CSE-403: Machine Learning

Chapter 6: Neural Networks

Md Atikuzzaman

Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

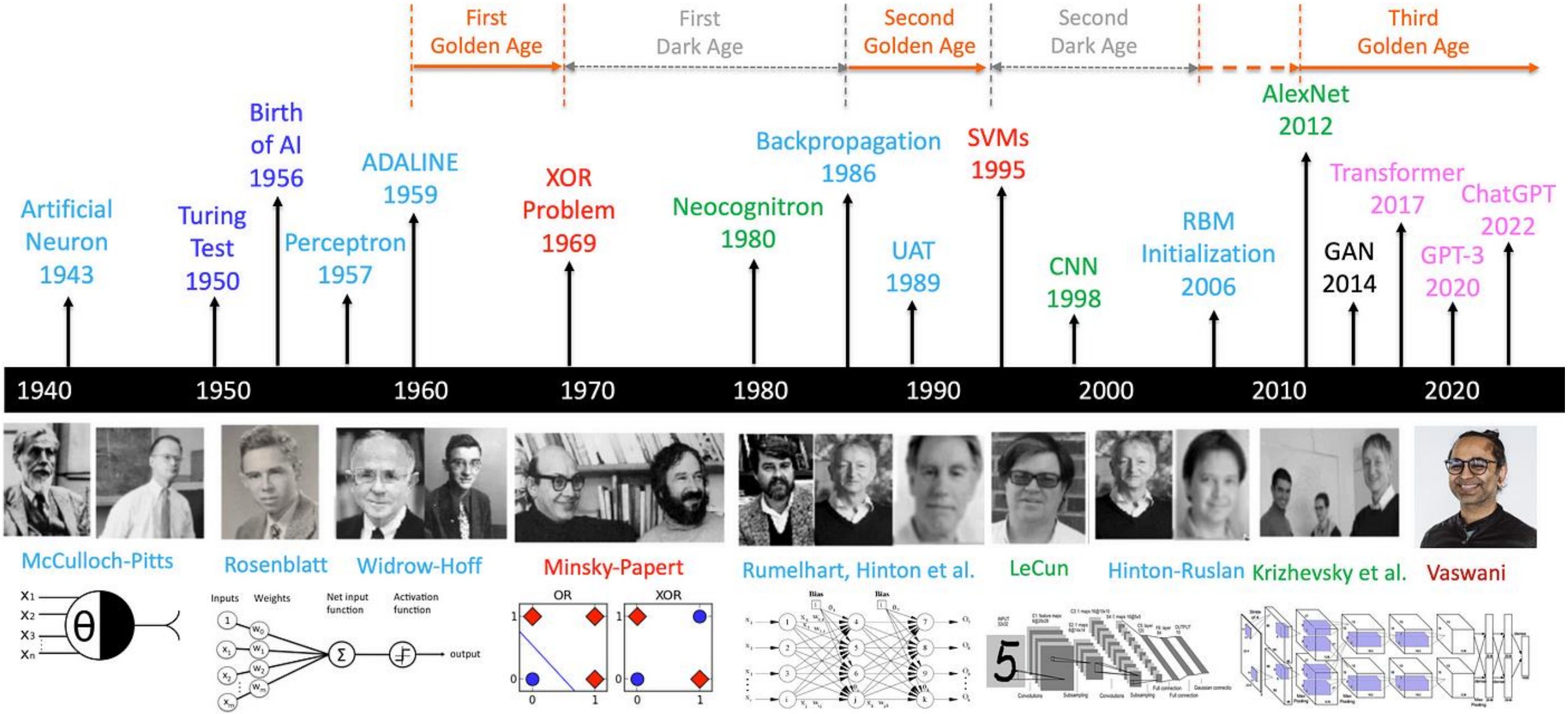
Outline

1) The Neuron Model and Activation Functions
2) Architecture of Neural Networks
3) Forward Propagation
4) Loss Functions for Neural Networks
5) Backpropagation and Gradient Descent
6) Initialization and Normalization Techniques
7) Vanishing and Exploding Gradients
8) Regularization in Neural Networks (Dropout, Weight Decay)
9) Optimization Algorithms (SGD, Adam, RMSprop)
10) Batch, Mini-batch, and Stochastic Training
11) Hyperparameter Tuning
12) Performance Evaluation and Model Interpretation

Why Neural Networks?

- ❑ Many real problems are too complex for simple linear models.
- ❑ Neural networks can learn rich nonlinear relationships from data.
- ❑ They are widely used in:
  - image classification,
  - speech recognition,
  - natural language processing,
  - forecasting and anomaly detection

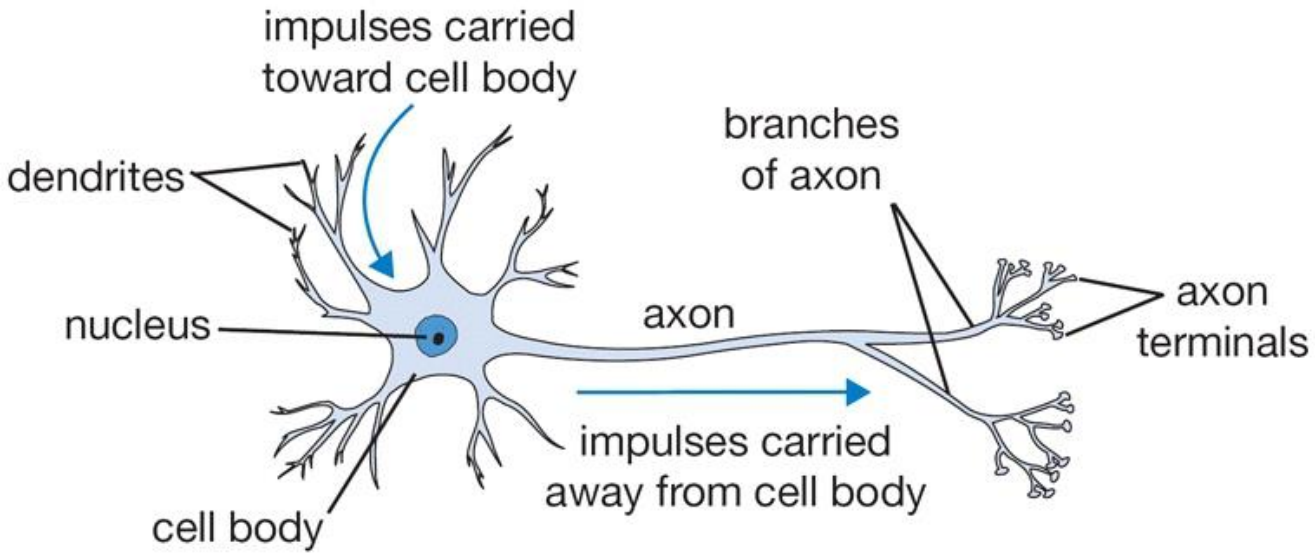
History of artificial intelligence

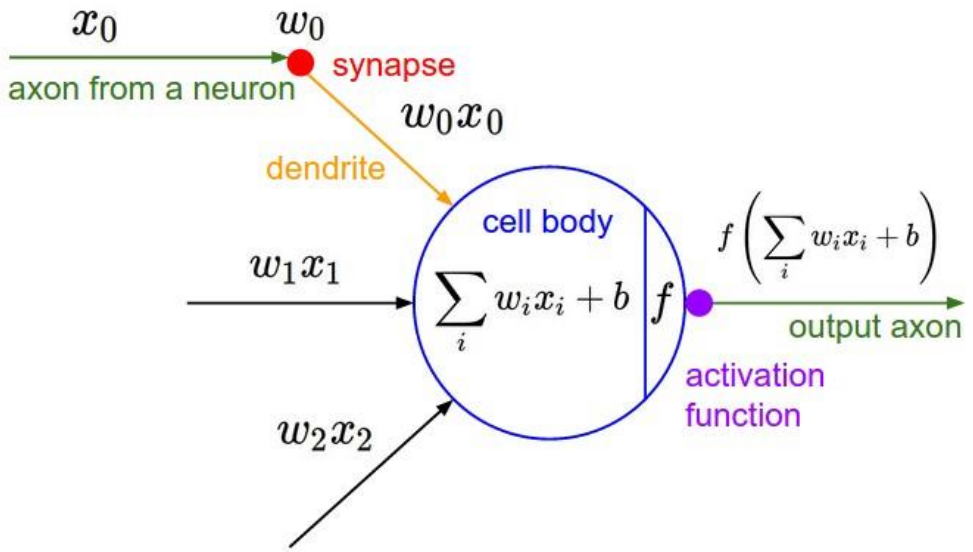


Neuron

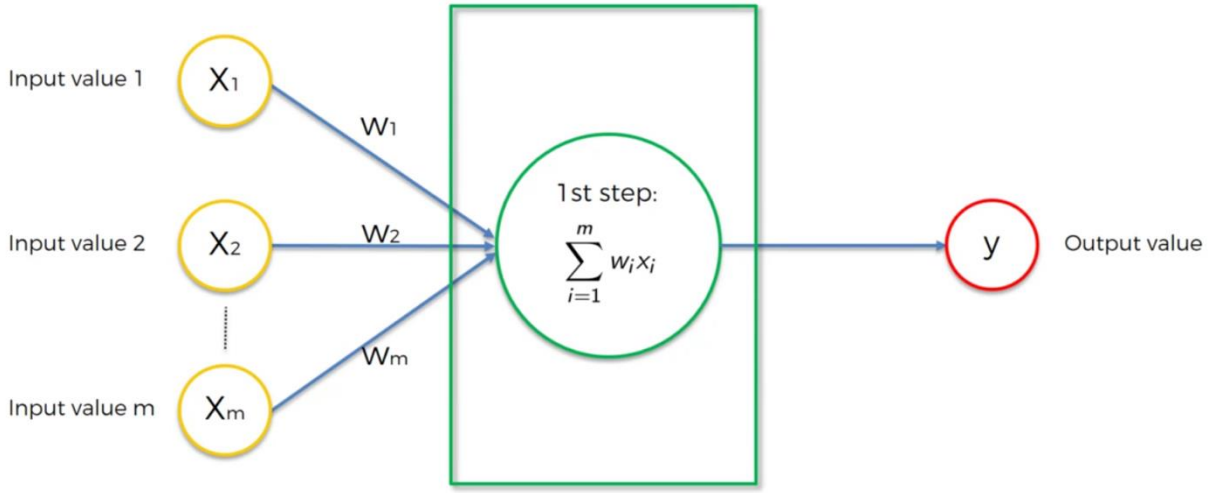
In machine learning, specifically within artificial neural networks, a **neuron** (often called a **node** or a **perceptron**) is the fundamental computational building block.

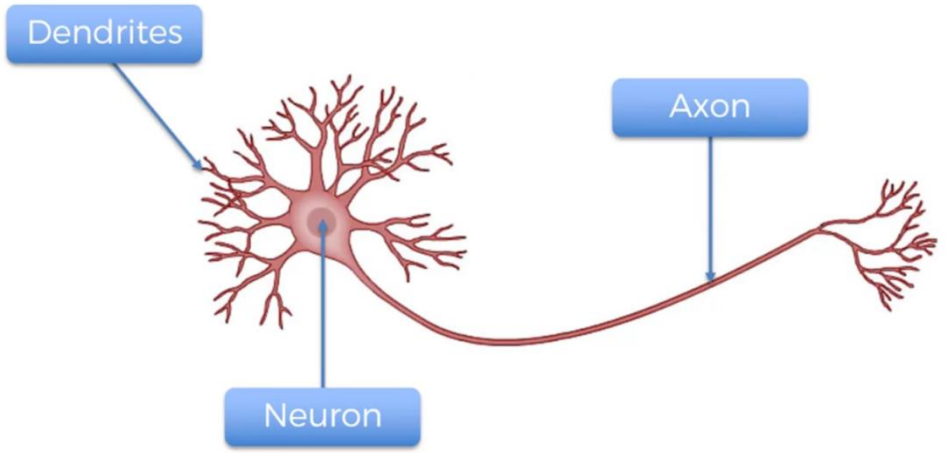
While loosely inspired by biological neurons in the human brain, an artificial neuron is ultimately just a mathematical function. Its primary job is to take in multiple pieces of information, weight their importance, and output a single decision or signal.

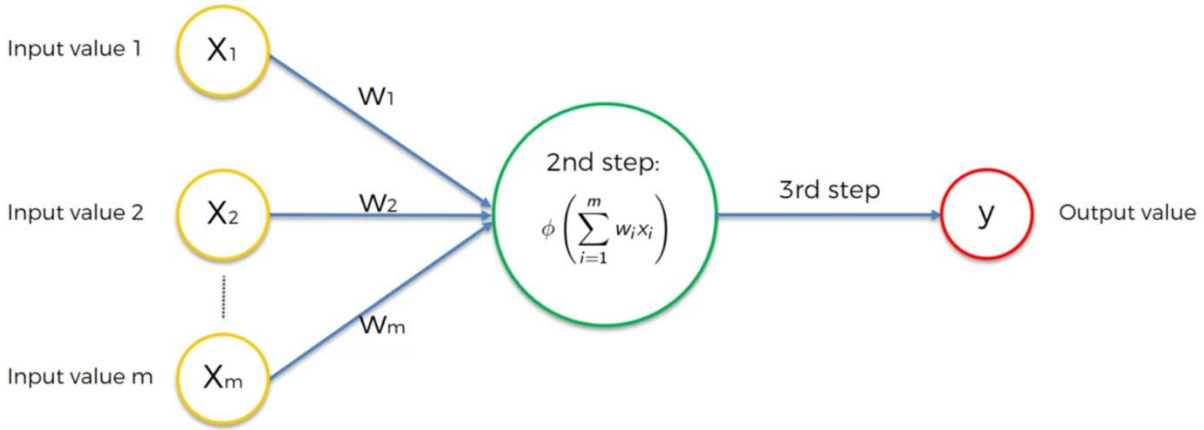


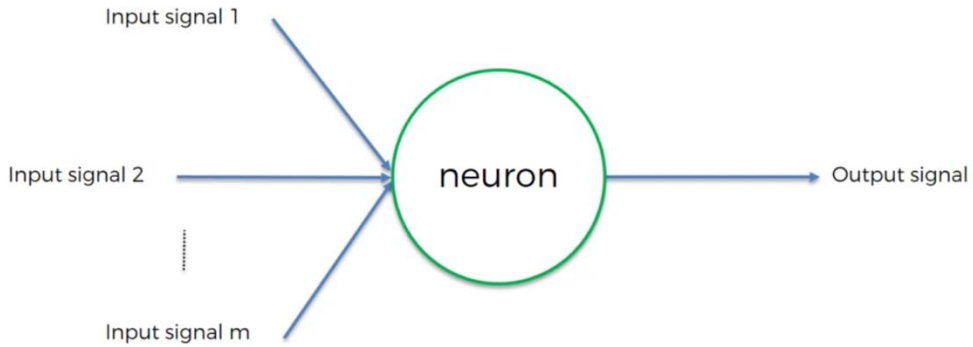


Neuron


Input value 1
X1
W1
Input value 2
X2
W2
Wm
Input value m
Xm
1st step:
$\sum_{i=1}^{m} w_i x_i$
y
Output value

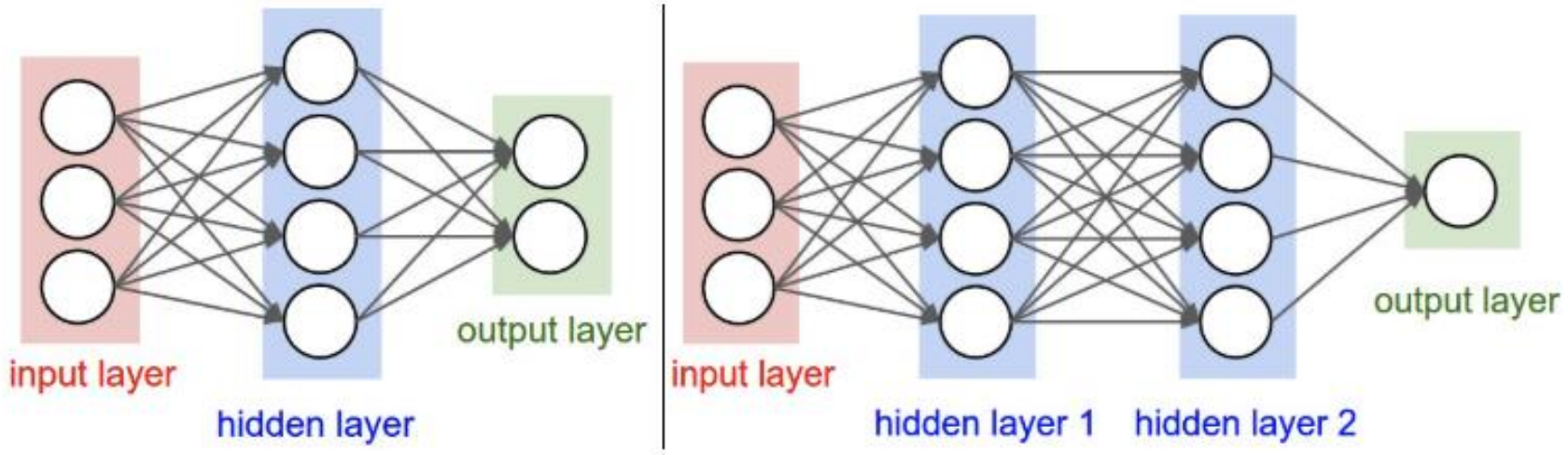

Dendrites
Axon
Neuron


Input value 1
X1
W1
Input value 2
X2
W2
Wm
Input value m
Xm
2nd step:
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
3rd step
y
Output value


Input signal 1
Input signal 2
Input signal m
neuron
Output signal

Neural network

A neural network is a machine learning model that stacks simple "**neurons**" in layers and learns pattern-recognizing weights and biases from data to map inputs to outputs.

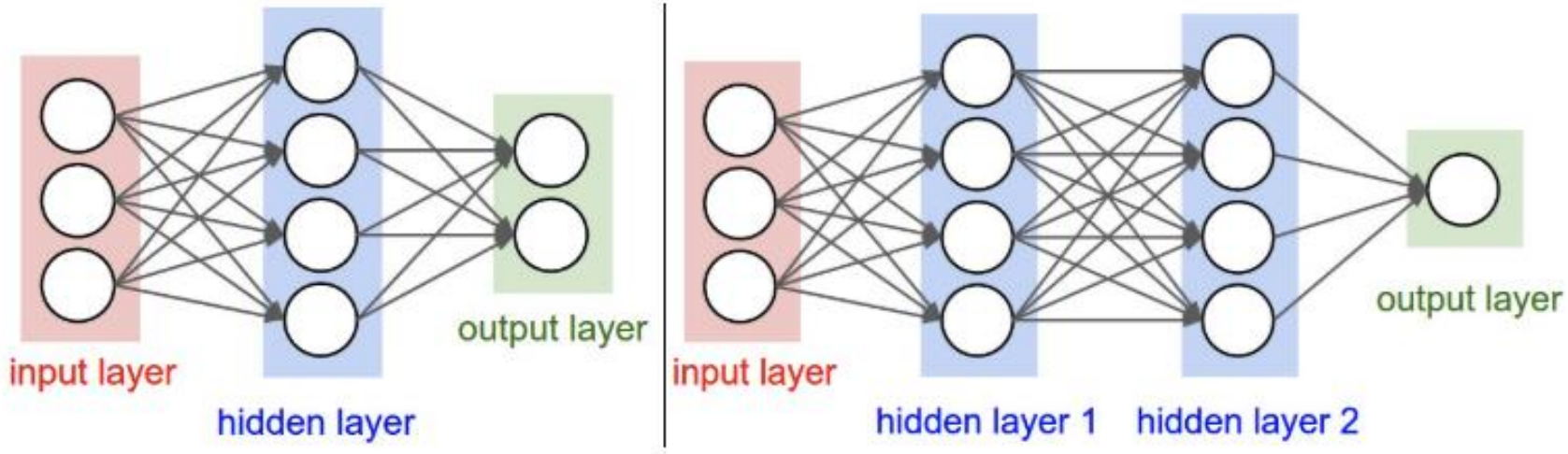


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Naming conventions. Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). You may also hear these networks interchangeably referred to as *"Artificial Neural Networks"* (ANN) or *"Multi-Layer Perceptrons"* (MLP).

Neural network

Output layer. Unlike all layers in a Neural Network, the output layer neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function). This is because the last output layer is usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression).

