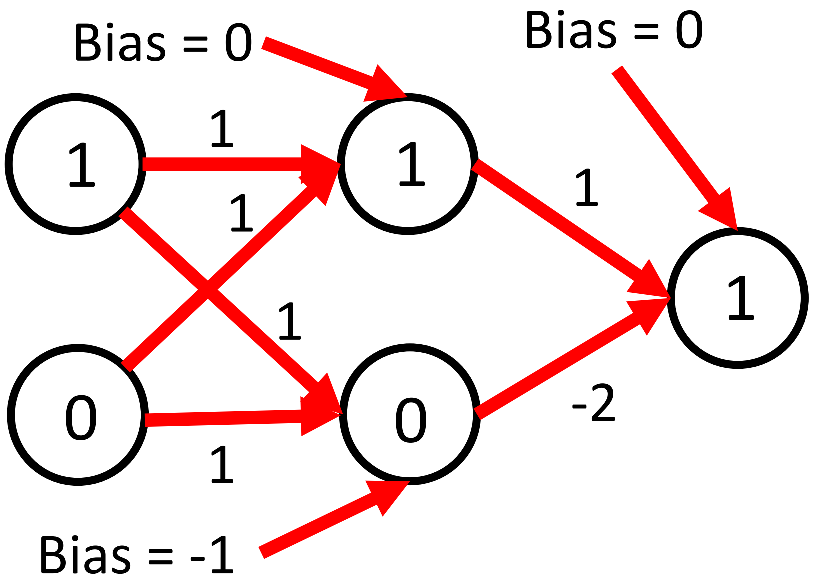
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



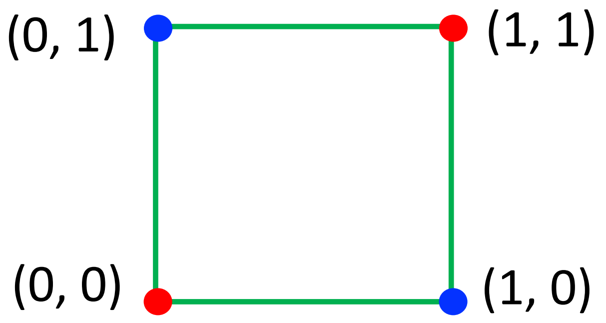
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



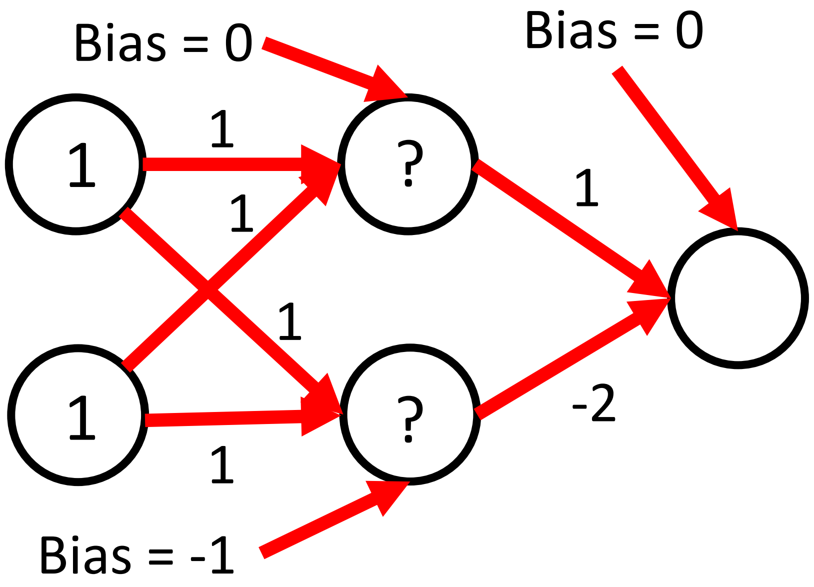
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



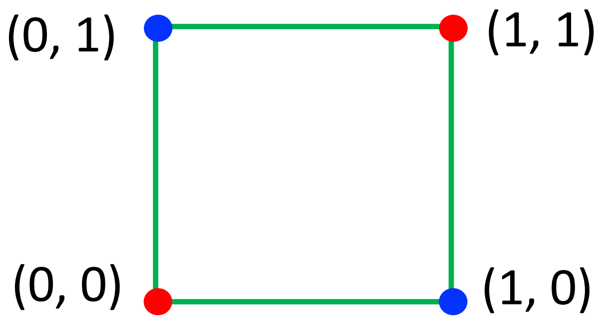
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



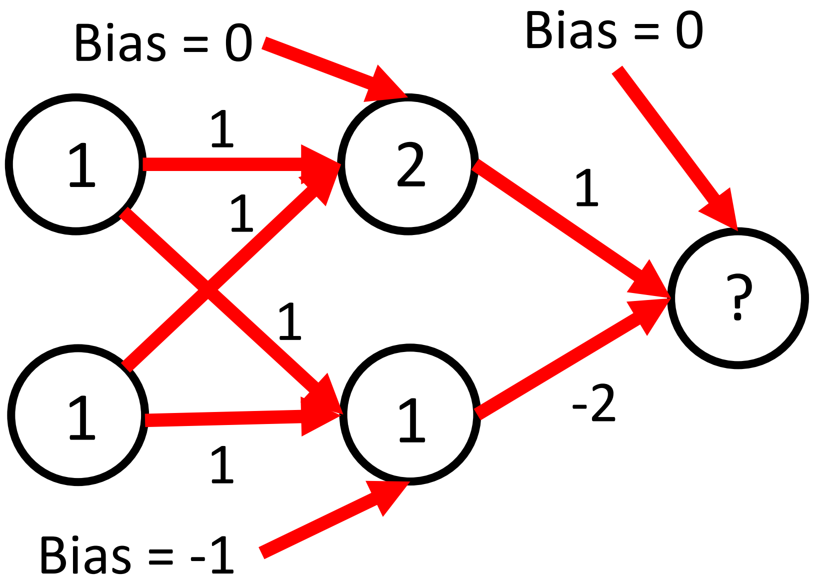
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example: Revisiting XOR problem

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



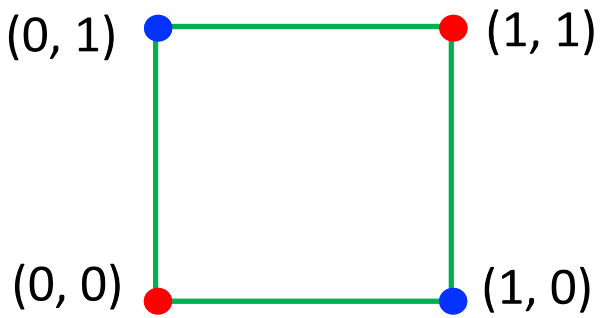
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



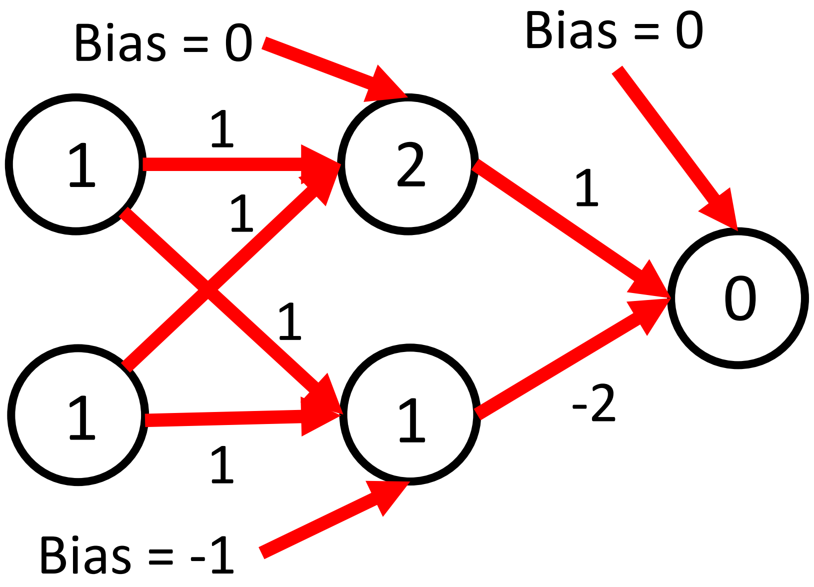
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
$\boxed{1}$	$\boxed{1}$	0

Non-Linear Example: Revisiting XOR problem

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



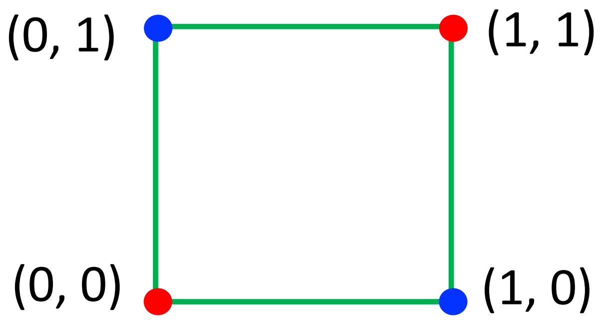
- Approach: ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with these parameters:



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 ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:

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

Today's Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss 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

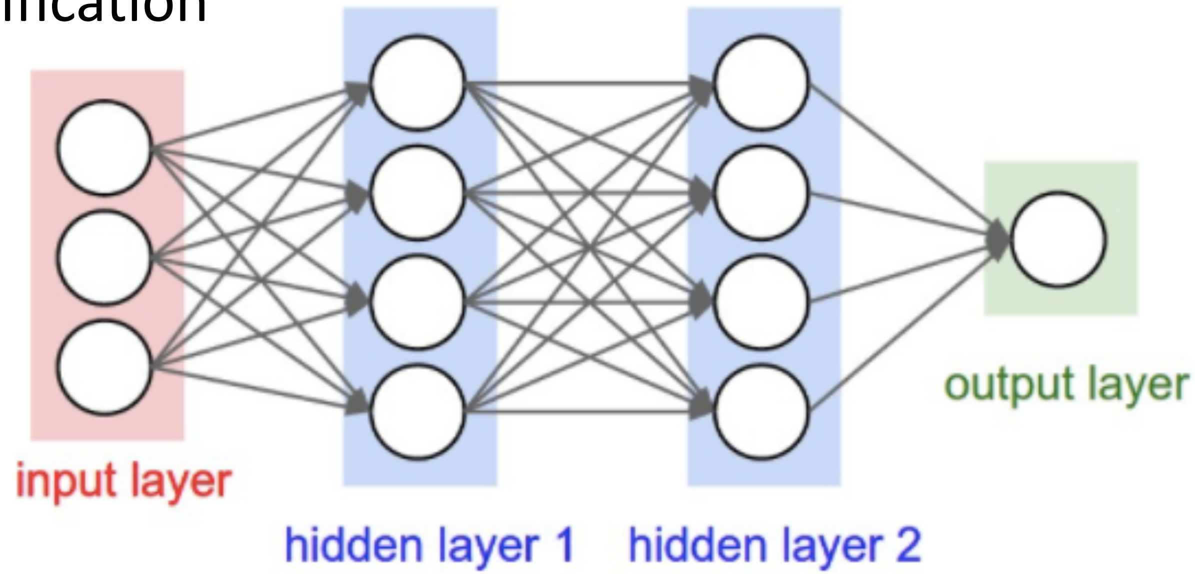
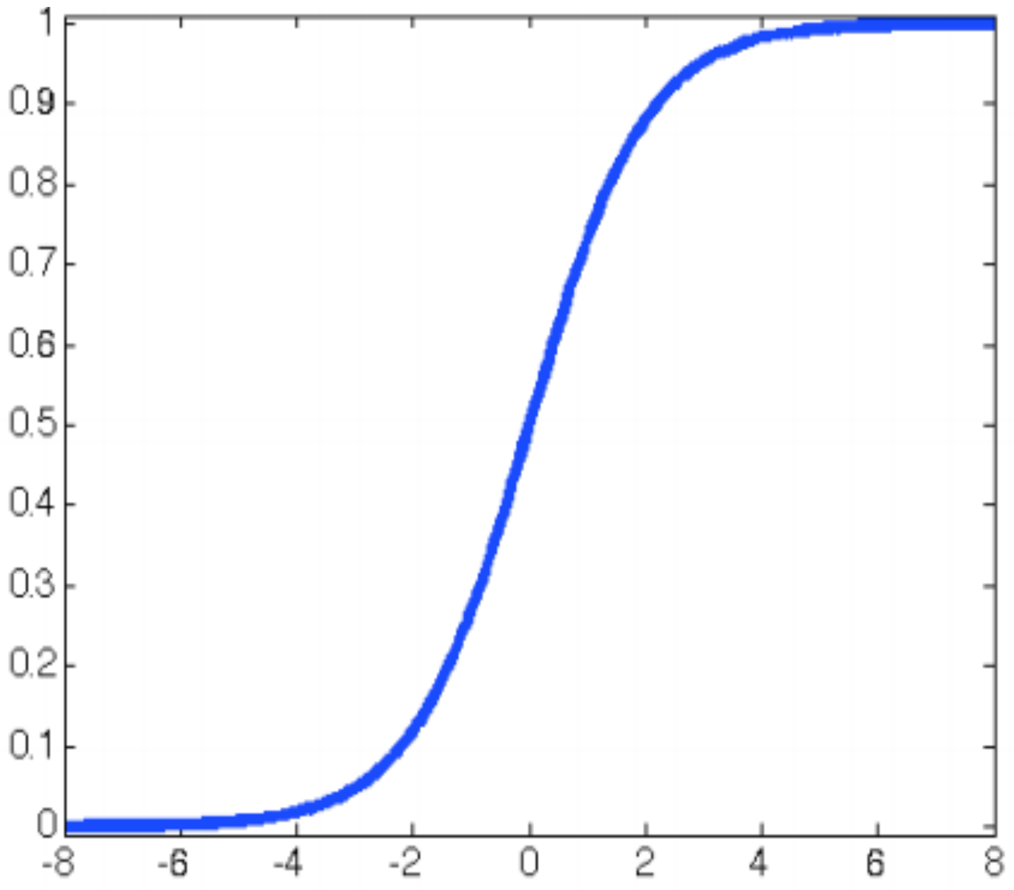


Figure Credit: http://cs231n.github.io/neural-networks-1/

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.,

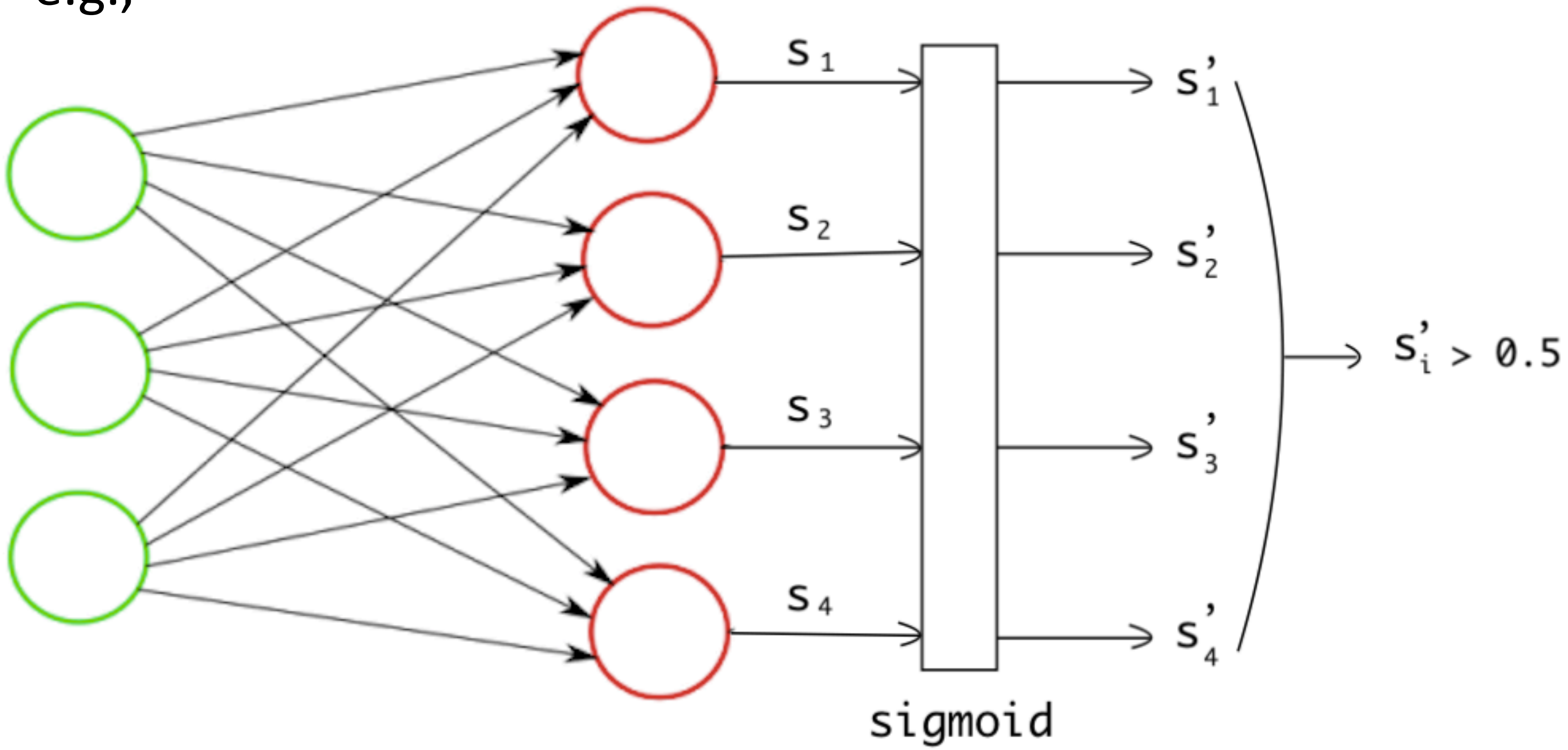


Figure Source: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede

Softmax (for Multiclass Classification)

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

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

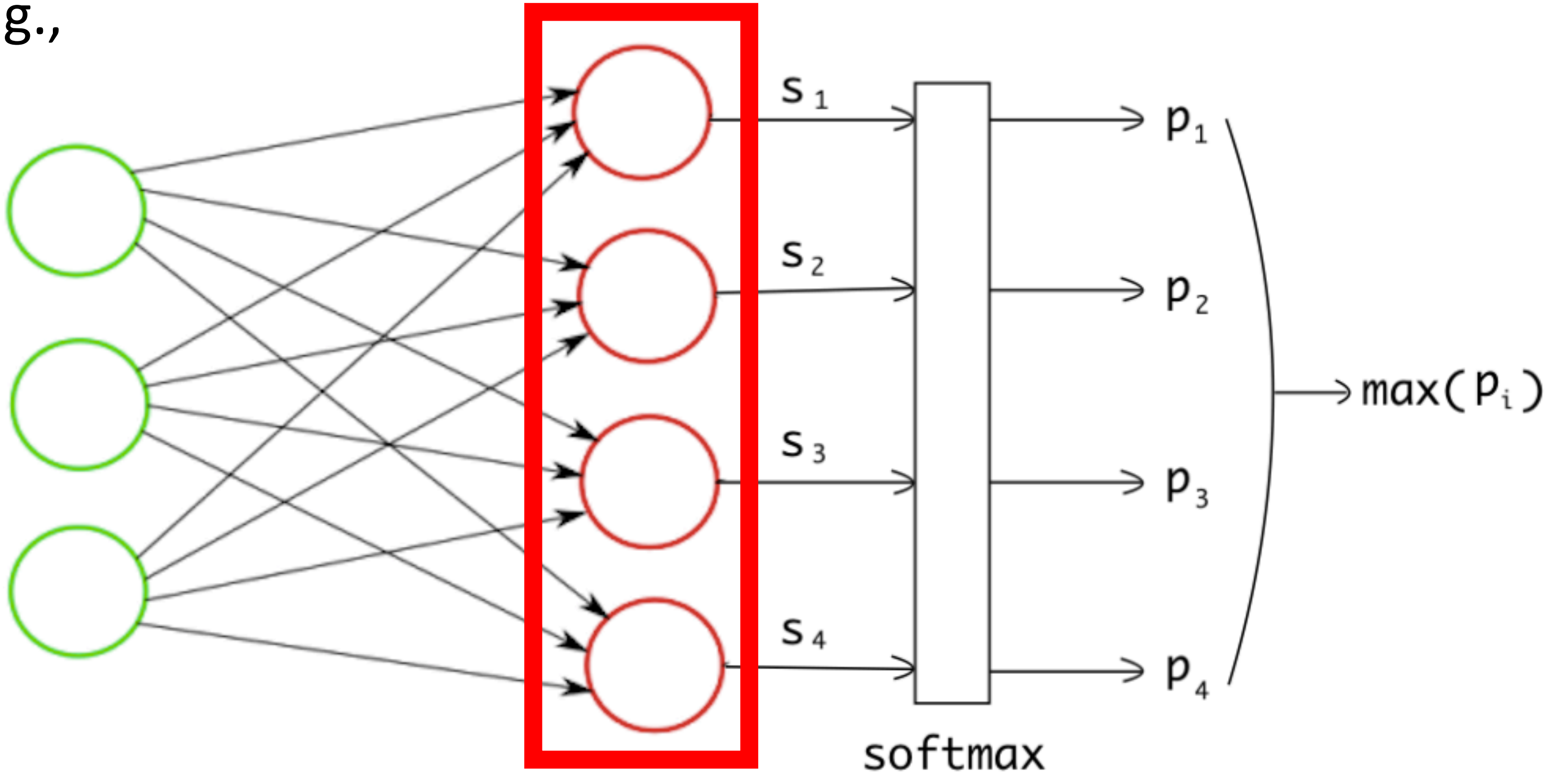


Figure Source: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede

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.,

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

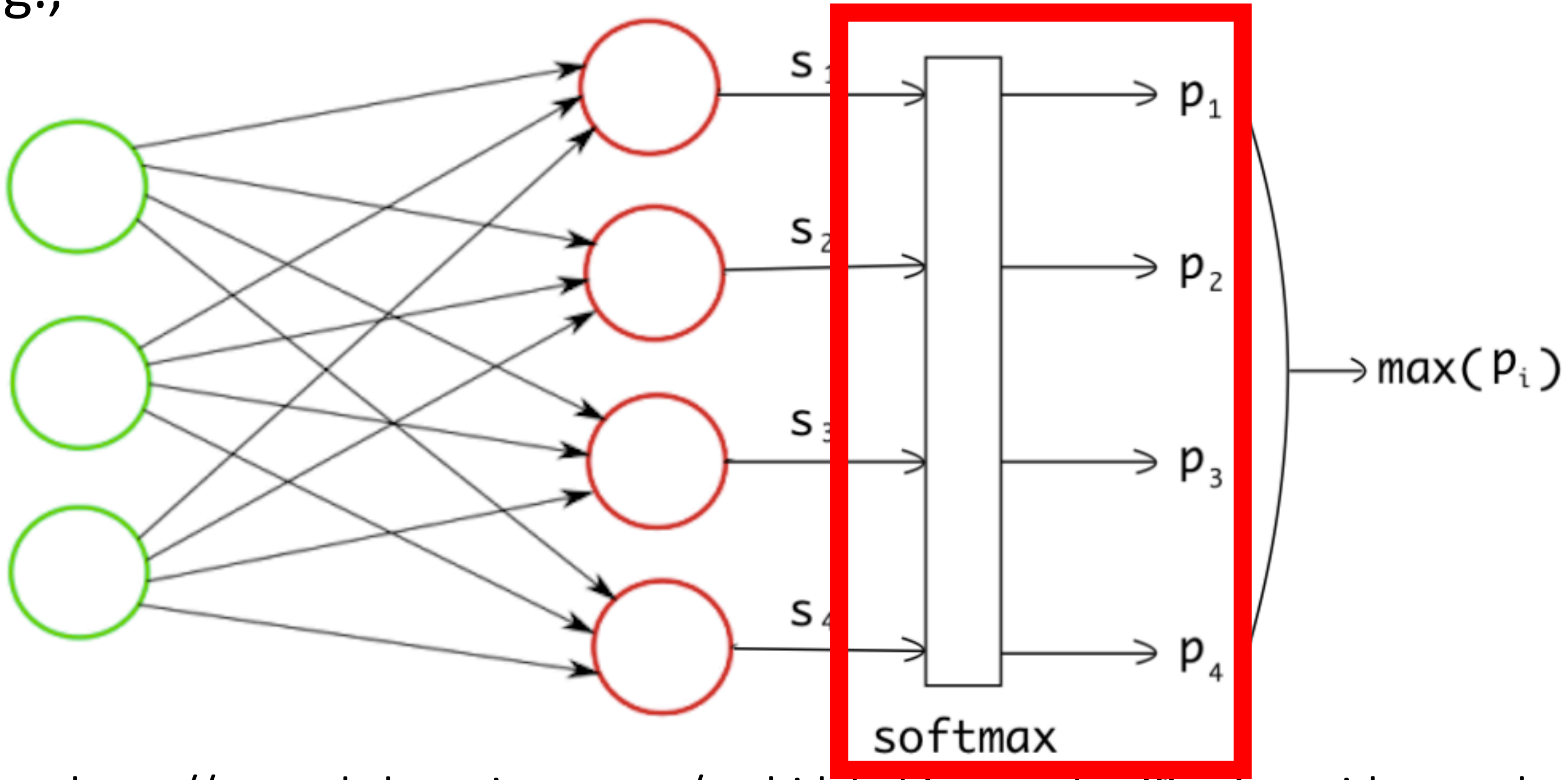


Figure Source: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede

Softmax (for Multiclass Classification)

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

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

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.,

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

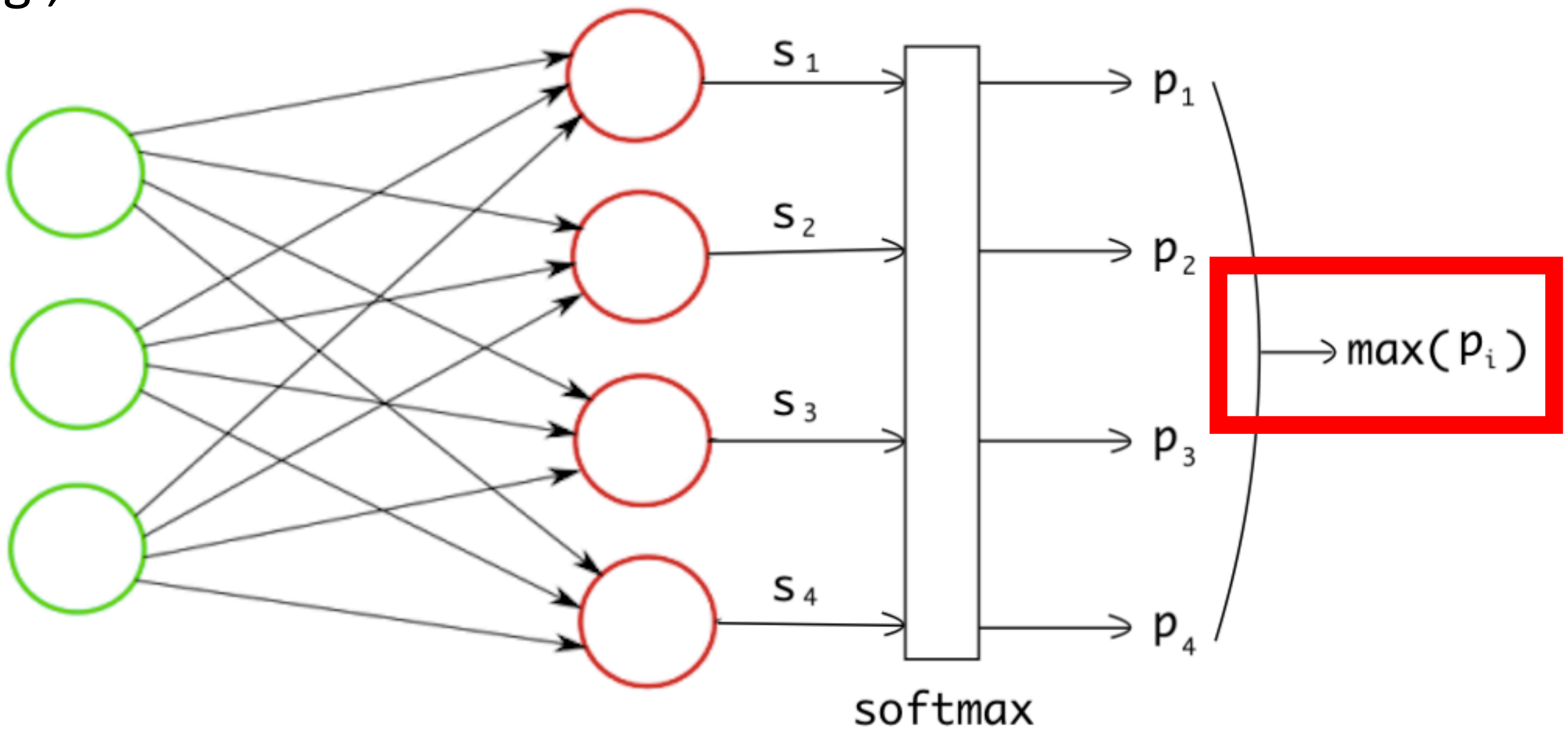


Figure Source: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede

Softmax (for Multiclass Classification)

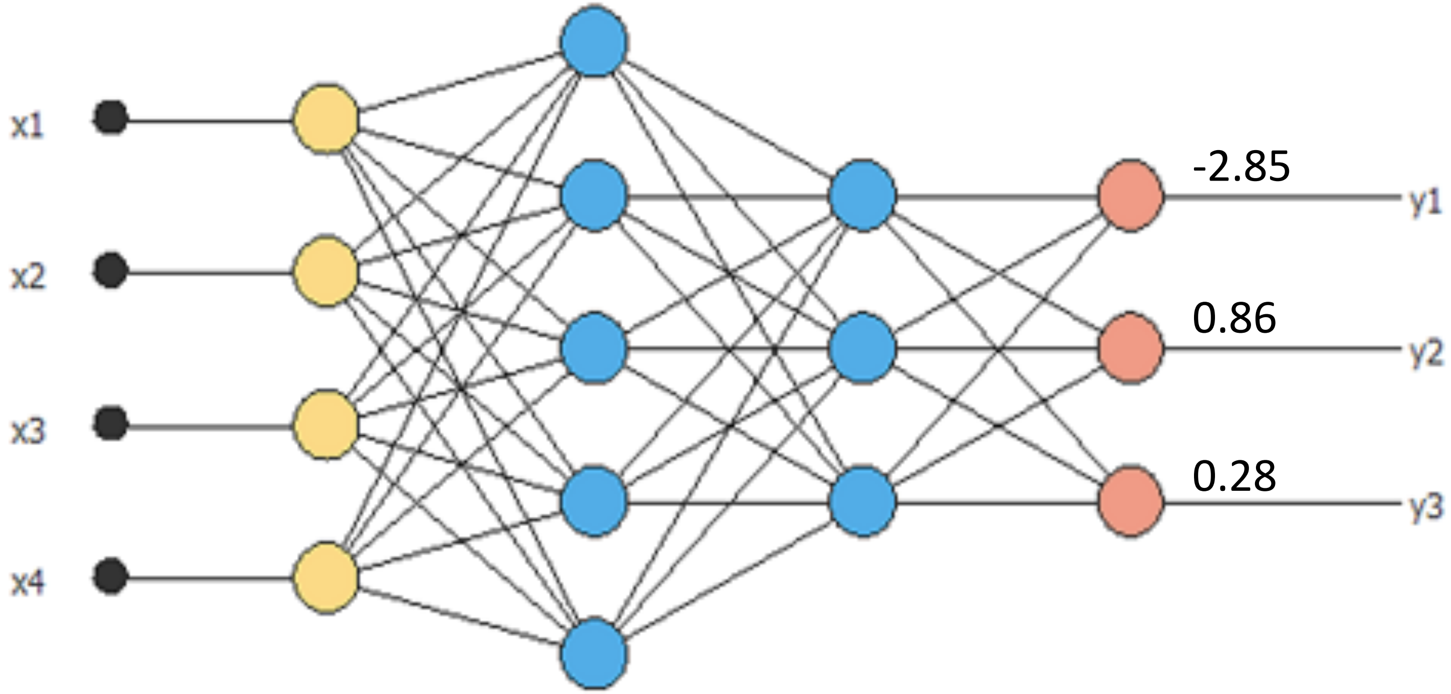
- Generalization of sigmoid that converts the input into a probability distribution that sums to 1:
  - e.g.,

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

	Scoring Function	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?