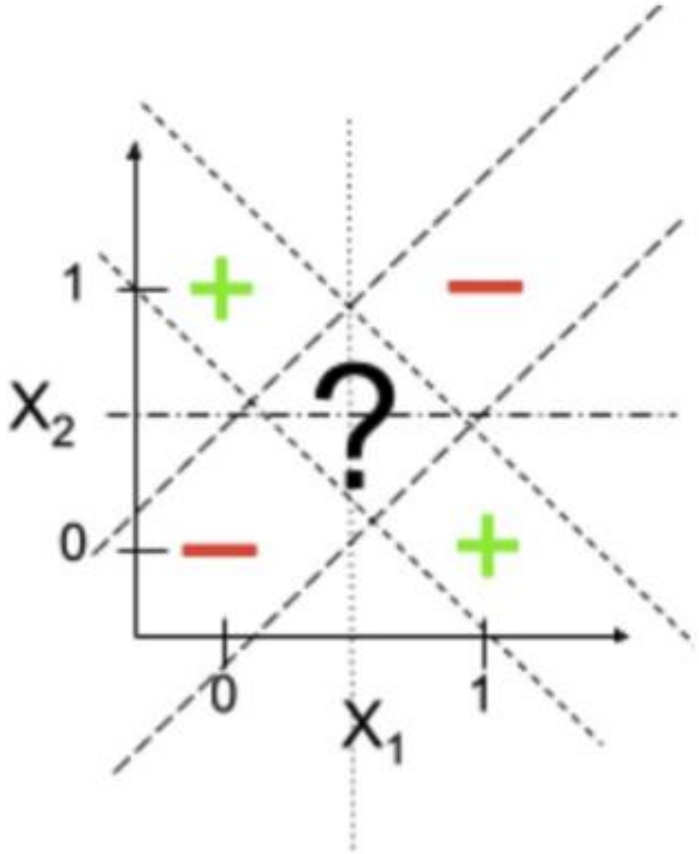


Sizing neural networks. The two metrics that people commonly use to measure the size of neural networks are the number of neurons, or more commonly the number of parameters. Working with the two example networks in the above picture:

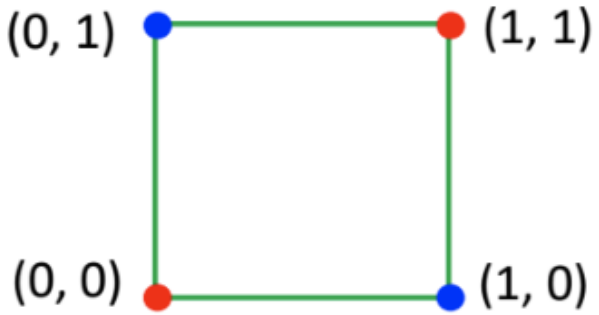
- The first network (left) has 4 + 2 = 6 neurons (not counting the inputs), [3 x 4] + [4 x 2] = 20 weights and 4 + 2 = 6 biases, for a total of 26 learnable parameters.
- The second network (right) has 4 + 4 + 1 = 9 neurons, [3 x 4] + [4 x 4] + [4 x 1] = 12 + 16 + 4 = 32 weights and 4 + 4 + 1 = 9 biases, for a total of 41 learnable parameters.

Perceptron Limitation

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



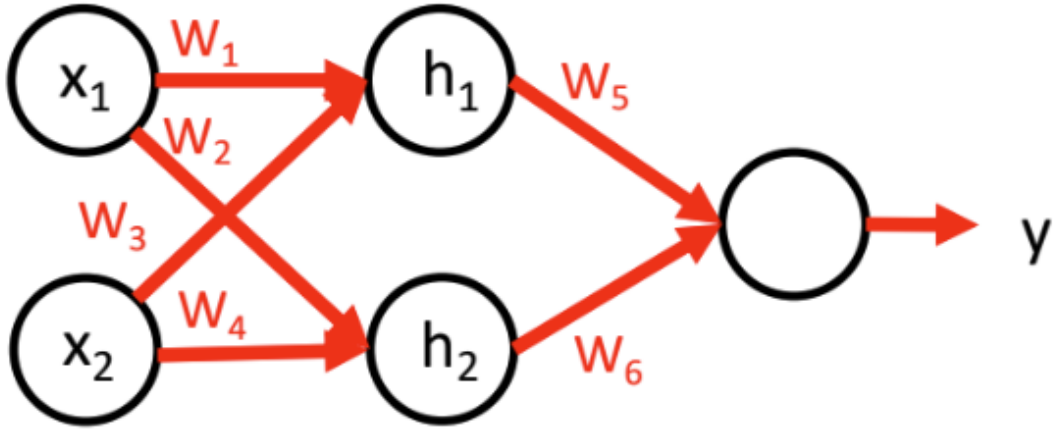
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



Non-Linear Activation Functions

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

Key Observation: feedforward networks are just functions chained together



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!

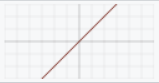
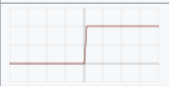
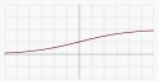
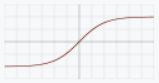
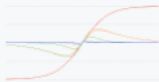
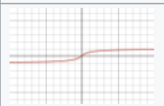
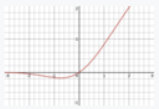
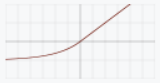
What is function for y?

$y = h_1w_5 + h_2w_6 + b_3$

Constant x linear function = linear function

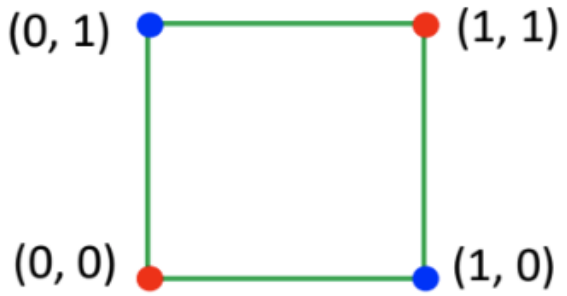
Non-Linear Activation Functions

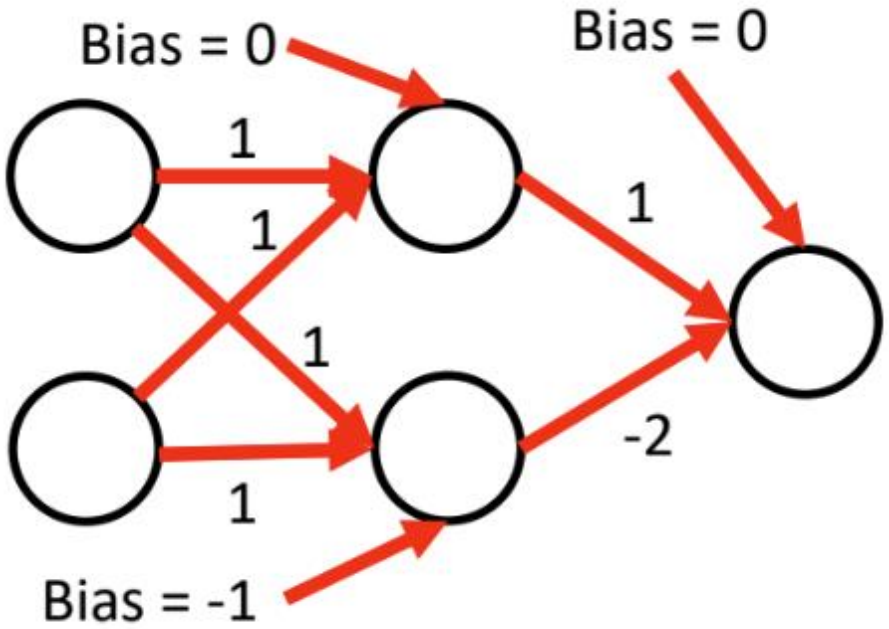
In machine learning, an **activation function** is the mathematical "gatekeeper" of an artificial neuron. Once a neuron calculates the weighted sum of its inputs and adds the bias ($z = \sum w_i x_i + b$) the activation function steps in to decide what the neuron's final output should be.

Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	0	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$	$(0, 1)$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$
Soboleva modified hyperbolic tangent (smht)		$\operatorname{smht}(x) \doteq \frac{e^{ax} - e^{-bx}}{e^{cx} + e^{-dx}}$		$(-1, 1)$
Softsign		$\frac{x}{1+\lvert x\rvert}$	$\frac{1}{(1+\lvert x\rvert)^2}$	$(-1, 1)$
Rectified linear unit (ReLU)[14]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0, x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0, \infty)$
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$ where erf is the gaussian error function.	$\Phi(x) + \frac{1}{2}x\phi(x)$ where $\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$ is the probability density function of standard gaussian distribution.	$(-0.17\ldots, \infty)$
Softplus[15]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$
Exponential linear unit (ELU)[16]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$(-\alpha, \infty)$

Non-Linear Example

Approach: Without any Activation Functions (we can't solve the XOR problem)

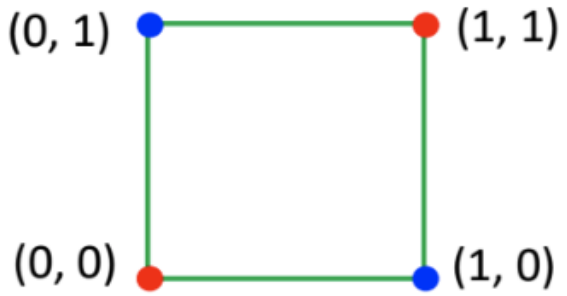


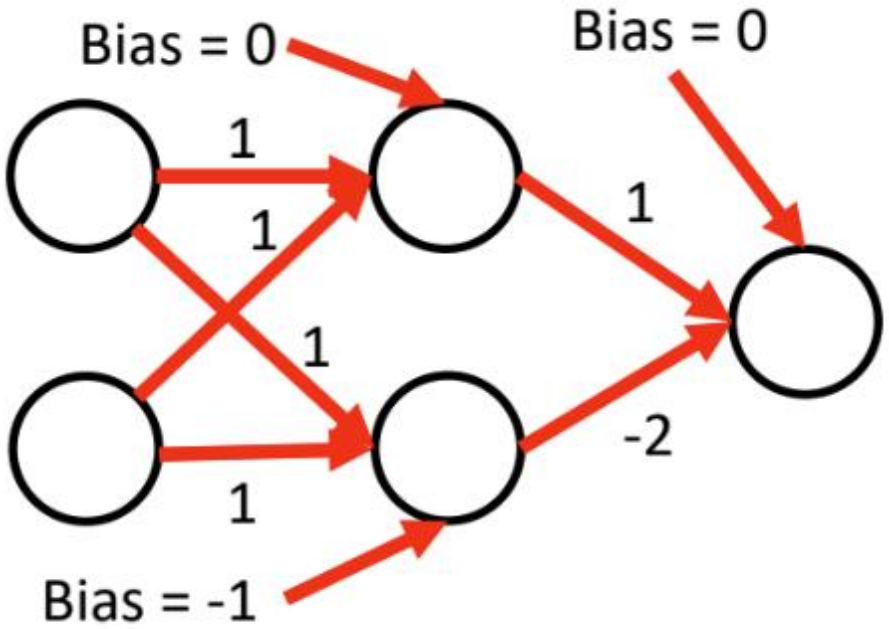


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Non-Linear Example

Approach: ReLU activation function [ReLU (z) = max(0, z)] with these parameters



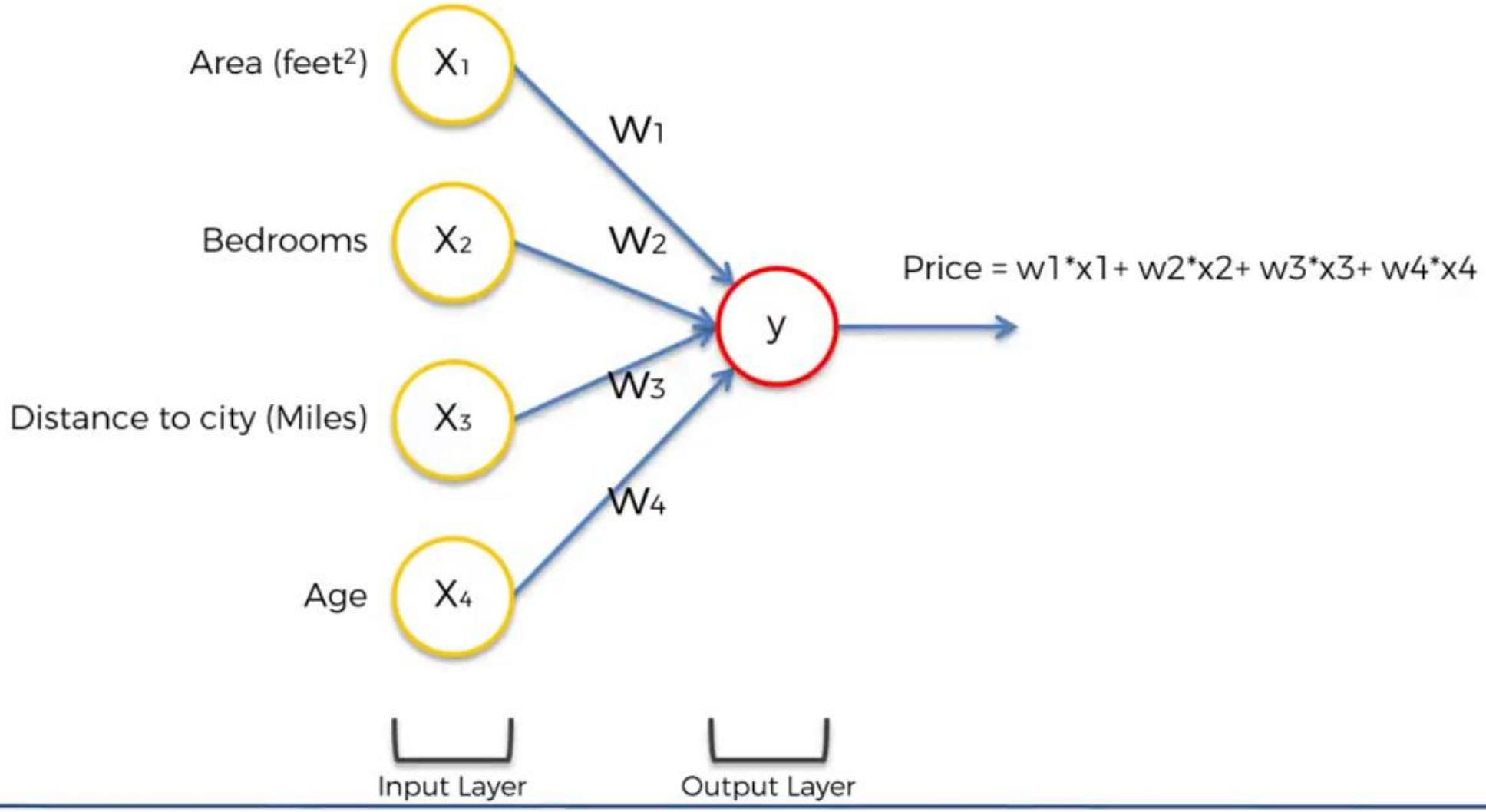


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

How Do Neural Networks Work?

How Do Neural Networks Work?

How Do Neural Networks Work?

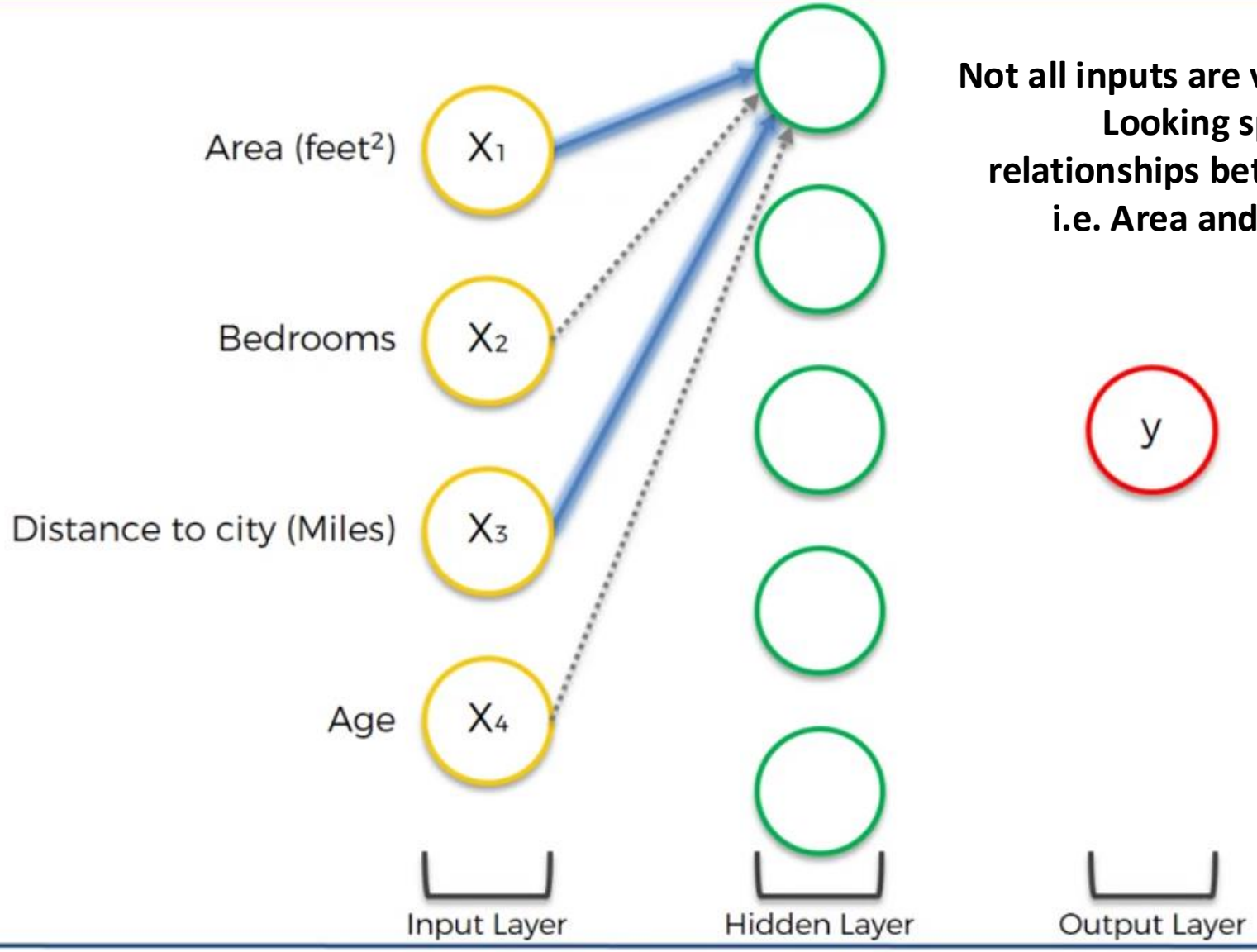


How Do Neural Networks Work?

Power comes from hidden layer

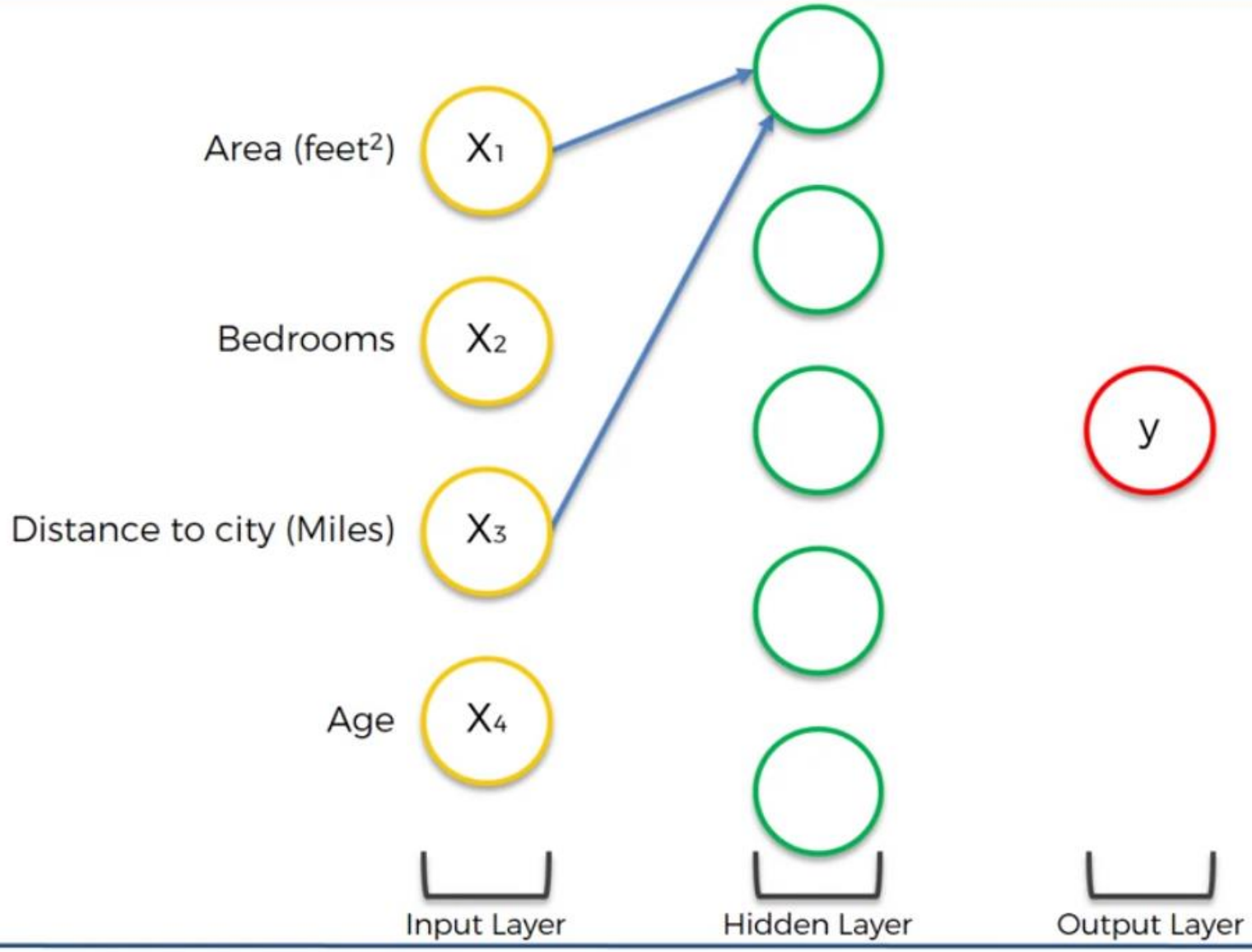
How Do Neural Networks Work?