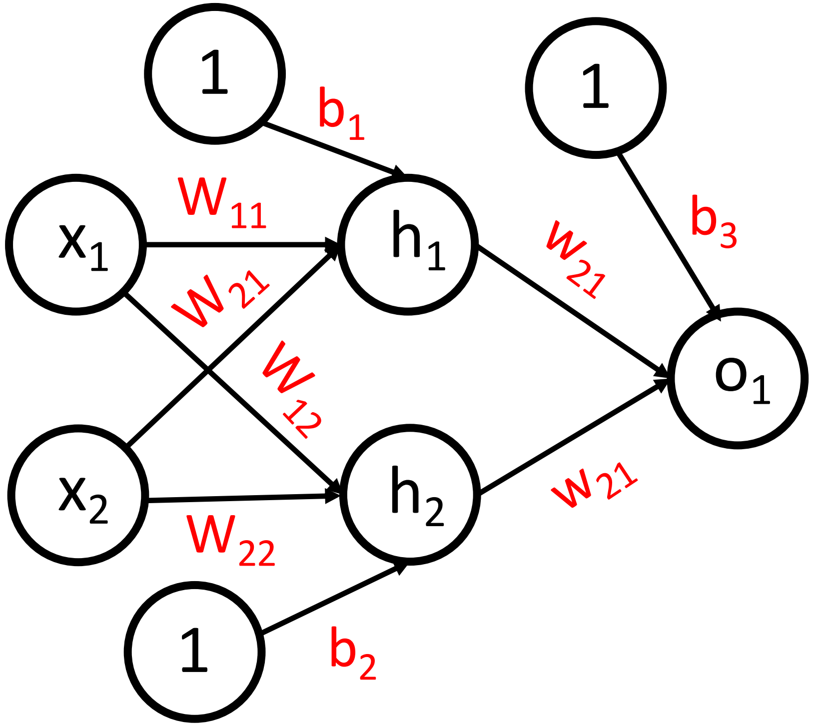
Today’s Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions

Recall: What to Learn in Neural Network?

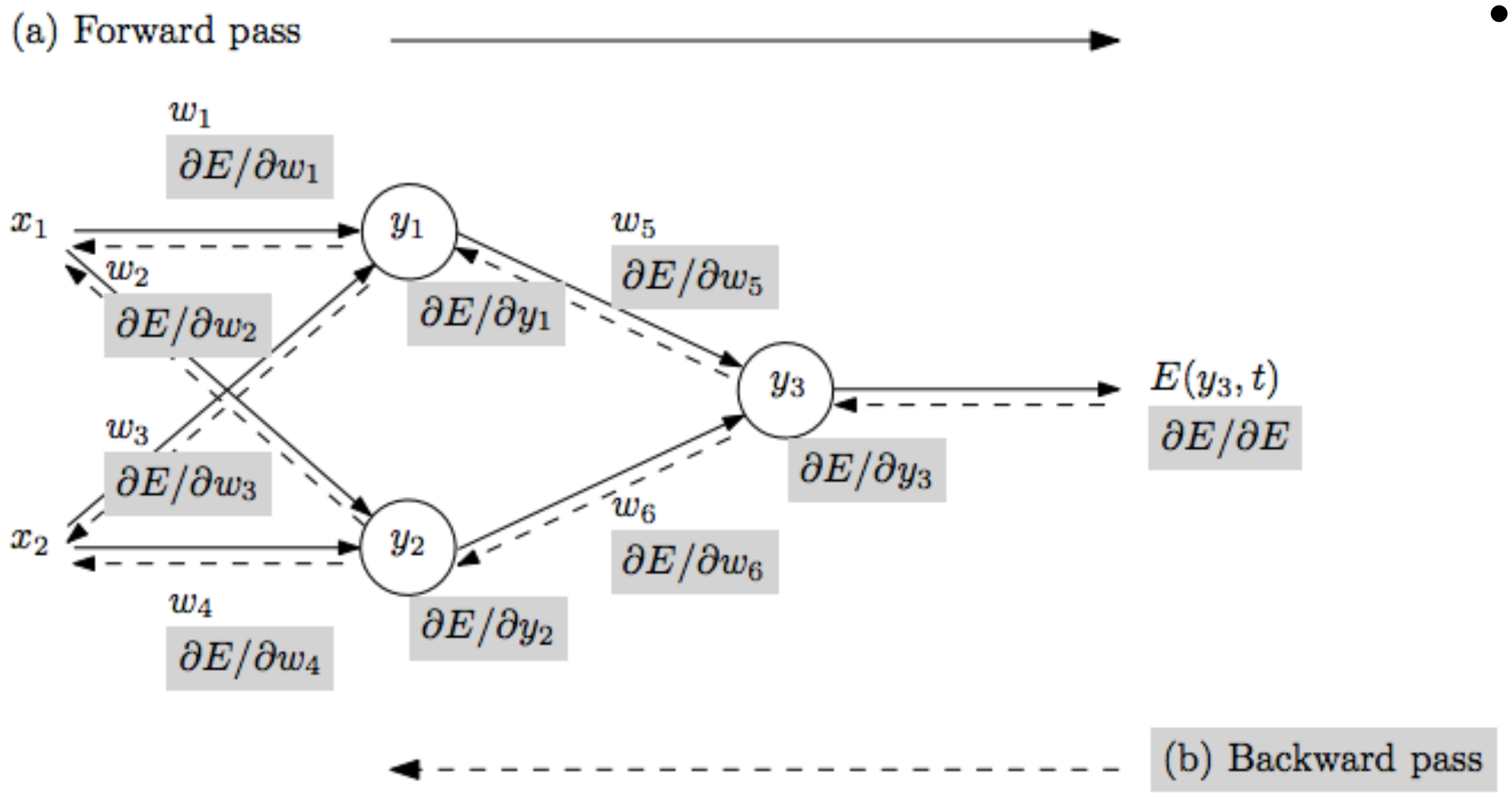
- Learn:
  - weights connecting units
  - bias for each unit

- e.g., 2 layer neural network:



- Algorithm decides how to use each layer to produce the output; for this reason, layers are called “hidden”

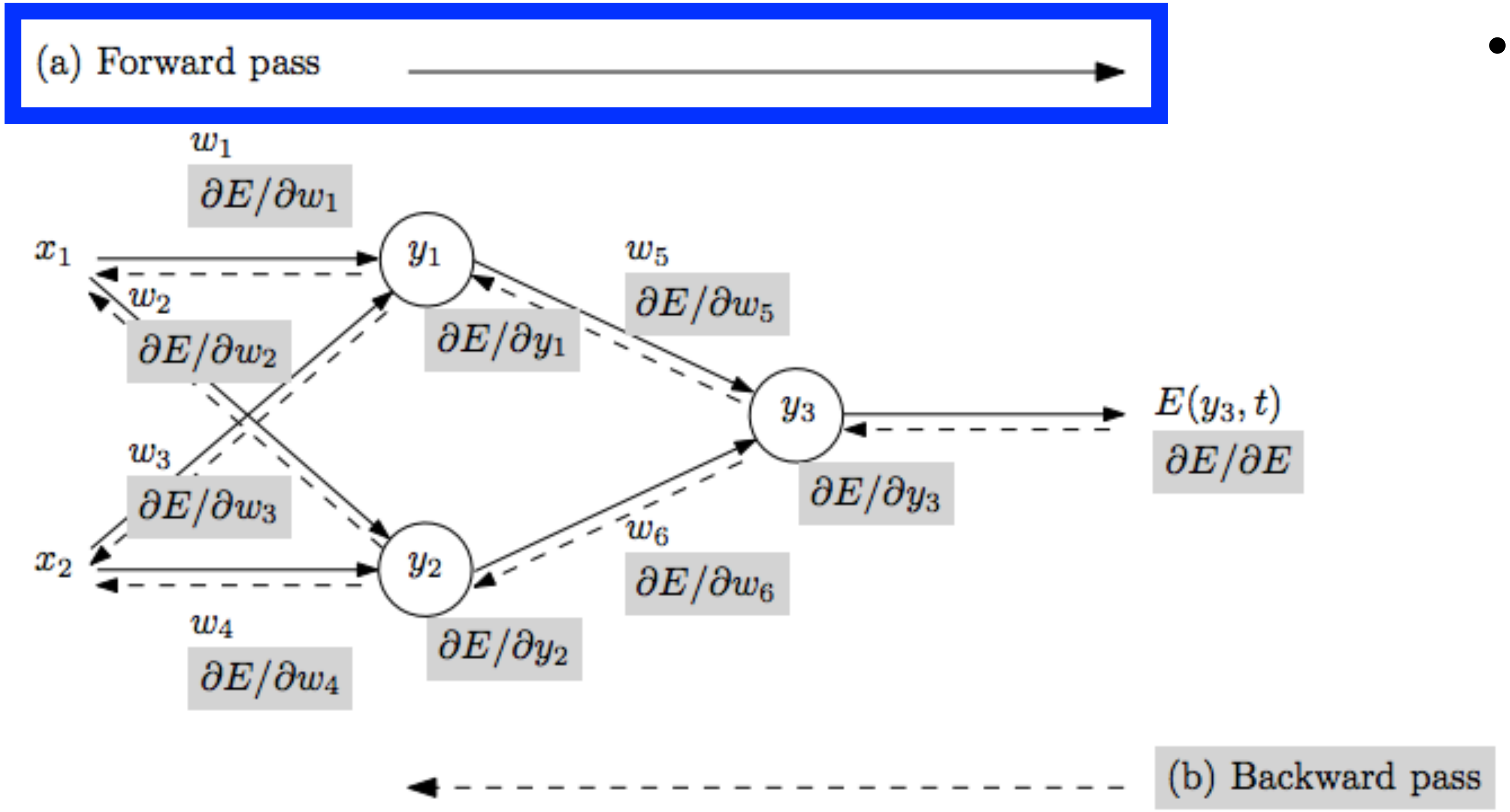
Neural Network: How to Learn?



- Repeat until stopping criterion met:

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

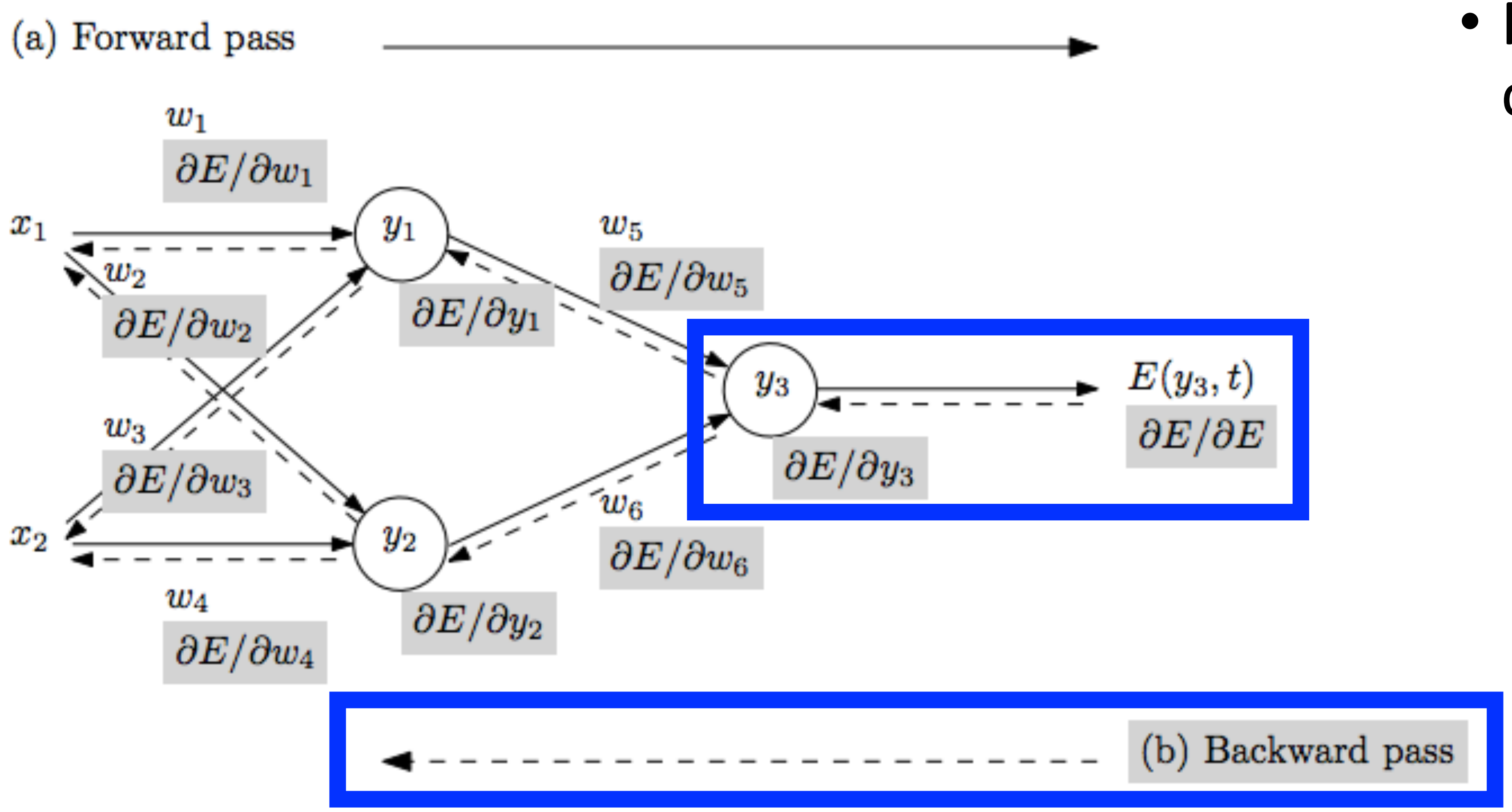
Neural Network: How to Learn?



- 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

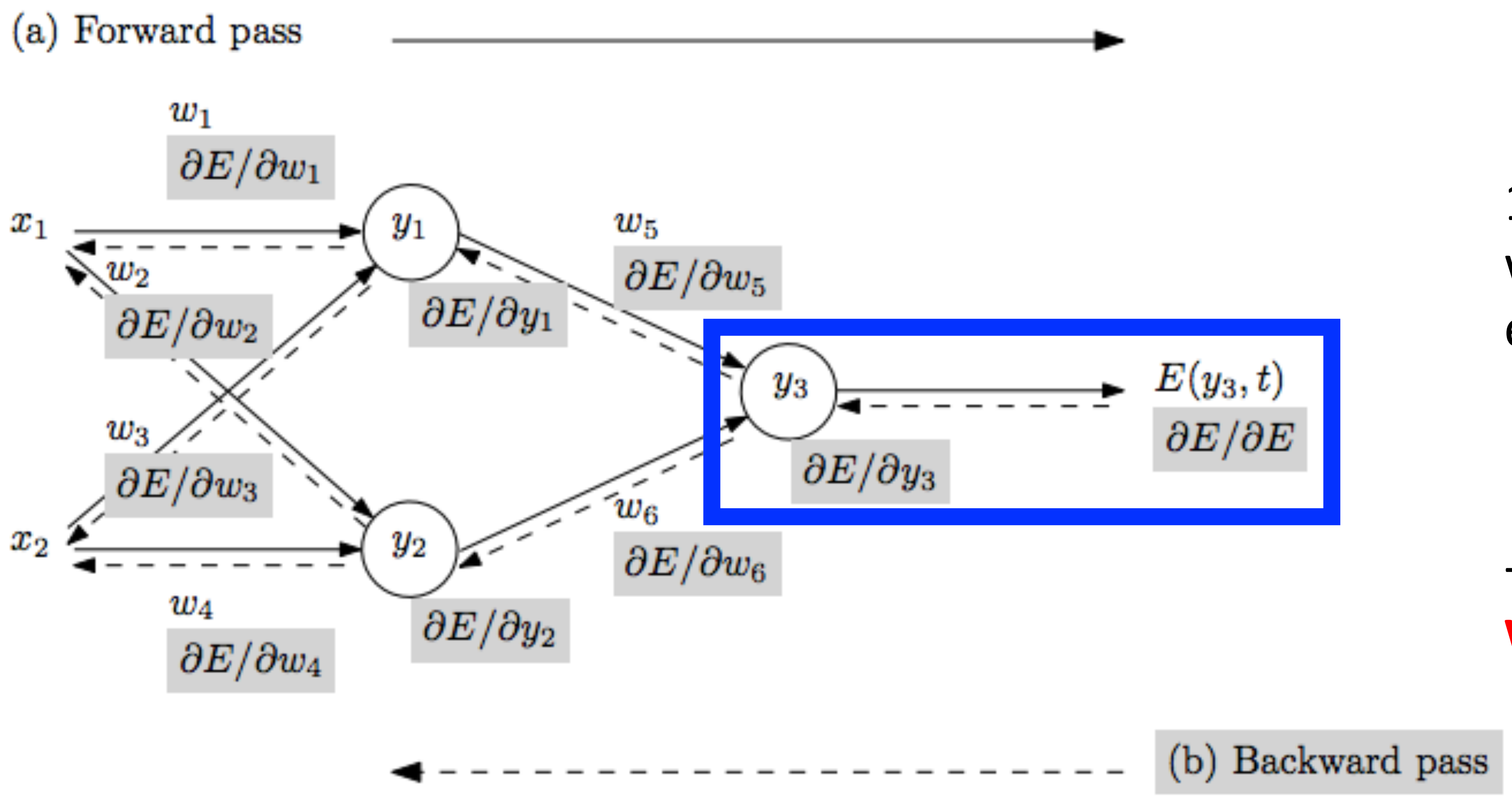
Neural Network: How to Learn?



- 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?



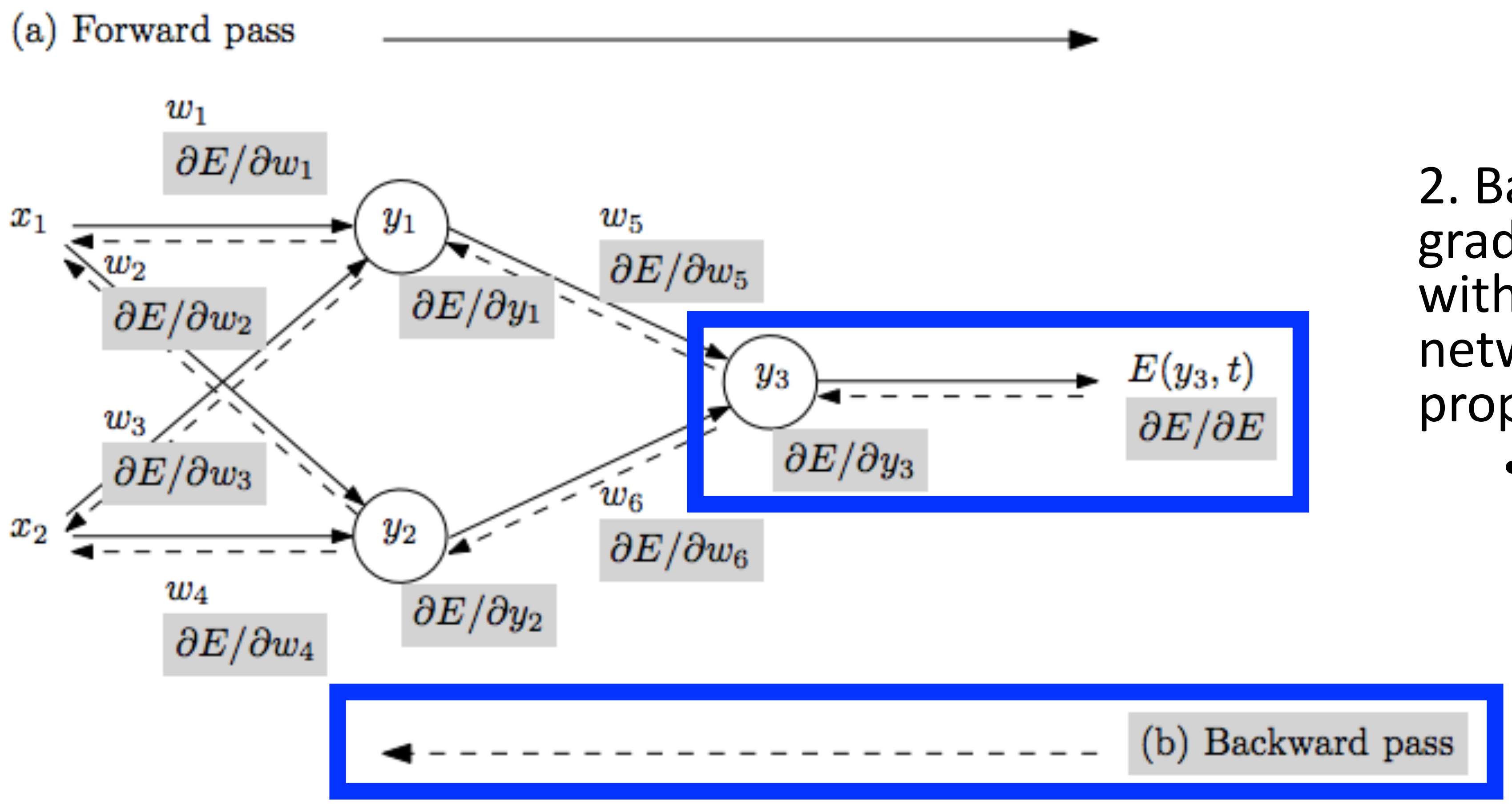
1. Define "loss" function, which quantifies the model's errors on training data

e.g., (Predicted – Actual)^2

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

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?

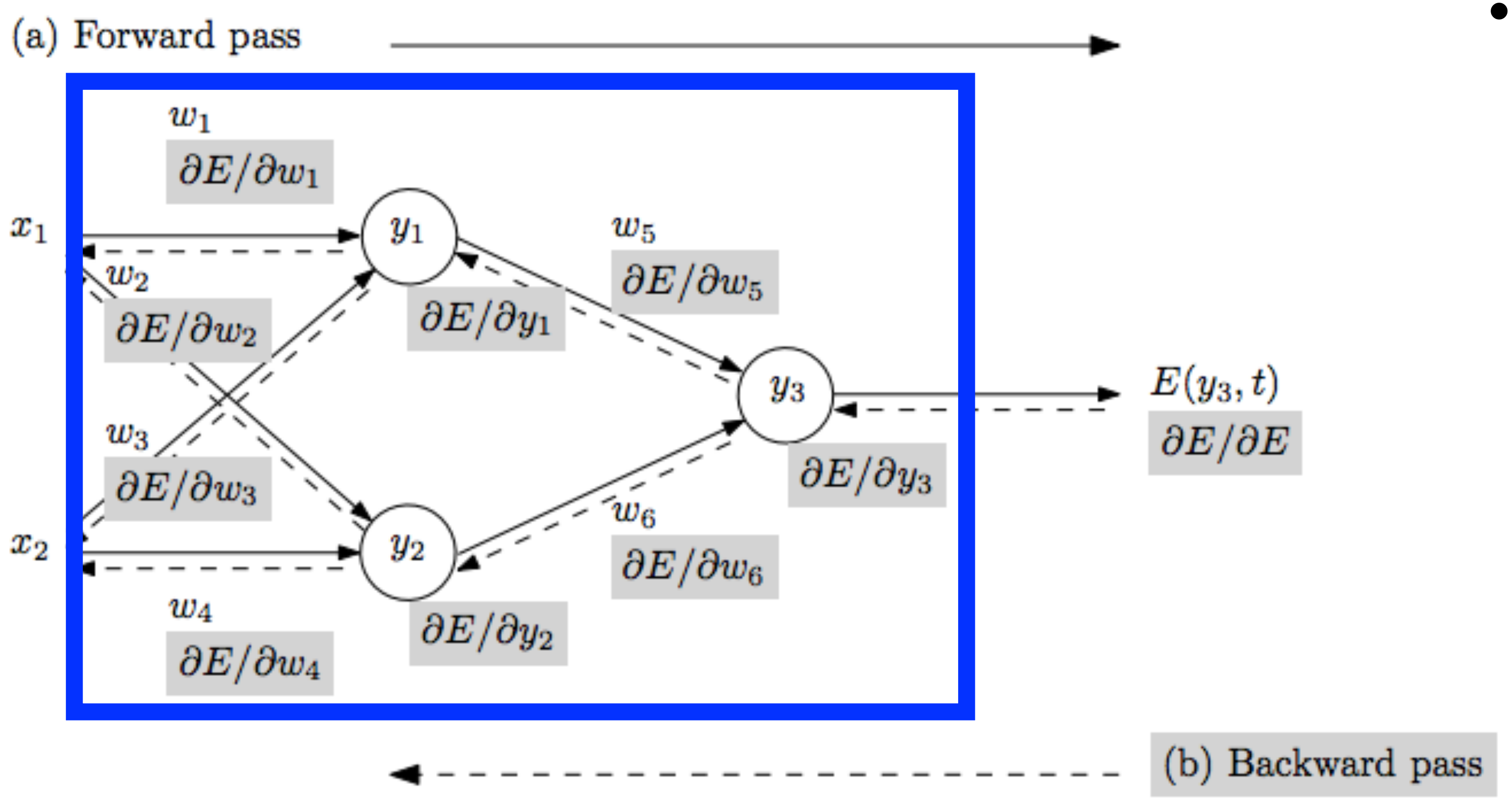


2. Backpropagation calculates gradient of the loss function with respect to the neural network's **weights** to propagate error backwards

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

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

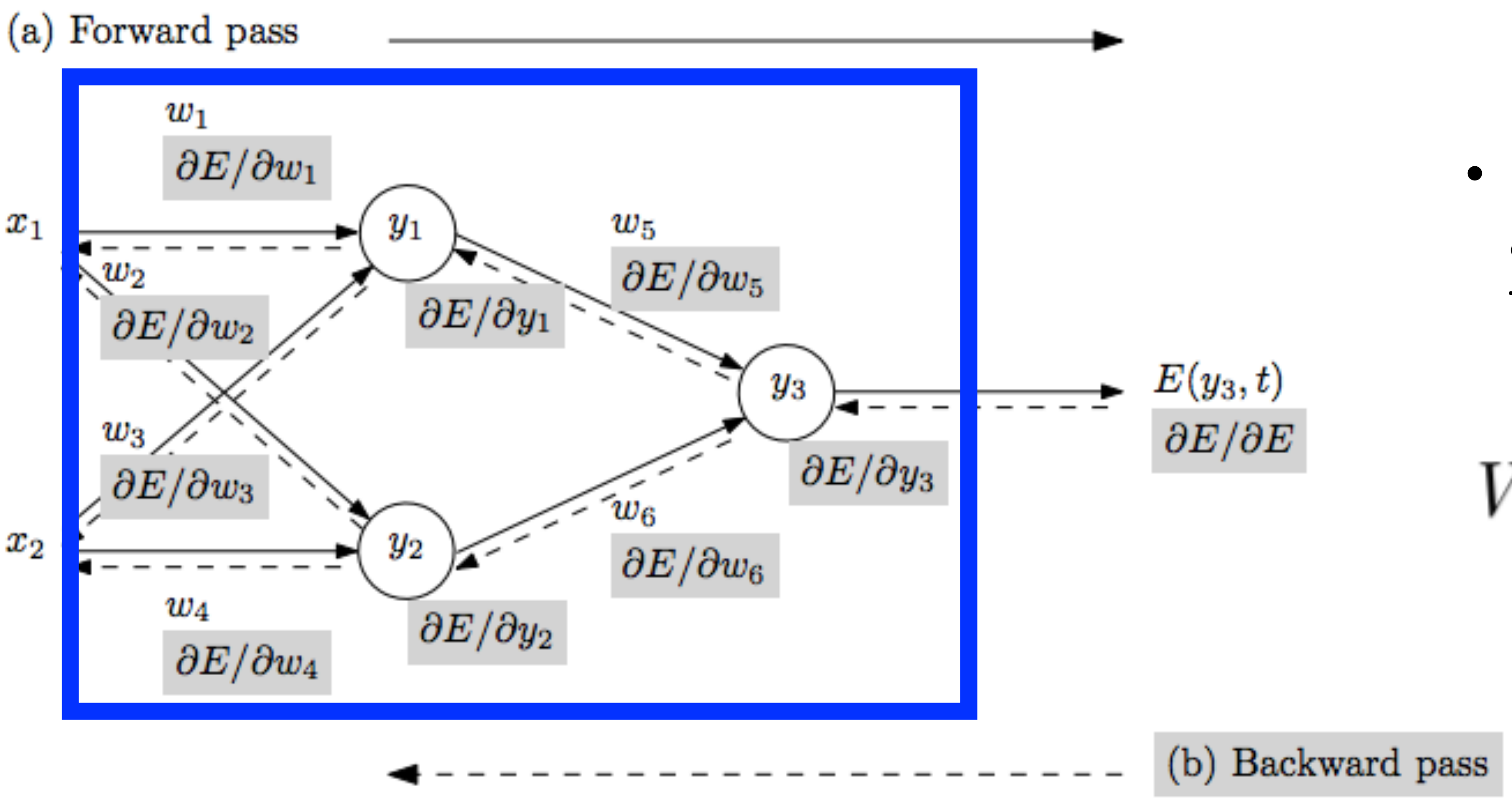
Neural Network: How to Learn?