How Do Neural Networks Work?

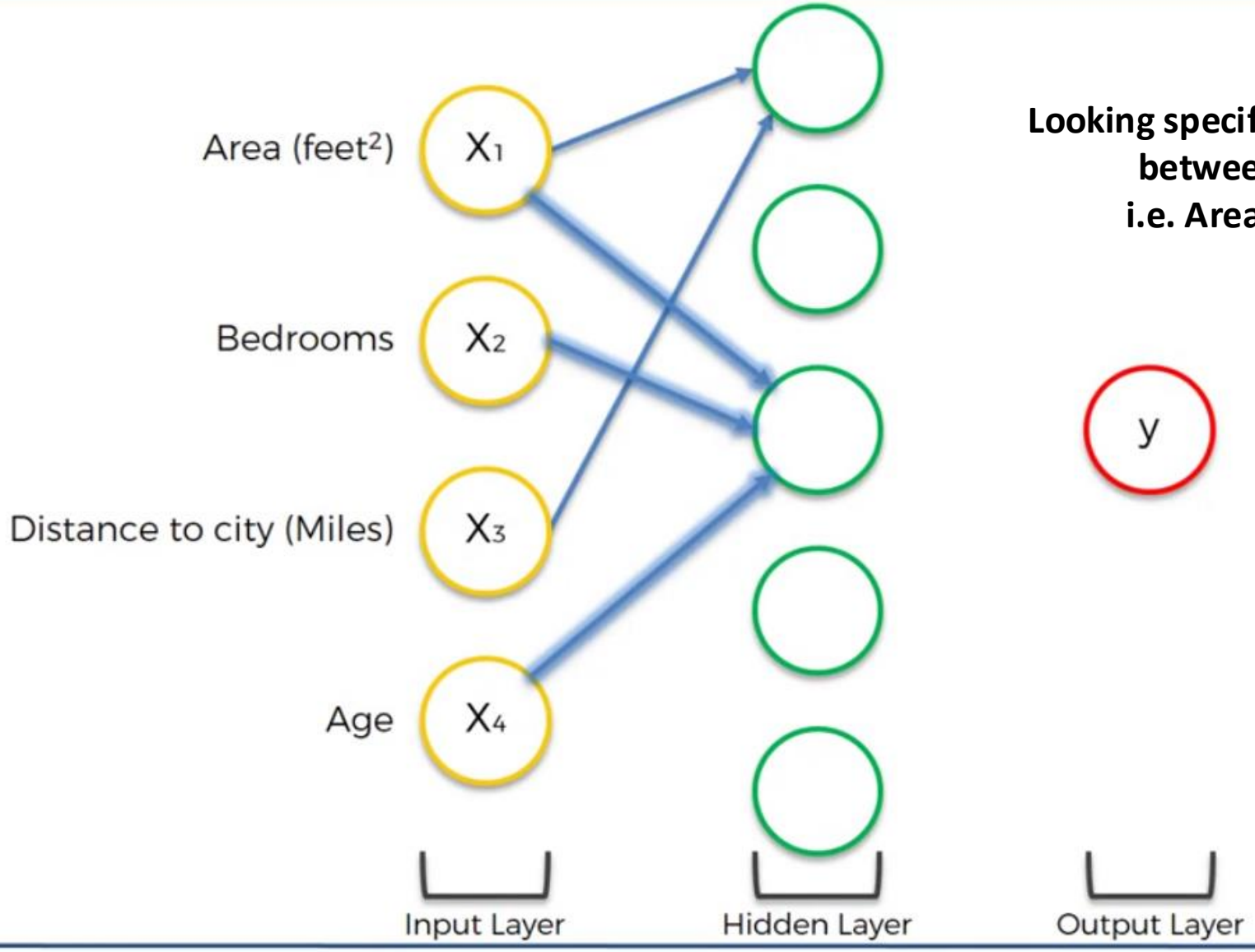


Not all inputs are valid to every neuron.
Looking specifically for
relationships between certain inputs
i.e. Area and Distance to City

How Do Neural Networks Work?



How Do Neural Networks Work?

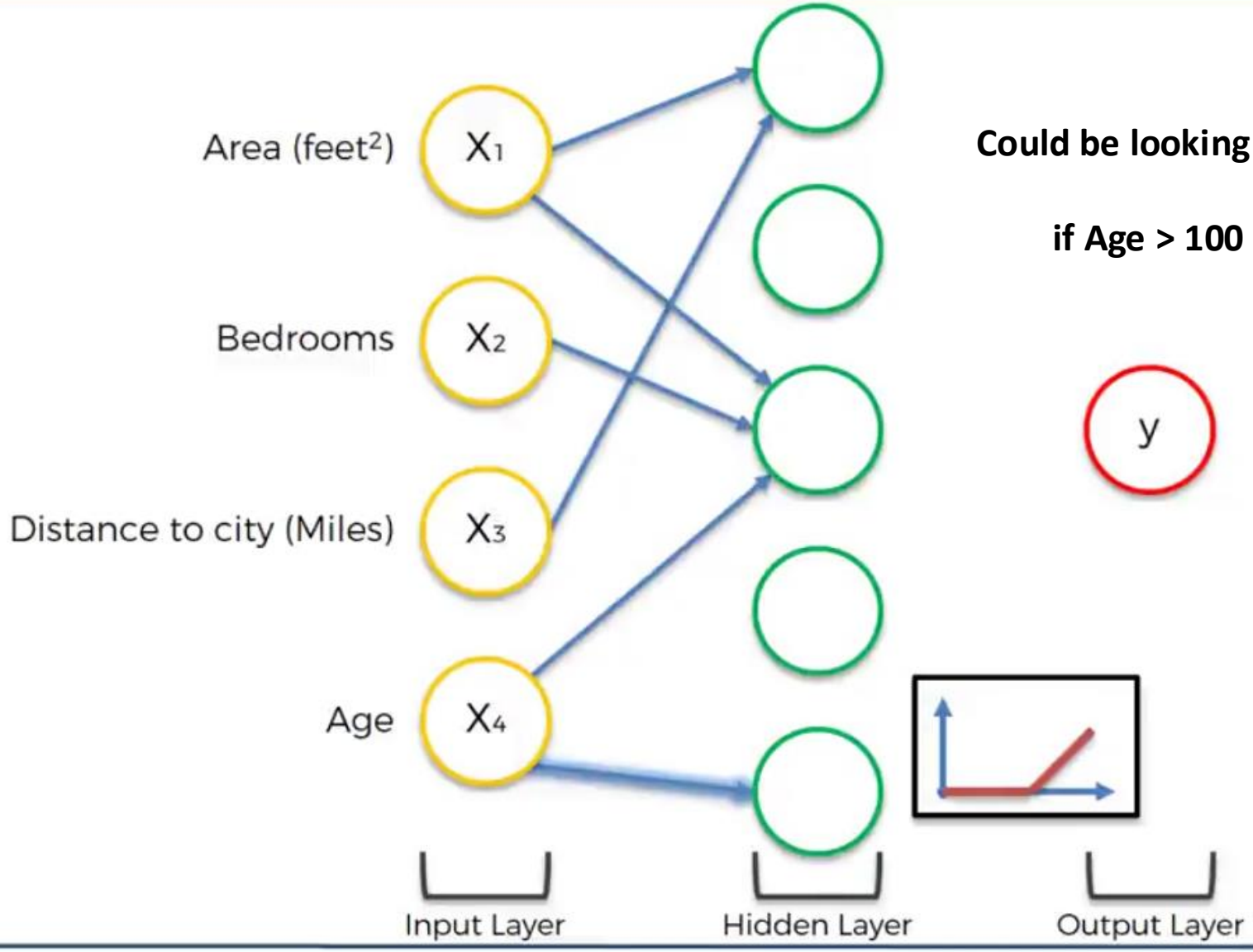


Looking specifically for relationships between certain inputs i.e. Area, Bedrooms, Age

How Do Neural Networks Work?

Could be looking for a certain single criteria
i.e.
if Age > 100 then historic property

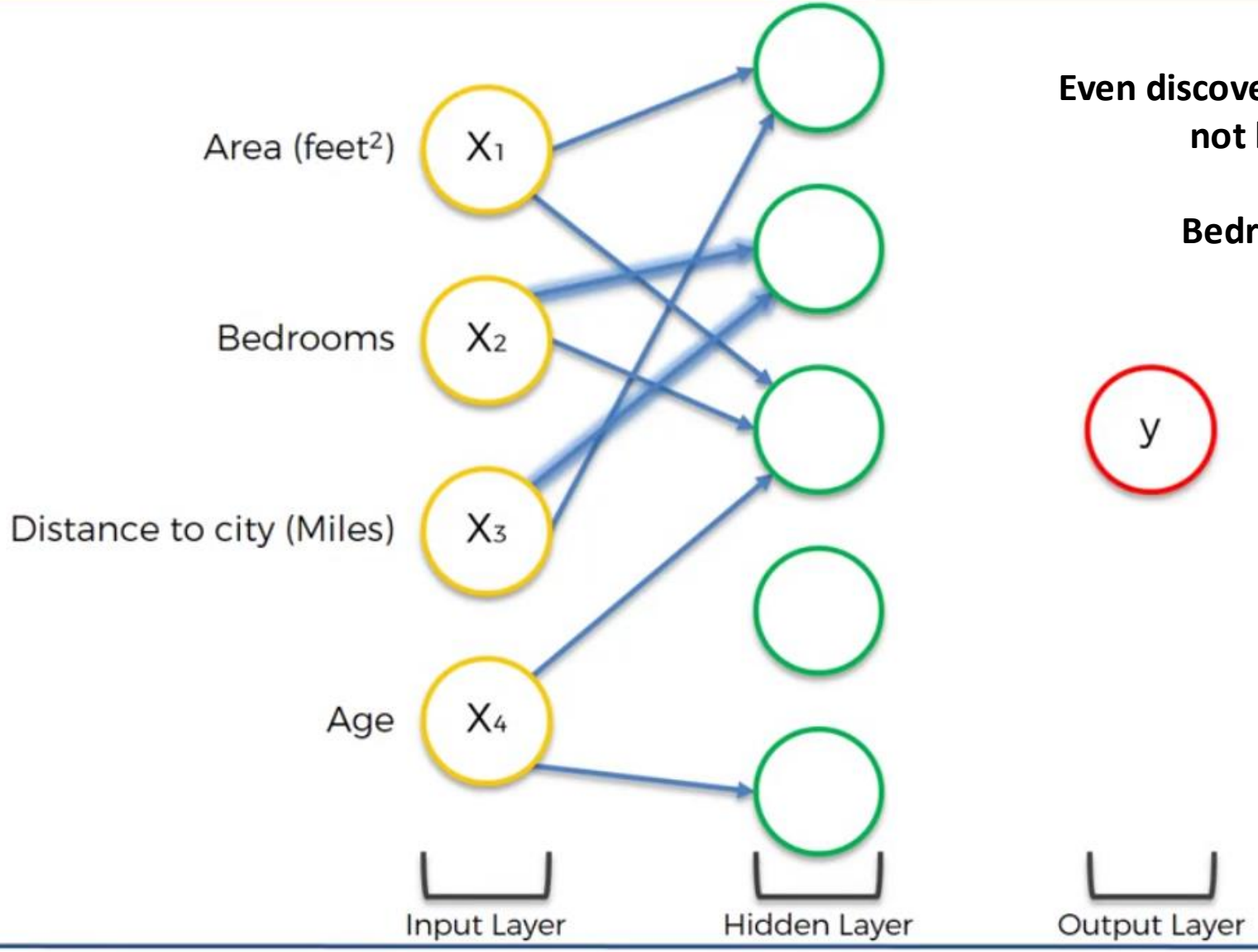
How Do Neural Networks Work?



Could be looking for a certain single criteria
i.e.
if Age > 100 then historic property

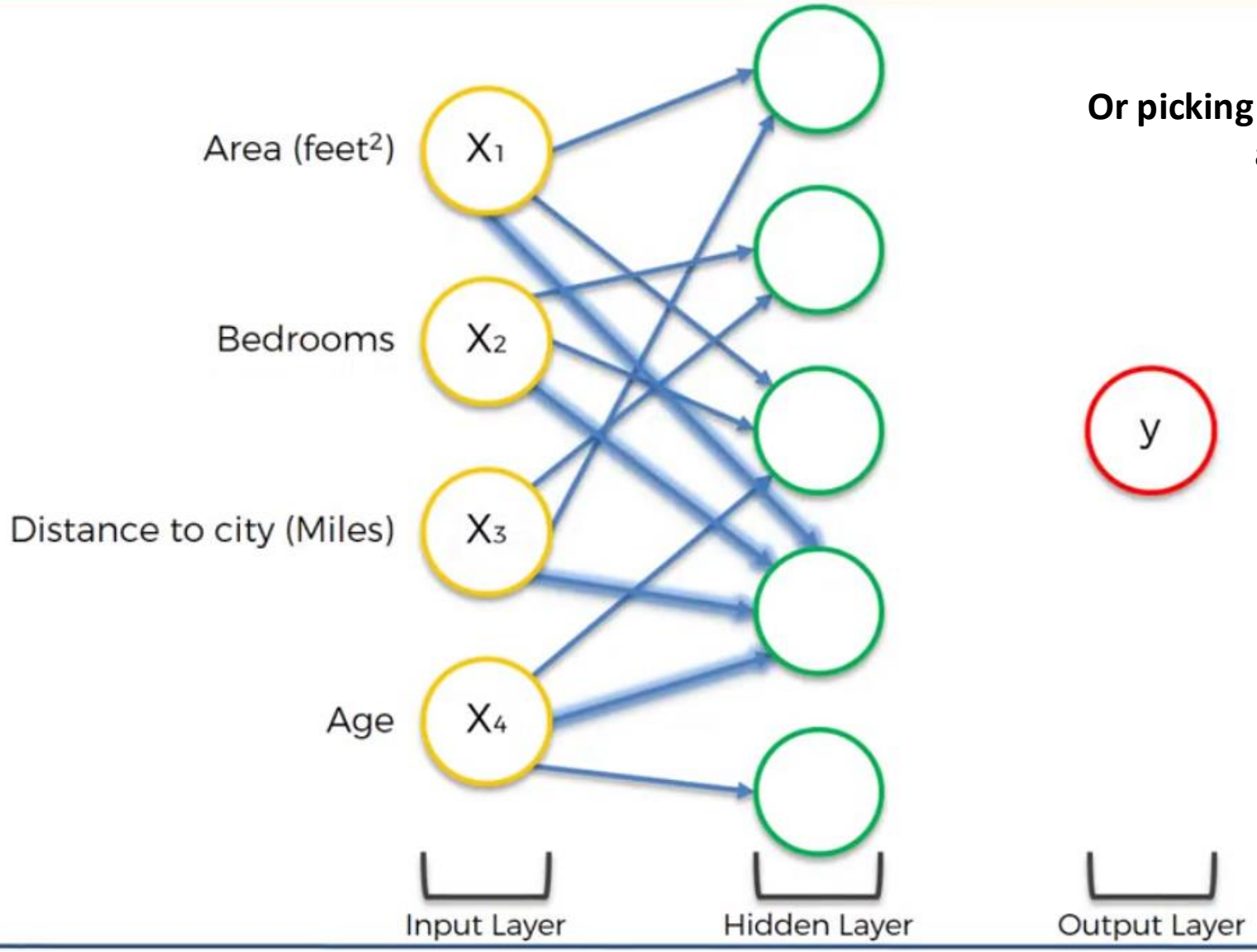
Good Example of a rectifier function example

How Do Neural Networks Work?



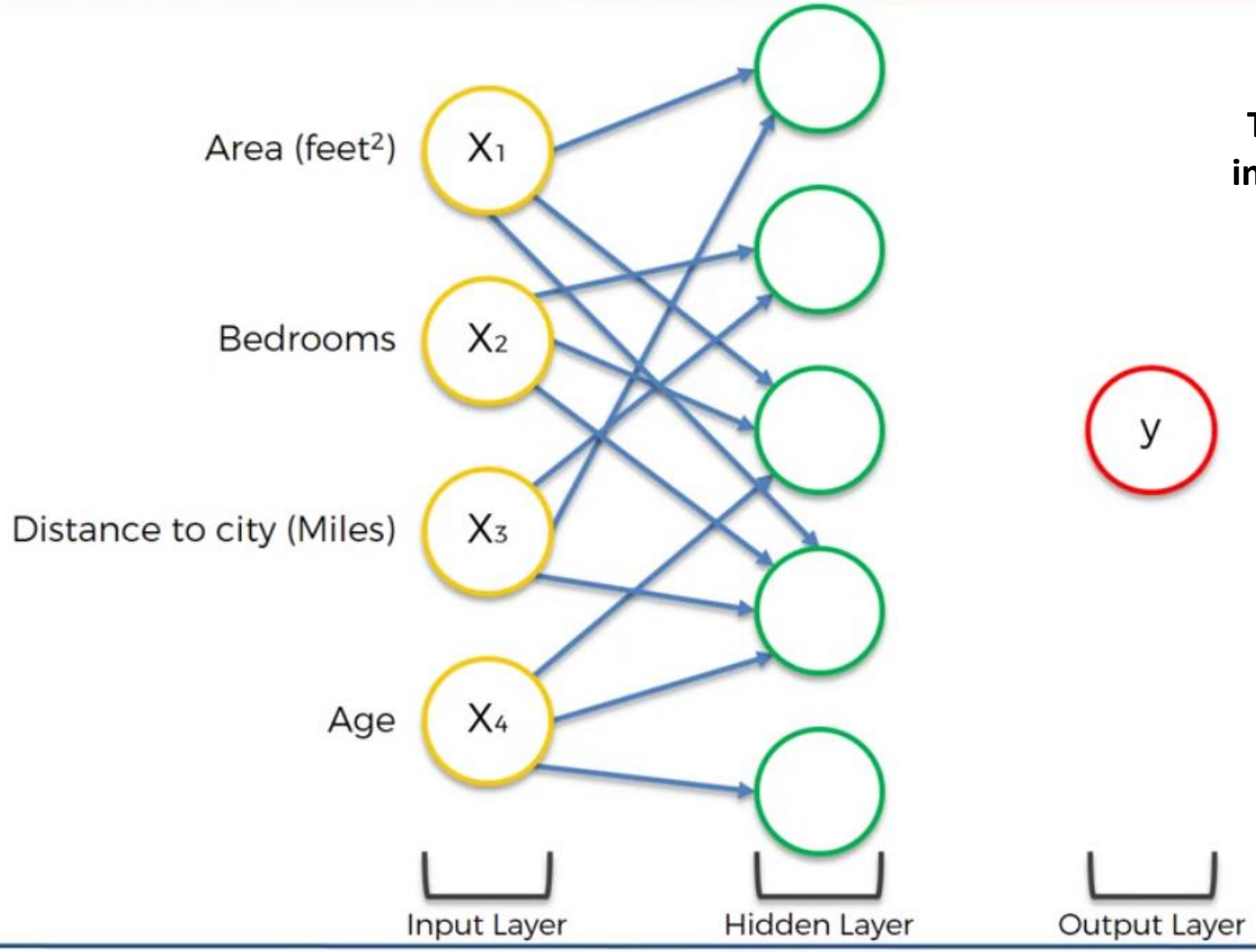
Even discovering relationships that may not be as intuitive to us
i.e.
Bedrooms & Dist. to City

How Do Neural Networks Work?



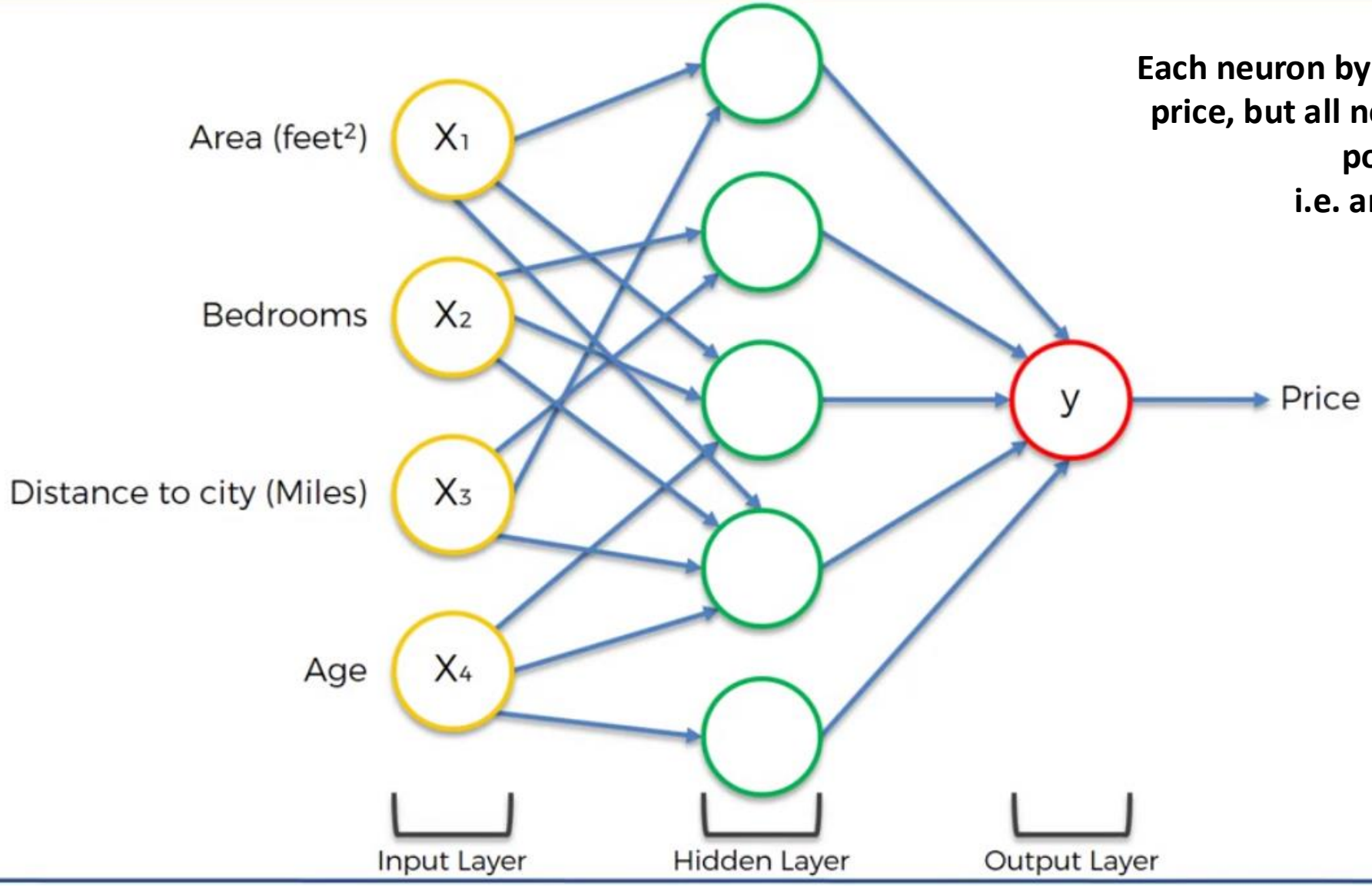
Or picking up a combination of all 4 inputs

How Do Neural Networks Work?