- 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?



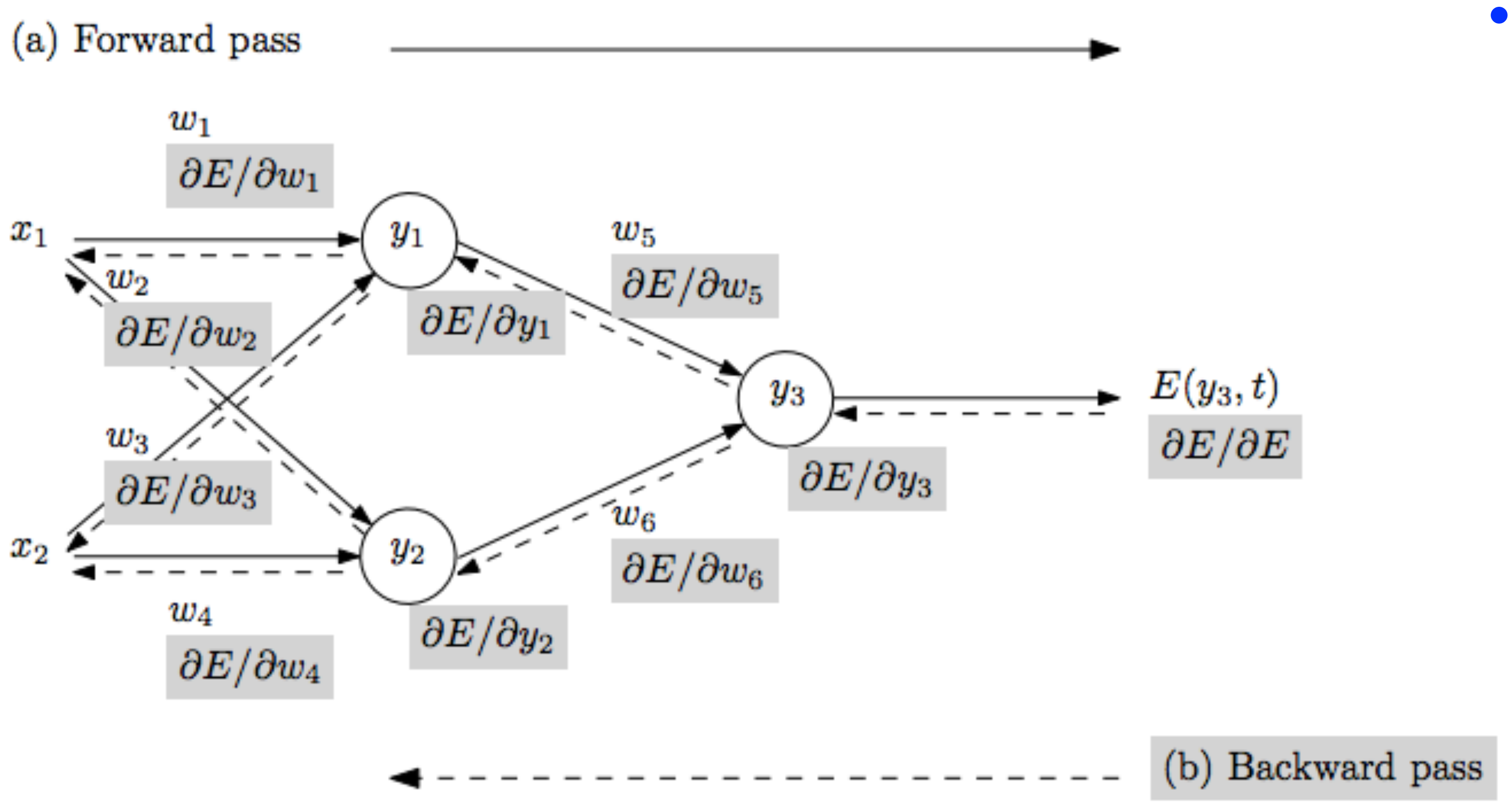
- Update weights by taking an opposite step towards the gradient for each layer

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

learning rate

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

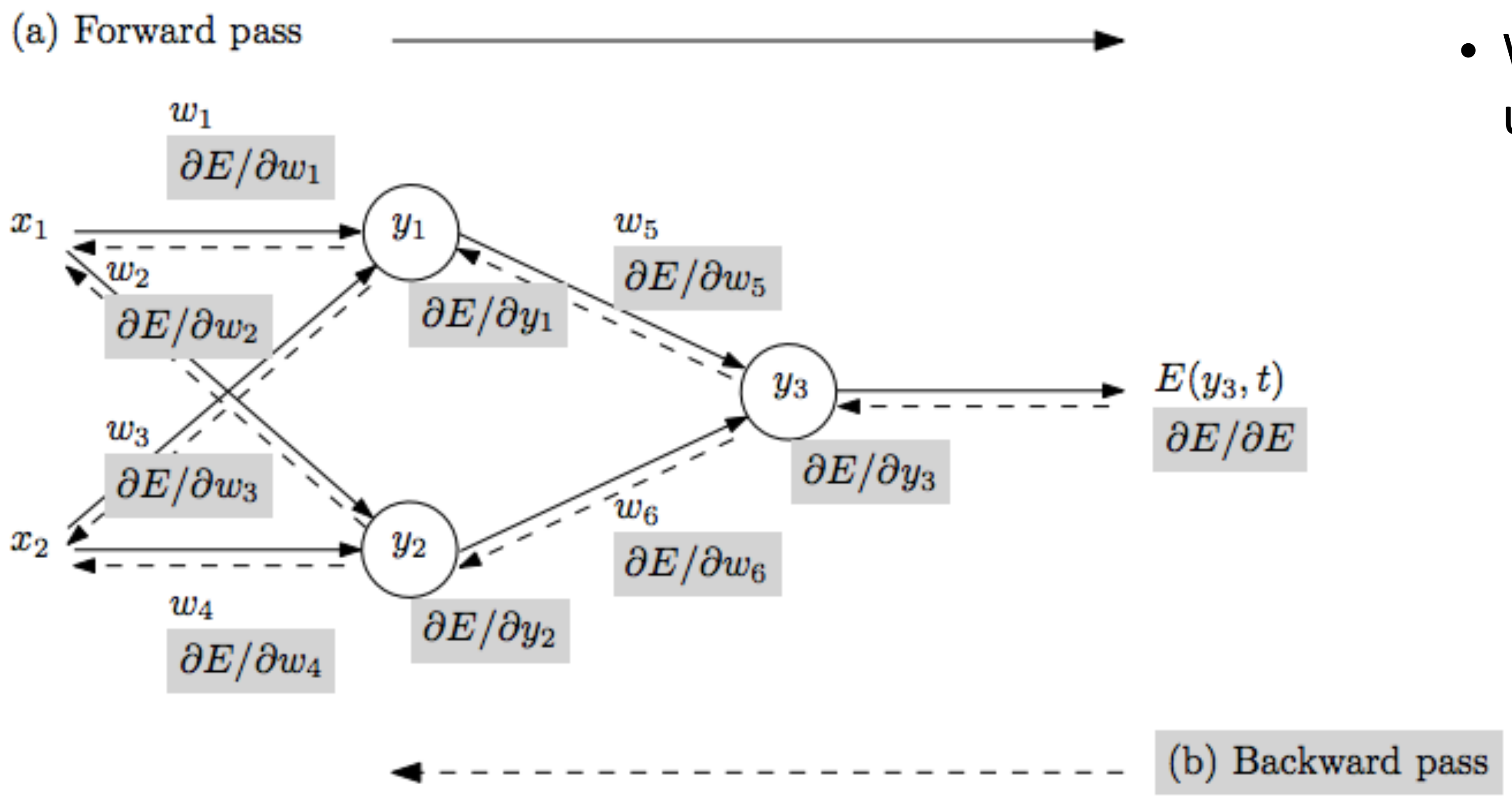
Neural Network: How to Learn?



- 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?



- 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)!

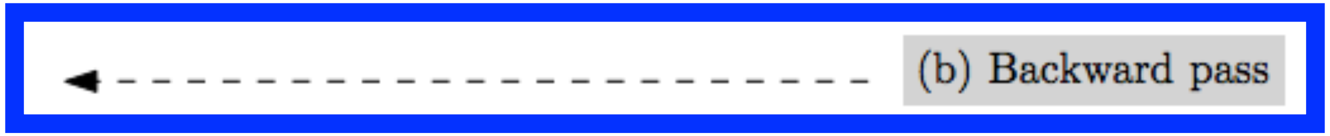
- When **minimizing** the objective function, it also is often called interchangeably the **cost function**, **loss function**, or **error function**.

Key Challenge: How to Compute Gradient?

Equation for calculating gradients depends on:

1) Loss function

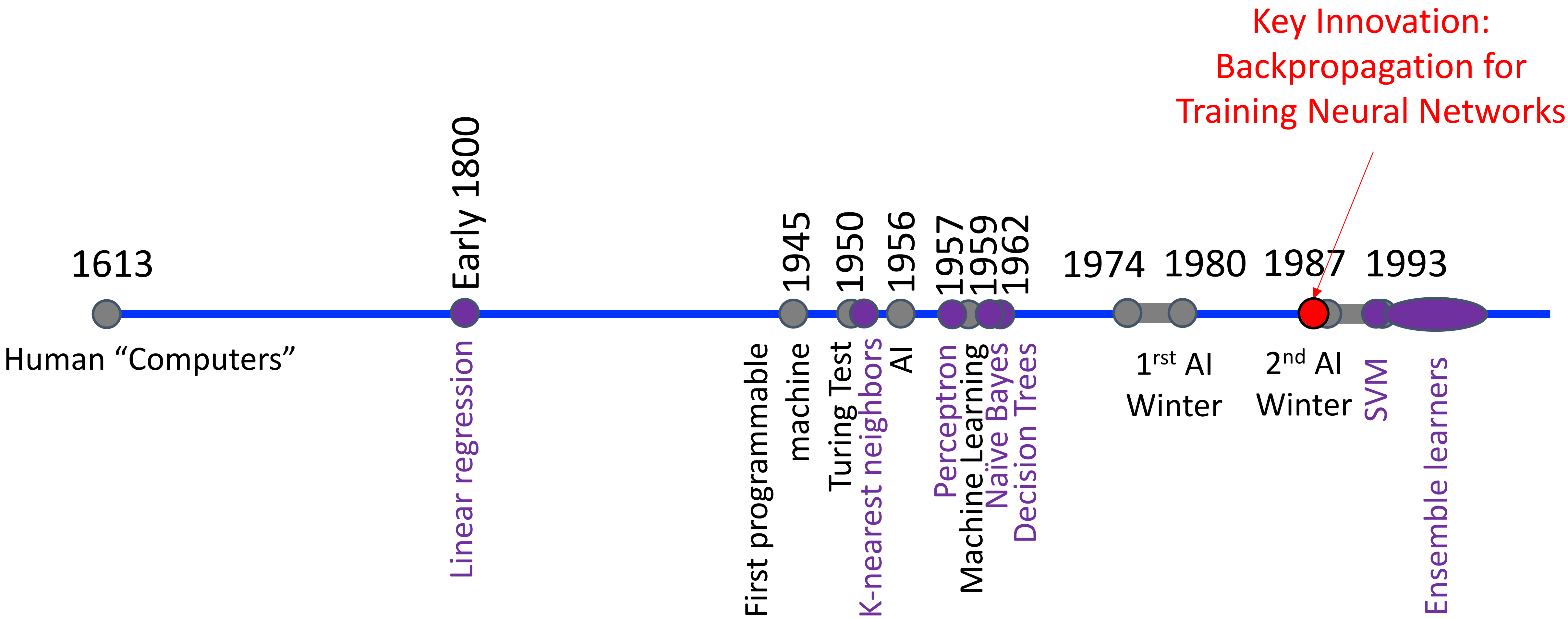
2) Network activation function



- 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

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

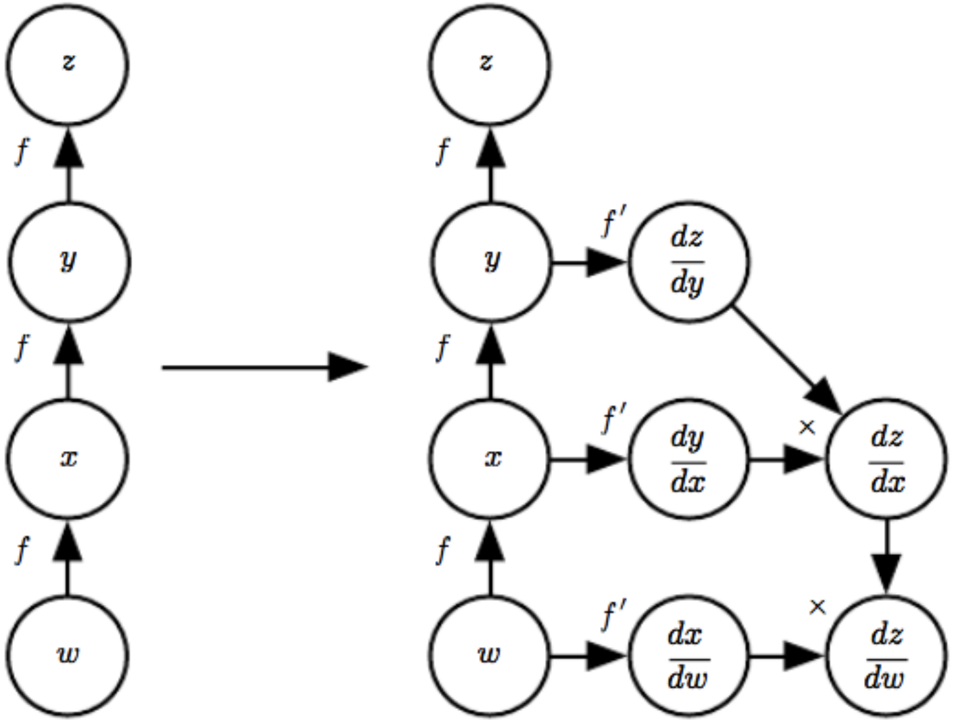
Neural Network Training: Backpropagation



D. Rulhart, G. Hinton, and R. Williams, Learning Internal Representations by Error Propagation, 1986.

Neural Network Training: Backpropagation

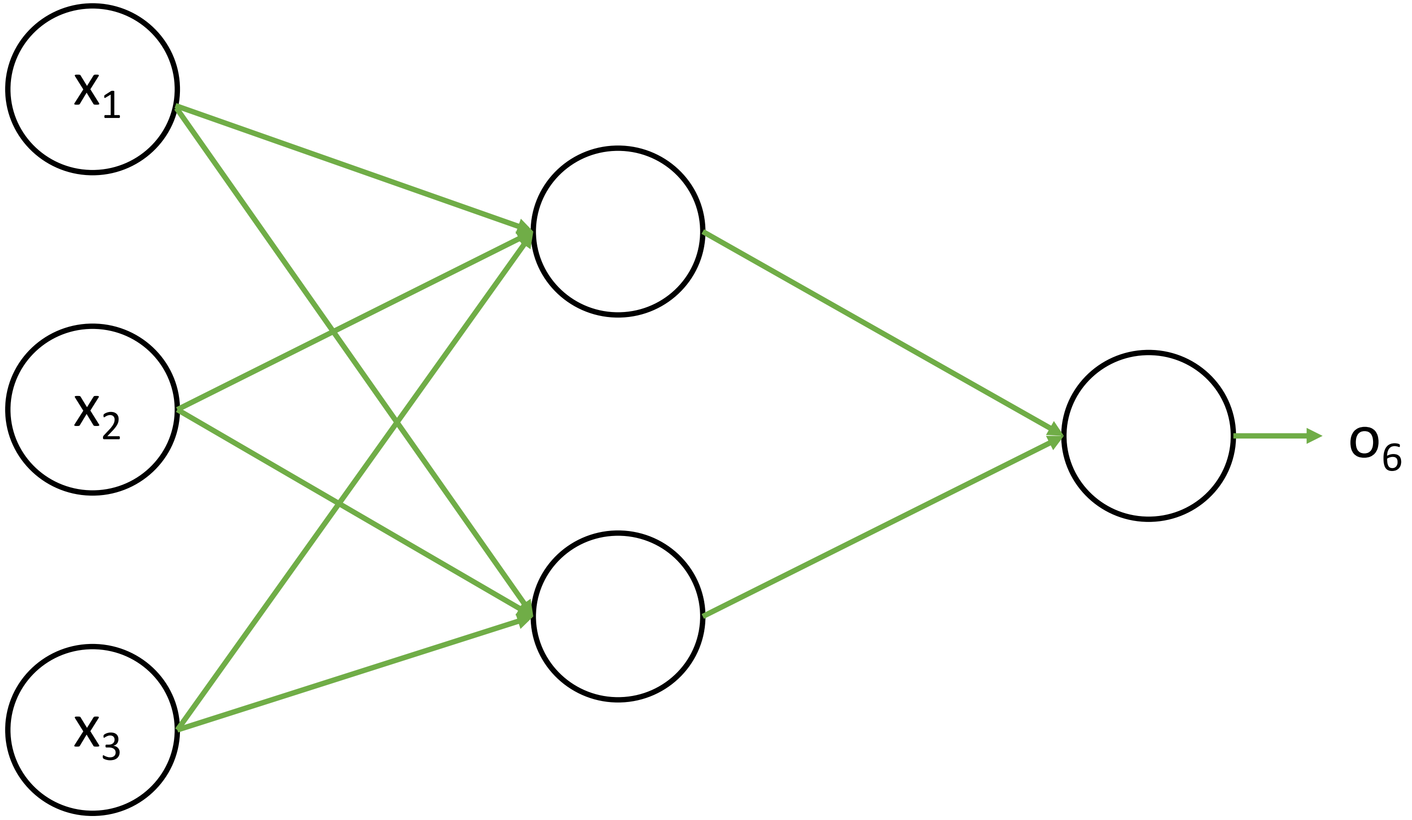
- Backpropagation idea: chain: $x = f(w), y = f(x), z = f(y)$



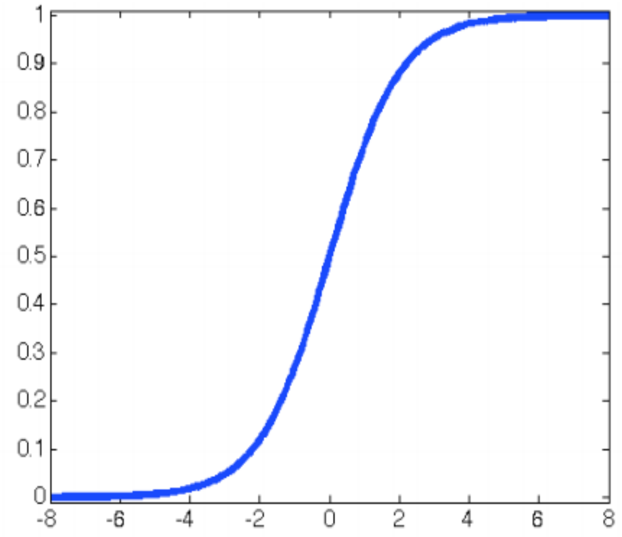
$$\frac{\partial z}{\partial w}$$
$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$
$$= f'(y) f'(x) f'(w)$$
$$= f'(f(f(w))) f'(f(w)) f'(w).$$

Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

Example: Choose Neural Network Architecture



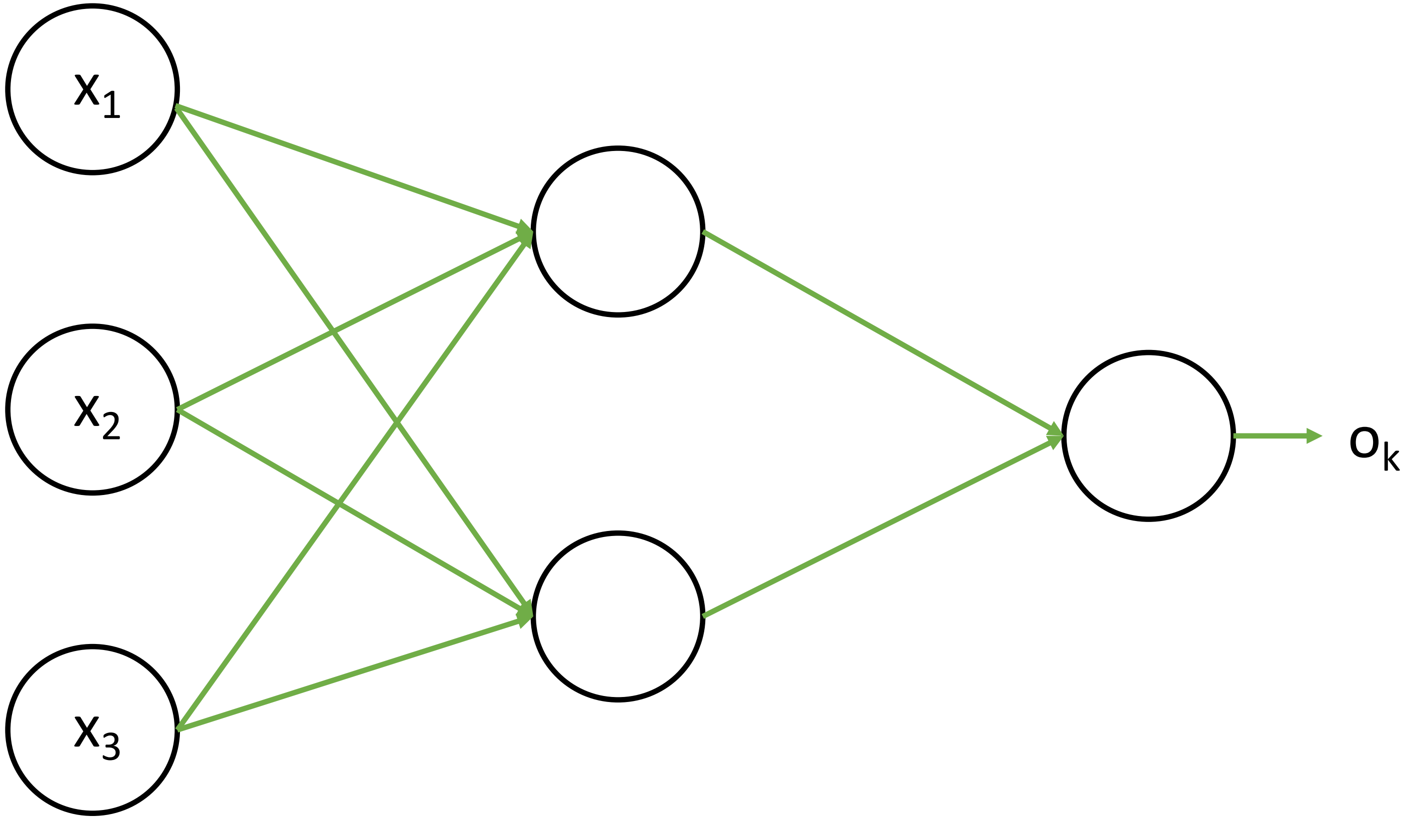
Sigmoid Activation Function



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

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

Example: Choose Loss Function for Training

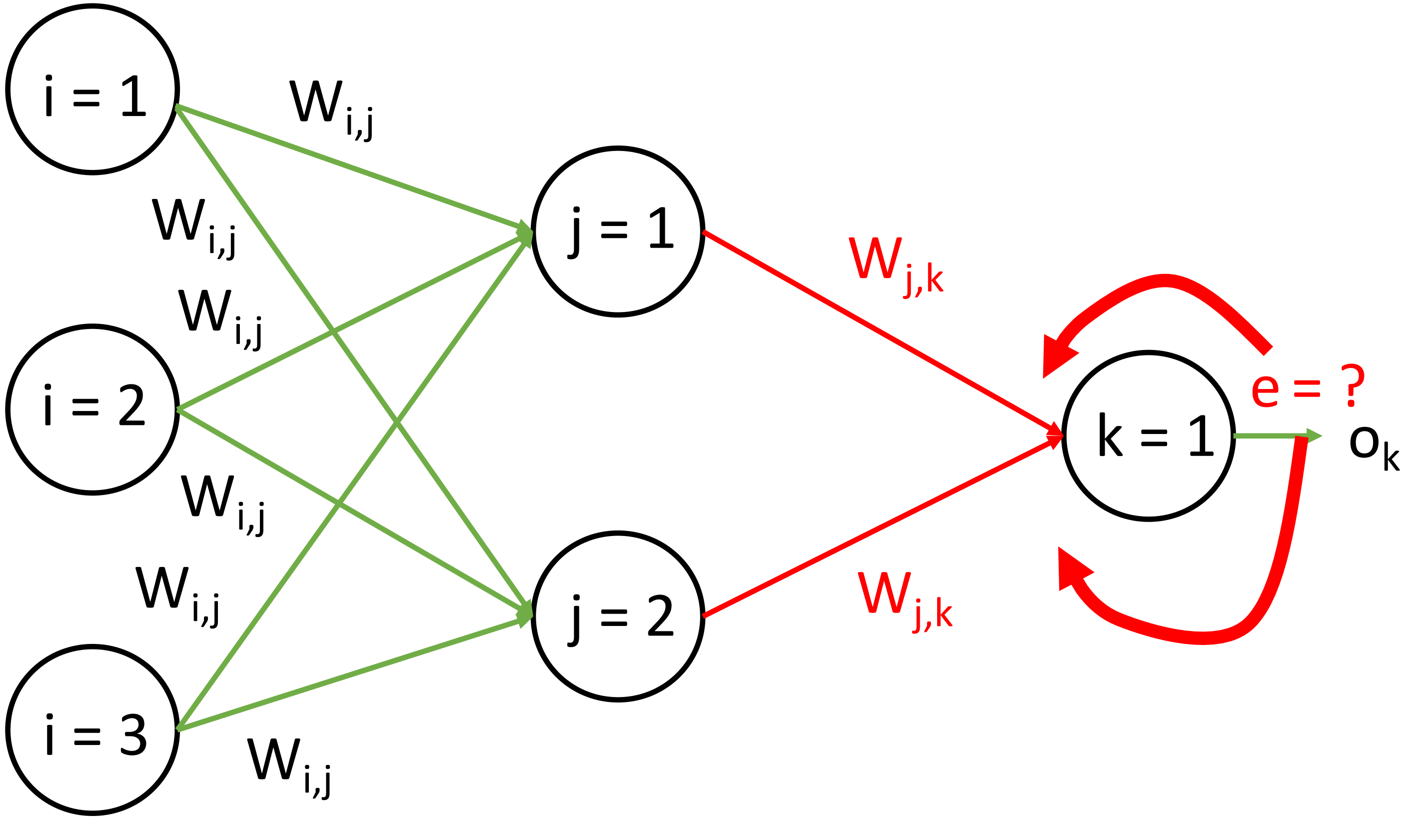


Squared Error Function

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

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

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



t is a constant value:

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

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

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

t is a constant value:

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

Using the following chain rule :

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

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

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

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

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

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

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

t is a constant value:

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

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

Using the following chain rule :

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

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

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

We can rewrite our function as follows:

For efficiency, compute last

$$\frac{\partial E}{\partial w_{jk}} = \boxed{-2(t_k - o_k)}\boxed{sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) * \boxed{o_j}}$$

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

t is a constant value:

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

Using the following chain rule :

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

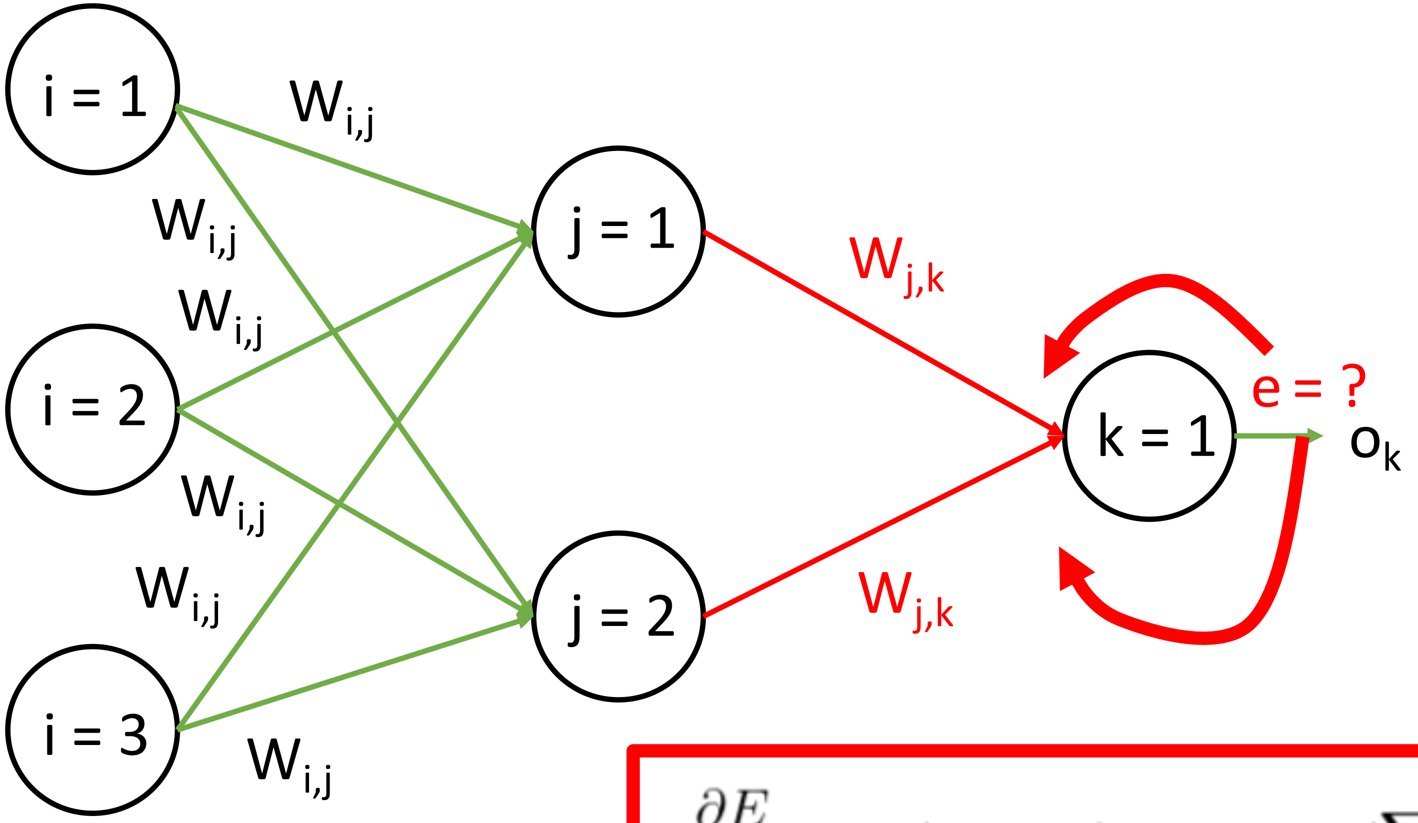
$$\boxed{\frac{\partial E}{\partial o_k} = -2(t_k - o_k)}$$

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

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

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

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

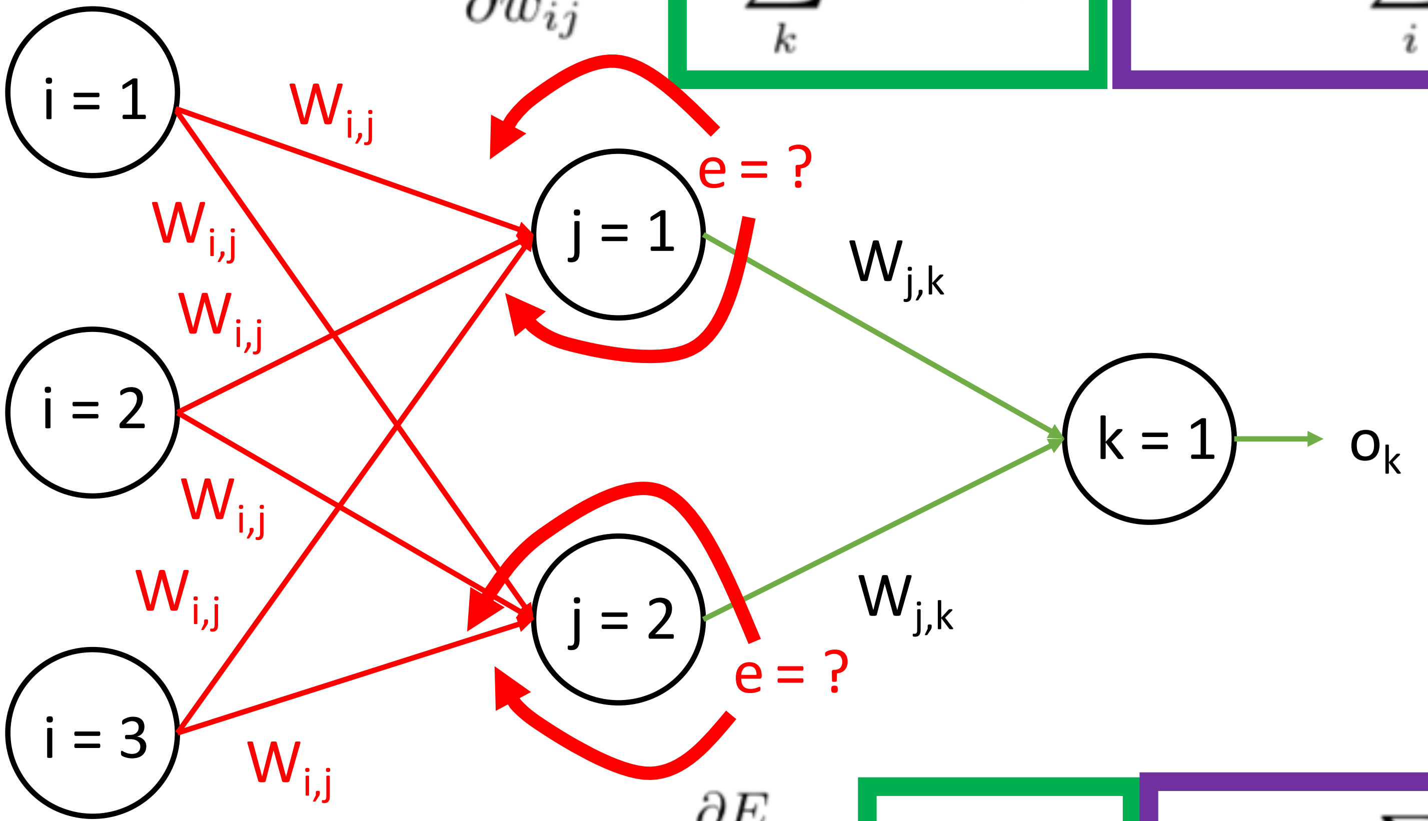


$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

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

Example: How to Compute Gradient? (Hidden Layer)

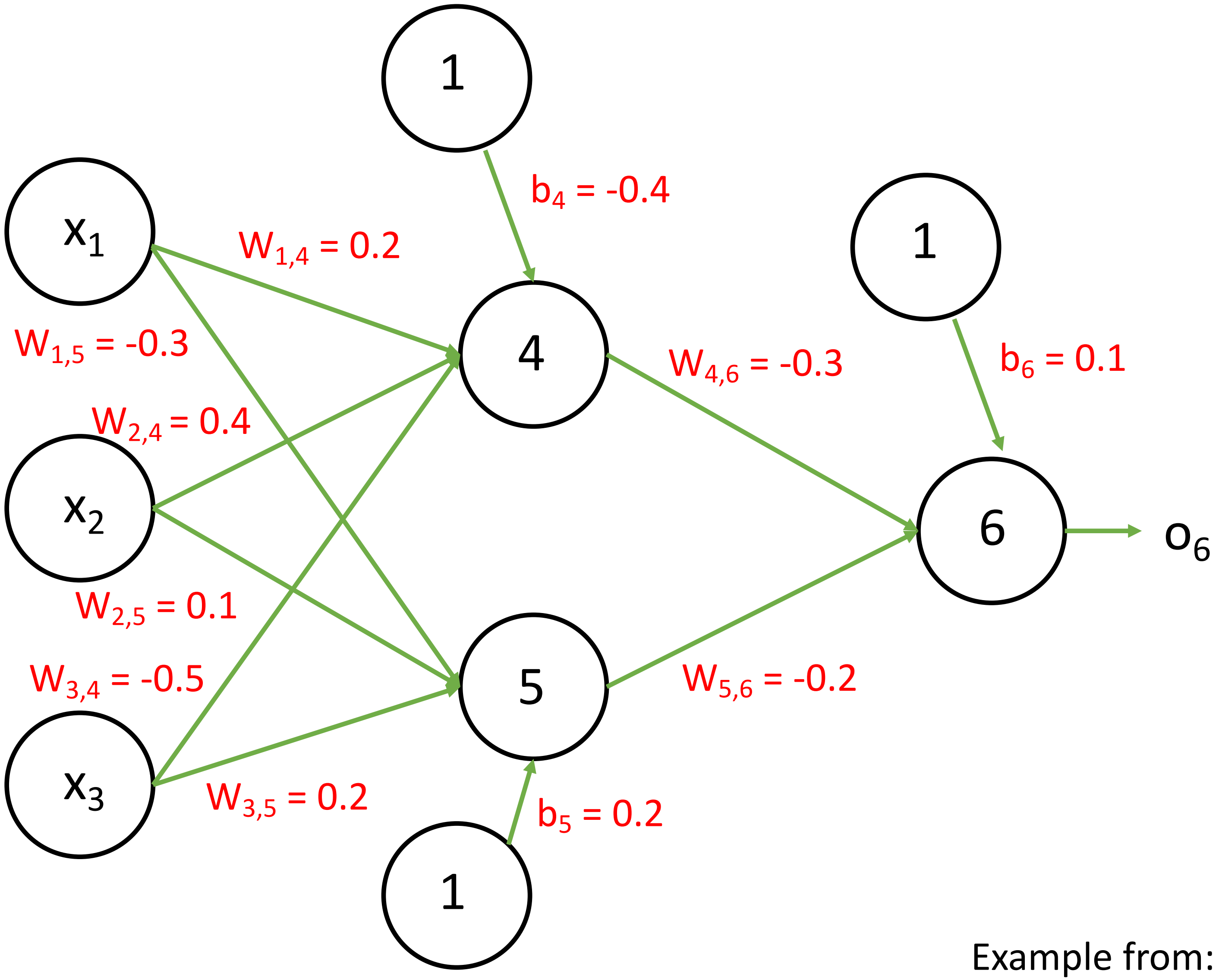
$$\frac{\partial E}{\partial w_{ij}} = \boxed{-(\sum_k Err_k w_{jk})} * \boxed{sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))}$$



Symmetry in how to solve for parameters in hidden layers

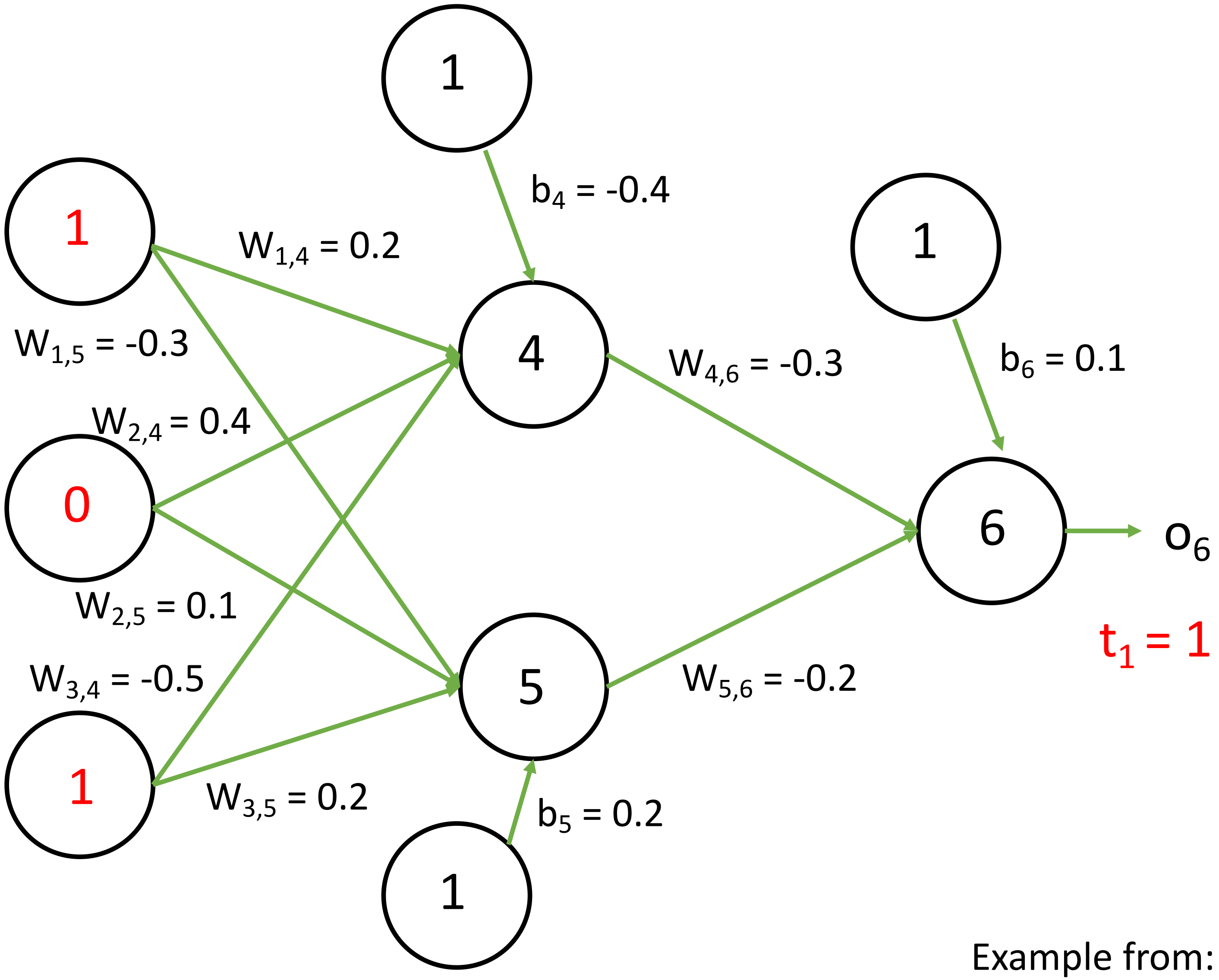
$$\frac{\partial E}{\partial w_{jk}} = \boxed{-(t_k - o_k)} \boxed{sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j))}$$

Example: Initialize Values (Weights, Biases)



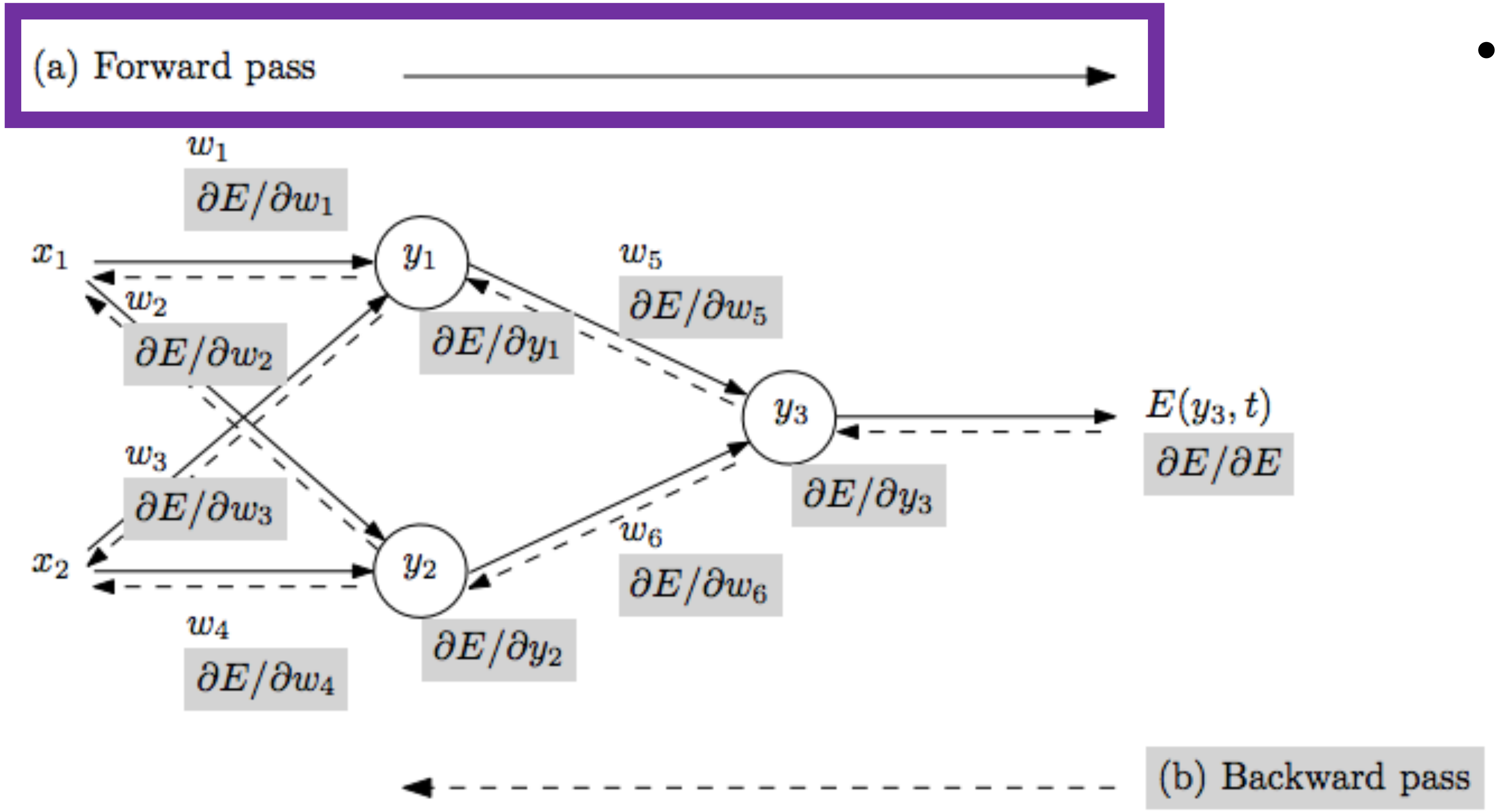
Example from: Jiawei Han and Micheline Kamber; Data Mining.

Example: Input Training Example



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

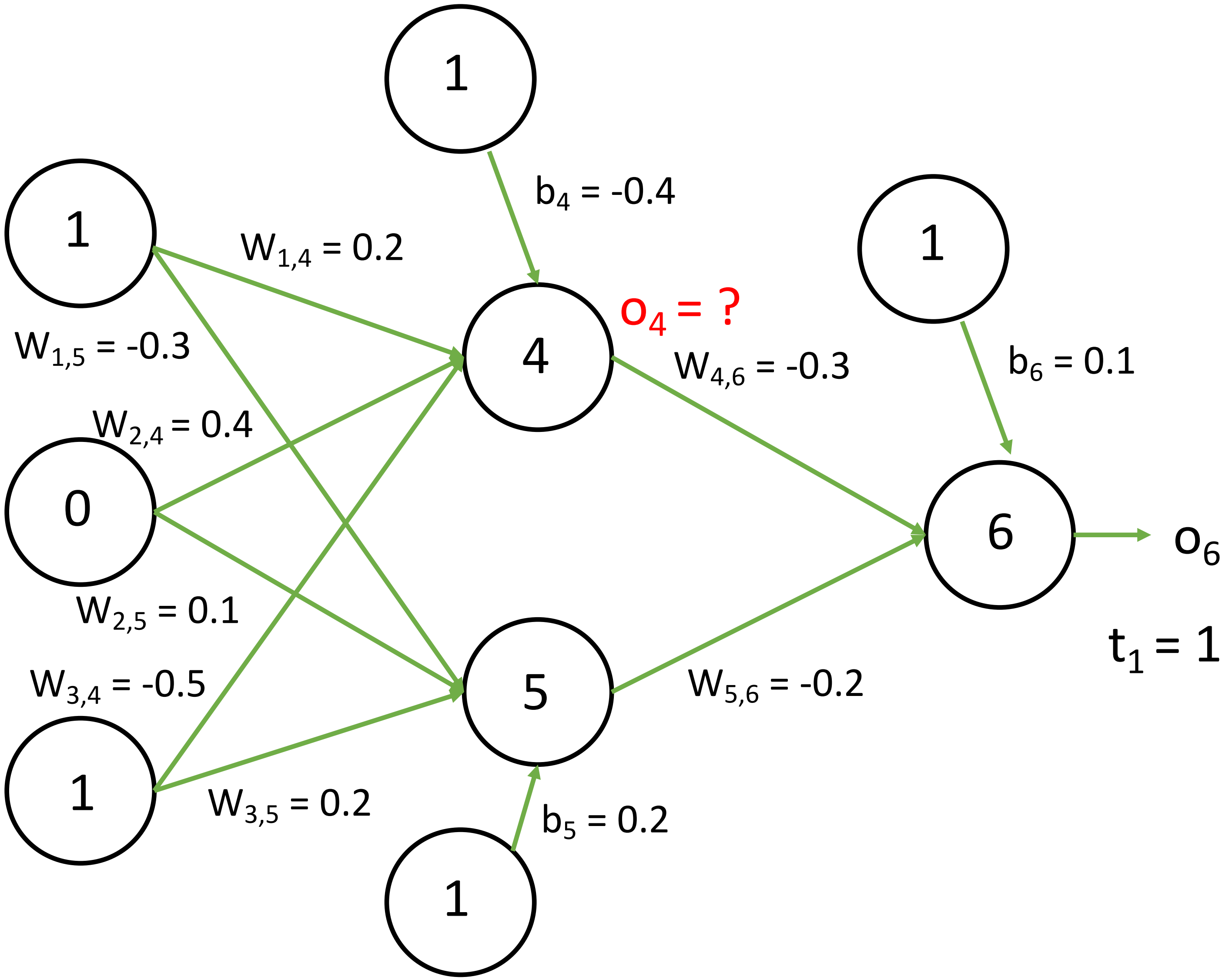
Example: Step 1 – Forward Pass



- 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

Example: Step 1 – Forward Pass



Input to node 4:

$i_4 = (1 \times 0.2 + 0 \times 0.4 + 1 \times -0.5) - 0.4$

$i_4 = -0.7$

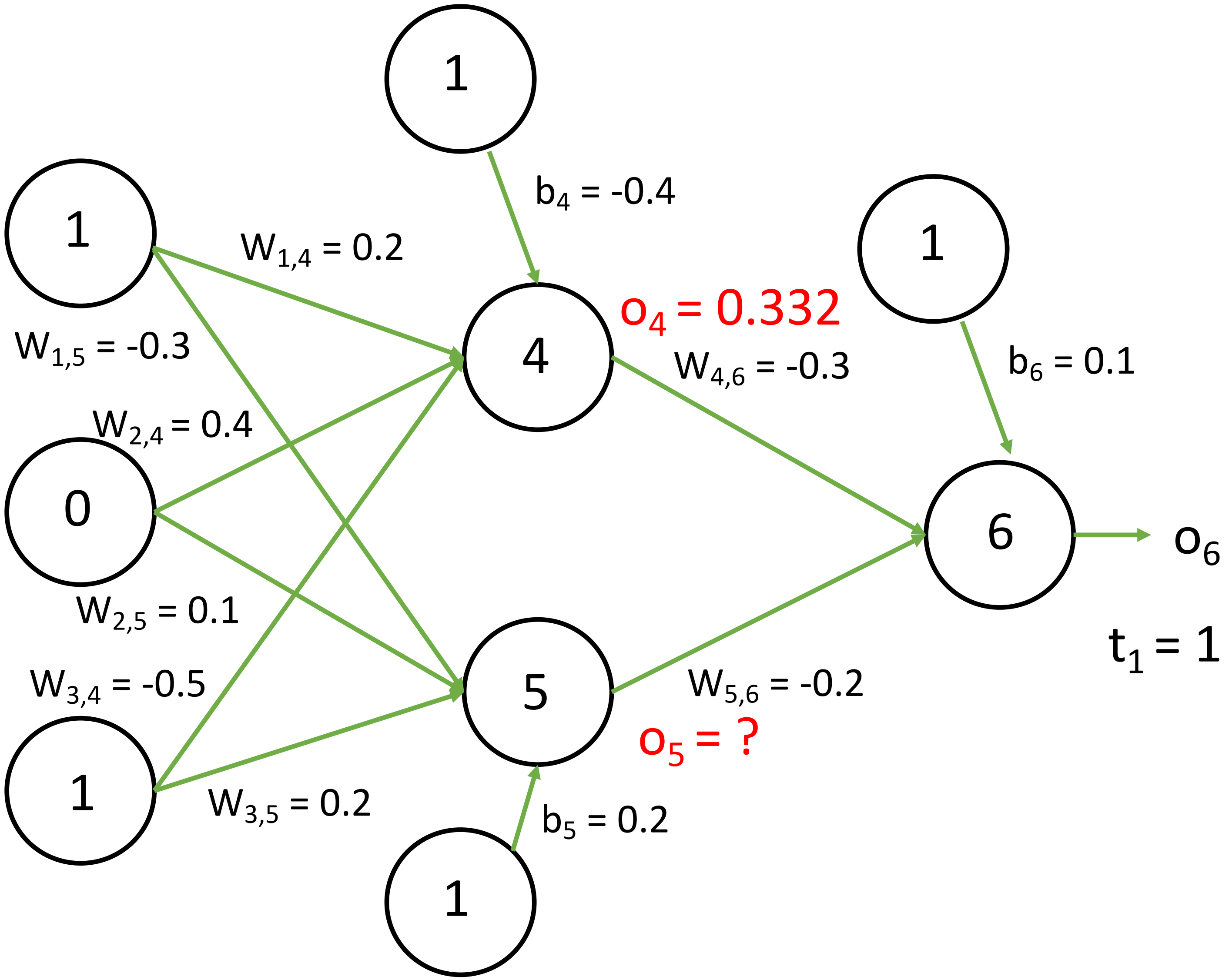
Output of node 4 (sigmoid function):

$o_4 = \textit{sigmoid}(-0.7)$

$o_4 = 1/(1+e^{-(-0.7)})$

$o_4 = 0.332$

Example: Step 1 – Forward Pass



Input to node 5:

$i_5 = (1 \text{ x } -0.3 + 0 \text{ x } 0.1 + 1 \text{ x } 0.2) + 0.2$

$i_5 = 0.1$

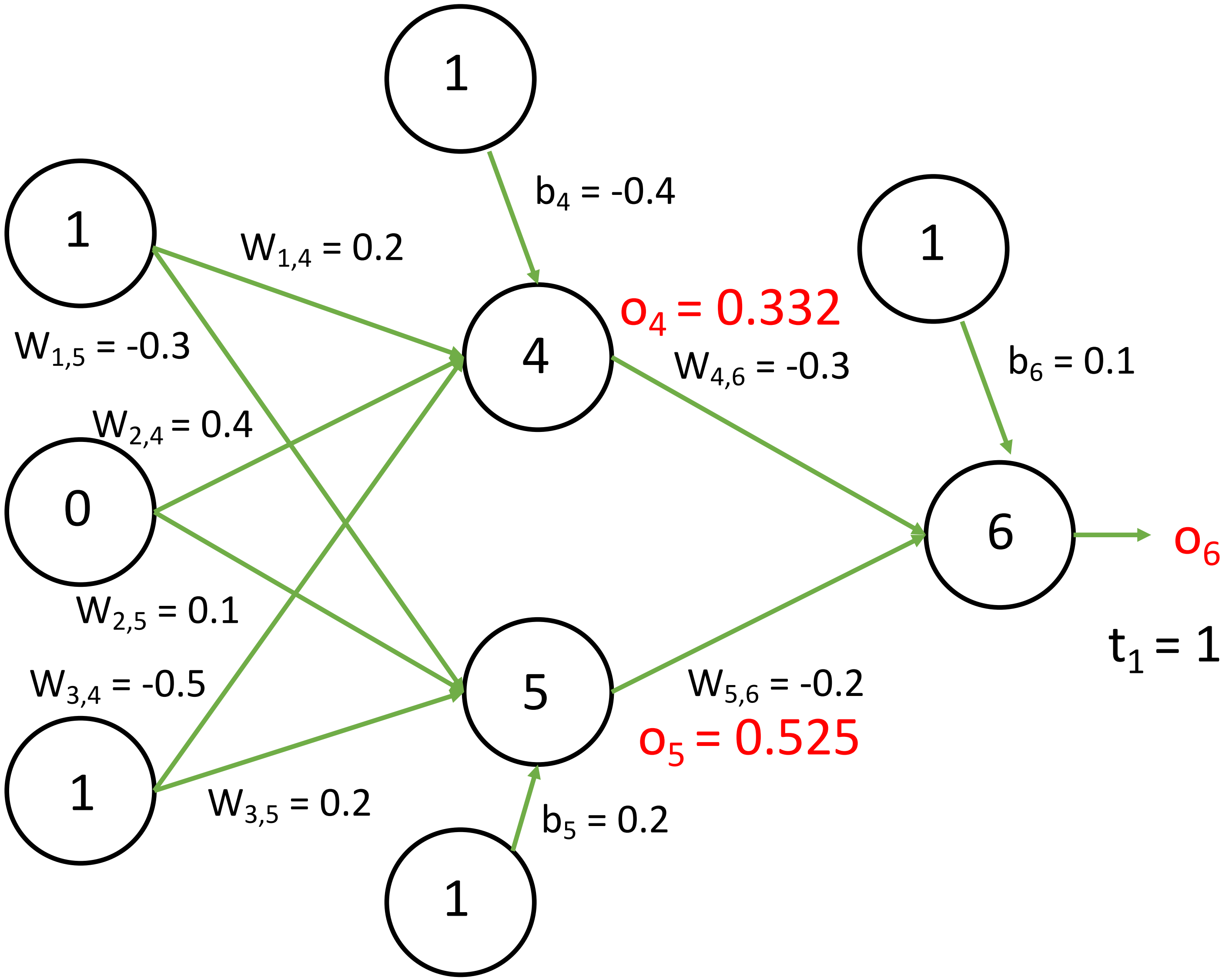
Output of node 5 (sigmoid function):

$o_5 = \mathit{sigmoid}(0.1)$

$o_5 = 1/(1+e^{-0.1})$

$o_5 = 0.525$

Example: Step 1 – Forward Pass



Input to node 6:

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

$i_6 = -0.105$

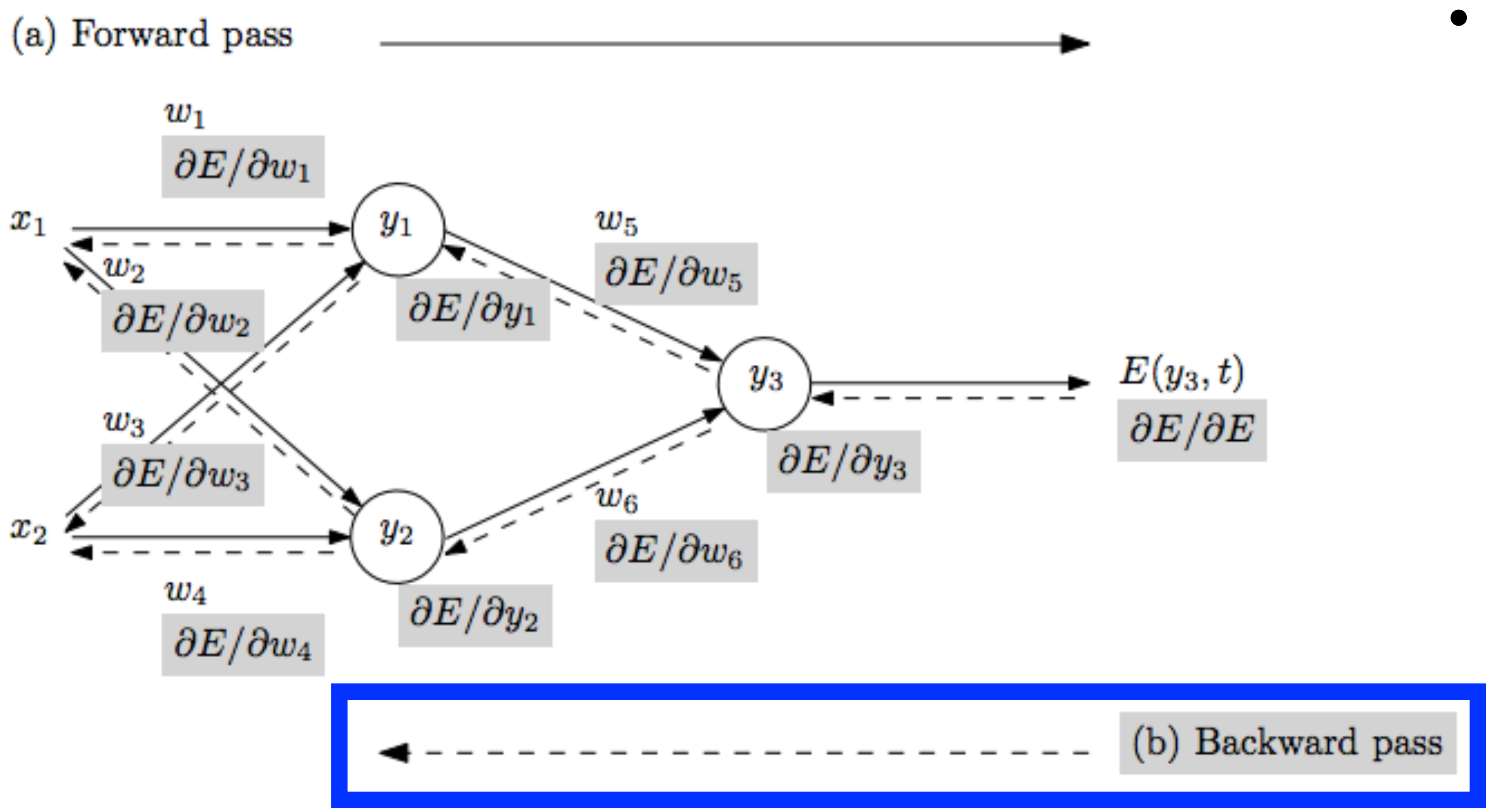
Output of node 6 (sigmoid function):