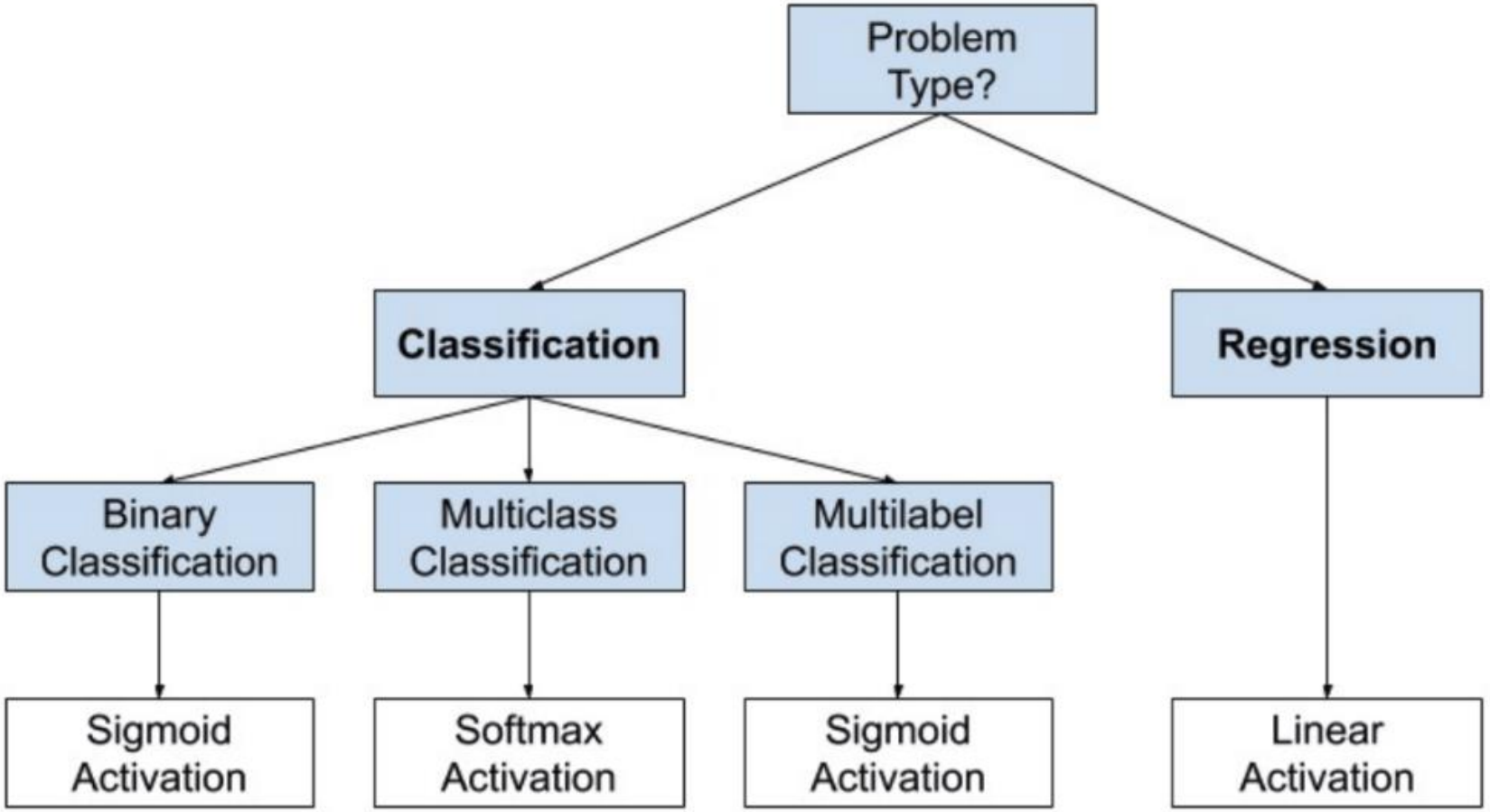
The hidden layer allows for increased flexibility of neural network

How Do Neural Networks Work?

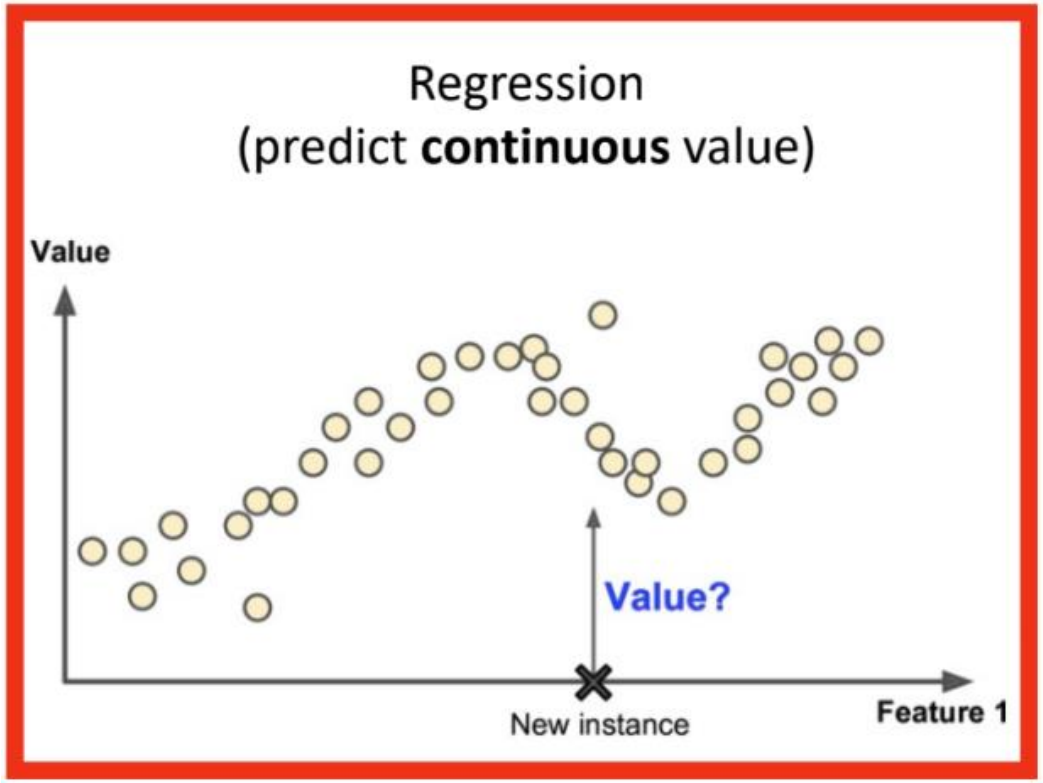


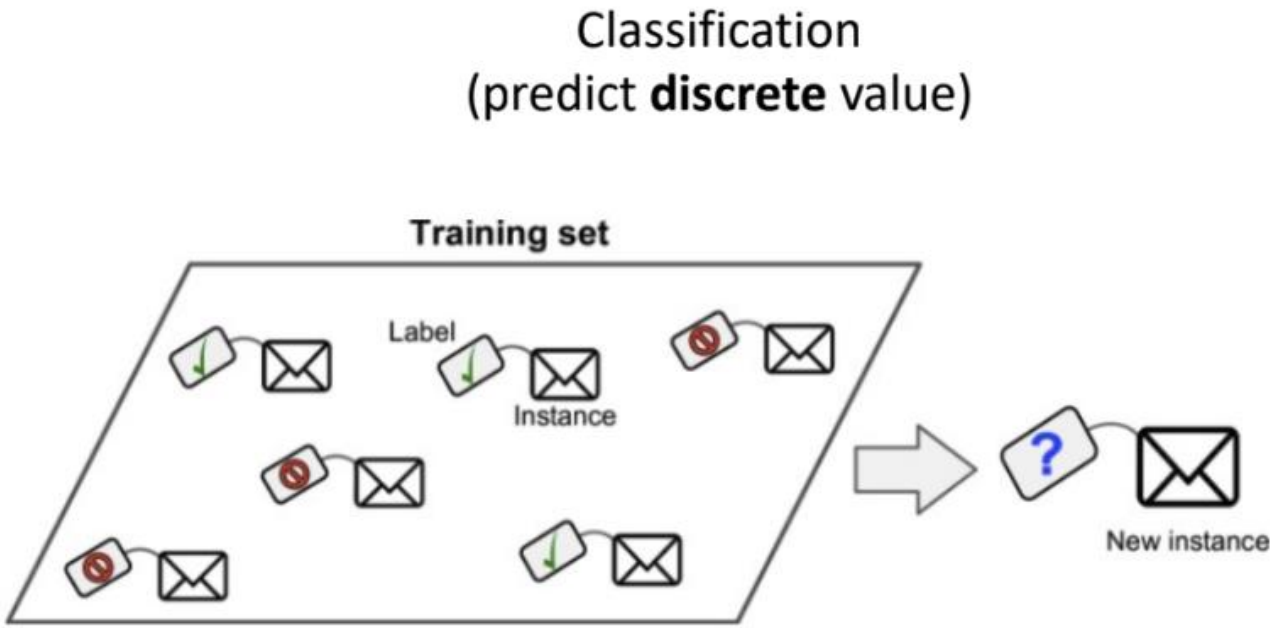
Each neuron by itself cannot predict price, but all neurons together are powerful
i.e. ant analogy

Desired Output Driven by Task

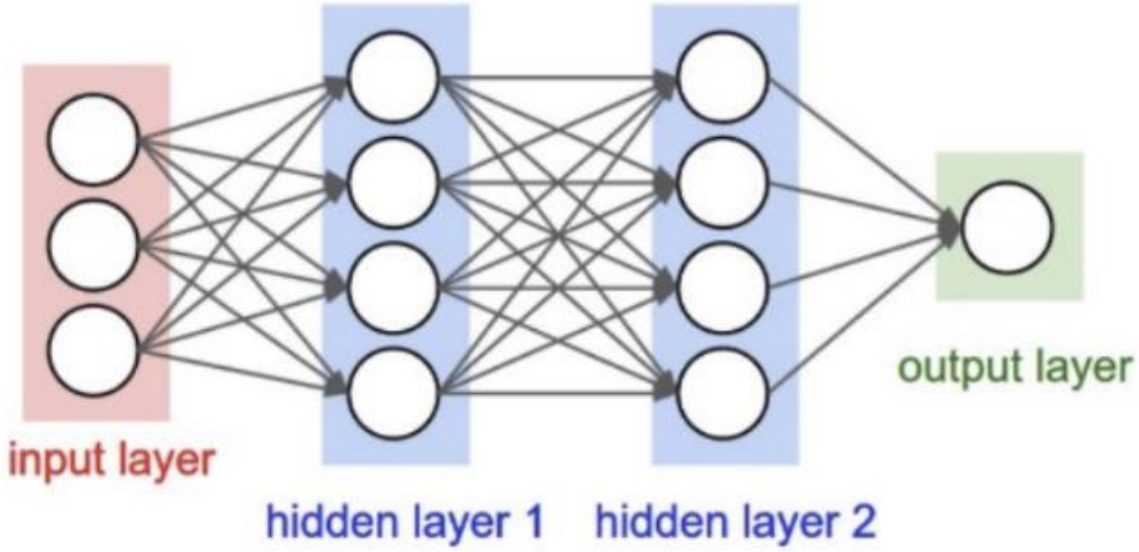


Objective Function





Objective Function



e.g., make as small as possible the squared error (aka, L2 loss, quadratic loss)

Mean taken over *n* instances

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

True value

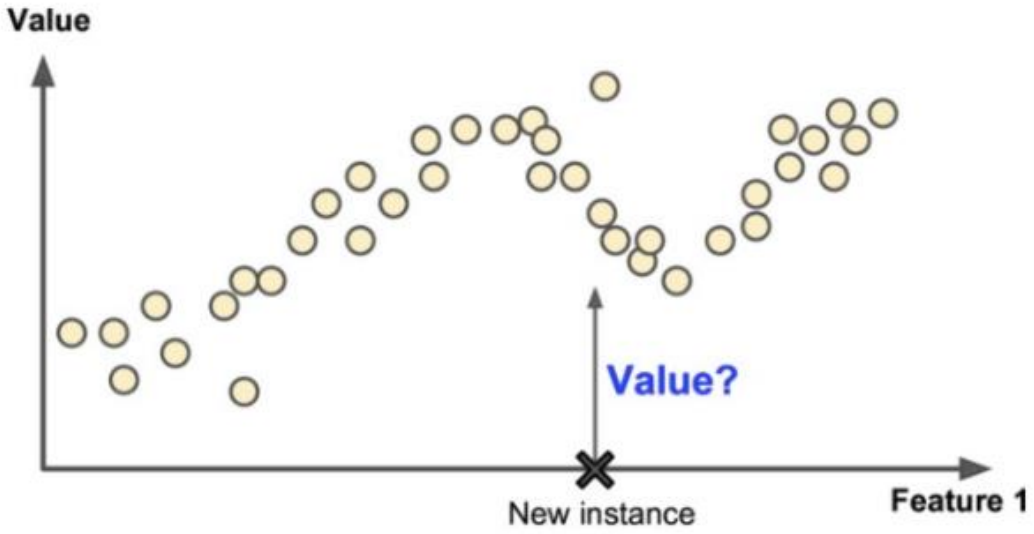
Predicted value

What is the range of possible values?

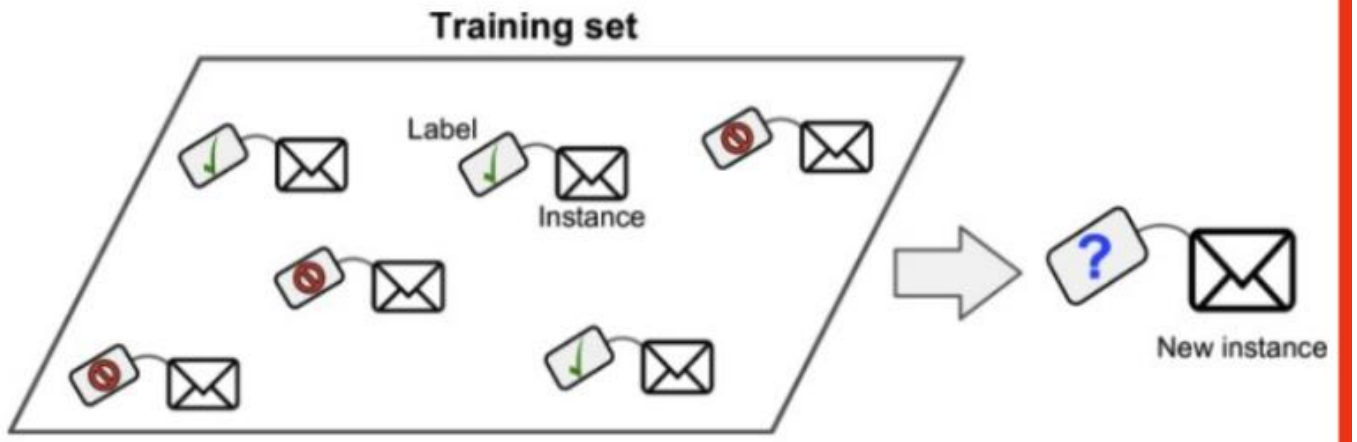
- Minimum: 0
  - i.e., all correct predictions
- Maximum: Infinity
  - i.e., incorrect predictions

Objective Function

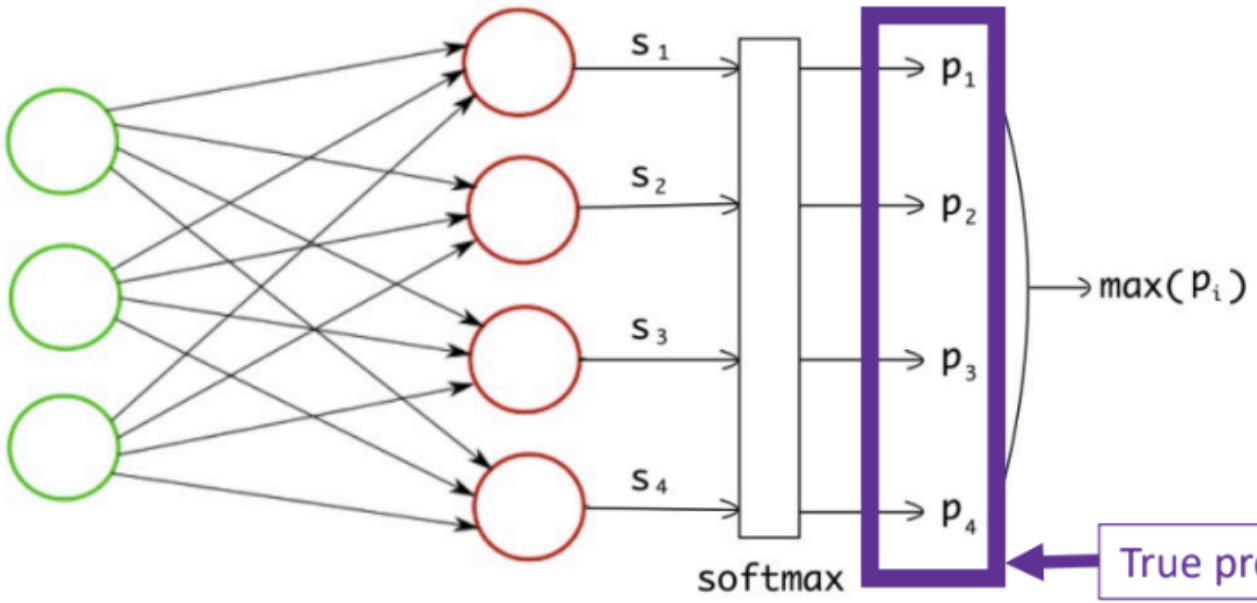
Regression
(predict **continuous** value)



Classification
(predict **discrete** value)



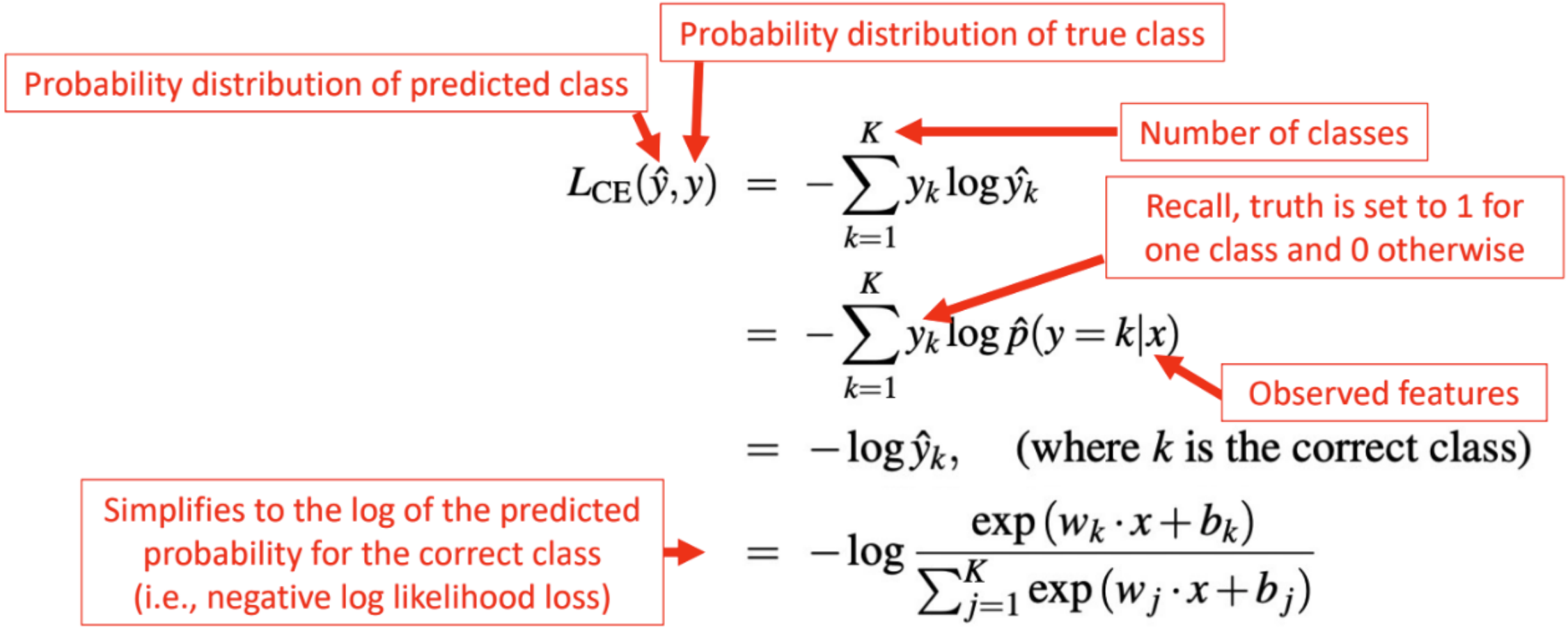
Objective Function



e.g., make as small as possible the distance between predicted and true class distributions with cross entropy loss

True prediction is 1 hot vector (i.e., one 1 and the rest 0s)

Objective Function



Objective Function

Probability distribution of predicted class

Probability distribution of true class

Number of classes?