$o_6 = \textit{sigmoid}(-0.105)$

$o_6 = 1/(1+e^{-(-0.105)})$

$o_6 = 0.474$

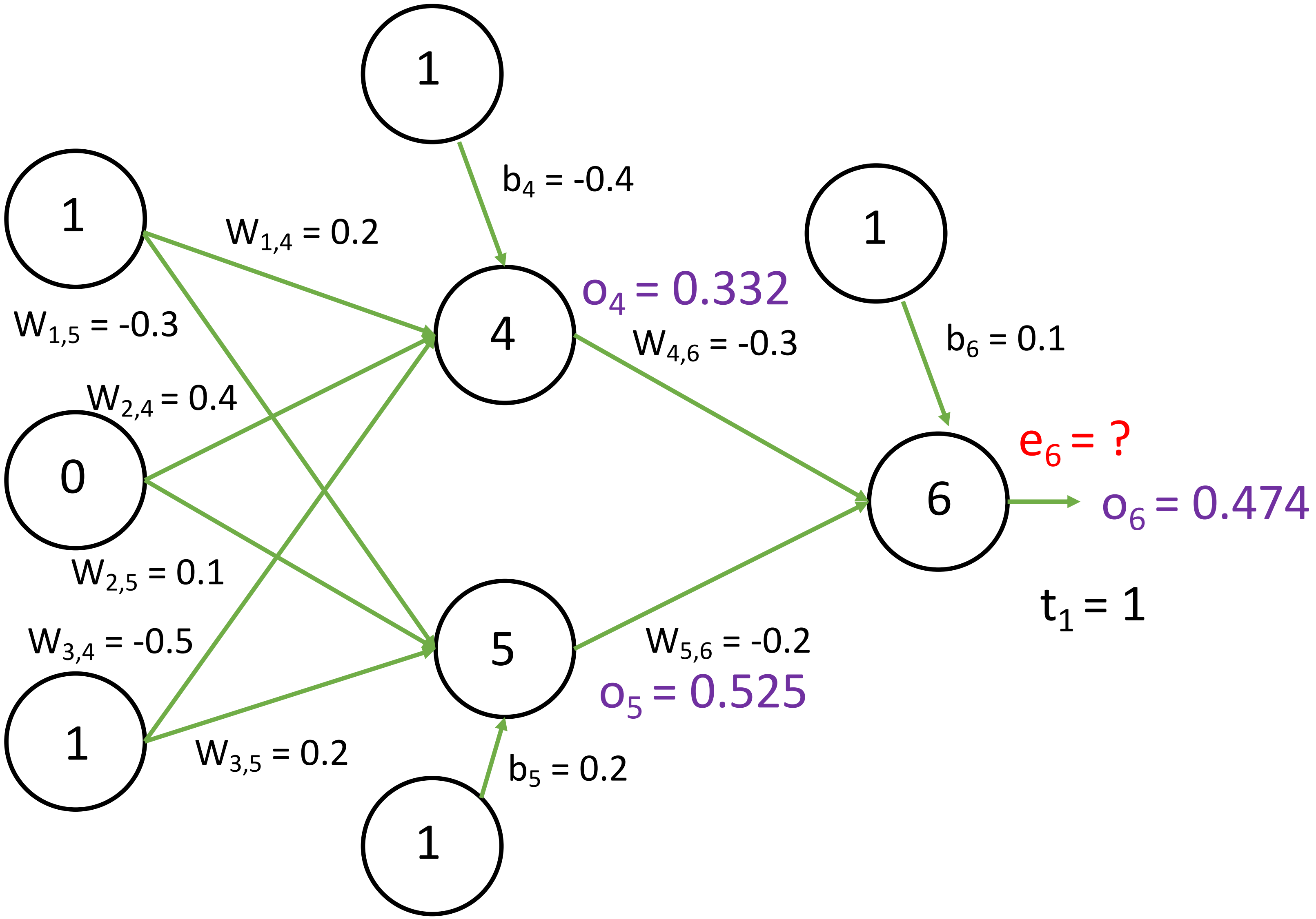
Example: Step 2 – Backward Pass



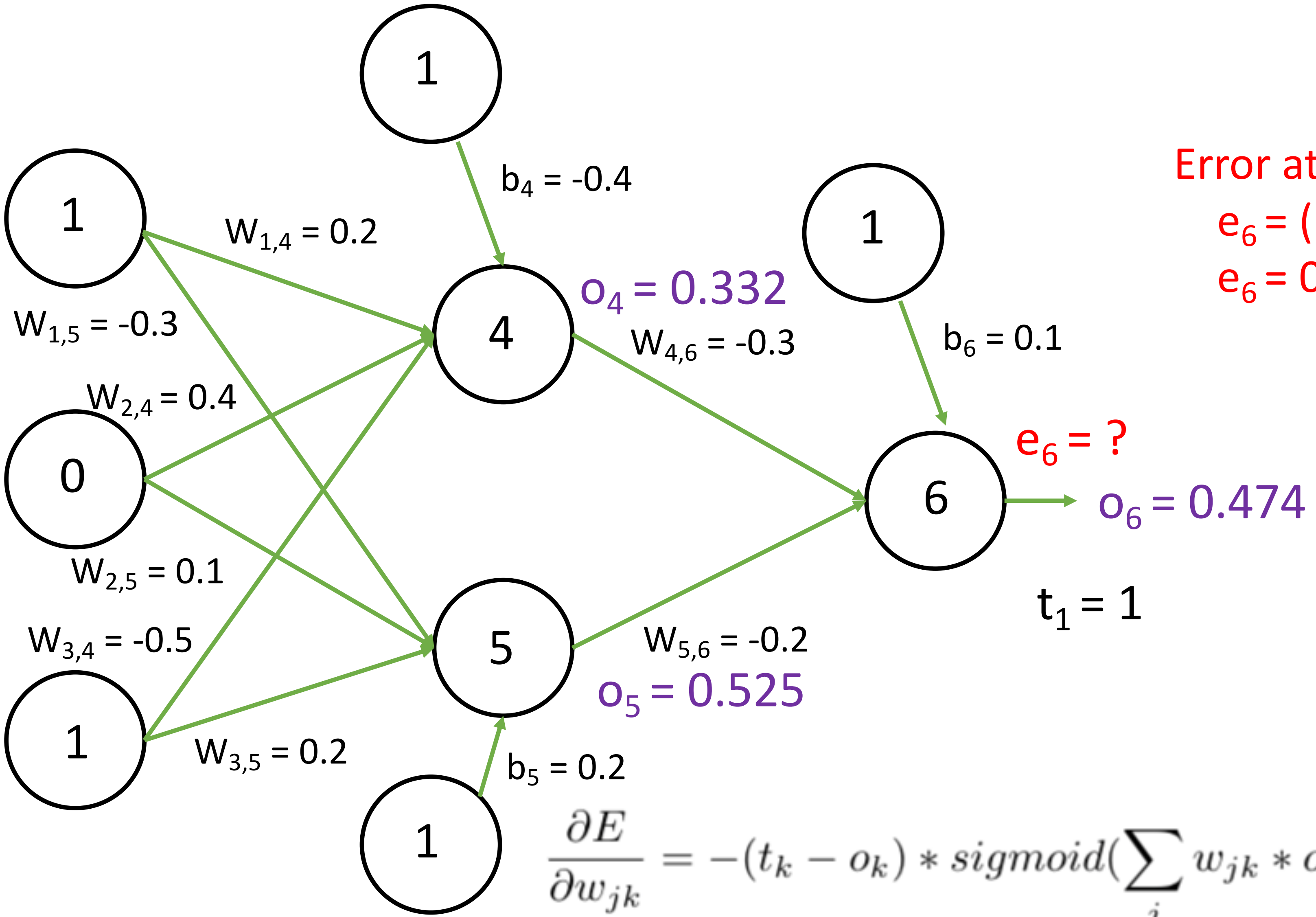
- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward

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

Example: Step 2 – Backward Pass



Example: Step 2 – Backward Pass



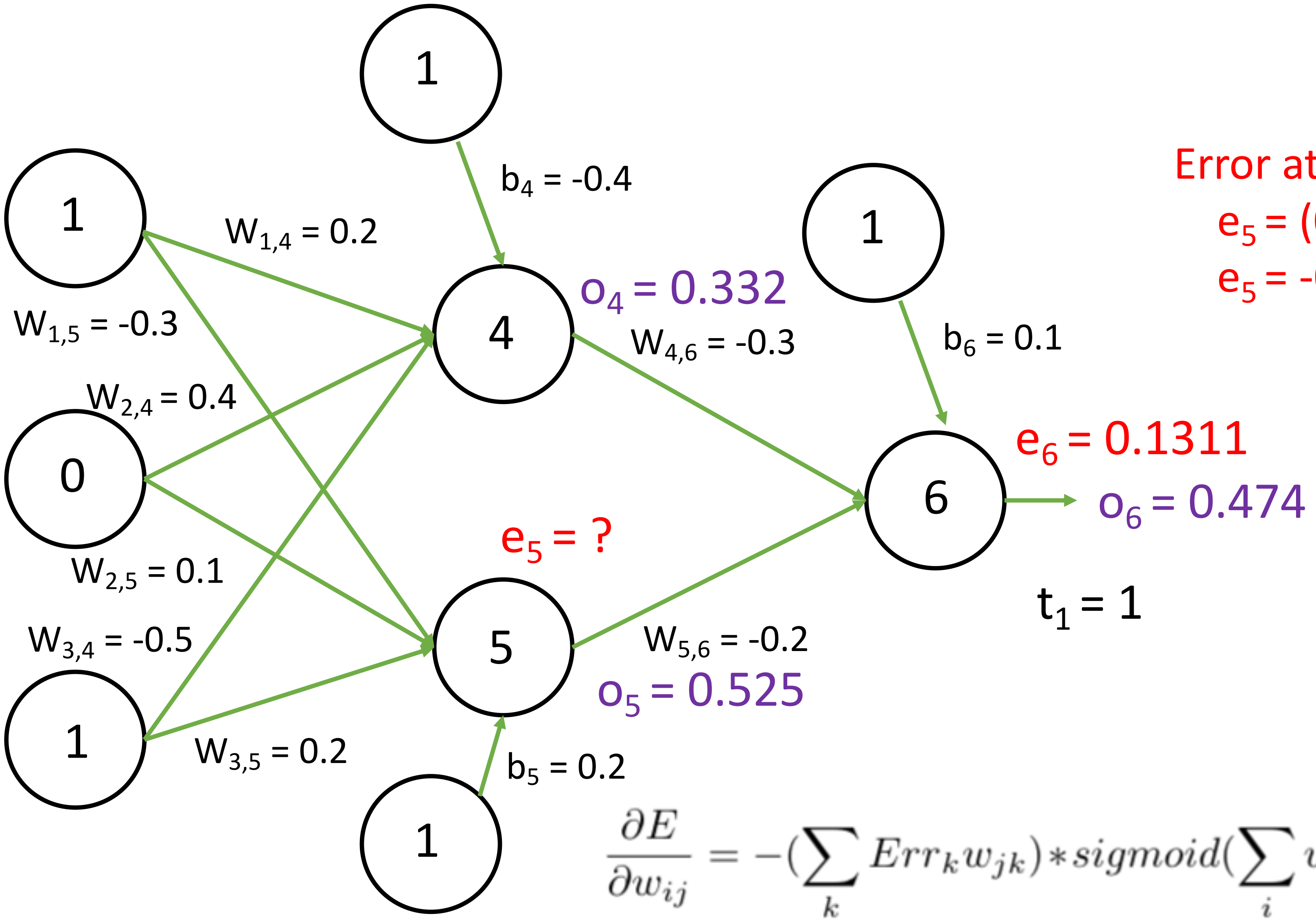
Error at node 6:

$e_6 = (1\text{-}0.474)\ (0.474)(1\text{-}0.474)$

$e_6 = 0.1311$

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

Example: Step 2 – Backward Pass



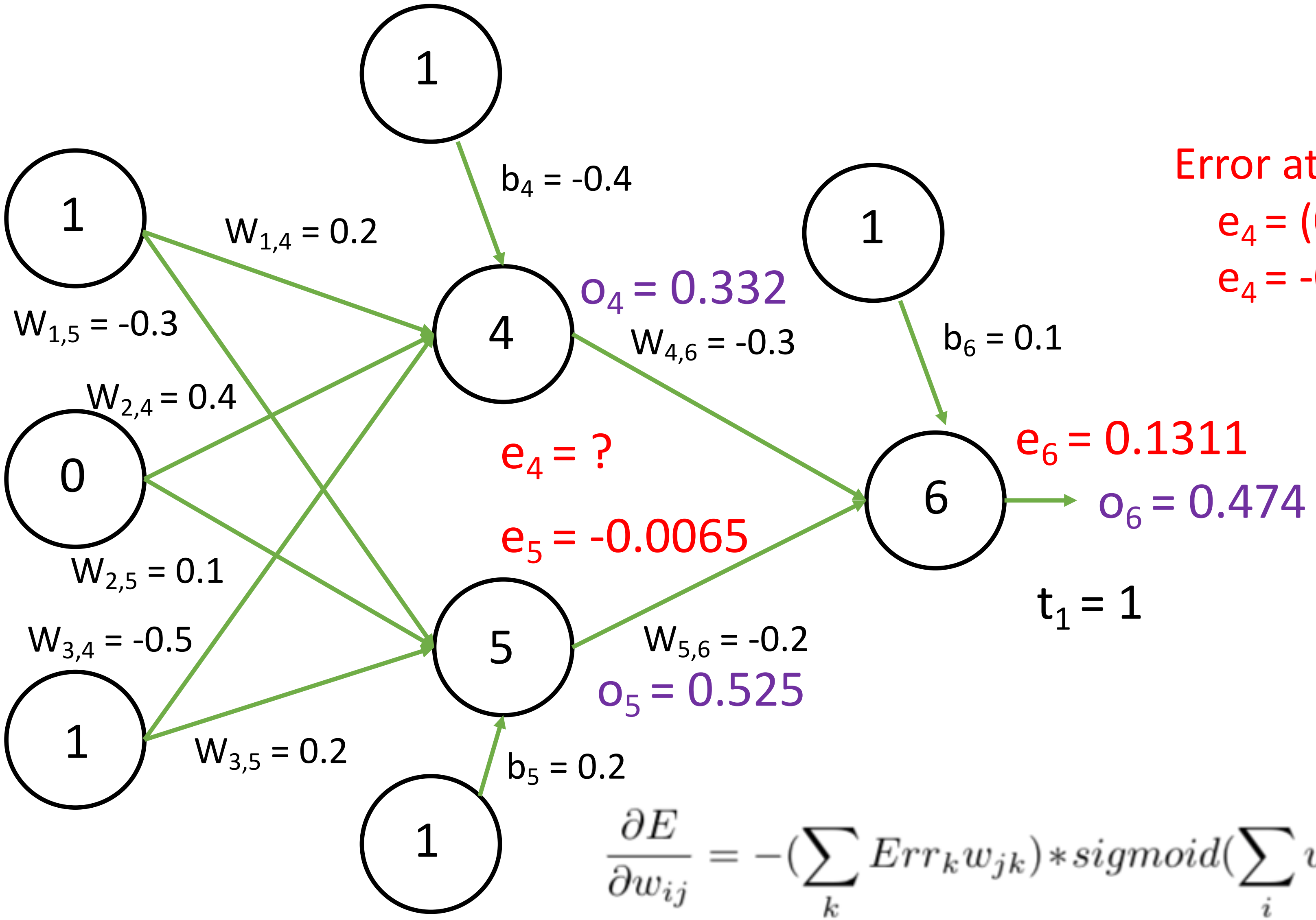
Error at node 5:

e_5 = (0.1311)(-0.2)(0.525)(1-0.525)

e_5 = -0.0065

$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$

Example: Step 2 – Backward Pass



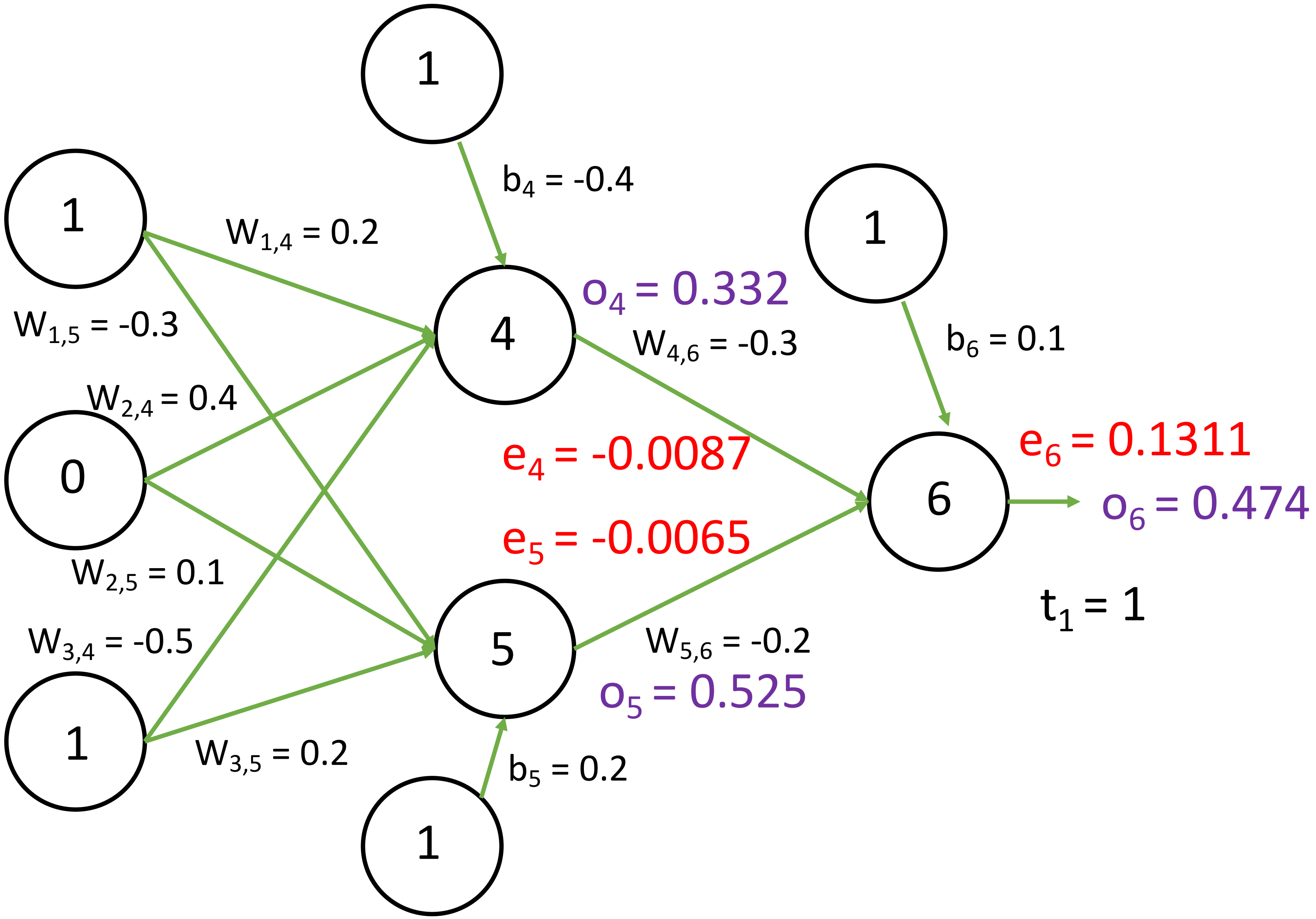
Error at node 4:

$e_4 = (0.1311)(-0.3)(0.332)(1-0.332)$

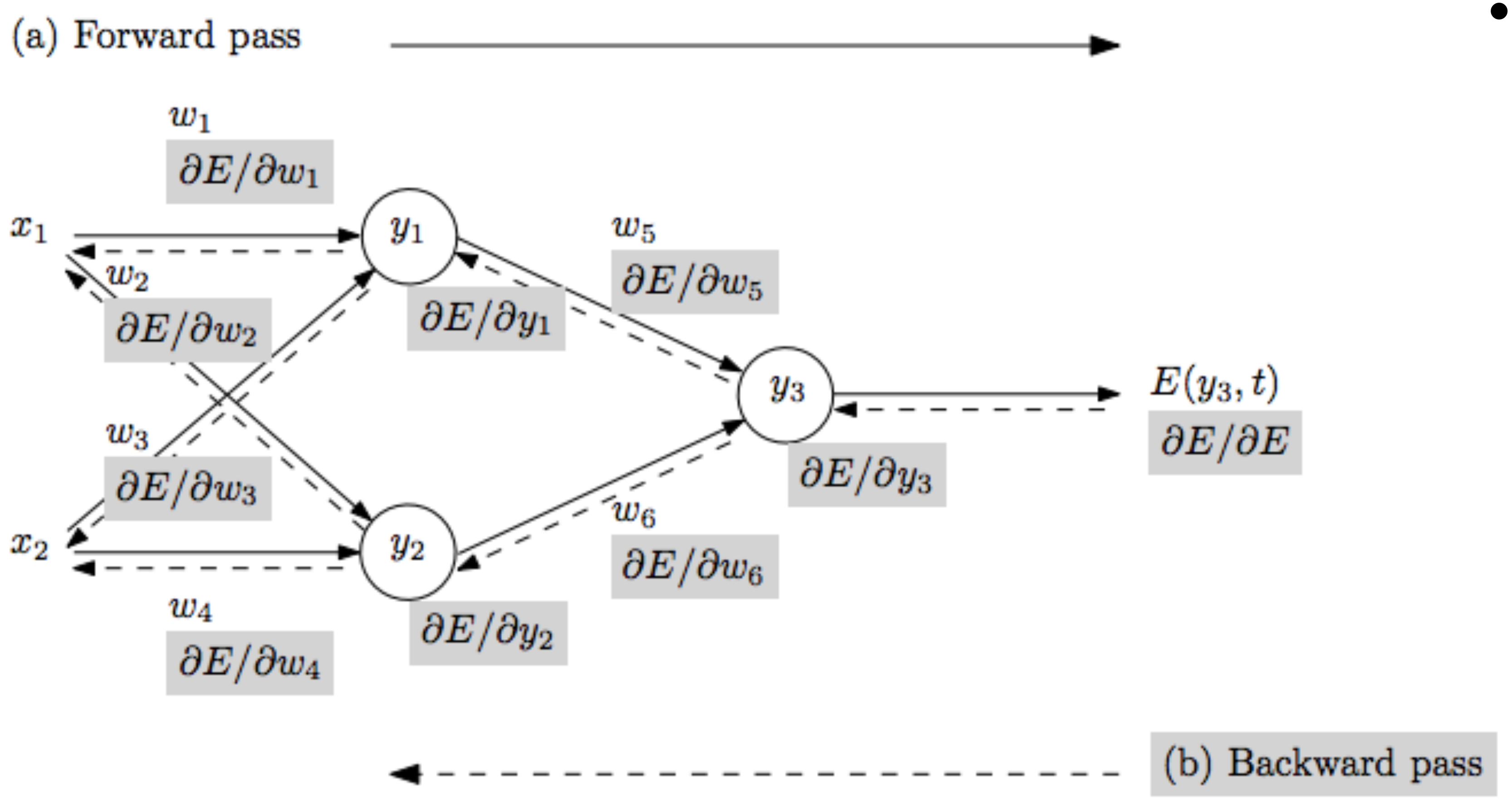
$e_4 = -0.0087$

$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$

Example: Step 2 – Backward Pass



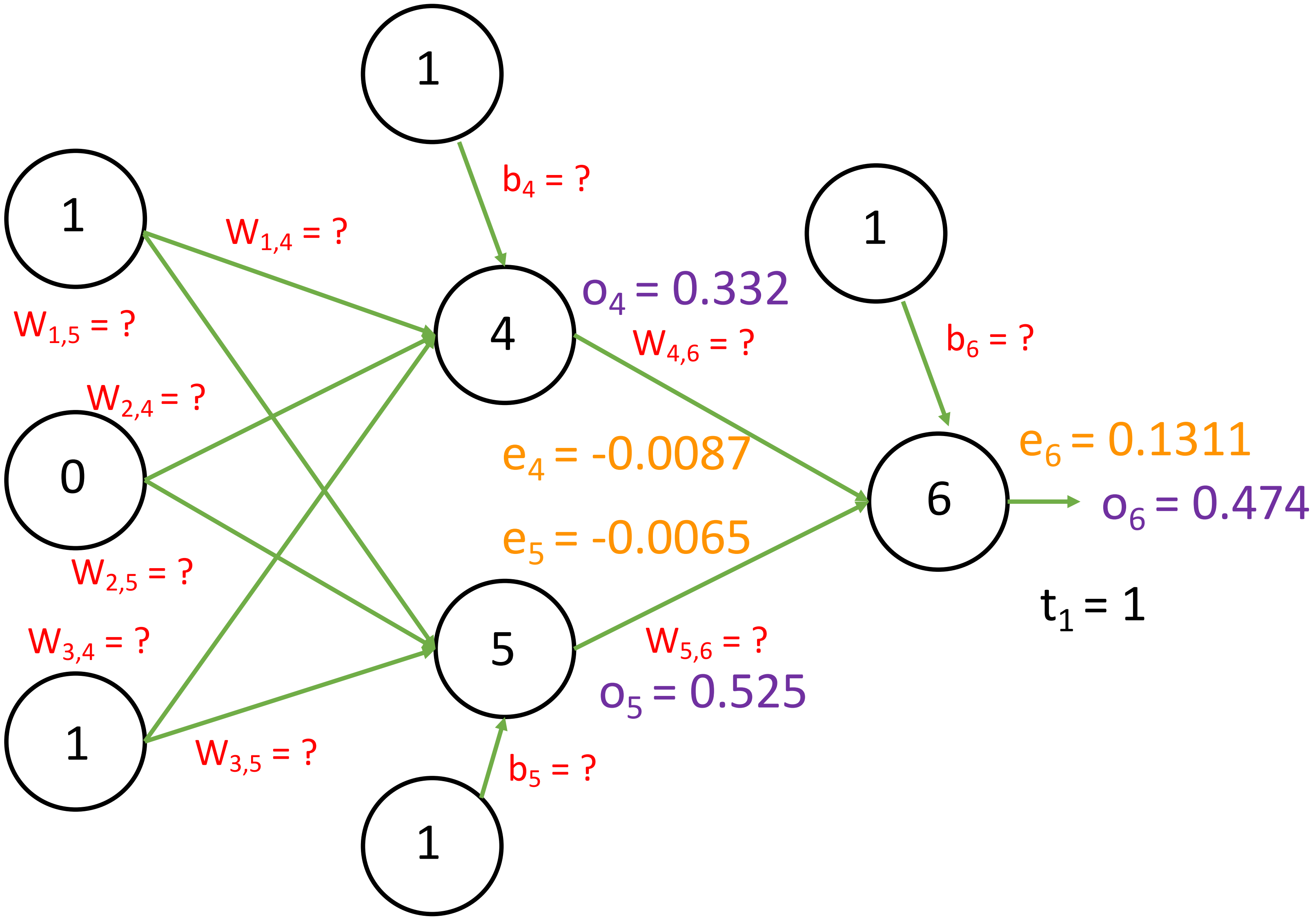
Example: Step 3 – Update Weights



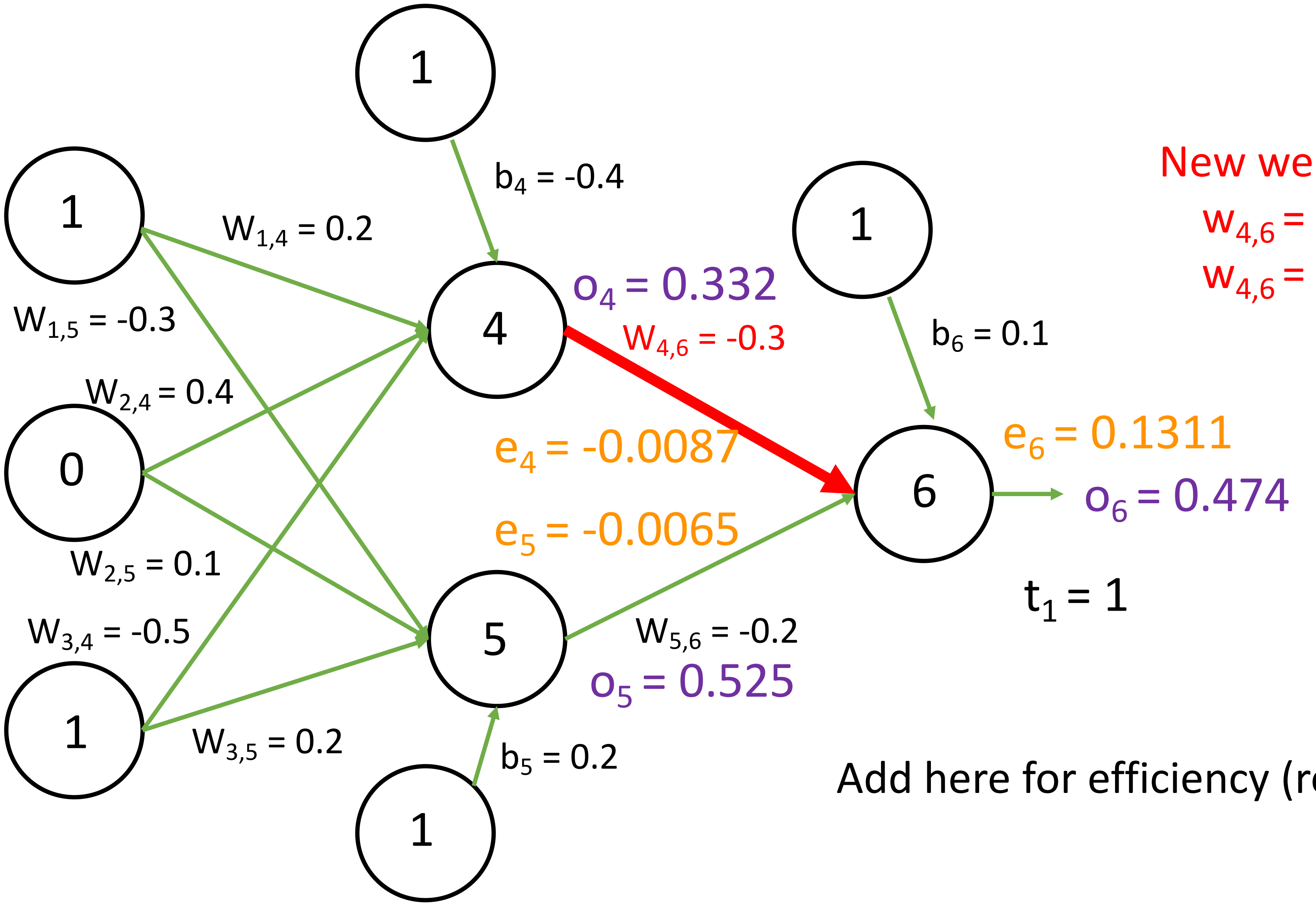
- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