Recall, truth is set to 1 for one class and 0 otherwise

Observed features

$$L_{\text{CE}}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

$$= -\sum_{k=1}^{K} y_k \log \hat{p}(y = k|x)$$

$$= -\log \hat{y}_k, \quad \text{(where } k \text{ is the correct class)}$$

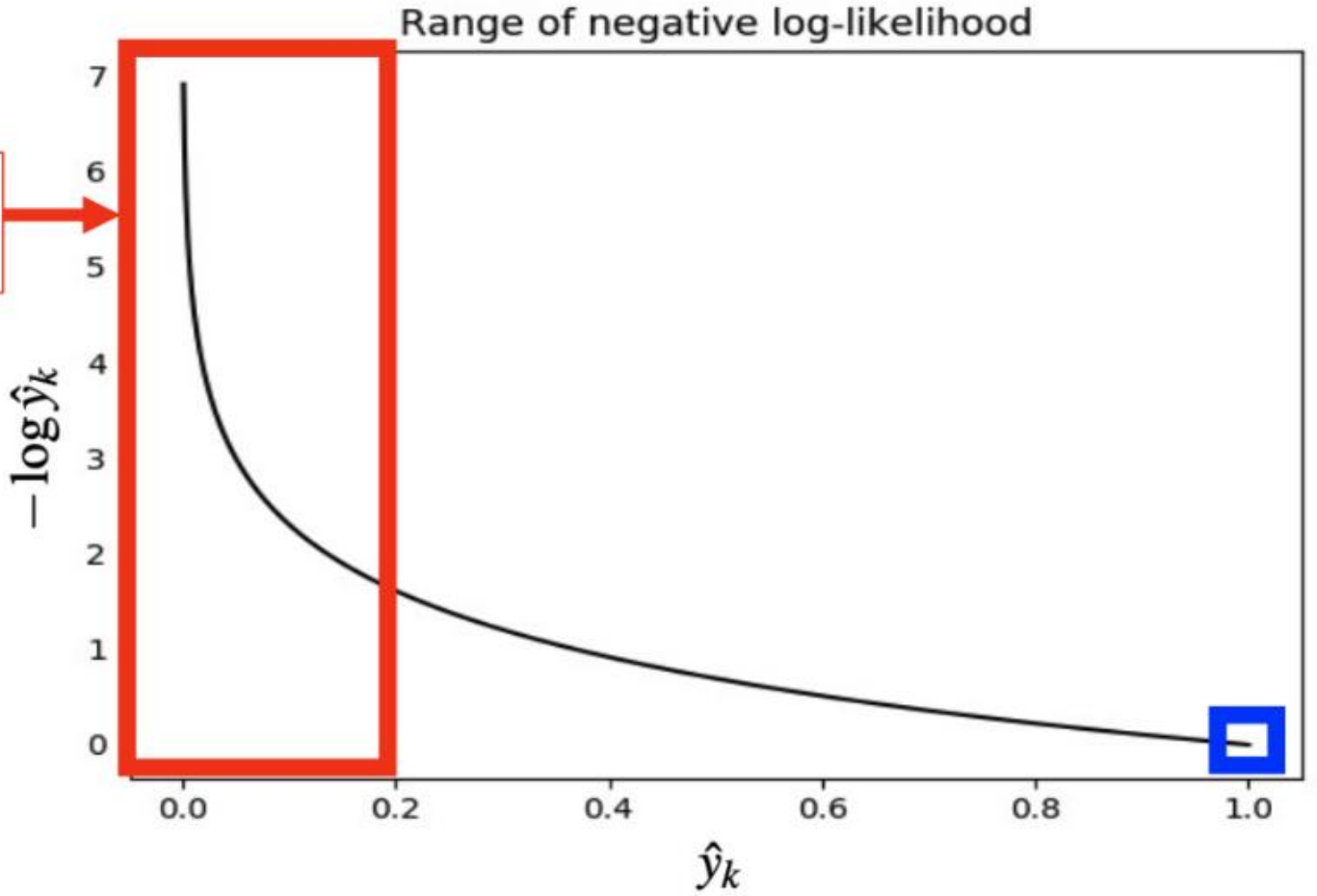
$$= -\log \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)}$$

What is the range of possible values?

- Minimum: 0
  - i.e., correct prediction: negative log of 1
- Maximum: Infinity
  - i.e., incorrect prediction: negative log of 0

Objective Function

More confidently wrong predictions lead to greater error



What is the range of possible values?

- Minimum: 0
  - i.e., correct prediction: negative log of 1
- Maximum: Infinity
  - i.e., incorrect prediction: negative log of 0

Gradient Descent: How Often to Update?

- Use calculations over **all training examples** (**Batch gradient descent**)
  - Less bouncing but can be slow or infeasible when dataset is large

- Use calculations from **one training example** (**Stochastic gradient descent**)
  - Fast to compute and can train using huge datasets (stores one instance in memory at each iteration) but updates are expected to bounce a lot

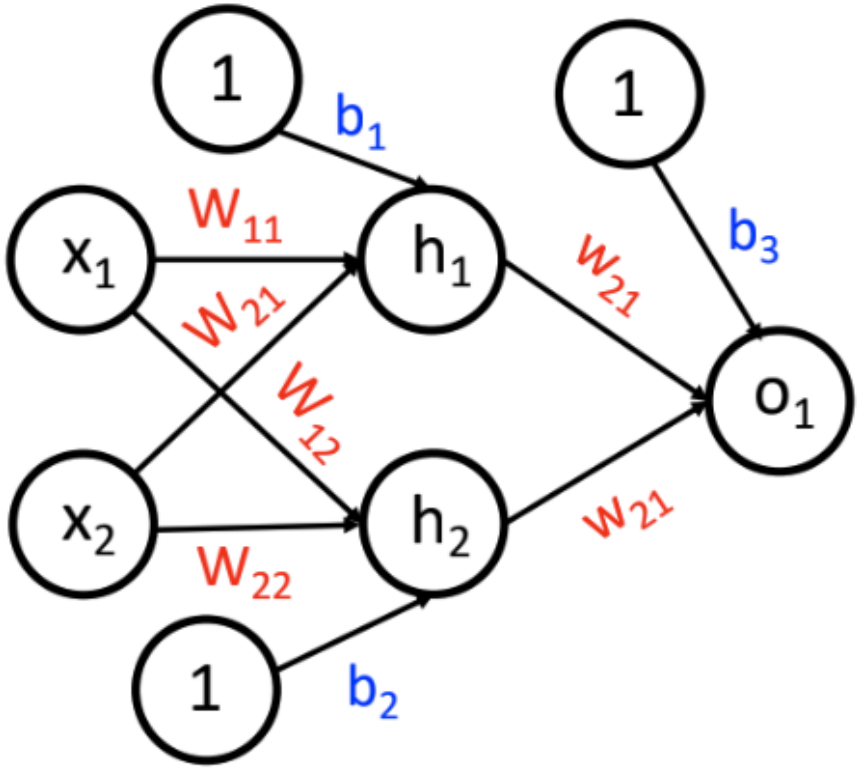
- Use calculations over **subset of training examples** (**Mini-batch gradient descent**)
  - Bounces less erratically than SGD and can train using huge datasets (store some instances in memory at each iteration) but can be slow or infeasible when dataset is large

- Often mini-batch gradient descent is used with maximum # of examples that fit in memory

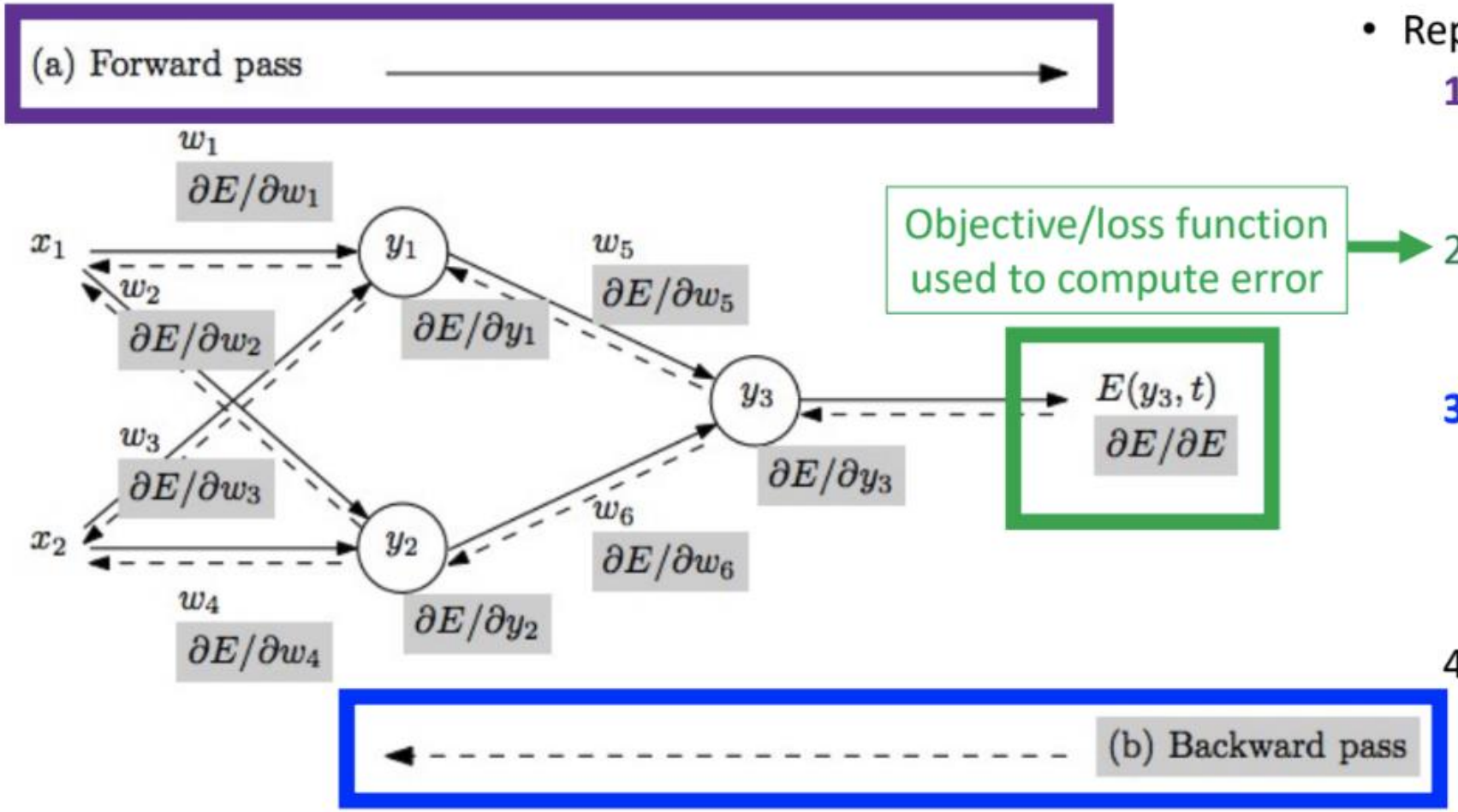
Recall: What to Learn in Neural Network?

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

- e.g., 2 layer neural network:



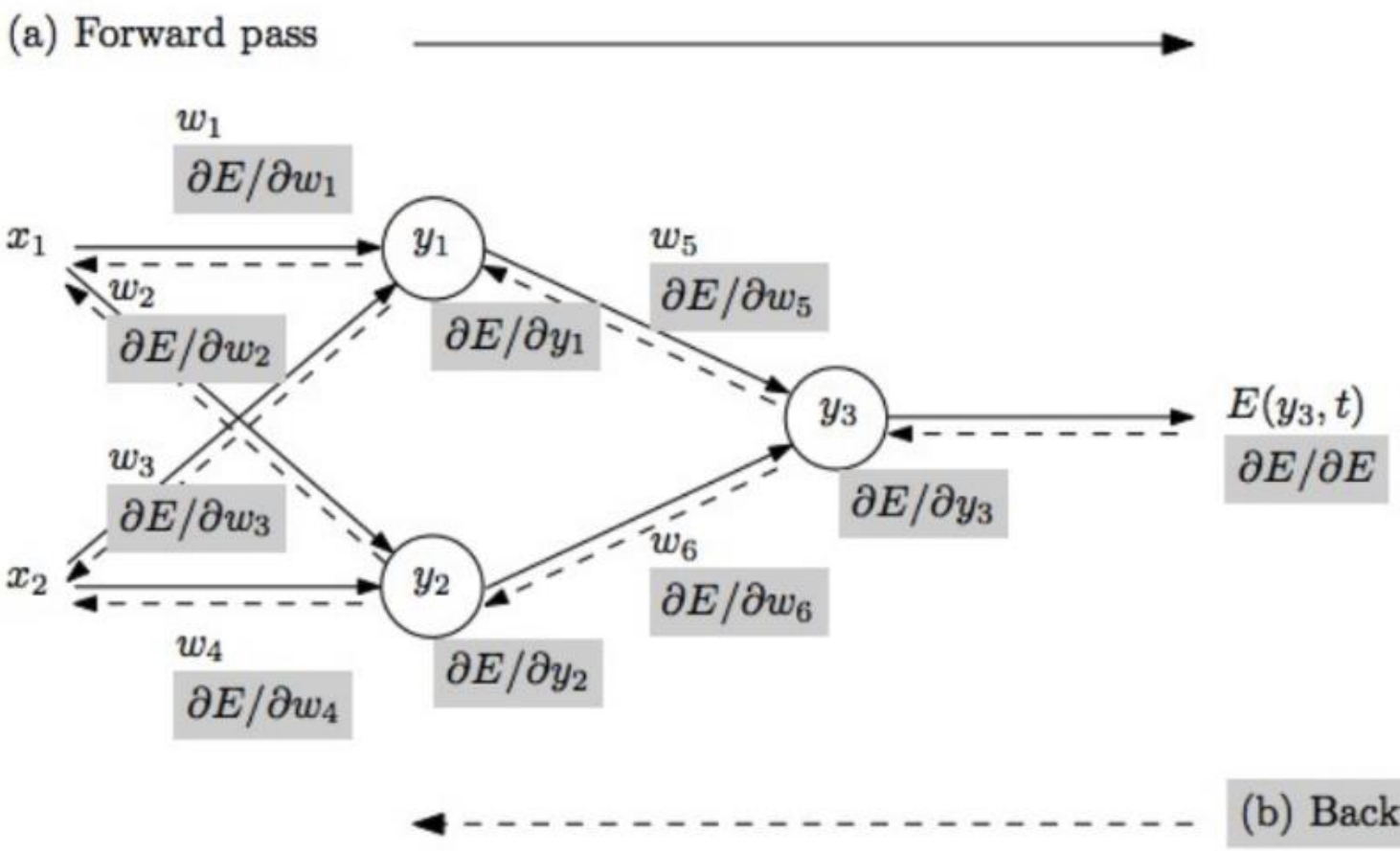
Training: How Neural Networks Learn



Objective/loss function used to compute error

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction
  2. Quantify the dissatisfaction with a model's results on the training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

When to Stop Training Neural Networks?



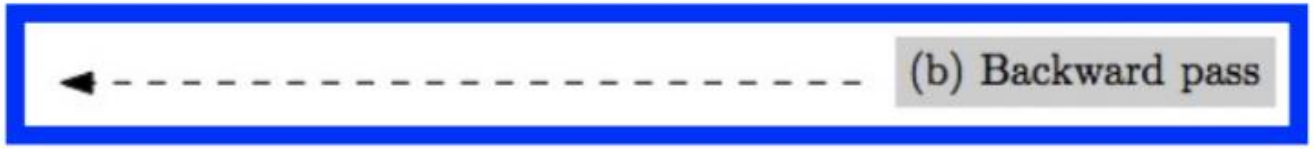
- What stopping criterion to use?
  - Weight changes are incredibly small
  - Finished a pre-specified number of epochs
  - Percentage of misclassified example is below some threshold
  - …

Key Challenge: How to Compute Gradient?

Equation for calculating gradients depends on:

1) Activation functions

2) Objective/loss function

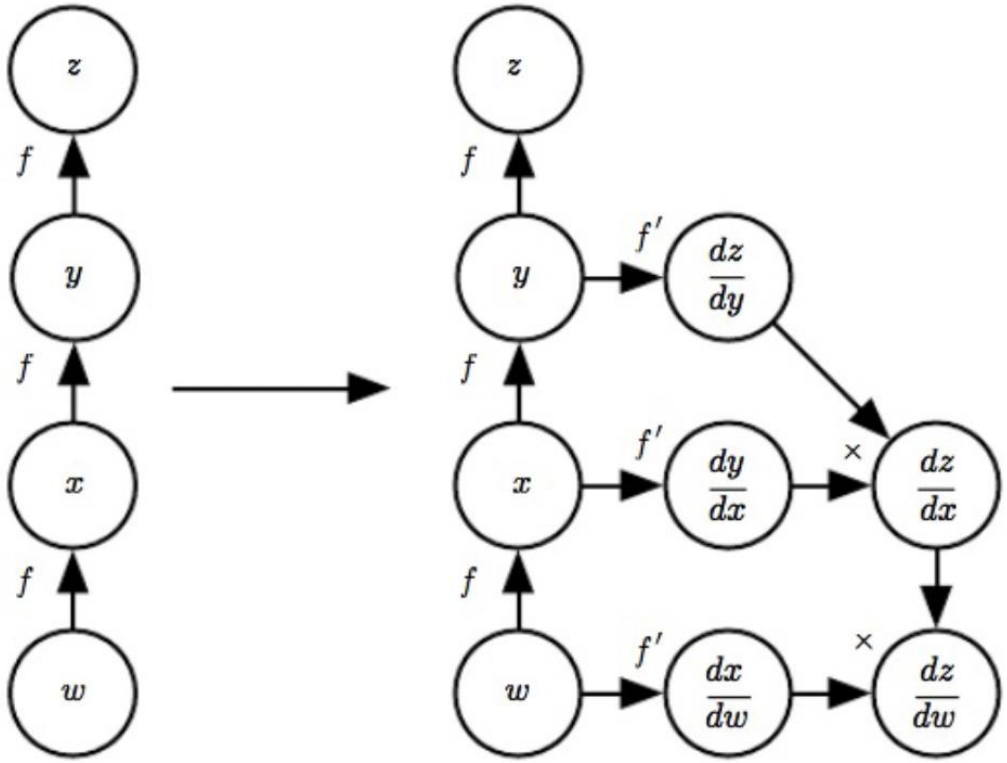


Repeat until stopping criterion met:

1. **Forward pass**: propagate training data through model to make prediction
2. Quantify the dissatisfaction with a model's results on the training data
3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
4. Update each parameter using calculated gradients