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

Example: Step 3 – Update Weights



Example: Step 3 – Update Weights



New weights (learning rate = 0.9):

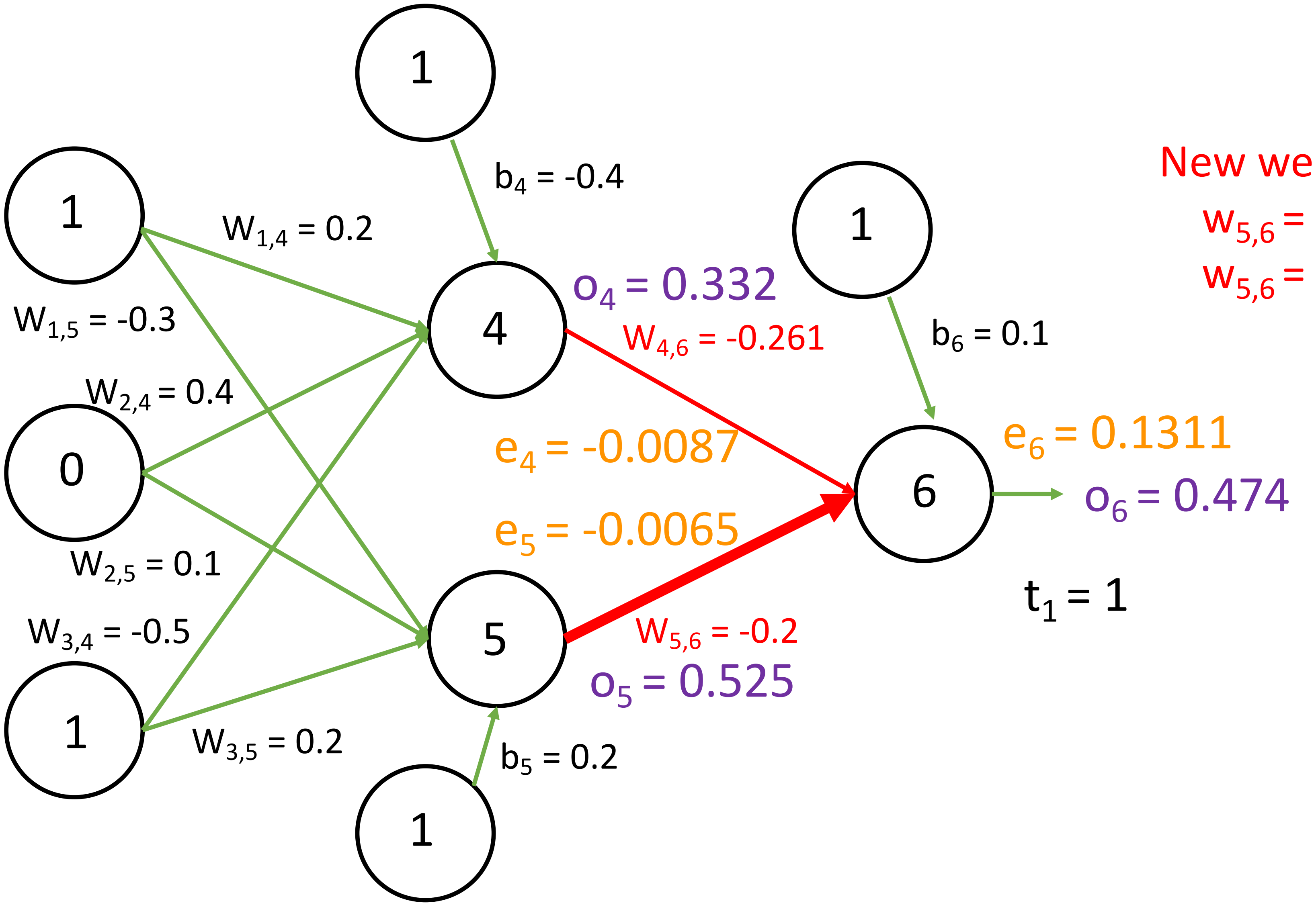
$w_{4,6}$ = -0.3 + (0.9)(0.1311)(0.332)

$w_{4,6}$ = -0.261

Add here for efficiency (removed from earlier equation)

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

Example: Step 3 – Update Weights



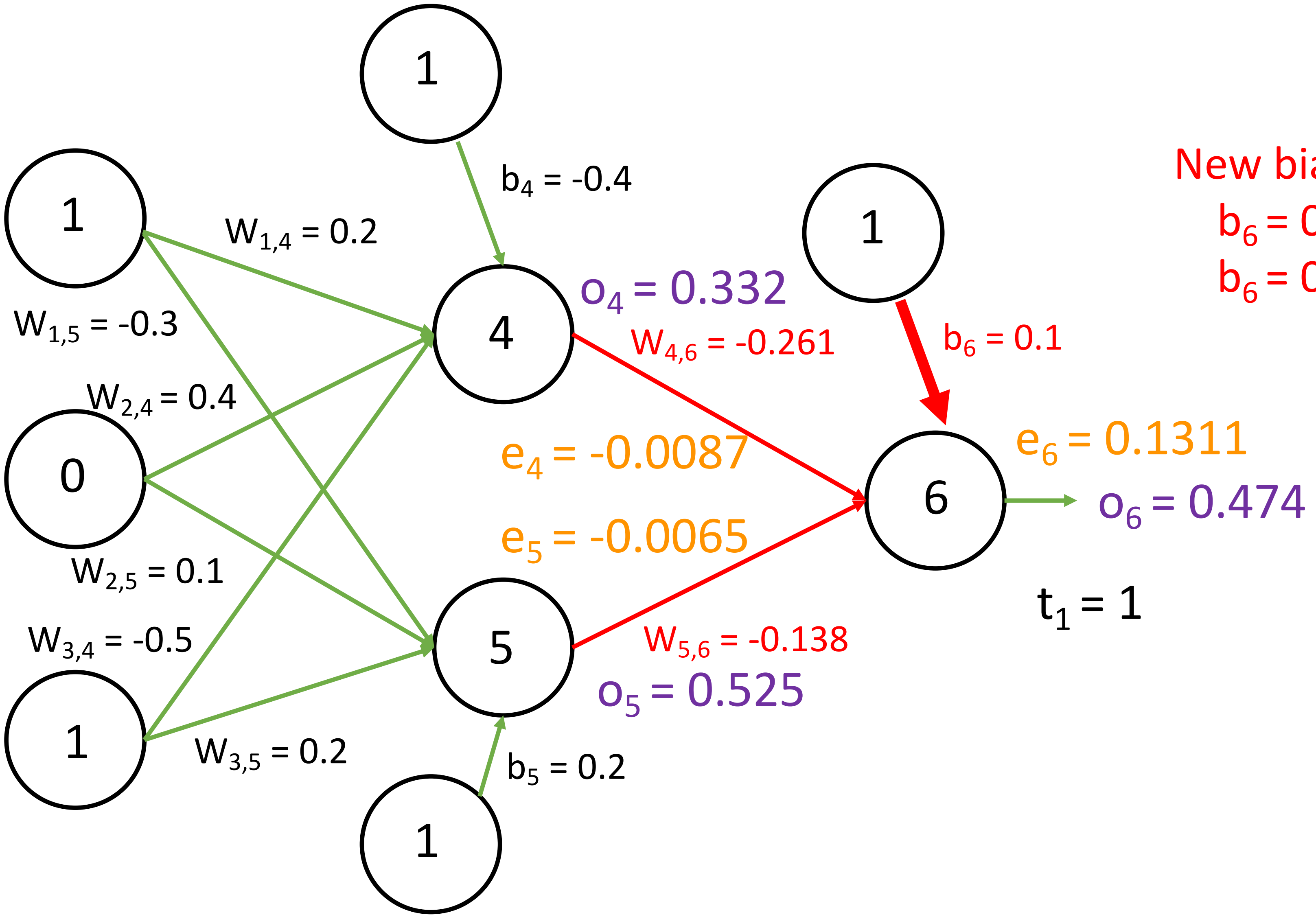
New weights (learning rate = 0.9):

$w_{5,6} = -0.2 + (0.9)(0.1311)(0.525)$

$w_{5,6} = -0.138$

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

Example: Step 3 – Update Weights



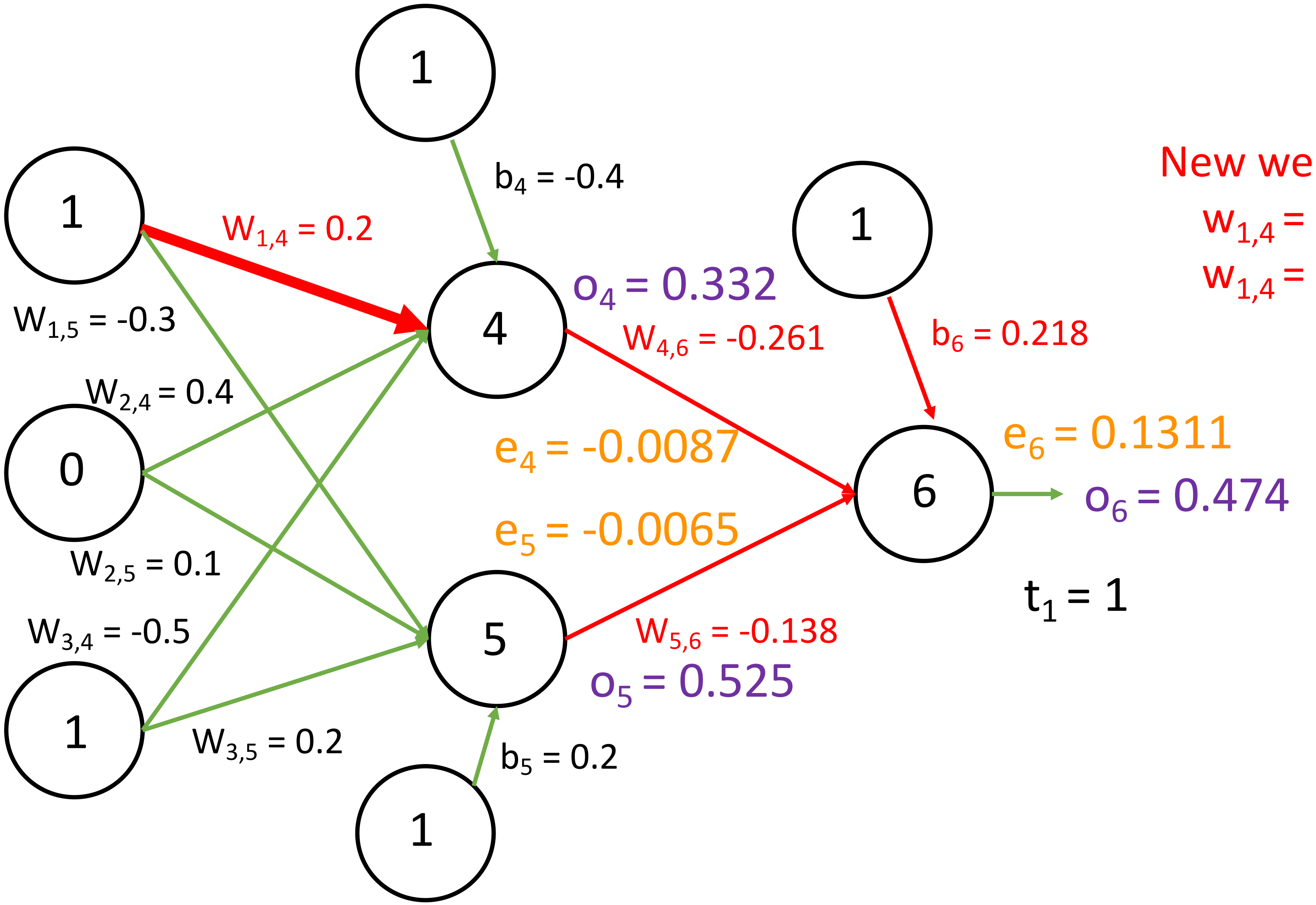
New bias (learning rate = 0.9):

$b_6 = 0.1 + (0.9)(0.1311)$

$b_6 = 0.218$

$$b_k = b_k + \eta Err_k$$

Example: Step 3 – Update Weights



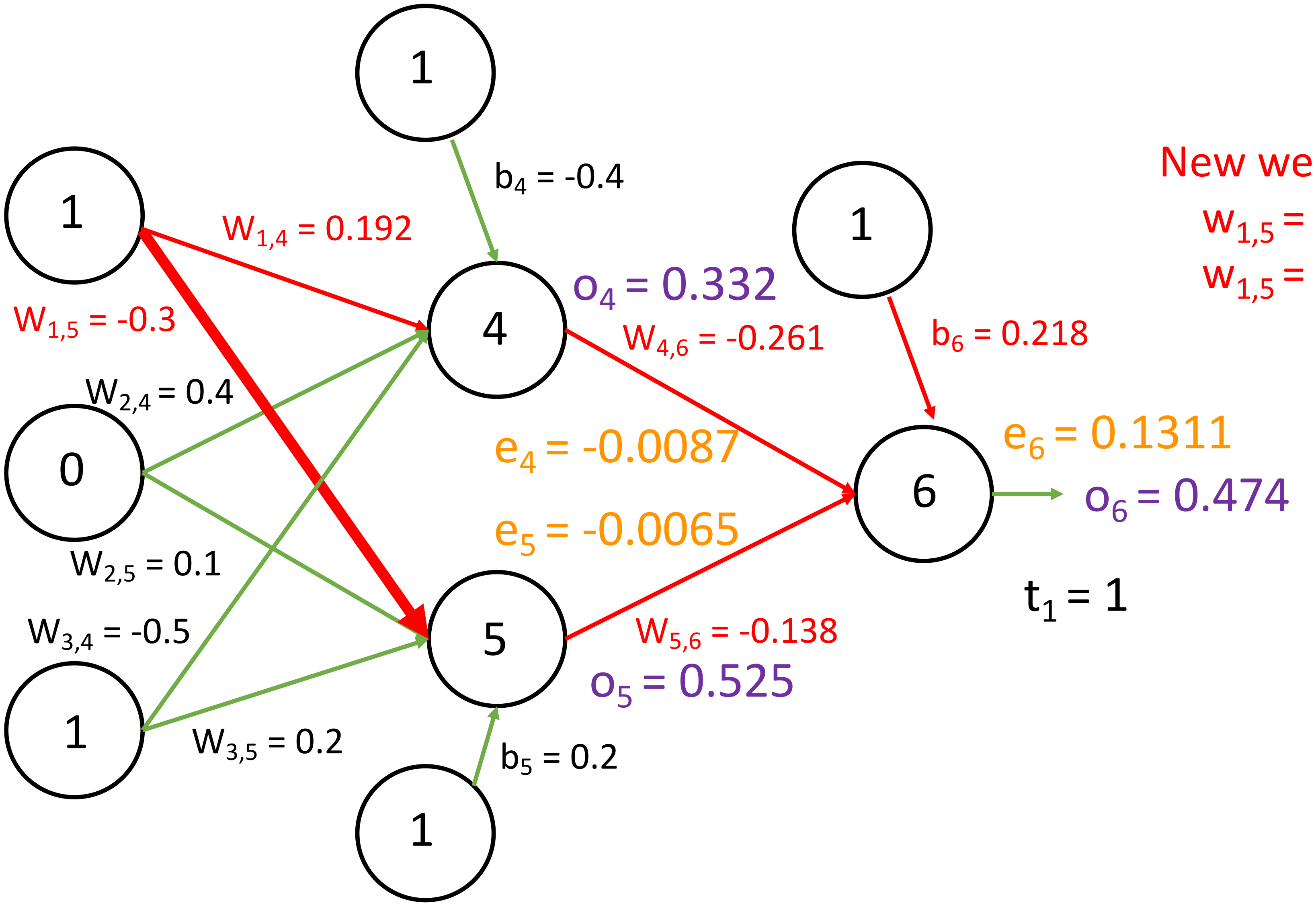
New weights (learning rate = 0.9):

$w_{1,4} = 0.2 + (0.9)(-0.0087)(1)$

$w_{1,4} = 0.192$

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

Example: Step 3 – Update Weights



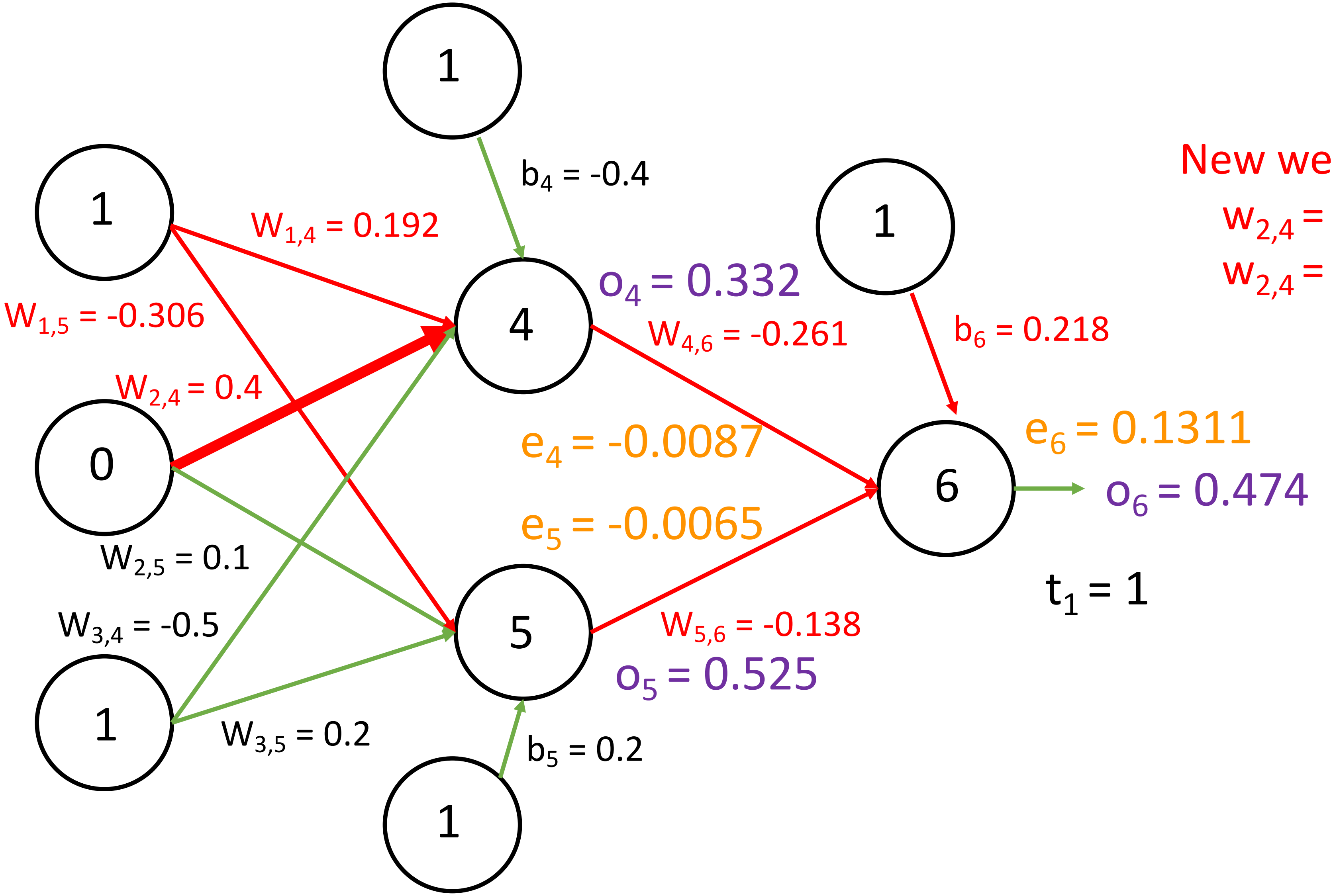
New weights (learning rate = 0.9):

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

$w_{1,5} = -0.306$

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

Example: Step 3 – Update Weights



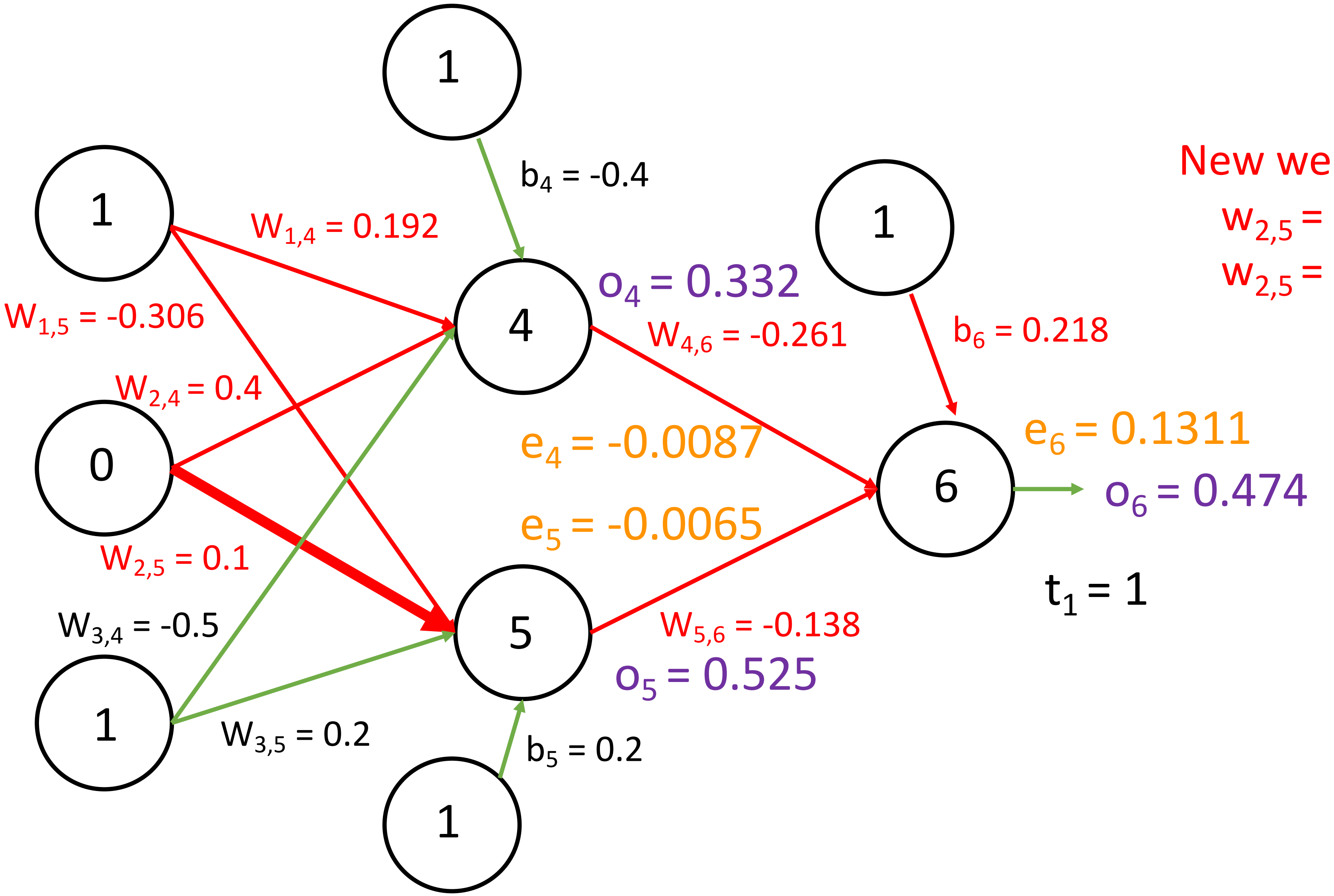
New weights (learning rate = 0.9):

$w_{2,4} = 0.4 + (0.9)(-0.0087)(0)$

$w_{2,4} = 0.4$

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

Example: Step 3 – Update Weights



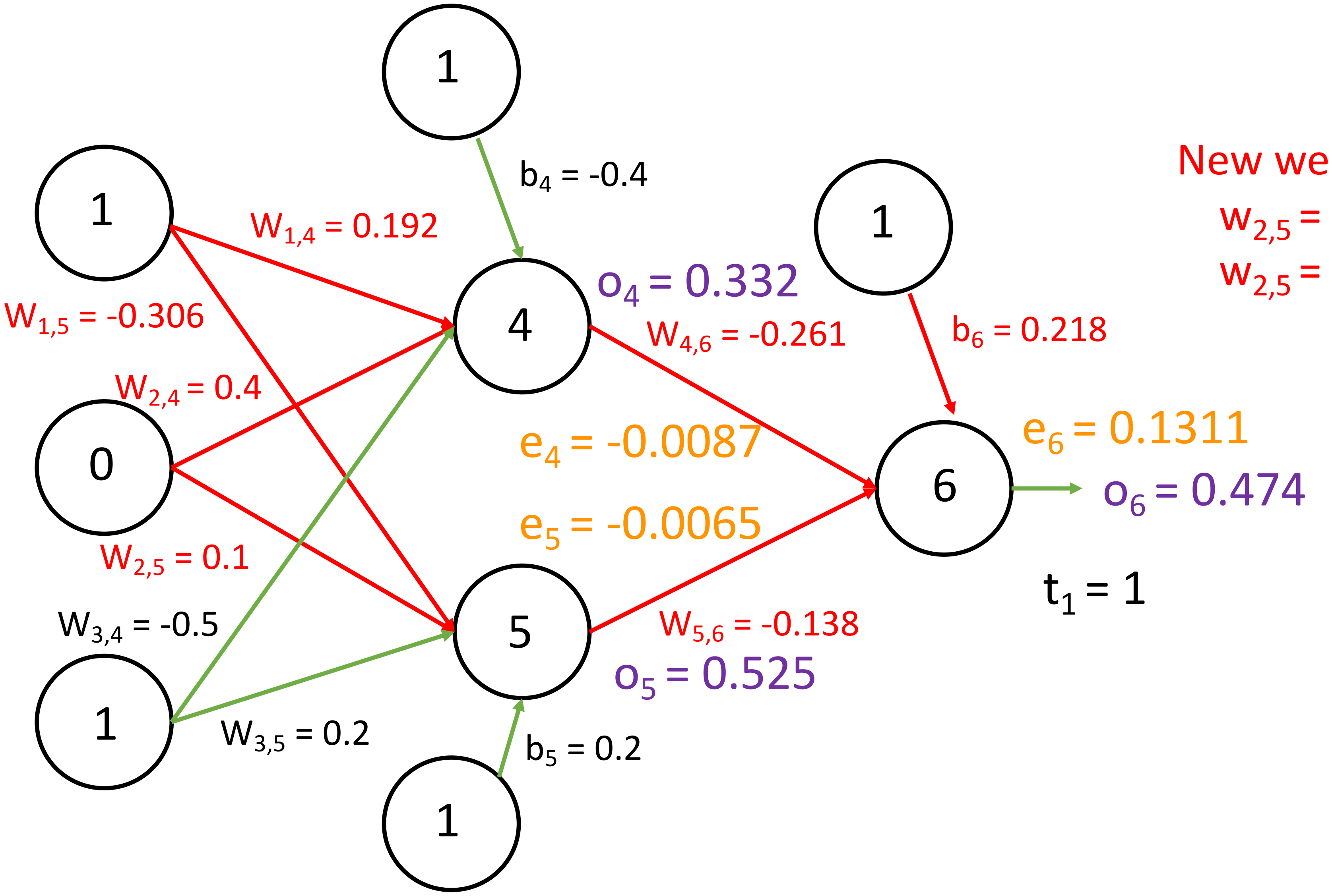
New weights (learning rate = 0.9):