Neural Network Training: Backpropagation

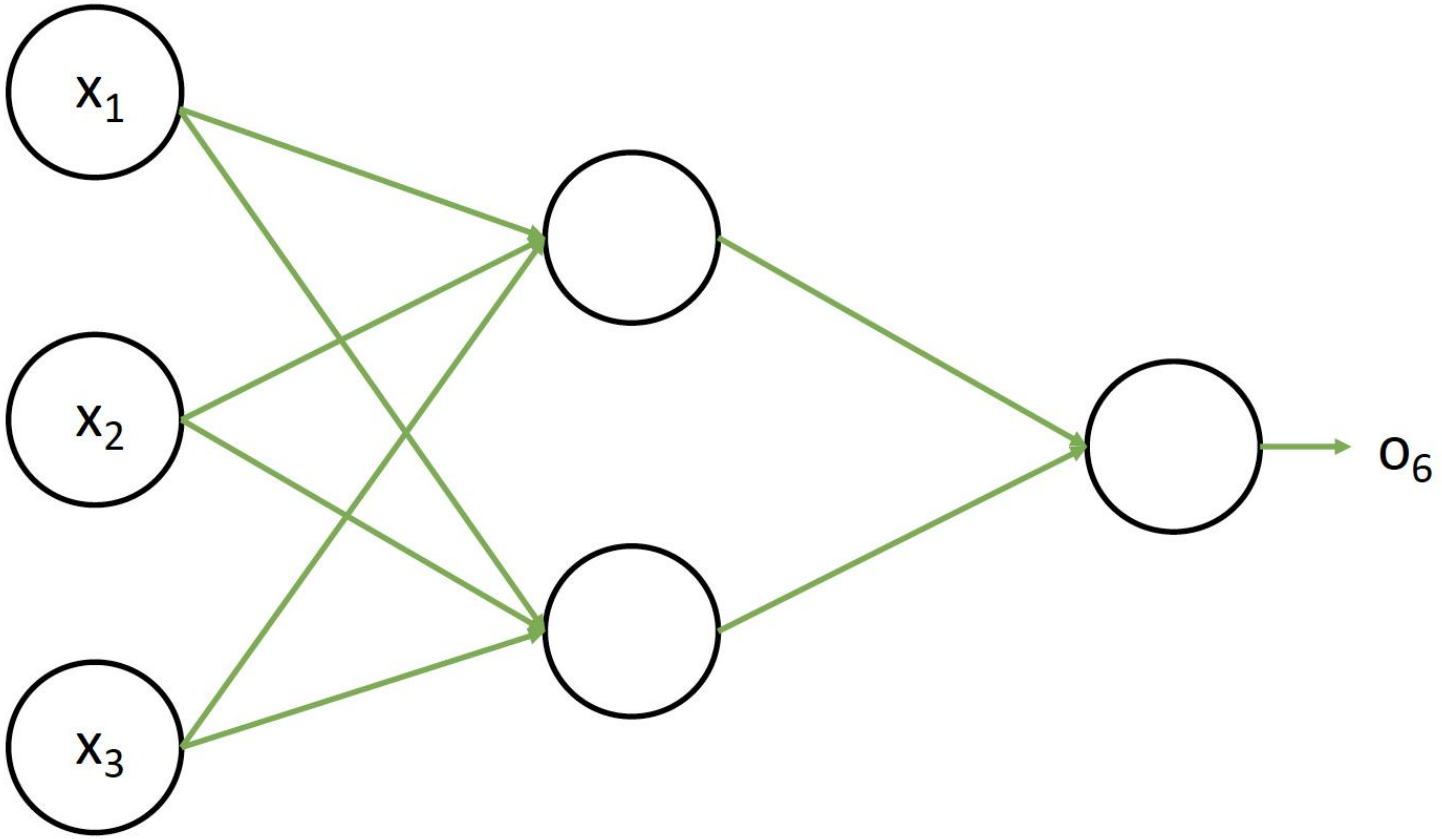
- Backpropagation idea: chain: $x = f(w),\ y = f(x),\ z = f(y)$



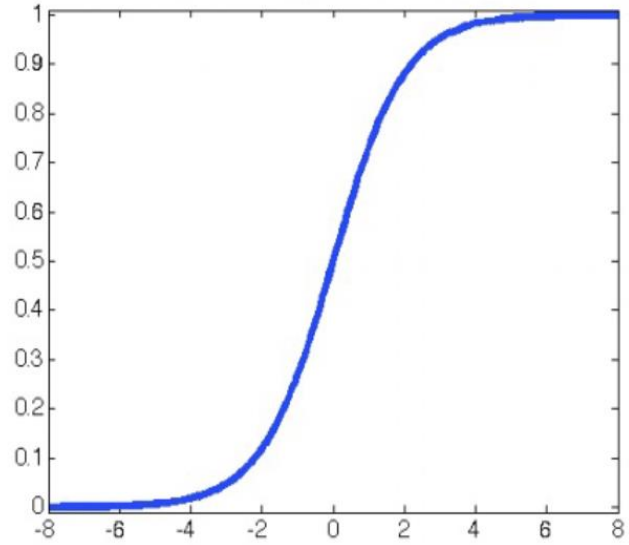
$$
\begin{aligned}
&\frac{\partial z}{\partial w} \\
=&\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
=&f'(y)f'(x)f'(w) \\
=&f'(f(f(w)))f'(f(w))f'(w).
\end{aligned}
$$

Neural Network Training: Backpropagation

Choose Neural Network Architecture



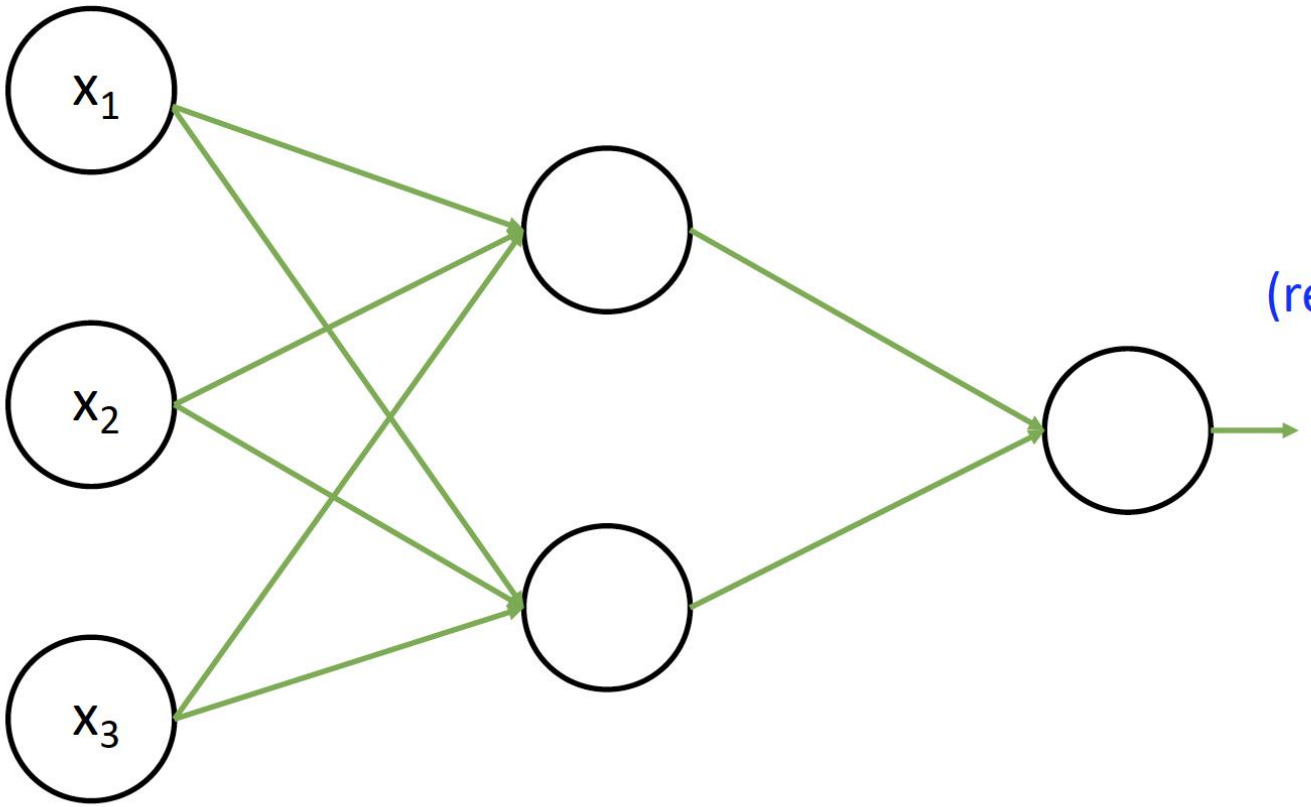
Sigmoid Activation Function



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

Neural Network Training: Backpropagation

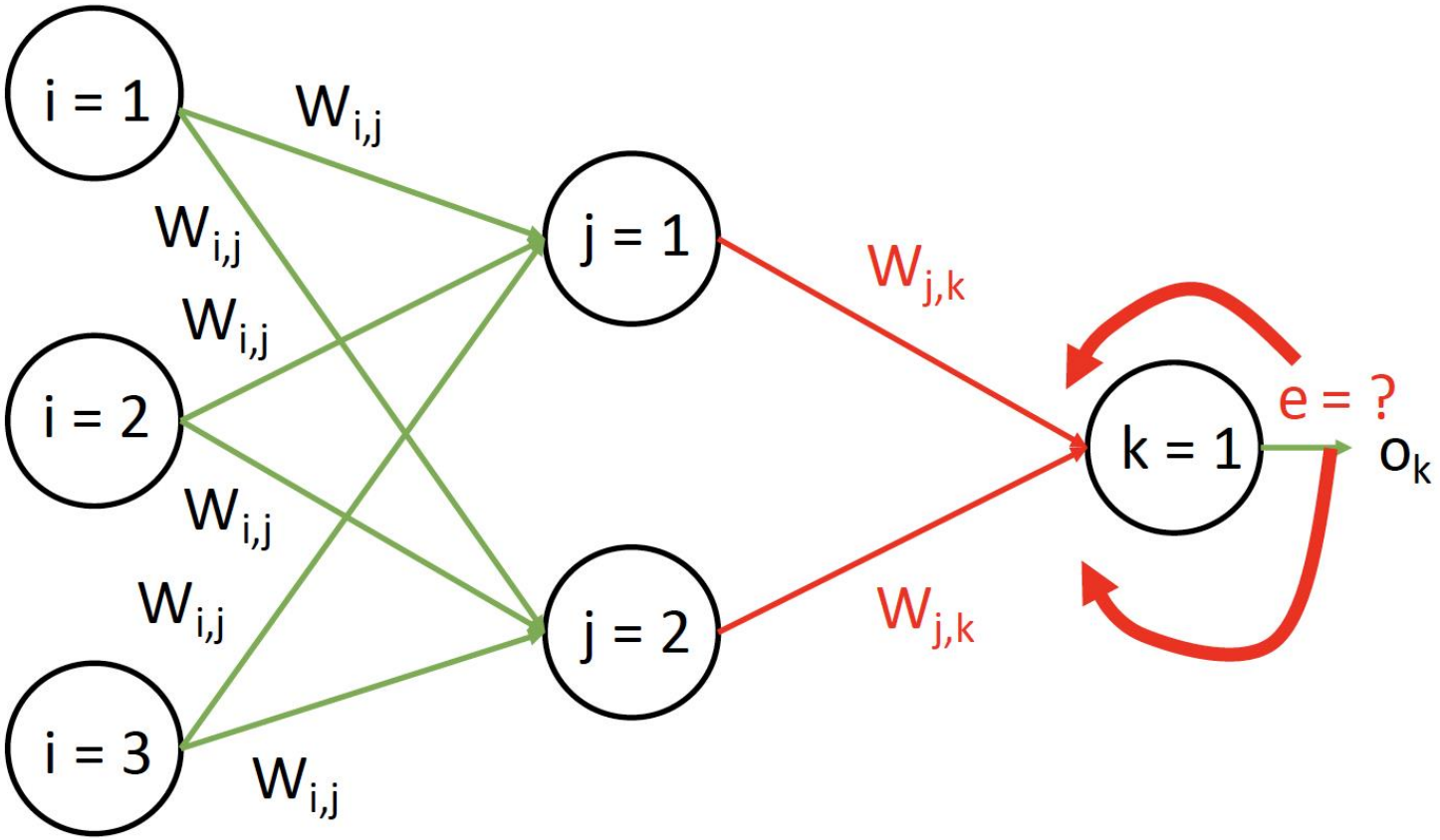
Choose Loss Function for Training



Squared Error Function
(regression or binary classification problem)

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

Neural Network Training: Backpropagation



t is a constant value:

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

Neural Network Training: Backpropagation

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 :

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

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

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

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

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

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

Neural Network Training: Backpropagation

t is a constant value:

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

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

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

Using the following chain rule :

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

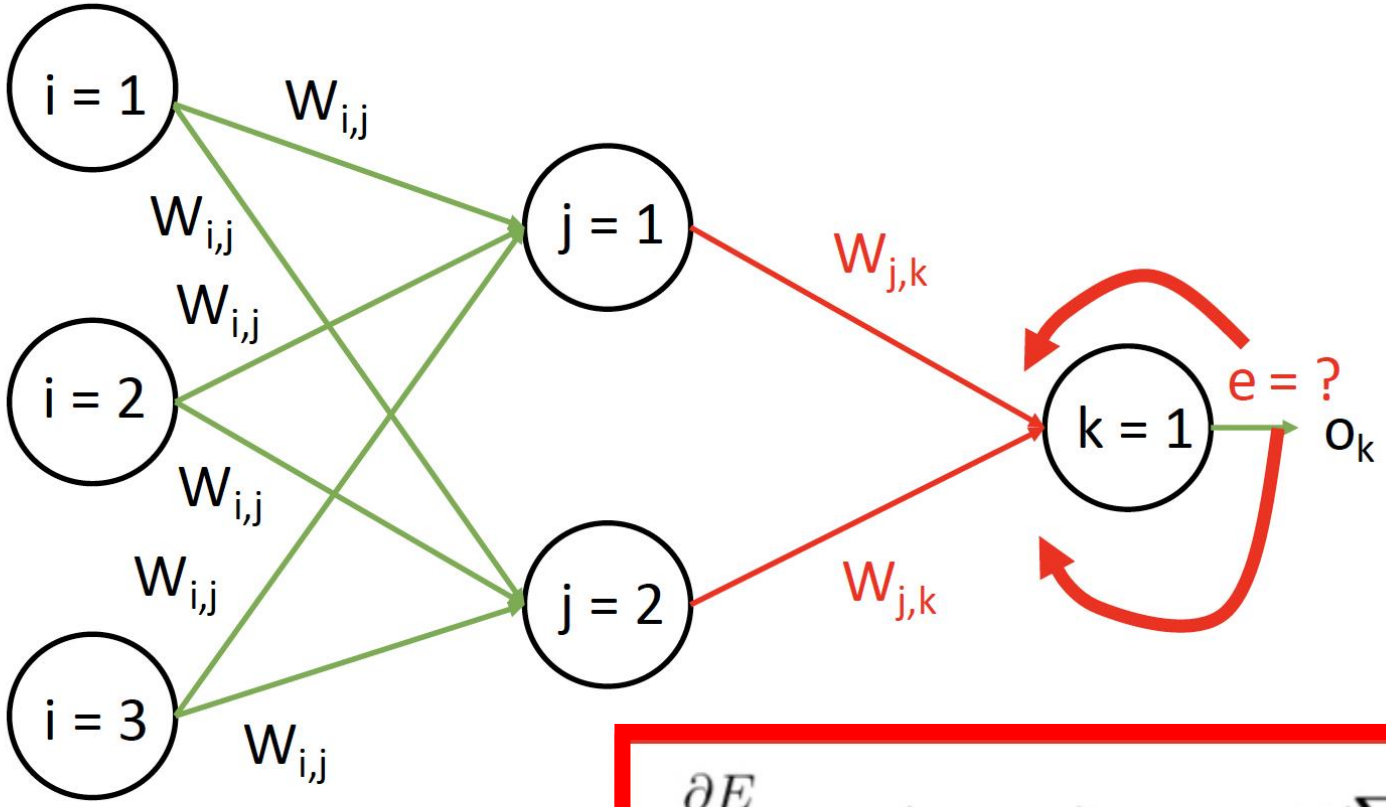
$$\frac{d\sigma(x)}{dx} = (1-\sigma(x))\,\sigma(x)$$

We can rewrite our function as follows:

For efficiency, compute last

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

Neural Network Training: Backpropagation

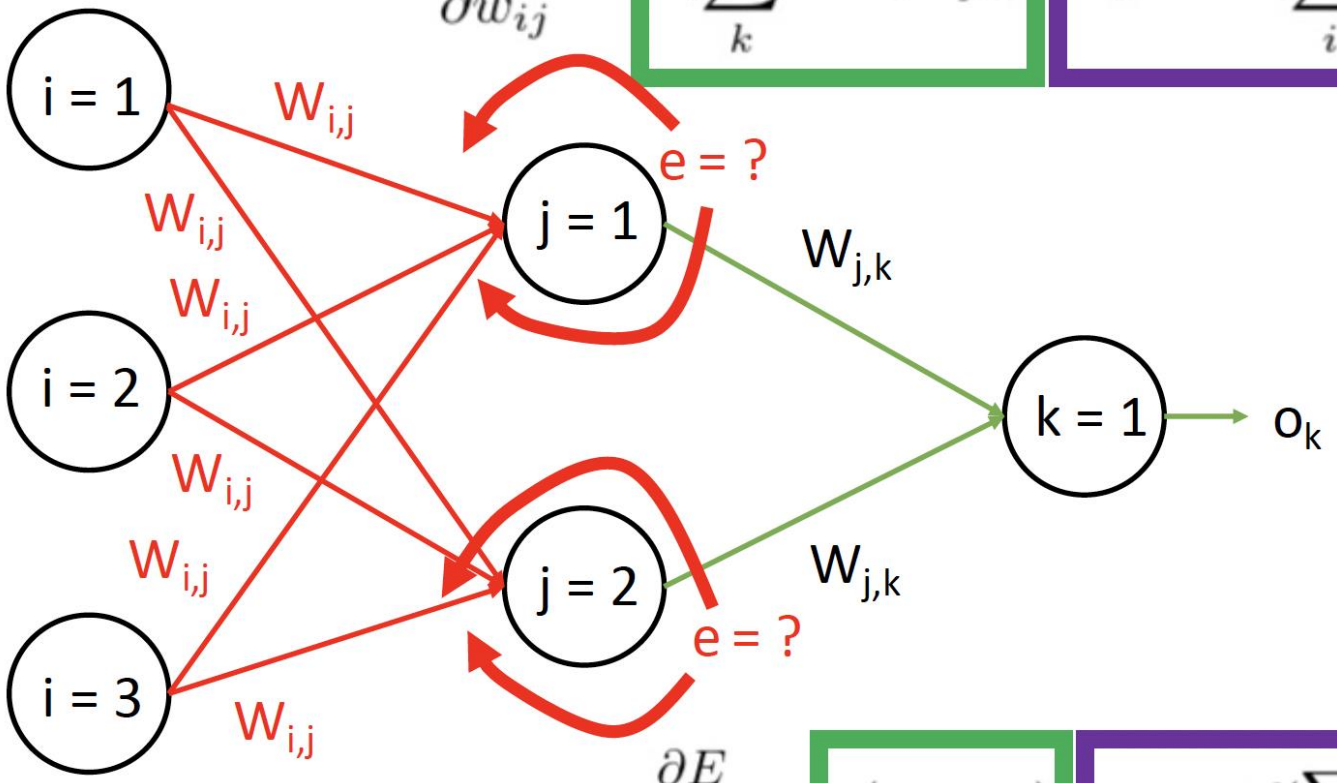


$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

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

Neural Network Training: Backpropagation

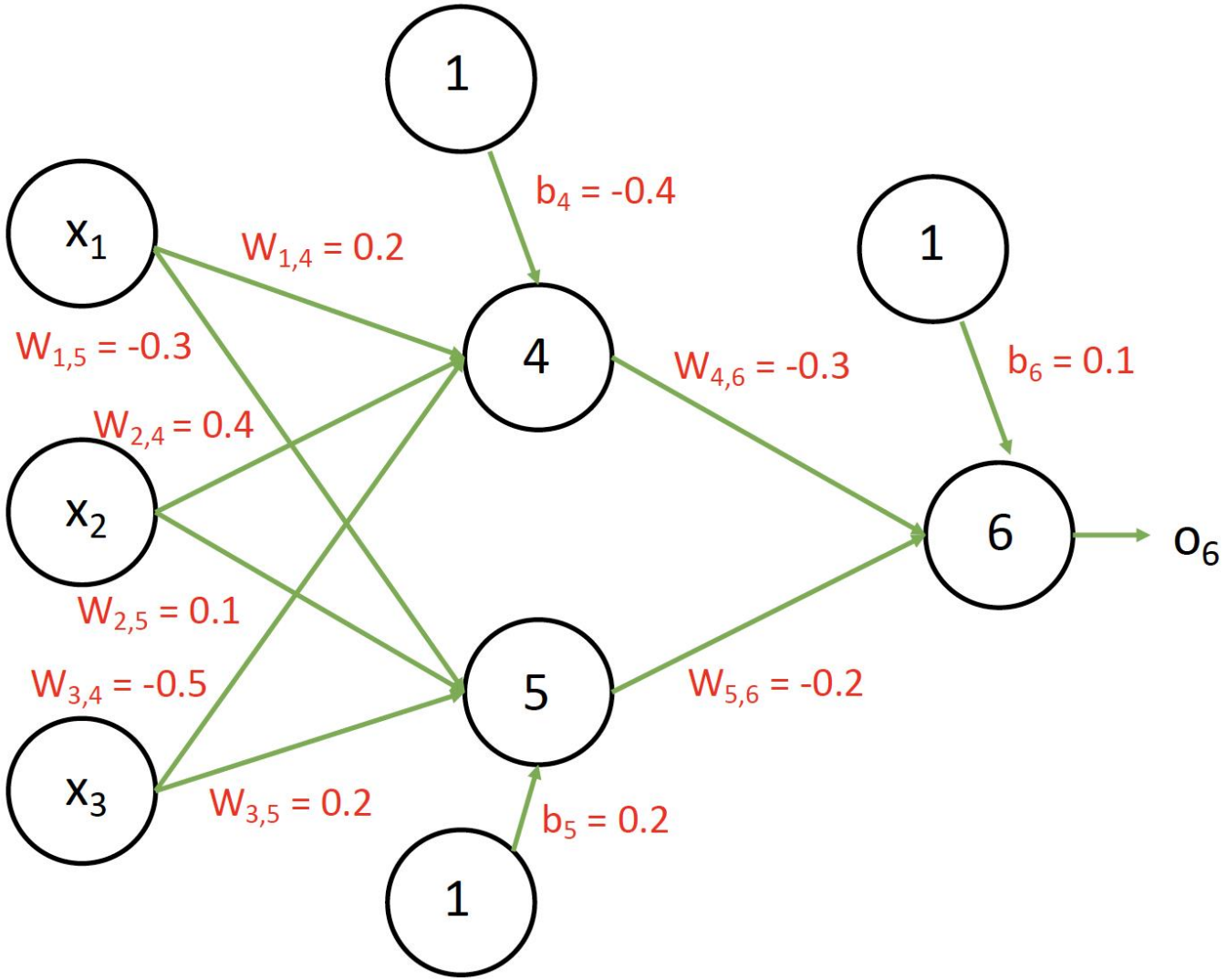
$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$