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

$w_{2,5} = 0.1$

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

Example: Step 3 – Update Weights



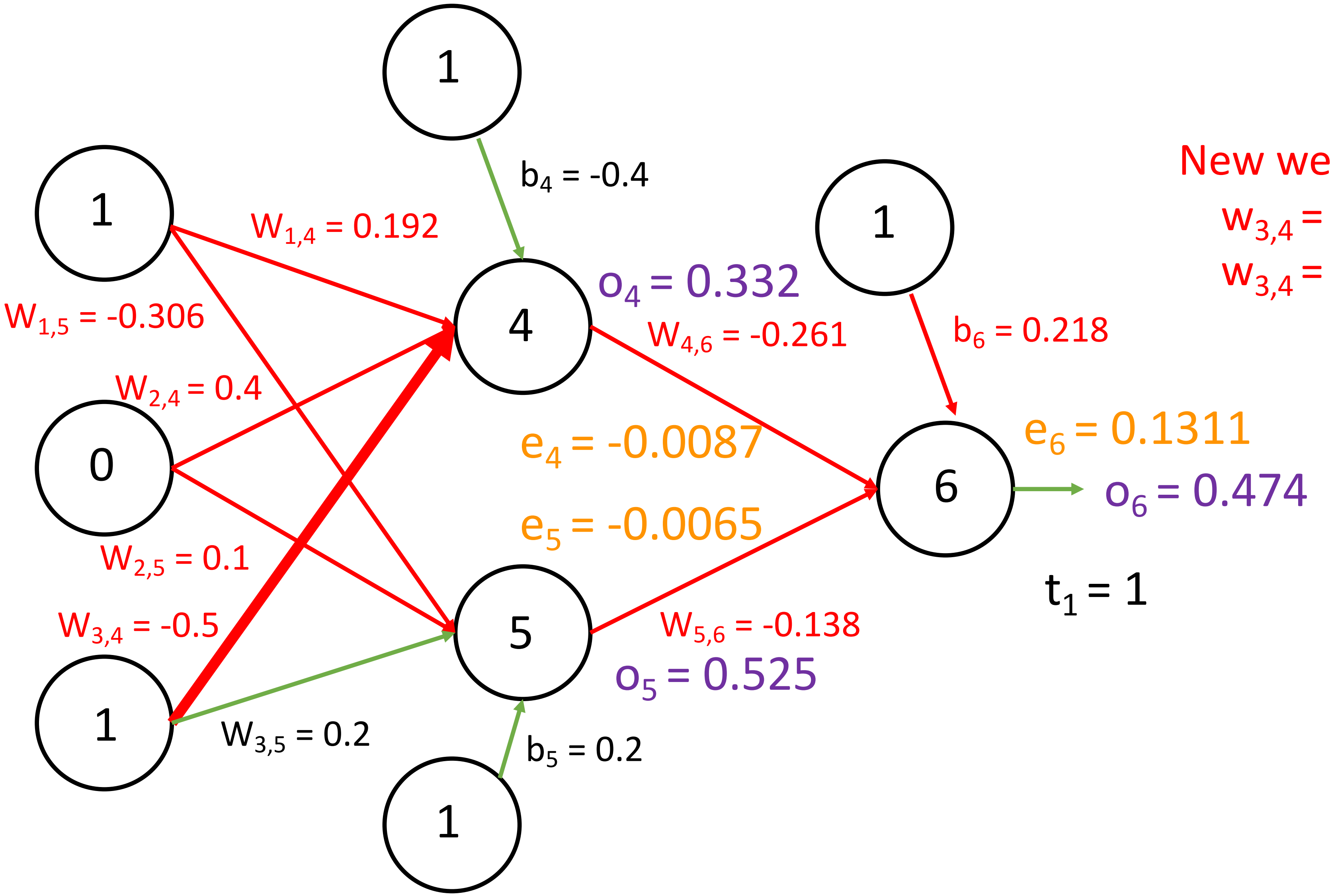
New weights (learning rate = 0.9):

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

$w_{2,5} = 0.1$

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

Example: Step 3 – Update Weights



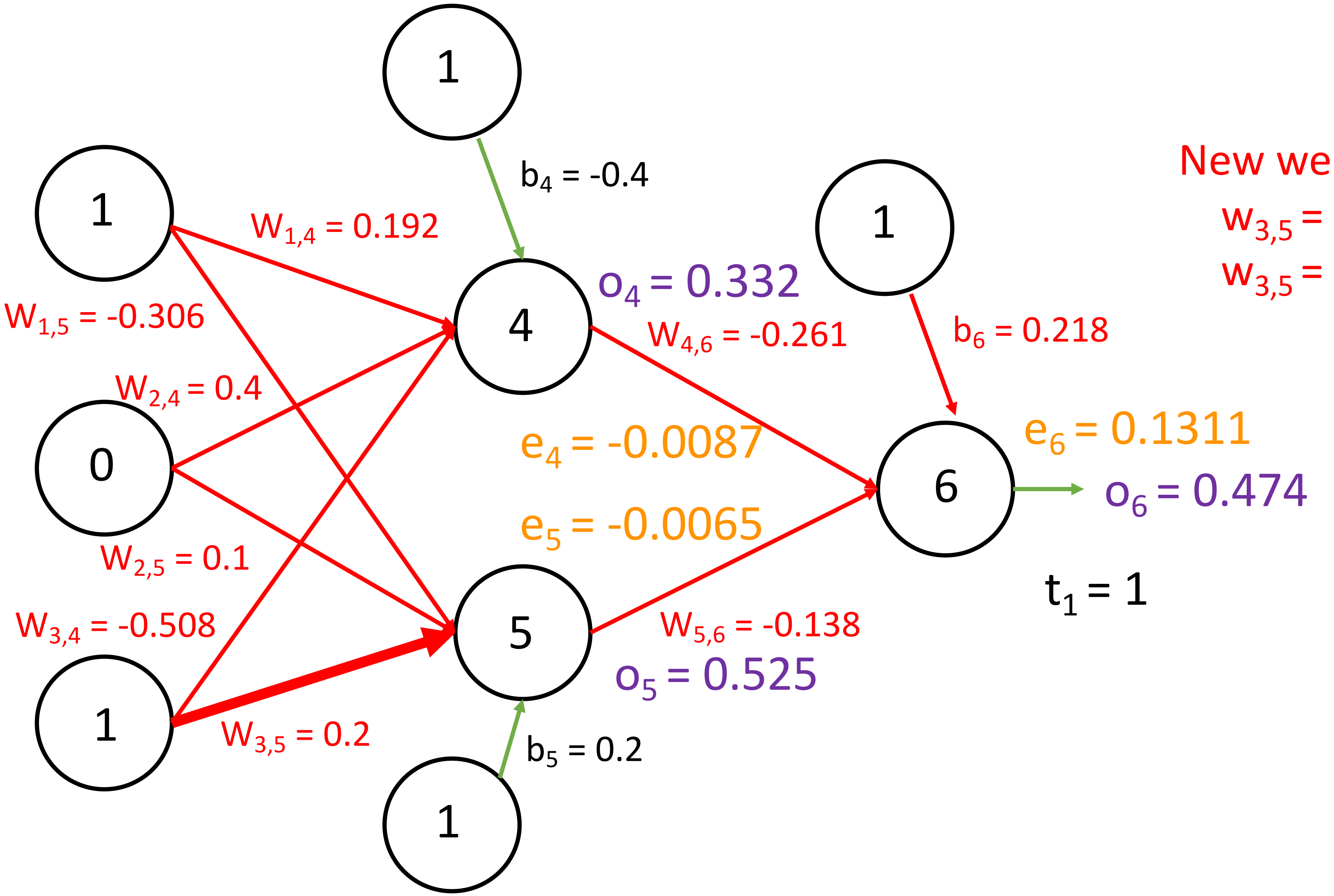
New weights (learning rate = 0.9):

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

$w_{3,4} = -0.508$

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

Example: Step 3 – Update Weights



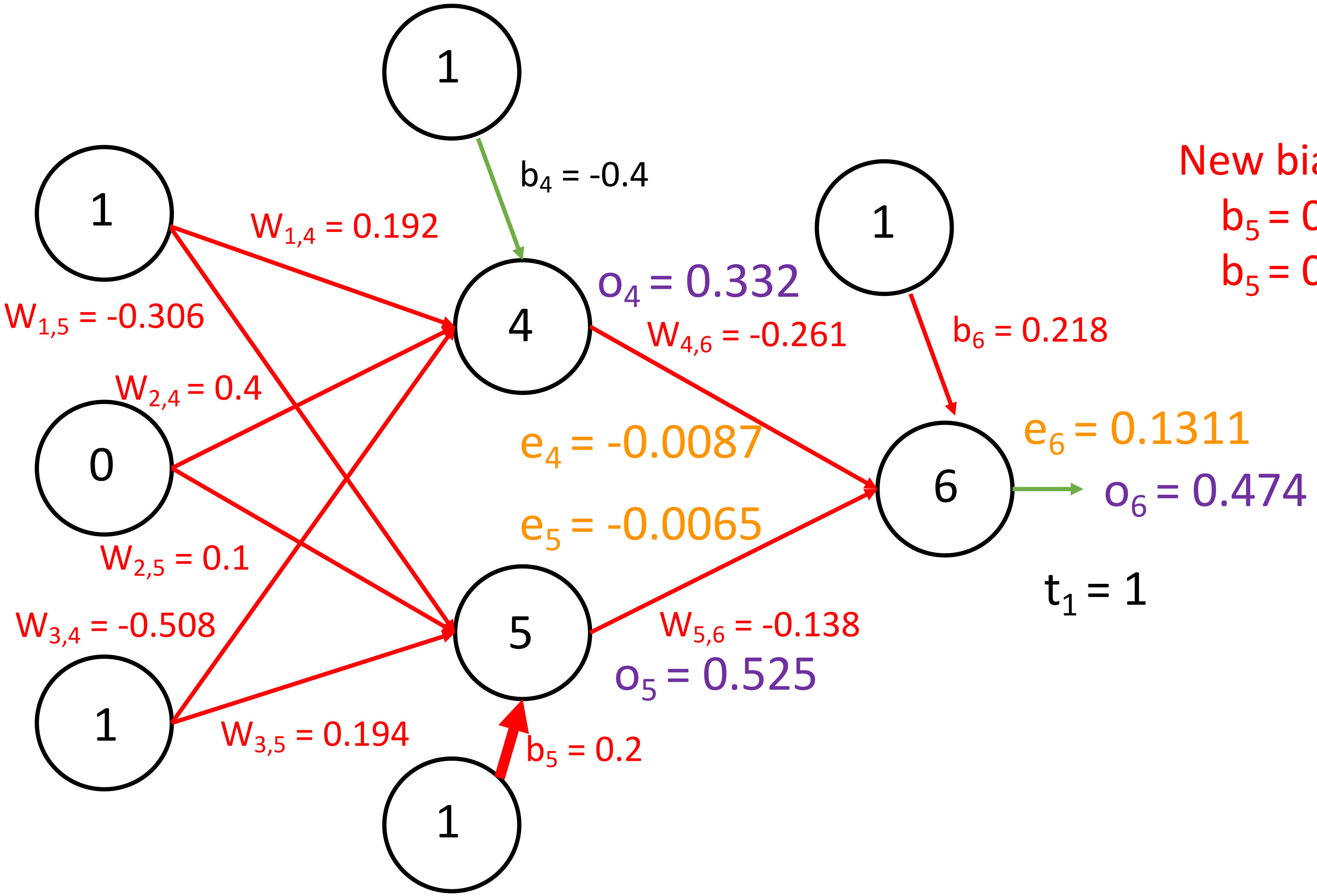
New weights (learning rate = 0.9):

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

$w_{3,5} = 01.94$

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

Example: Step 3 – Update Weights



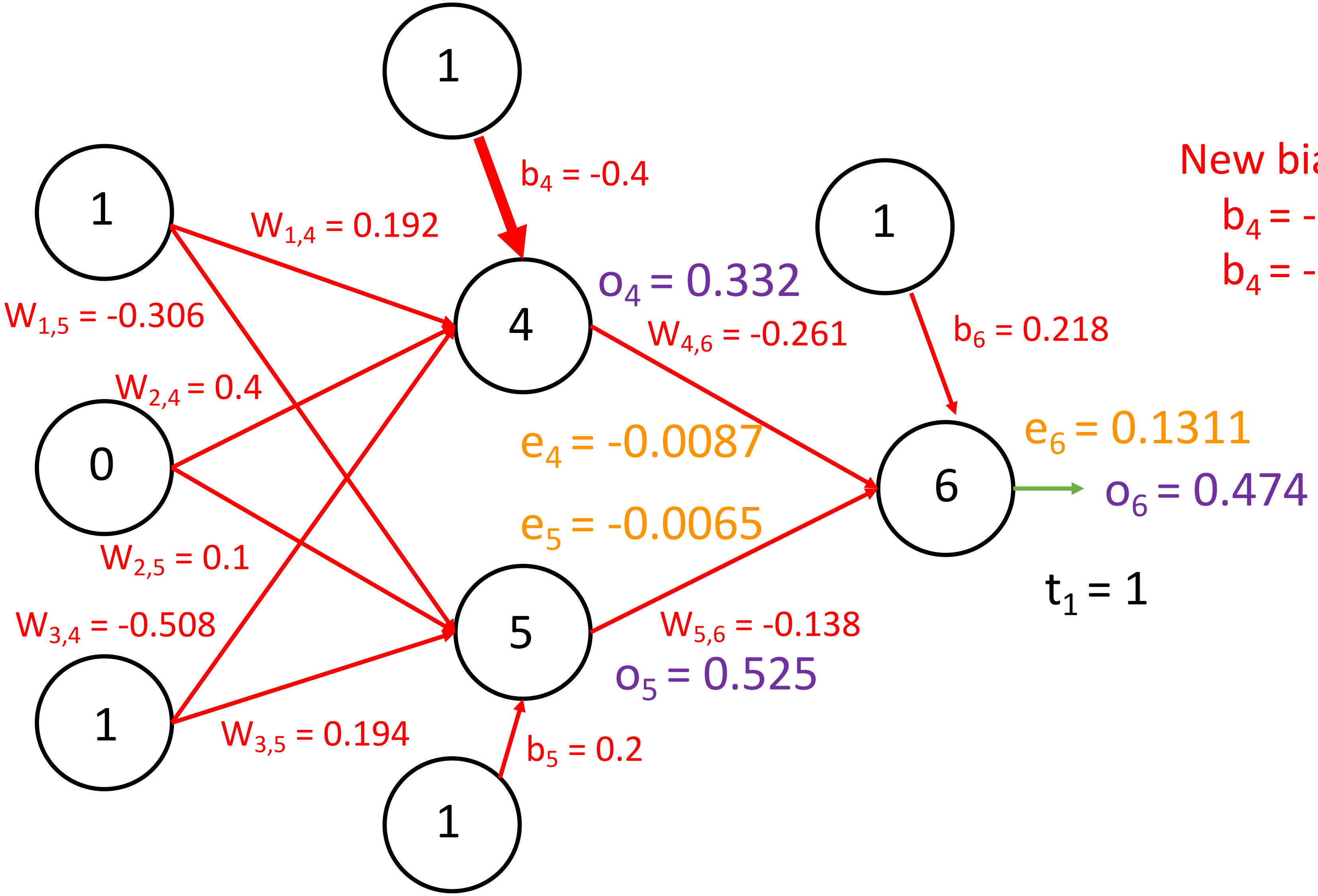
New bias (learning rate = 0.9):

$b_5 = 0.2 + (0.9)(-0.0065)$

$b_5 = 0.194$

$$b_k = b_k + \eta Err_k$$

Example: Step 3 – Update Weights



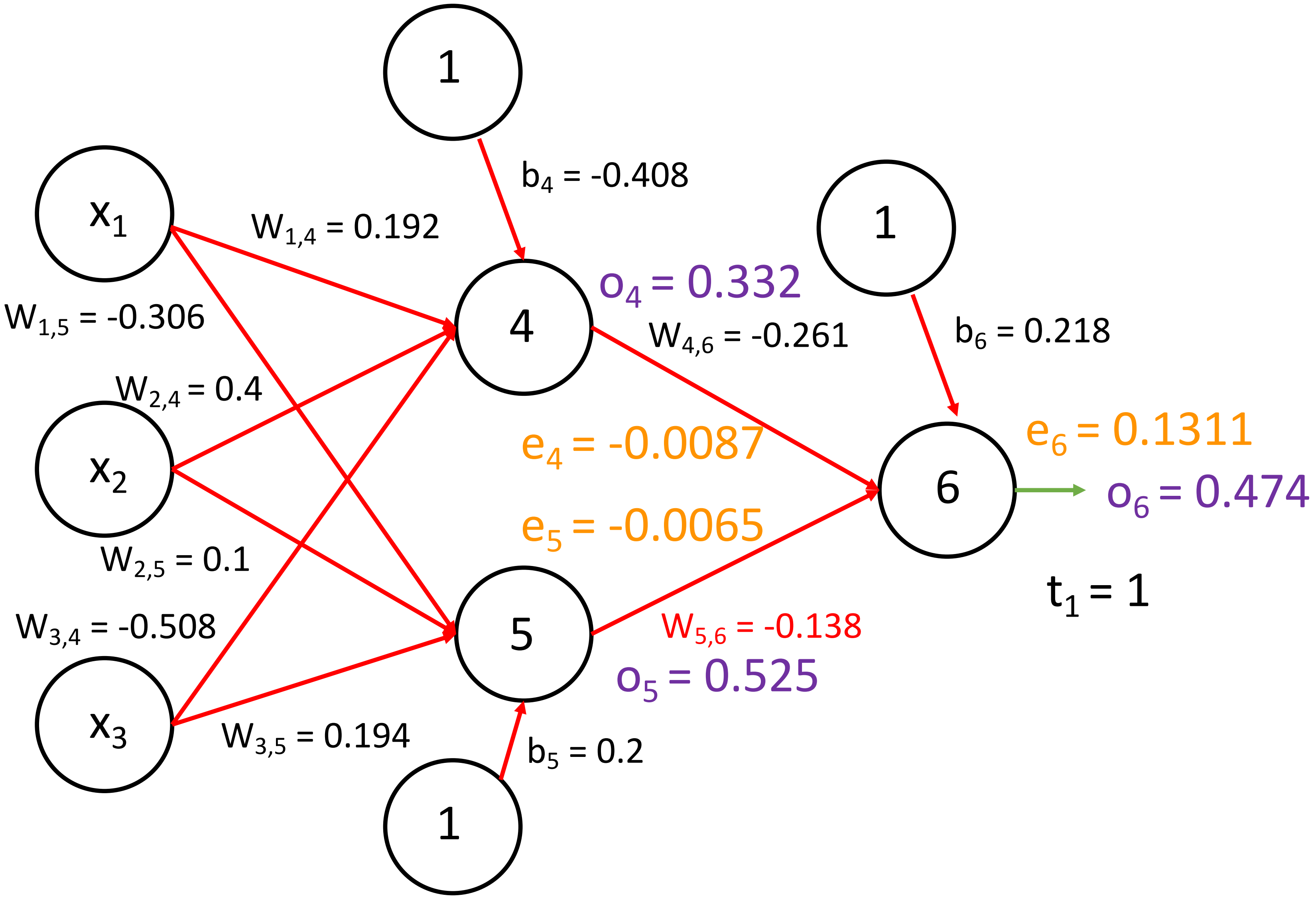
New bias (learning rate = 0.9):

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

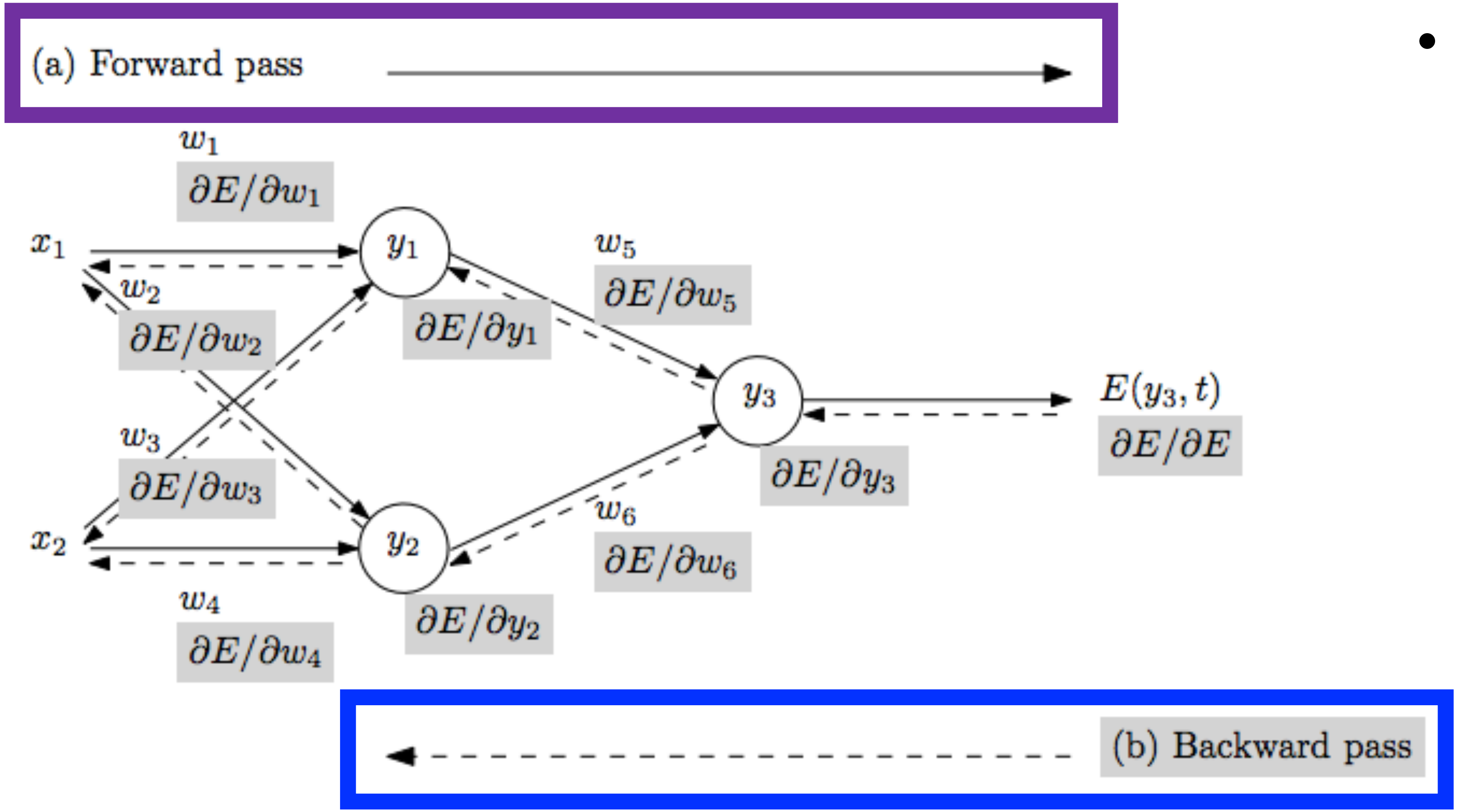
$b_4 = -0.408$

$$b_k = b_k + \eta Err_k$$

Repeat Steps 1-3 With New Examples



Repeat Steps 1-3 With New Examples



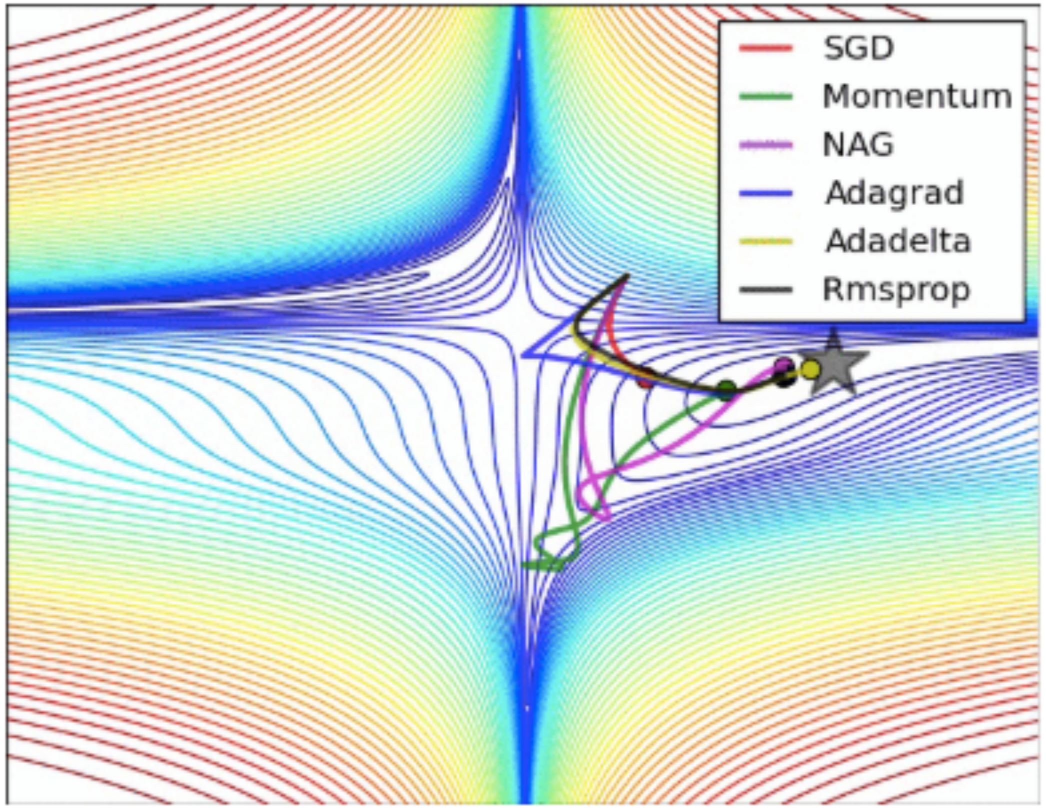
- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

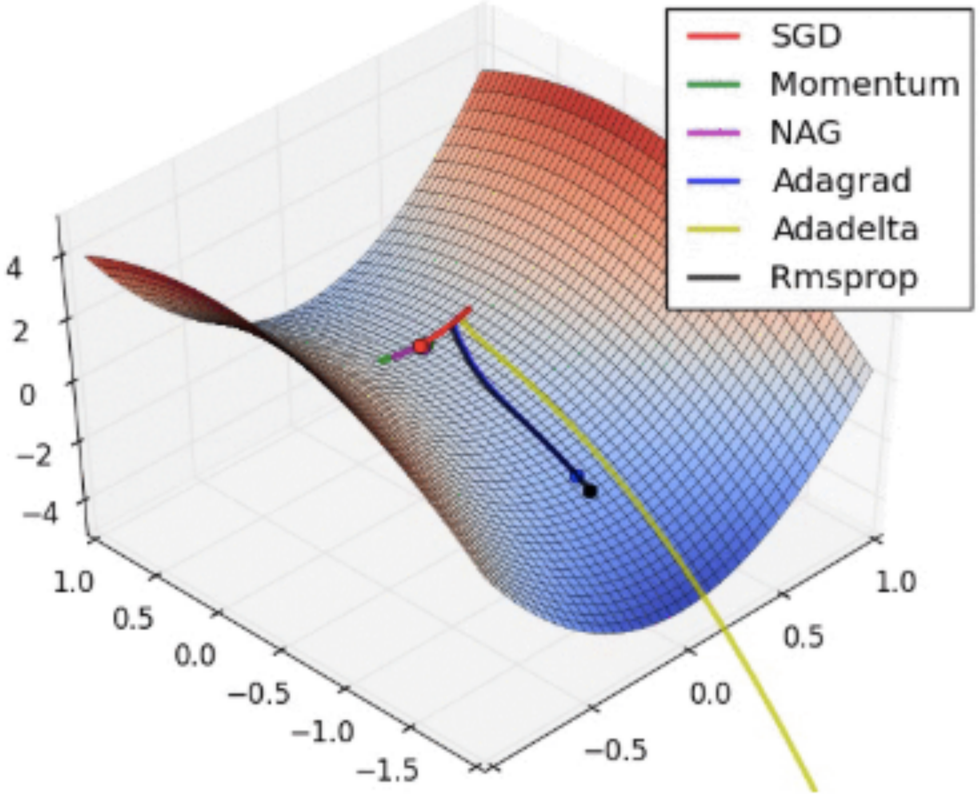
Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

Training Challenge: Train Faster!!!

- Can take hours, days, weeks, months, or more to train millions of parameters…

Weight Updates: How to Speed Up Training?





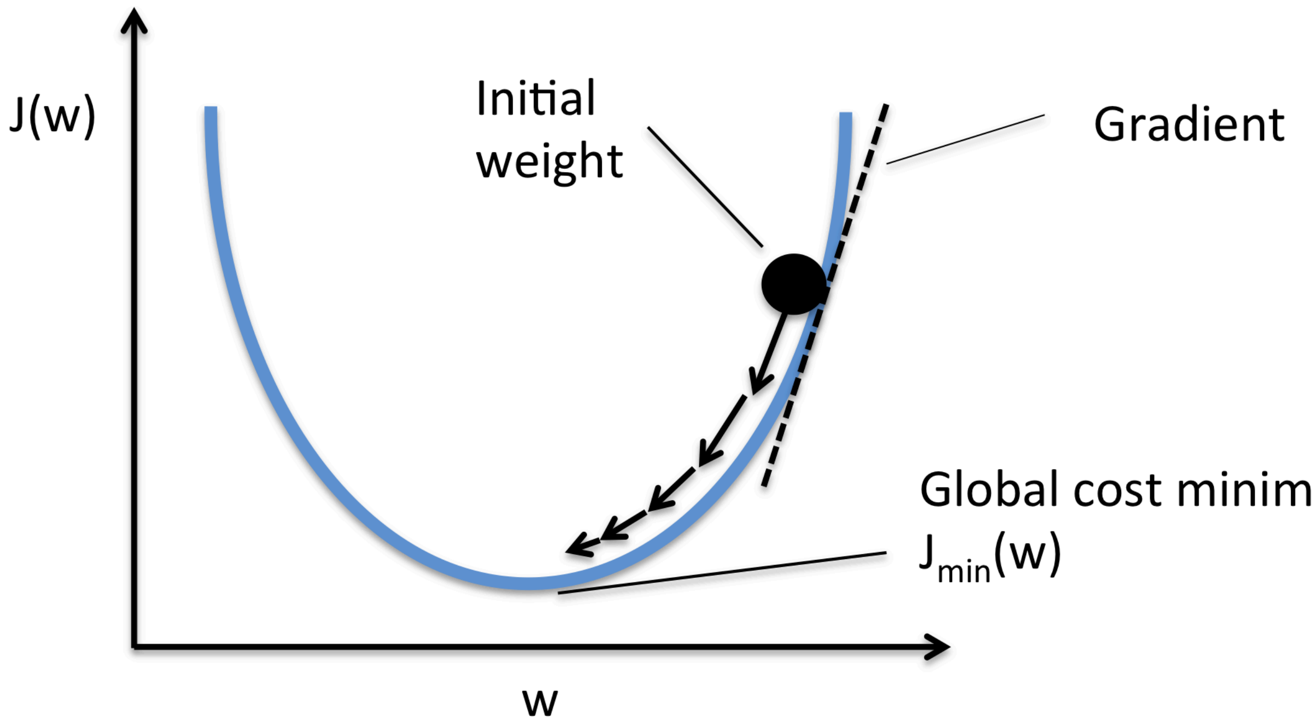
- Demo at http://cs231n.github.io/neural-networks-3/#update

Train Faster: How to Update Using Gradient?

- Vanilla Approach:

```
x += - learning_rate * dx
```

Recall: steps get smaller as gradient gets smaller



http://cs231n.github.io/neural-networks-3/#update

Figure from: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

Train Faster: How to Update Using Gradient?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

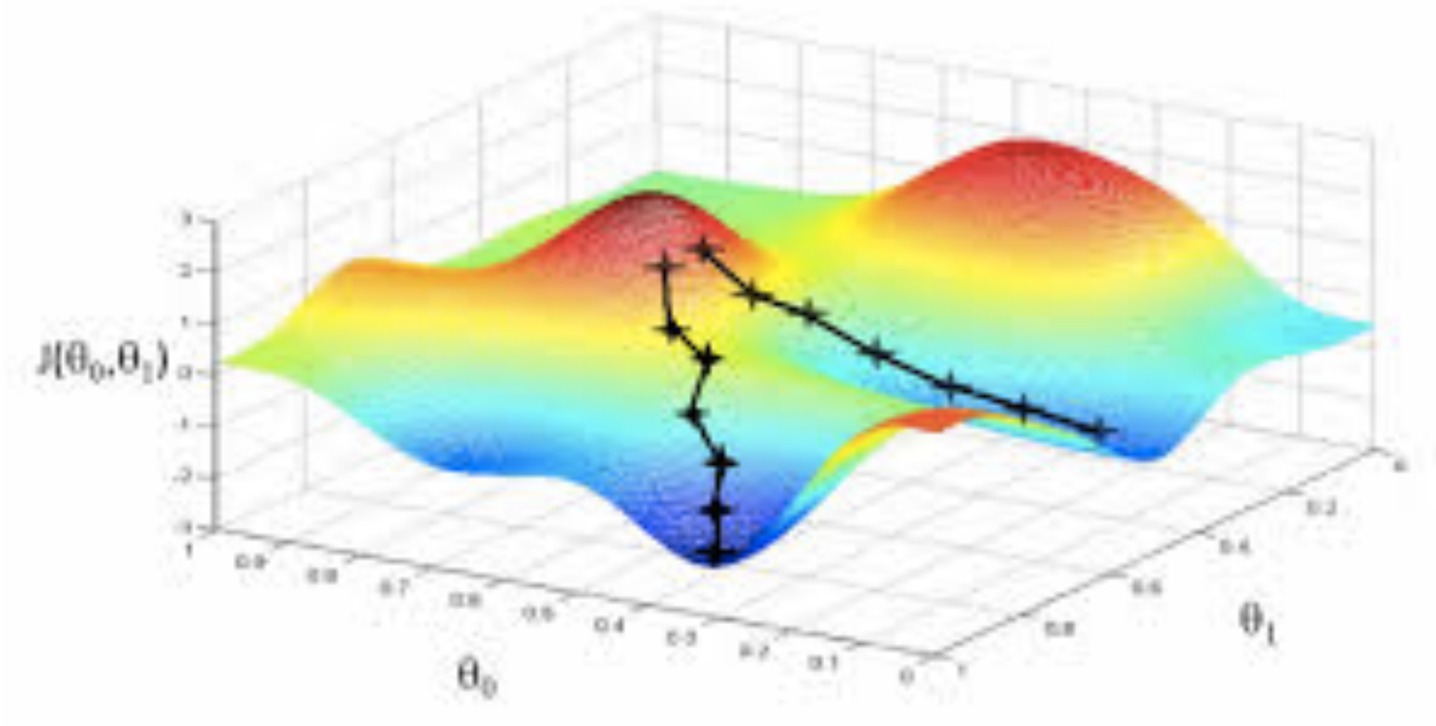


Figure from: https://medium.com/ai-society/hello-gradient-descent-ef74434bdfa5

Train Faster: How to Update Using Gradient?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

Gradient is used for acceleration rather than speed

Values range from 0 to 1 (larger values mean greater friction)

```
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

- What are advantages and disadvantages?
  - Can roll past local minima ☺
  - It may roll past optimum and oscillate around it ☹
  - Another hyperparameter to tune ☹

http://cs231n.github.io/neural-networks-3/#update

Train Faster: How to Update Using Gradient?

- Adapt learning rate per-parameter
- e.g., AdaGrad: decays faster when dimensions are steeper

```
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

- e.g., RMSprop:

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

- e.g., Adam:

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

http://cs231n.github.io/neural-networks-3/#update

Train Faster: How to Update Learning Rate?

- **Step decay**:
  - Reduce the learning rate by some factor every few epochs.

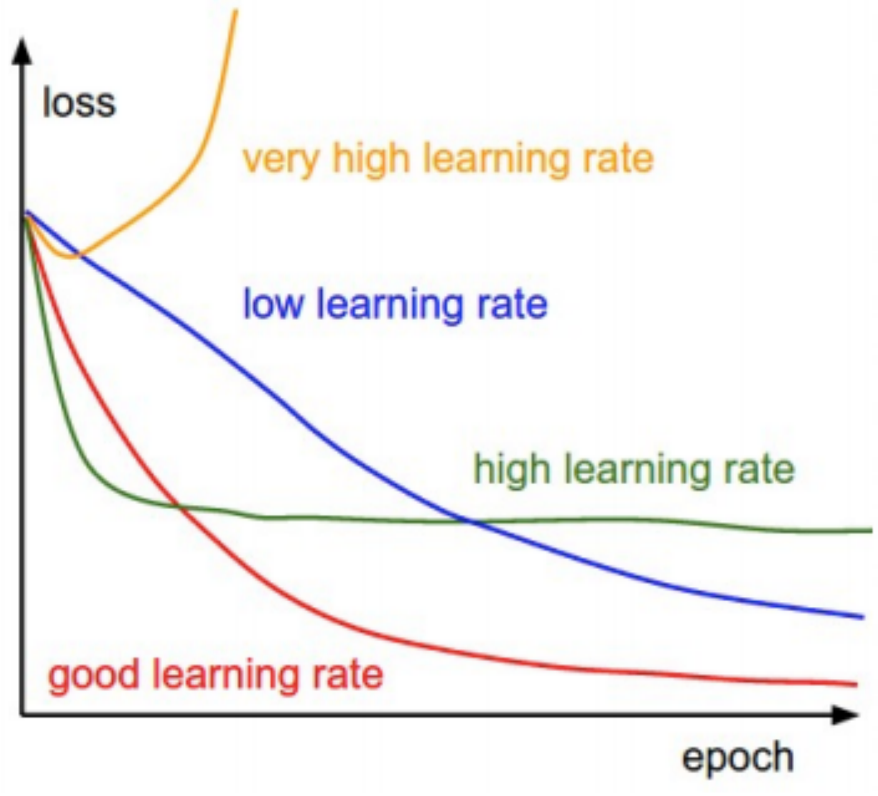
- **Exponential decay**

- **1/t decay**

http://cs231n.github.io/neural-networks-3/#update

Monitor Loss During Training

- What should happen to the loss function value during training?



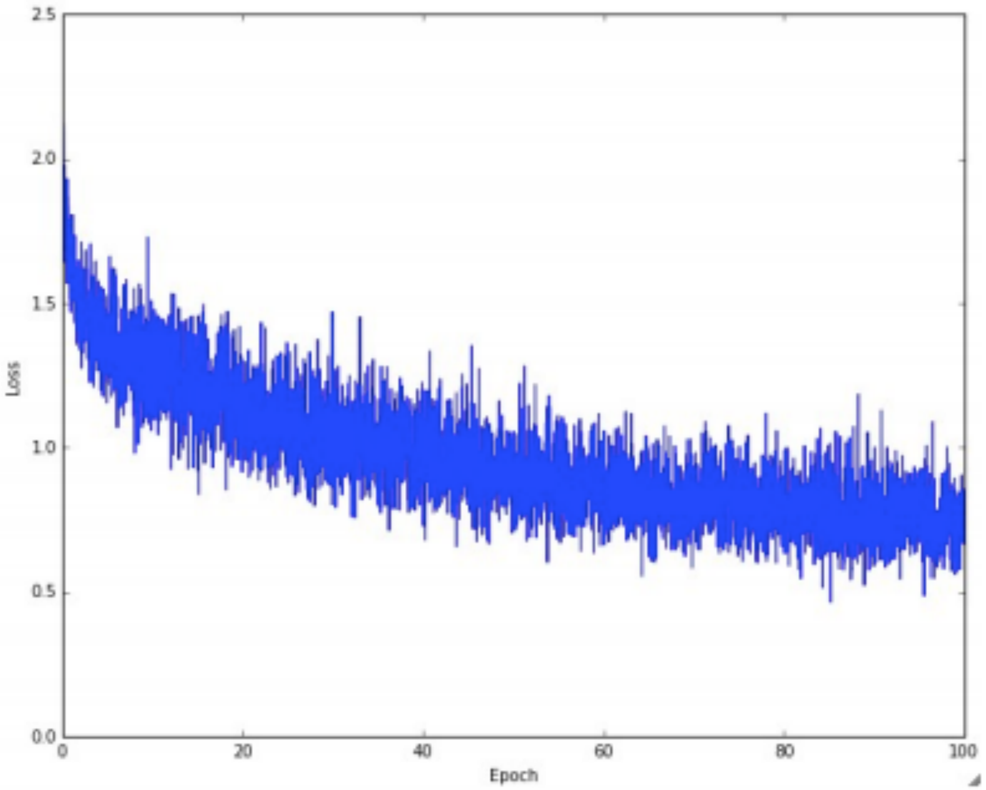


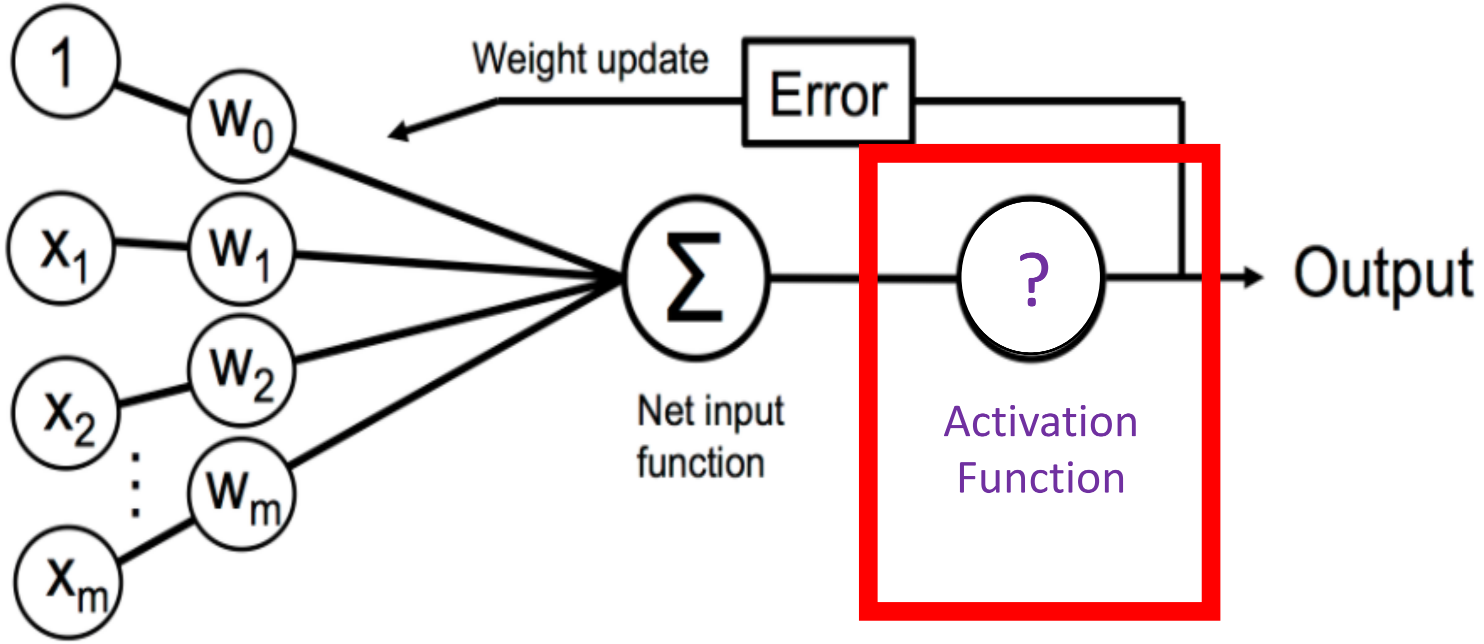
Image credit: http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

Today's Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions
- Lab

Recall: Non-Linear Activation Functions

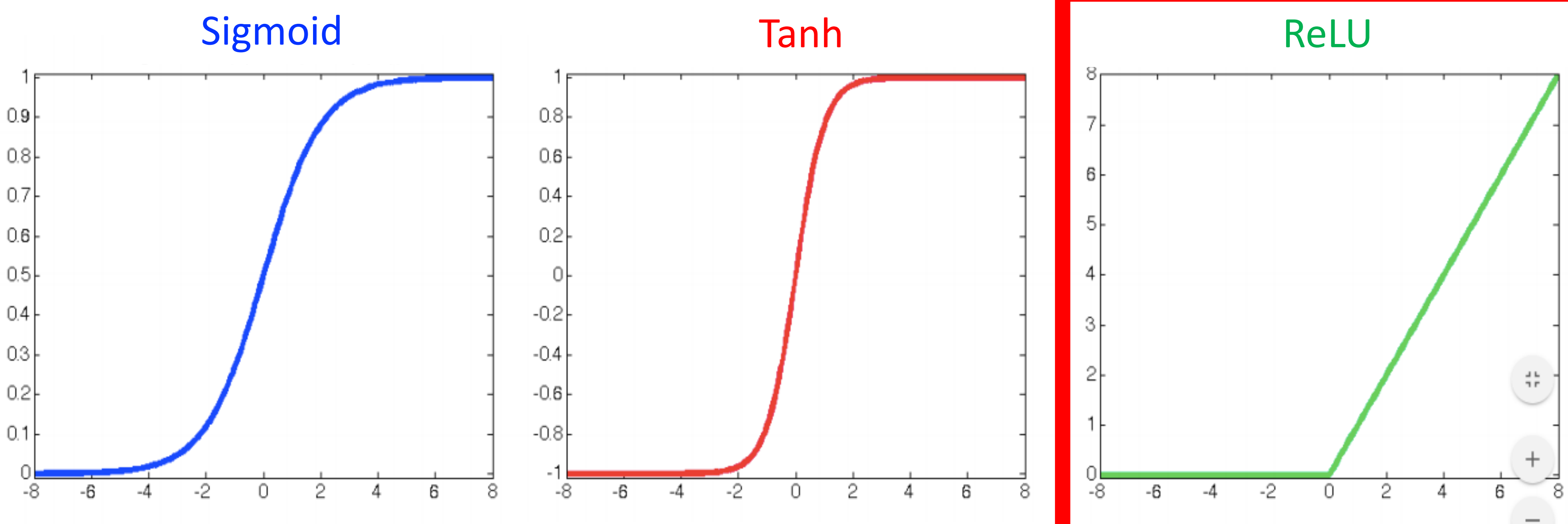
- Each unit applies a non-linear "activation" function to the weighted input to mimic a neuron firing



Python Machine Learning; Raschka & Mirjalili

Non-Linear Activation Functions

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



$$\sigma(z) = \frac{1}{1+\exp(-z)} \qquad \tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} \qquad \mathrm{ReLU}(z) = \max(0, z)$$

http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

Non-Linear Activation Functions

What is a limitation of ReLU?

- Neurons can die (they only output 0 when the weighted sum of its input is negative)

e.g., ReLU

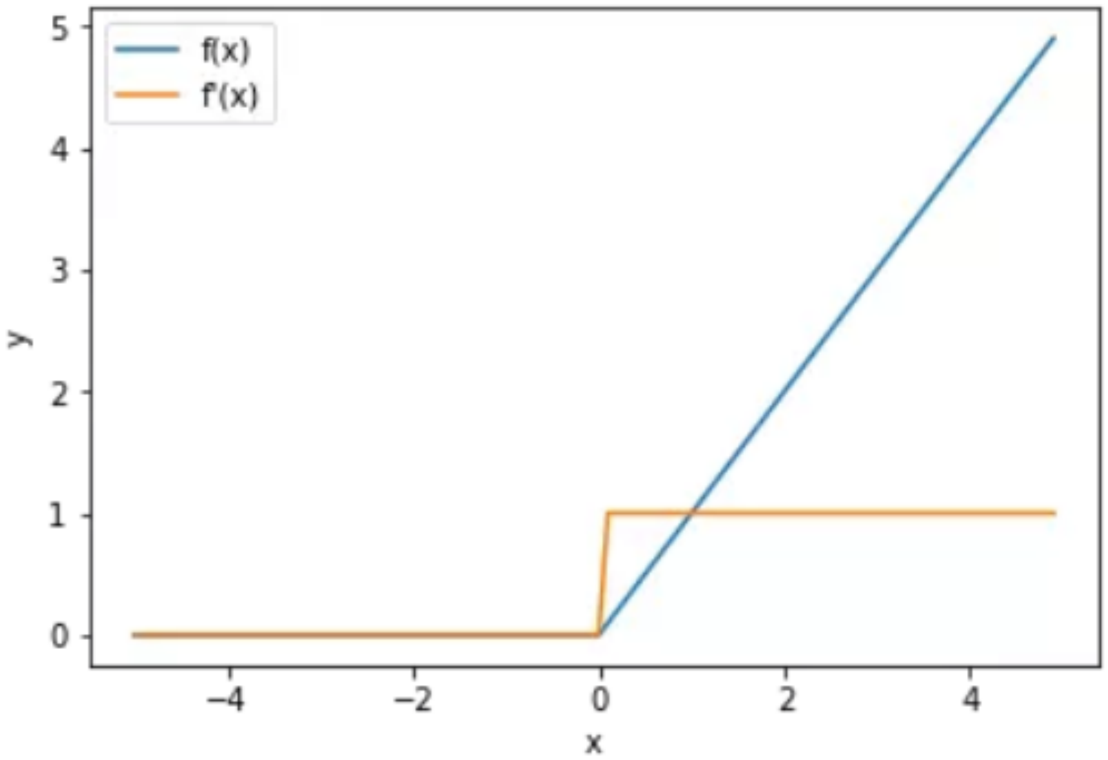
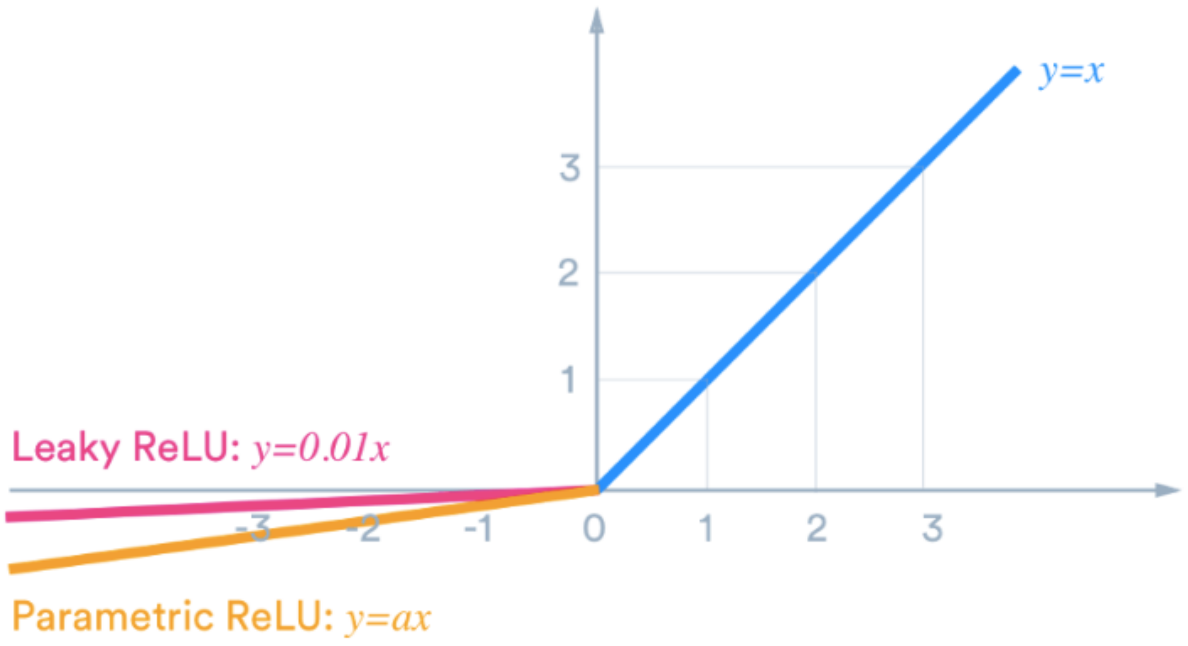


Figure Credit: https://adventuresinmachinelearning.com/vanishing-gradient-problem-tensorflow/

Non-Linear Activation Functions

Use activation functions that don't have small derivative values

e.g., Variants of ReLU



e.g., Exponential Linear

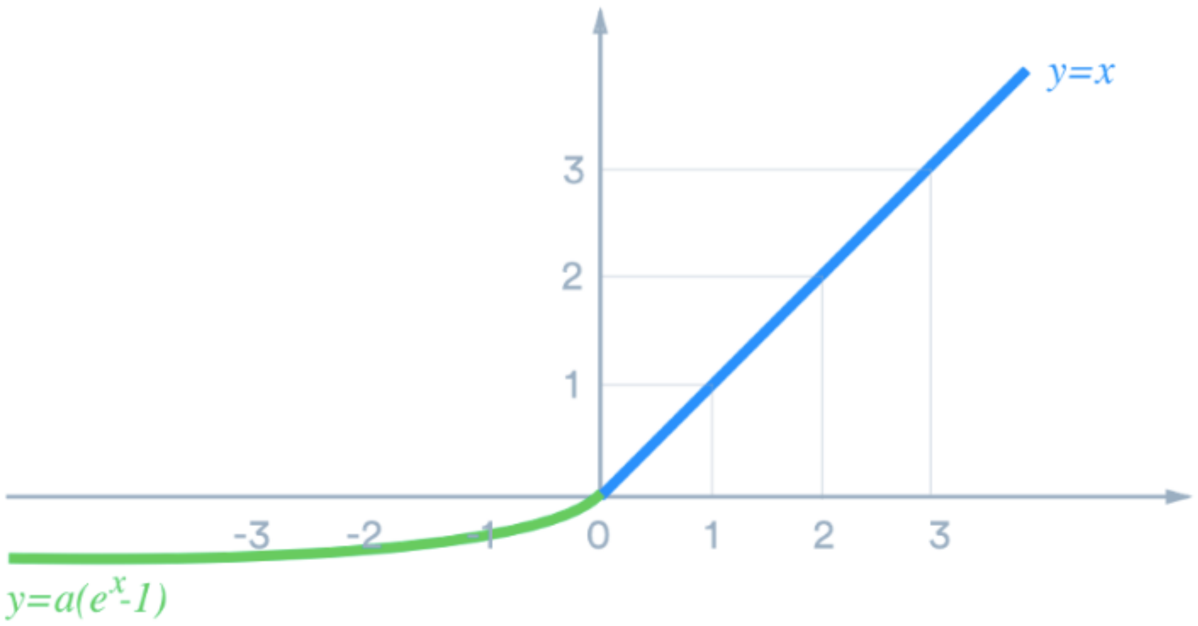
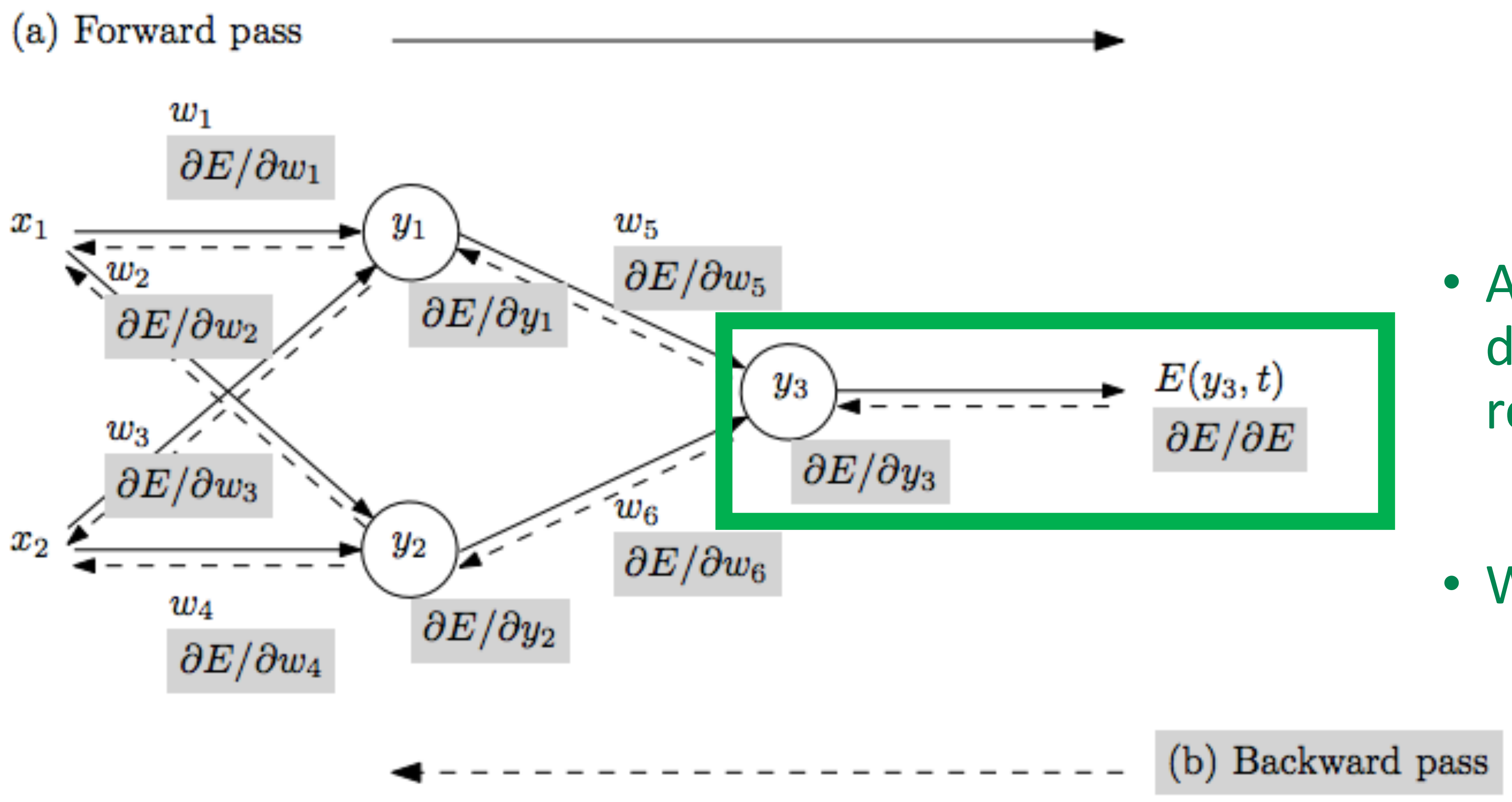


Figure Credit: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7

Clevert et al. Fast and Accurate deep network learning by exponential linear units. 2015

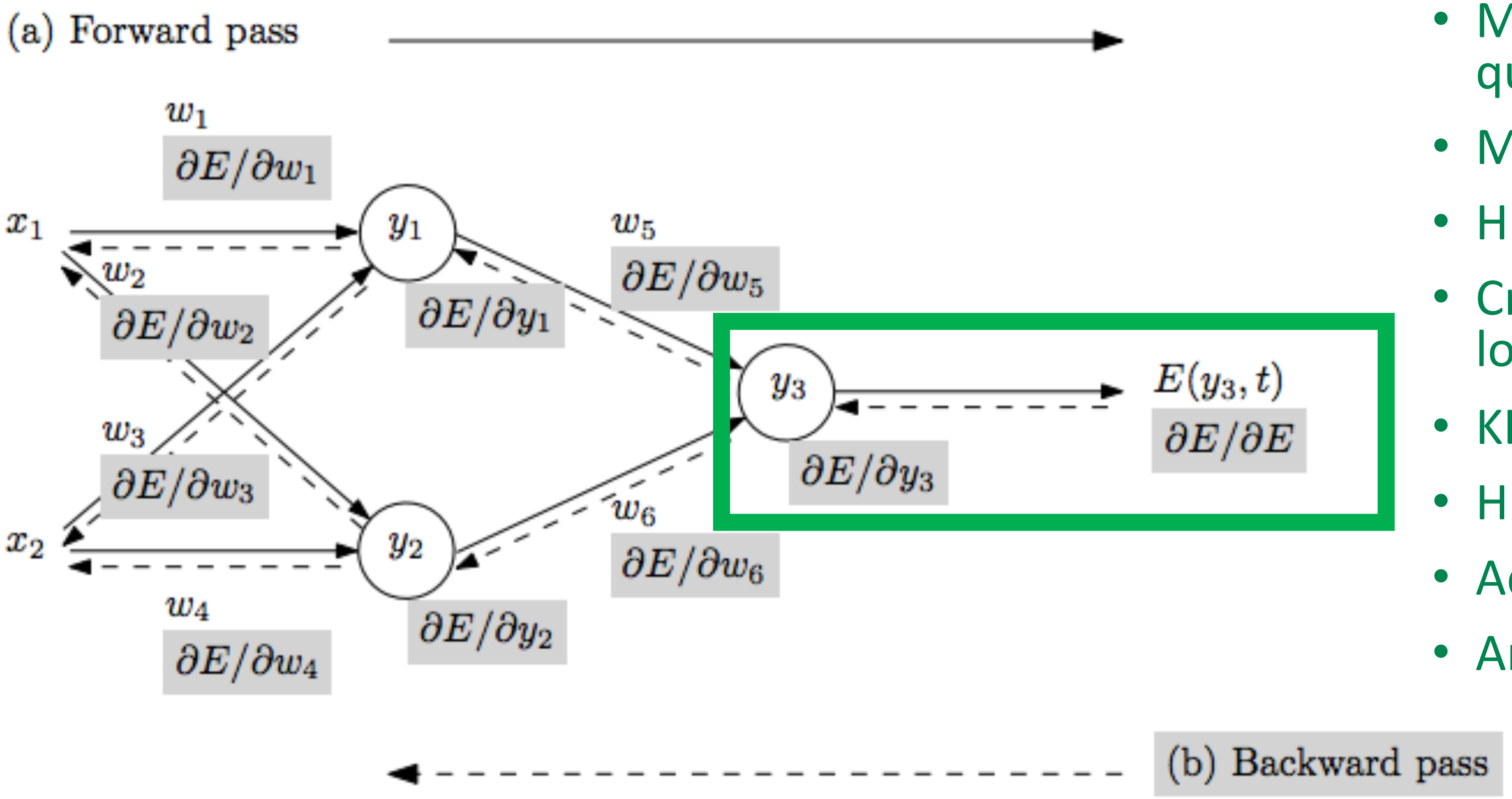
Recall: Loss Functions



- A loss function quantifies the dissatisfaction with a model's results on the training data.
- What loss function to use?

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

Loss Functions



- Mean squared error/L2 loss/ quadratic loss
- Mean absolute error/L1 loss
- Huber loss
- Cross entropy loss/logarithmic loss
- KL divergence loss
- Hinge loss
- Adversarial loss
- And many more options…

https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

Today’s Topics

- History of Neural Networks
- Neural Network Architecture – Hidden Layers and Solving XOR Problem
- Neural Network Architecture – Output Units
- Training a Neural Network – Optimization
- Training a Neural Network – Activation Functions & Loss Functions