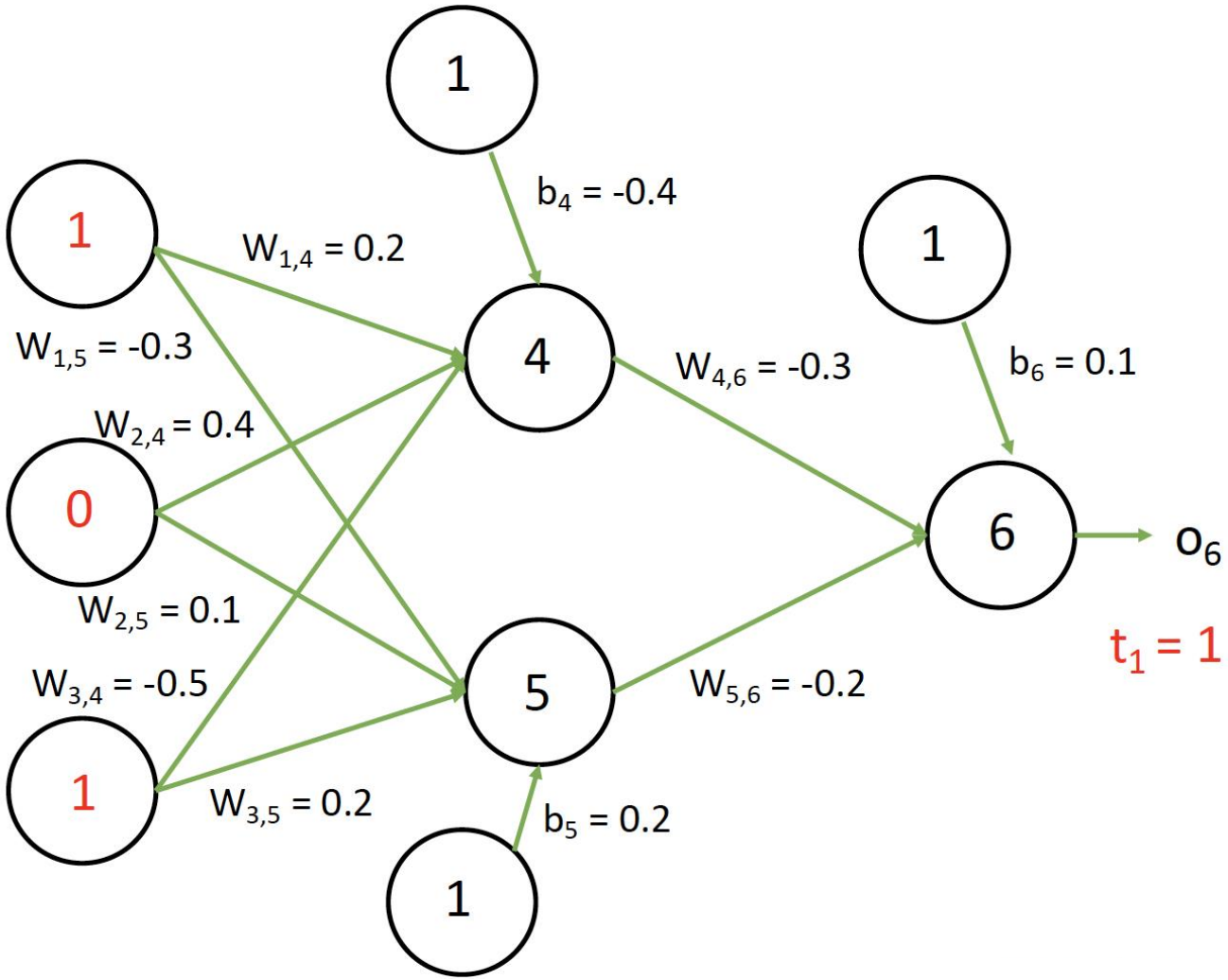
Symmetry in how to solve for parameters in hidden layers

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

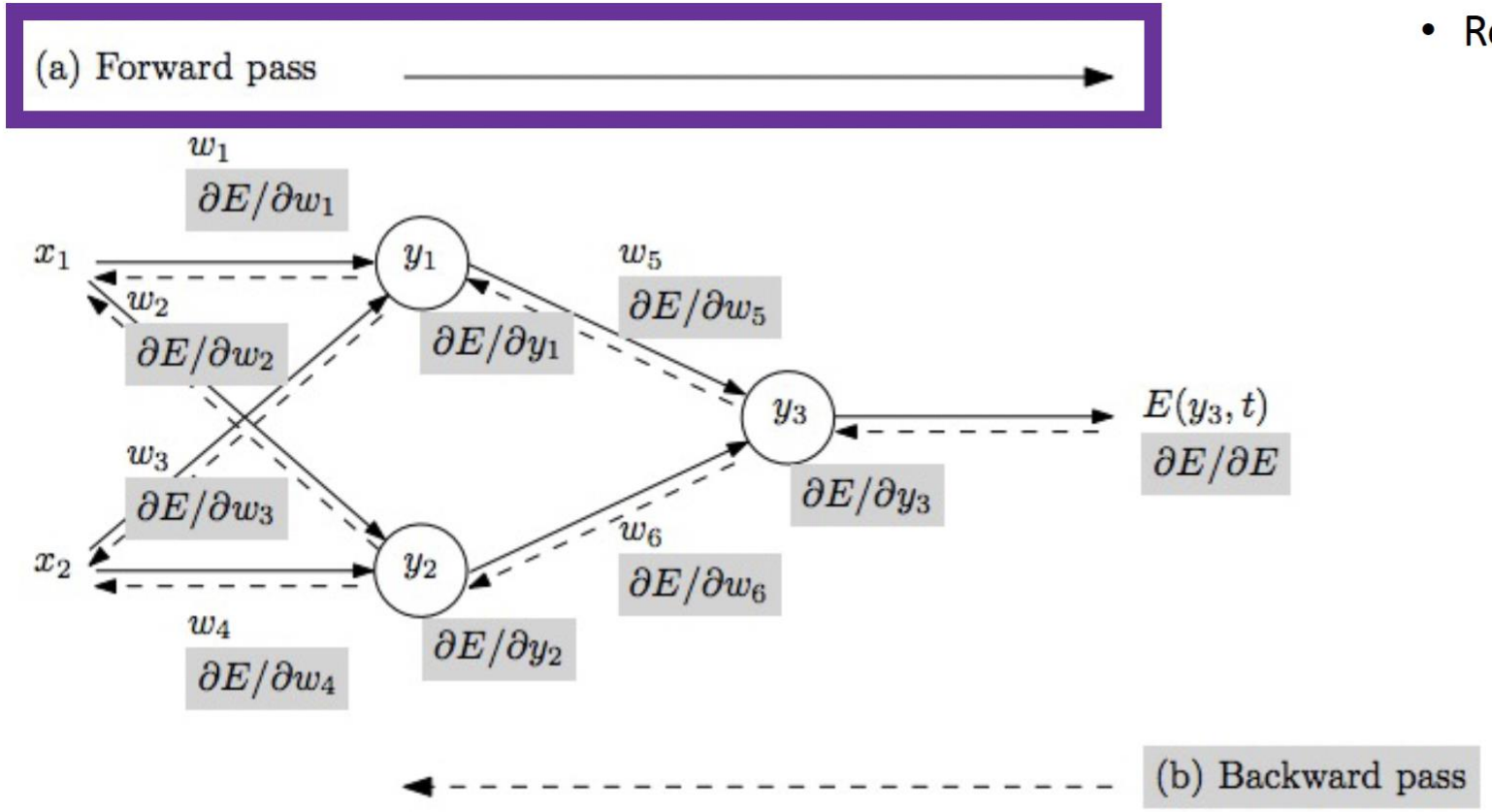
Neural Network Training: Backpropagation



Neural Network Training: Backpropagation

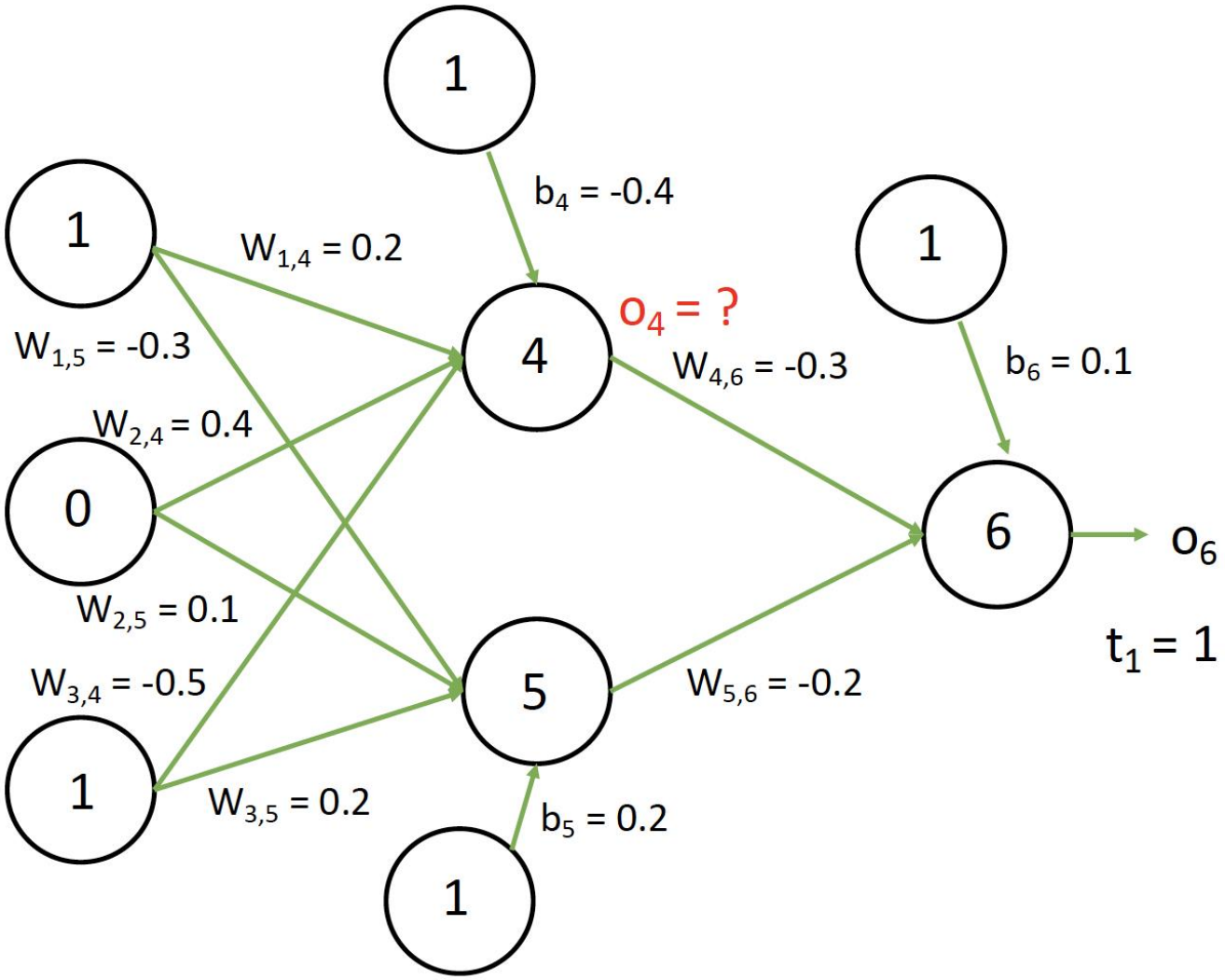


Neural Network Training: Backpropagation



- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction

Neural Network Training: Backpropagation



Input to node 4:

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

$i_4 = -0.7$

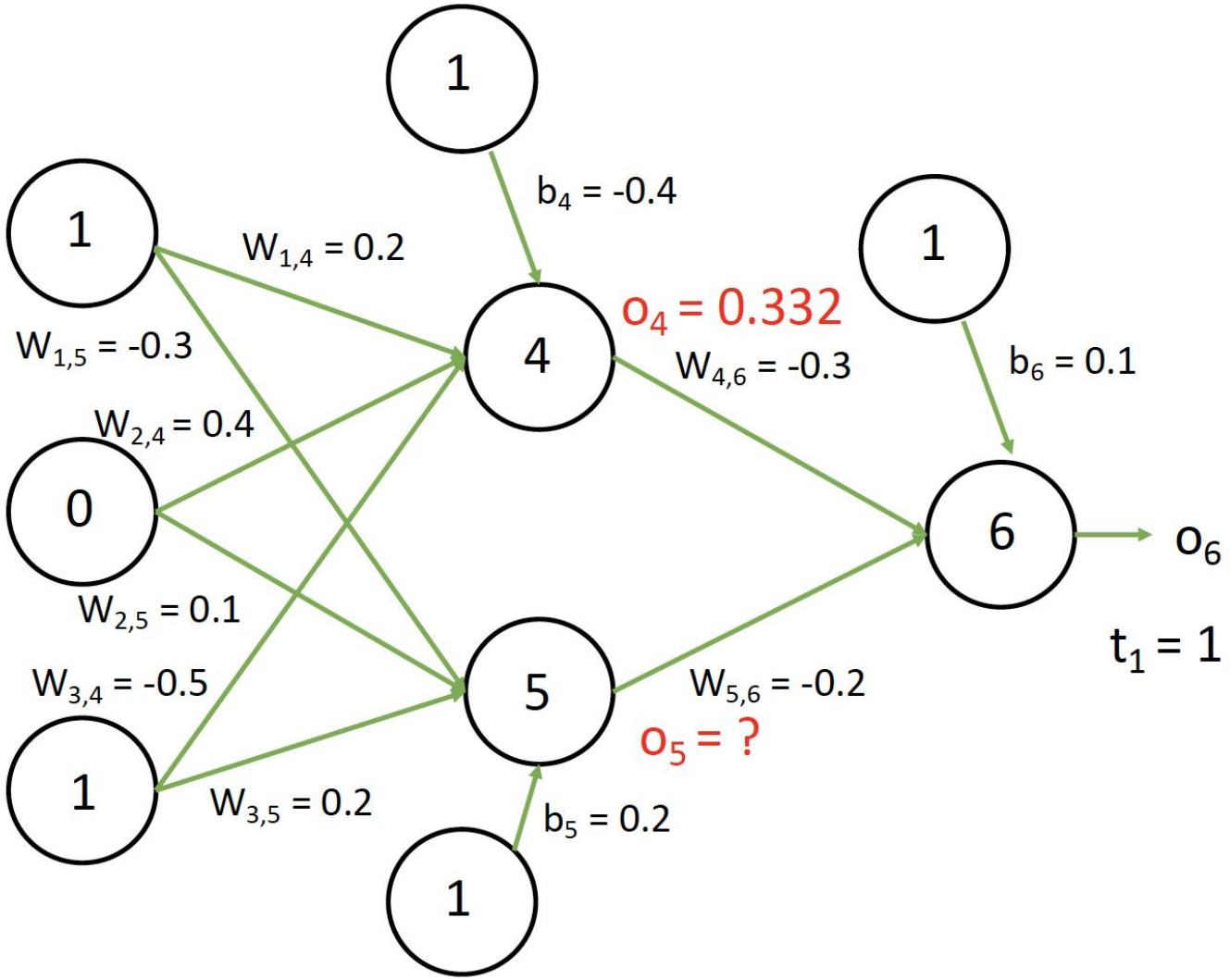
Output of node 4 (sigmoid function):

$o_4 = sigmoid(-0.7)$

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

$o_4 = 0.332$

Neural Network Training: Backpropagation



Input to node 5:

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

$i_5 = 0.1$

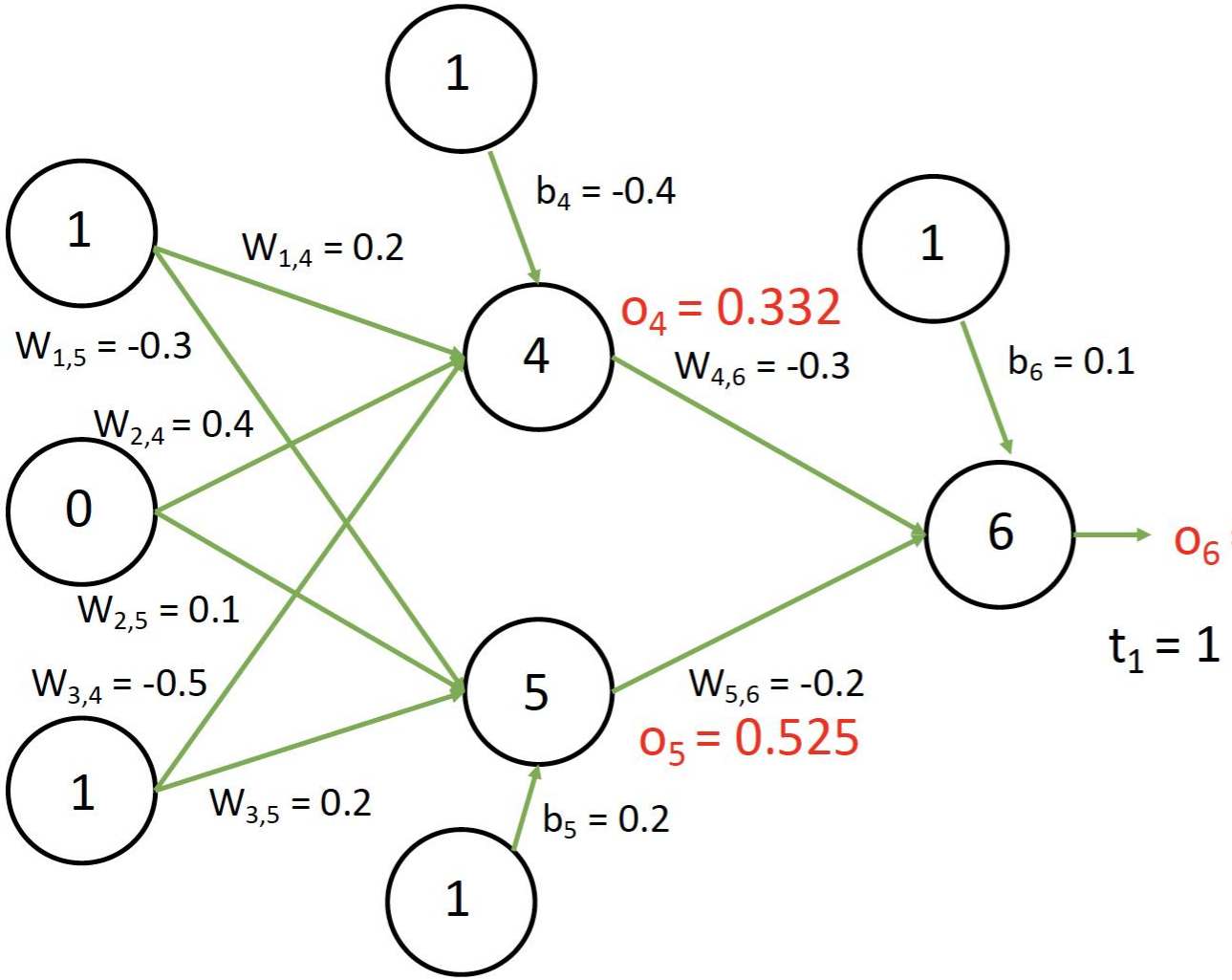
Output of node 5 (sigmoid function):

$o_5 = sigmoid(0.1)$

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

$o_5 = 0.525$

Neural Network Training: Backpropagation



Input to node 6:

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

$i_6 = -0.105$

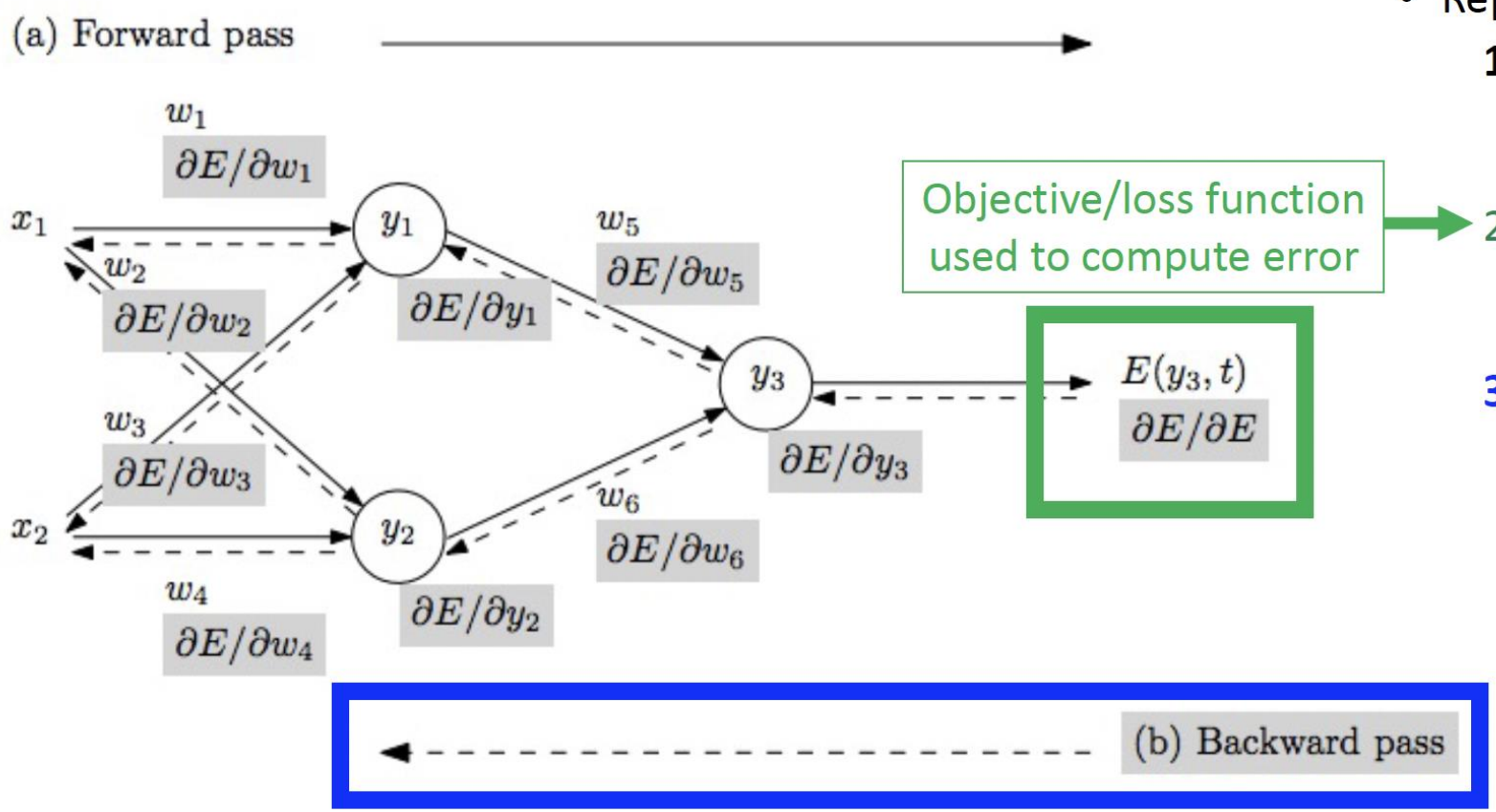
Output of node 6 (sigmoid function):

$o_6 = sigmoid(-0.105)$

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

$o_6 = 0.474$

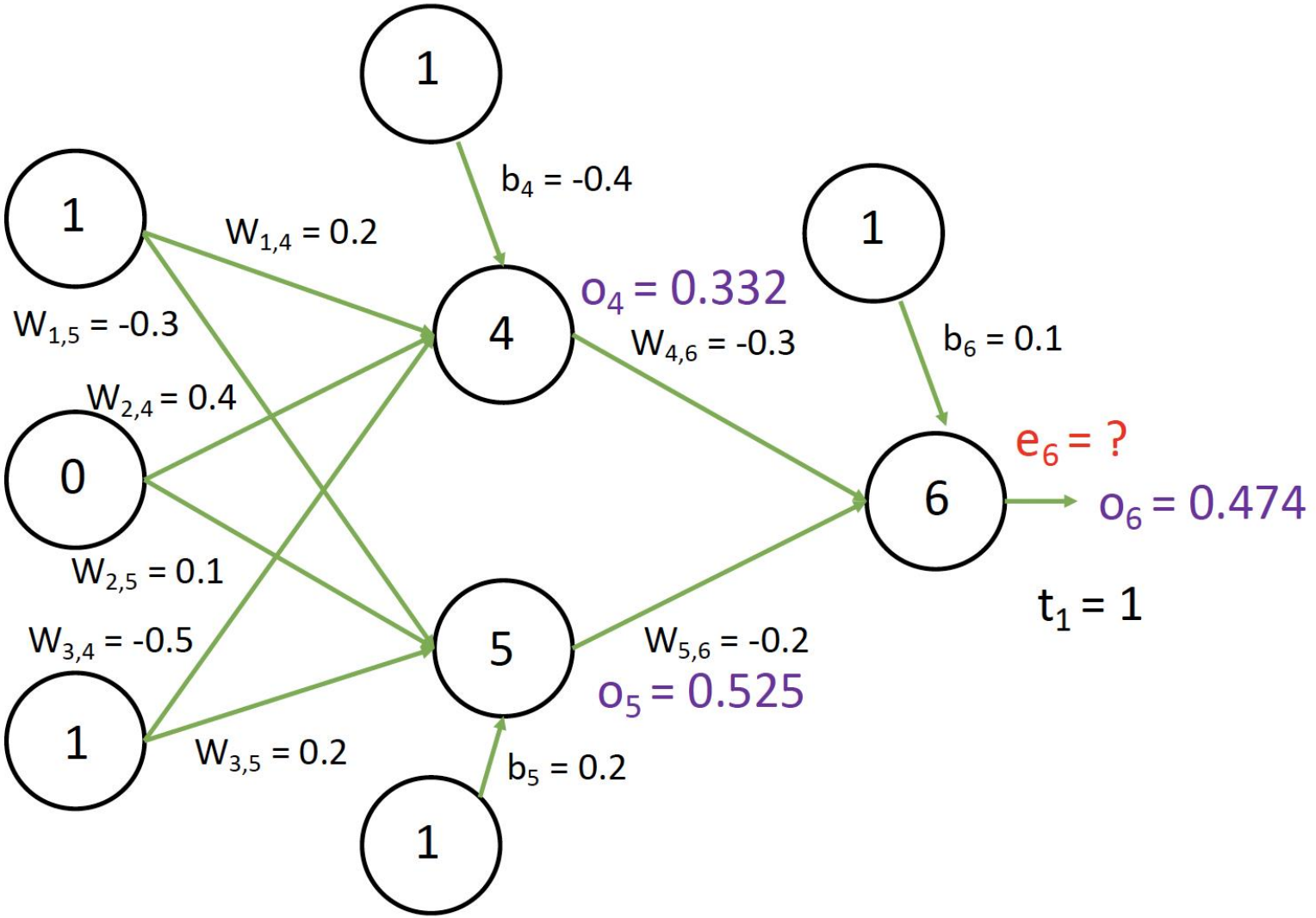
Neural Network Training: Backpropagation



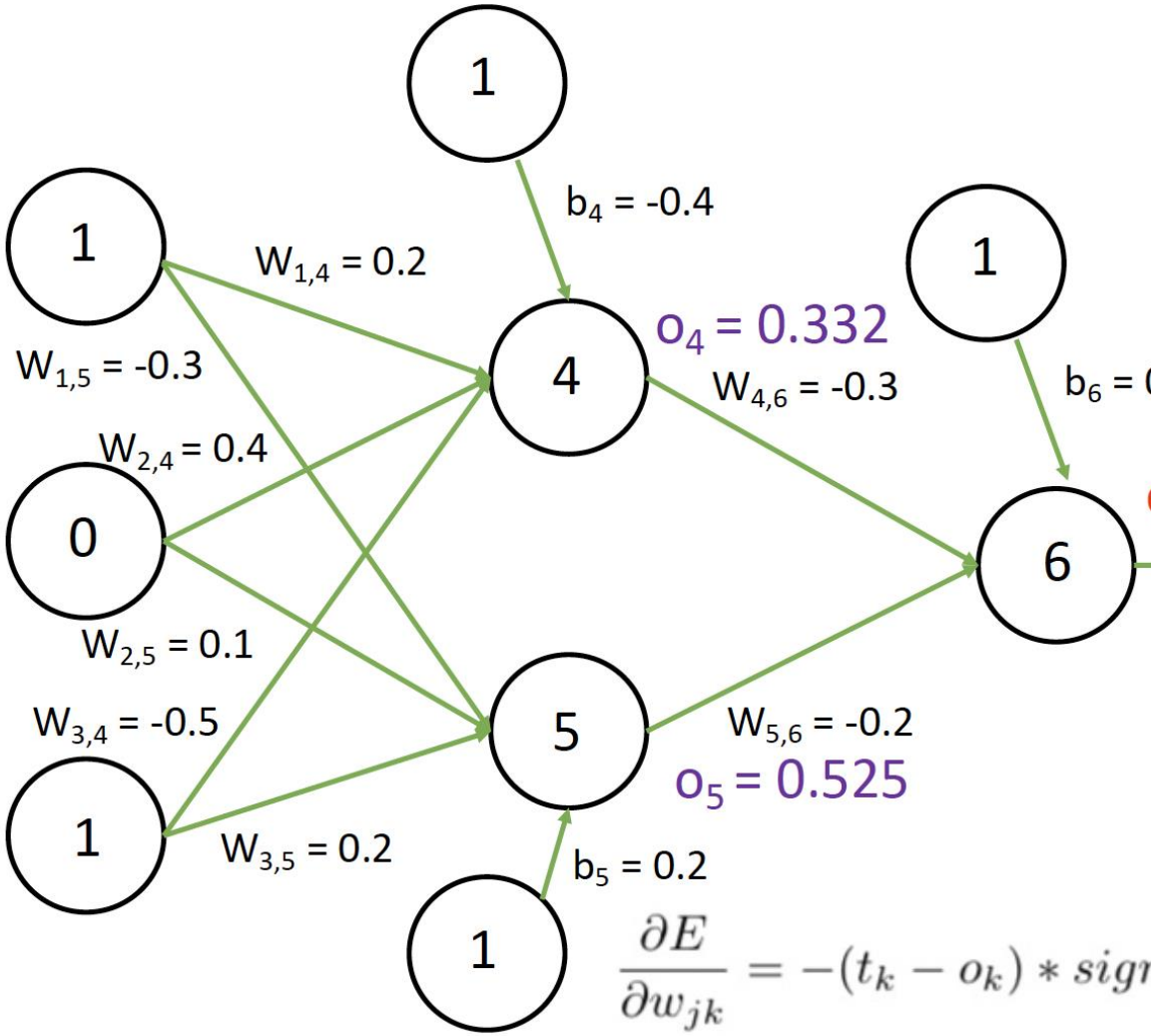
Objective/loss function used to compute error

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction
  2. Quantify the dissatisfaction with a model's results on the training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter

Neural Network Training: Backpropagation



Neural Network Training: Backpropagation



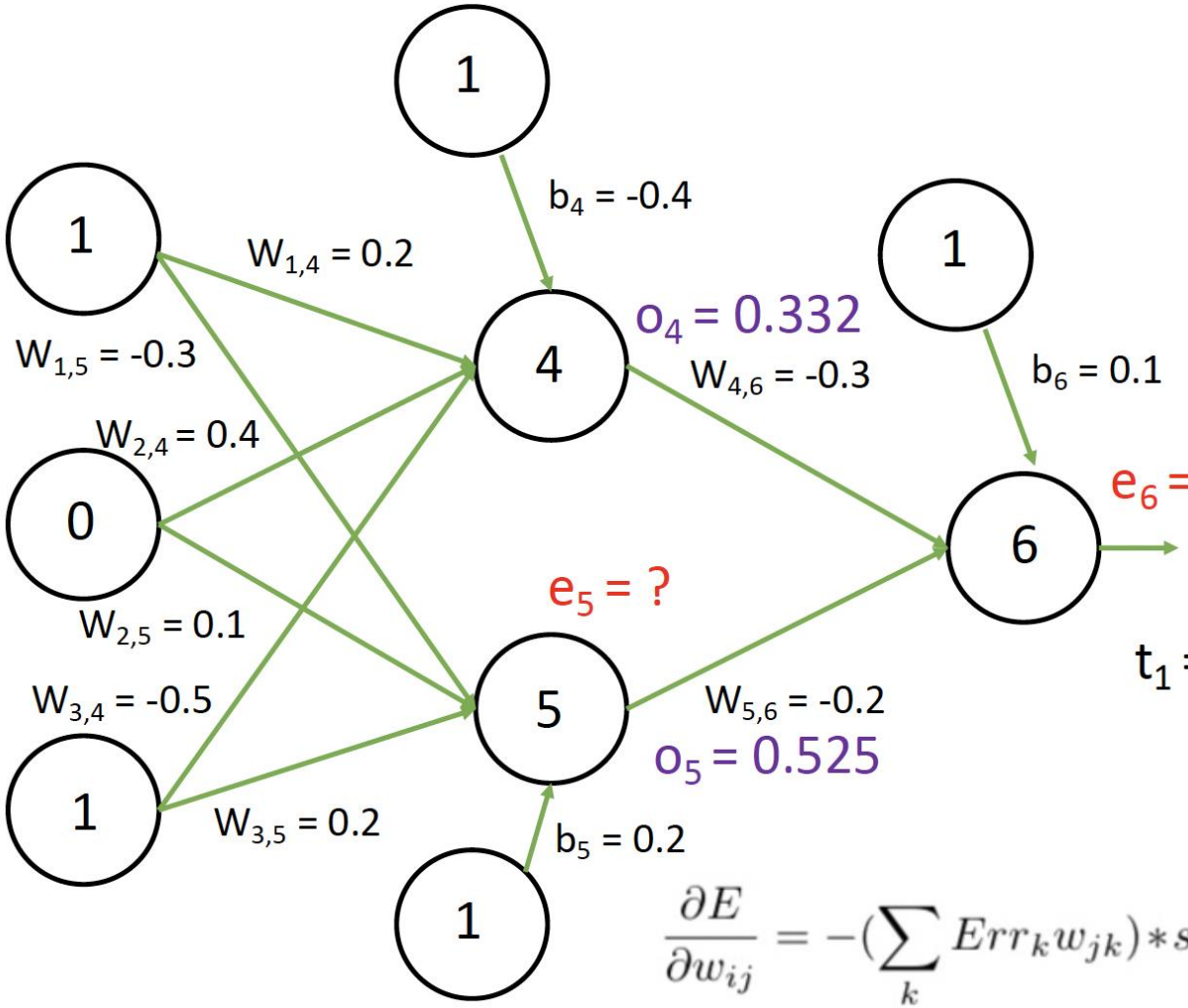
Error at node 6:

$e_6 = (1-0.474)\ (0.474)(1-0.474)$

$e_6 = 0.1311$

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

Neural Network Training: Backpropagation



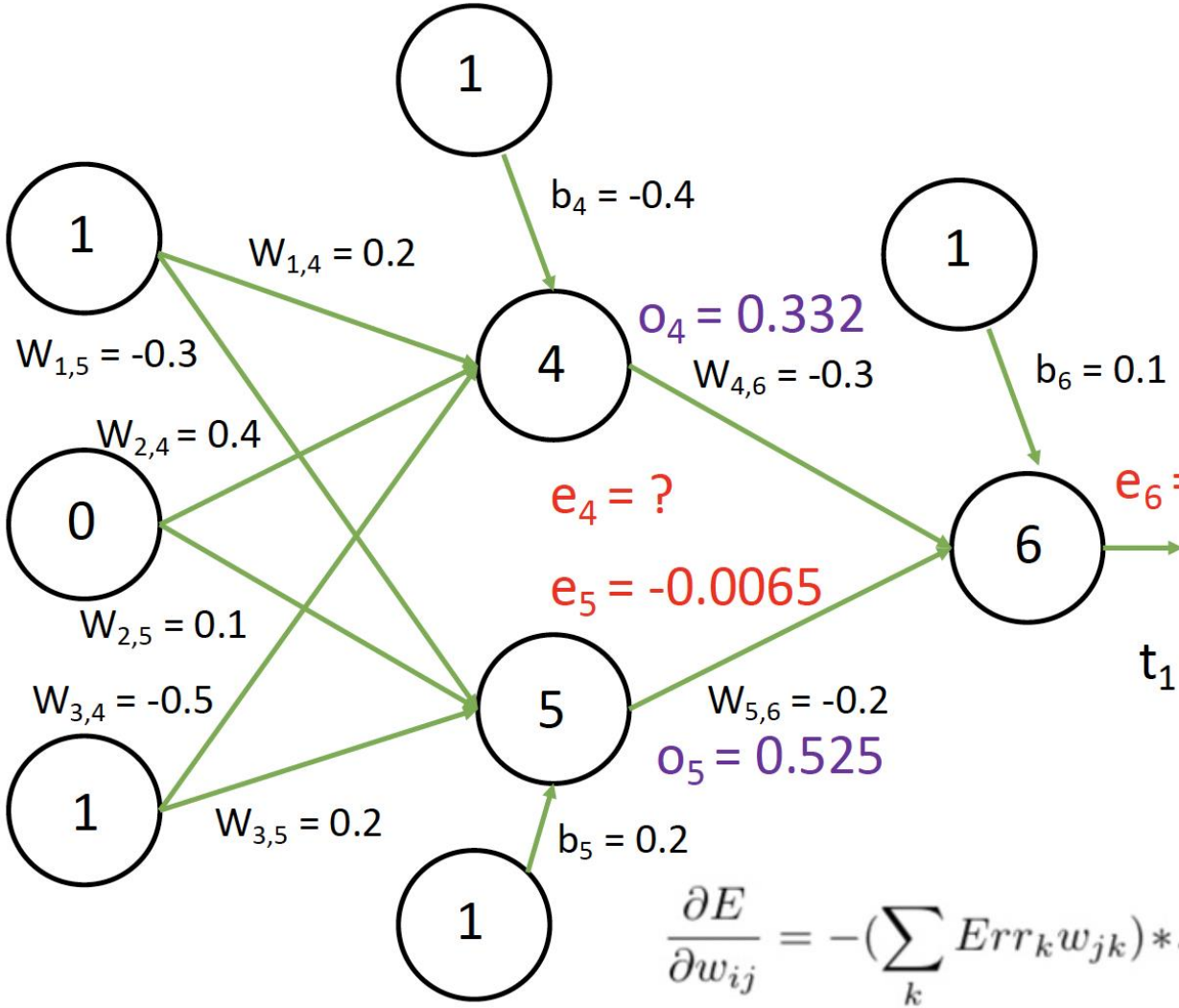
Error at node 5:

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

$e_5 = -0.0065$

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

Neural Network Training: Backpropagation



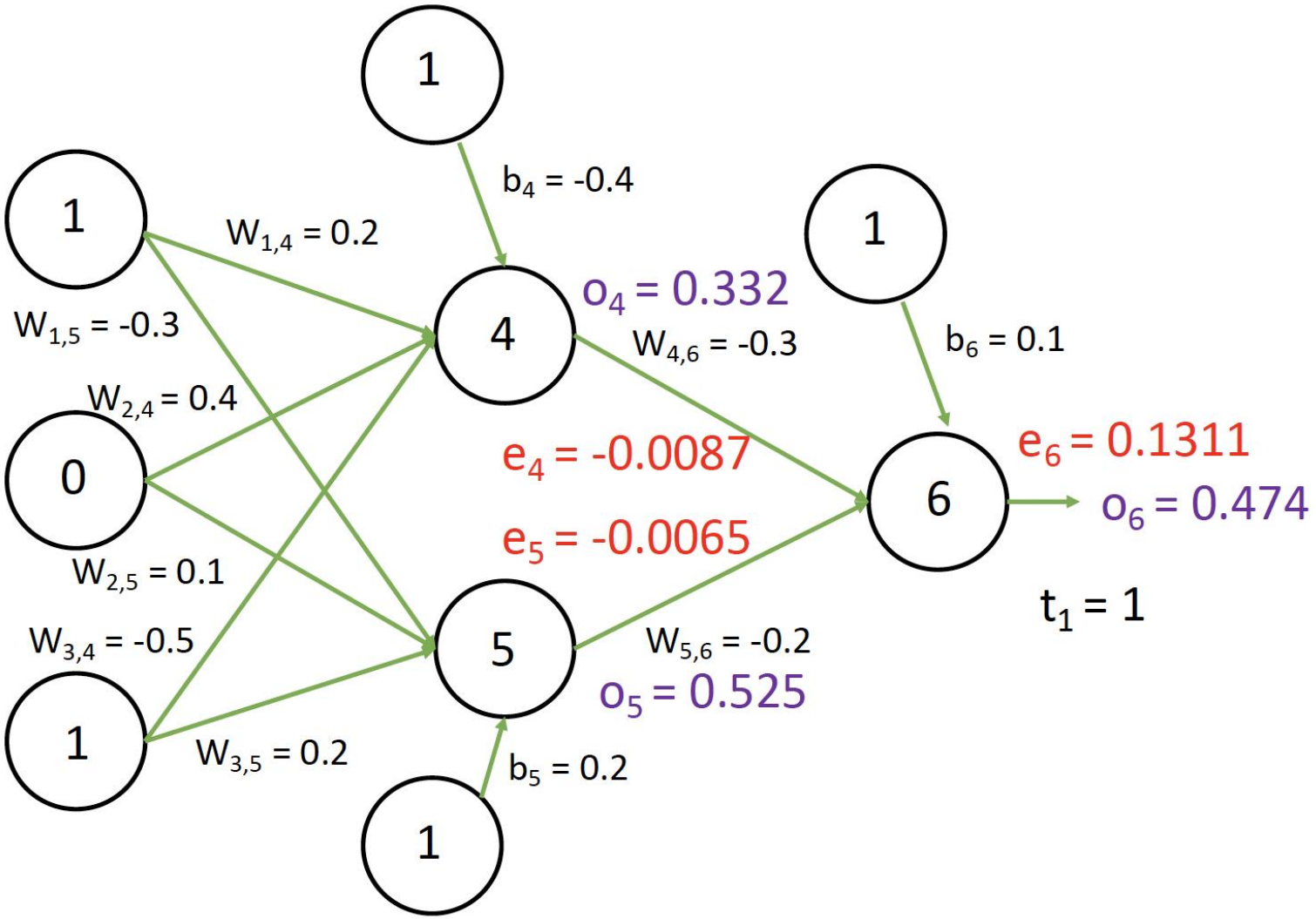
Error at node 4:

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

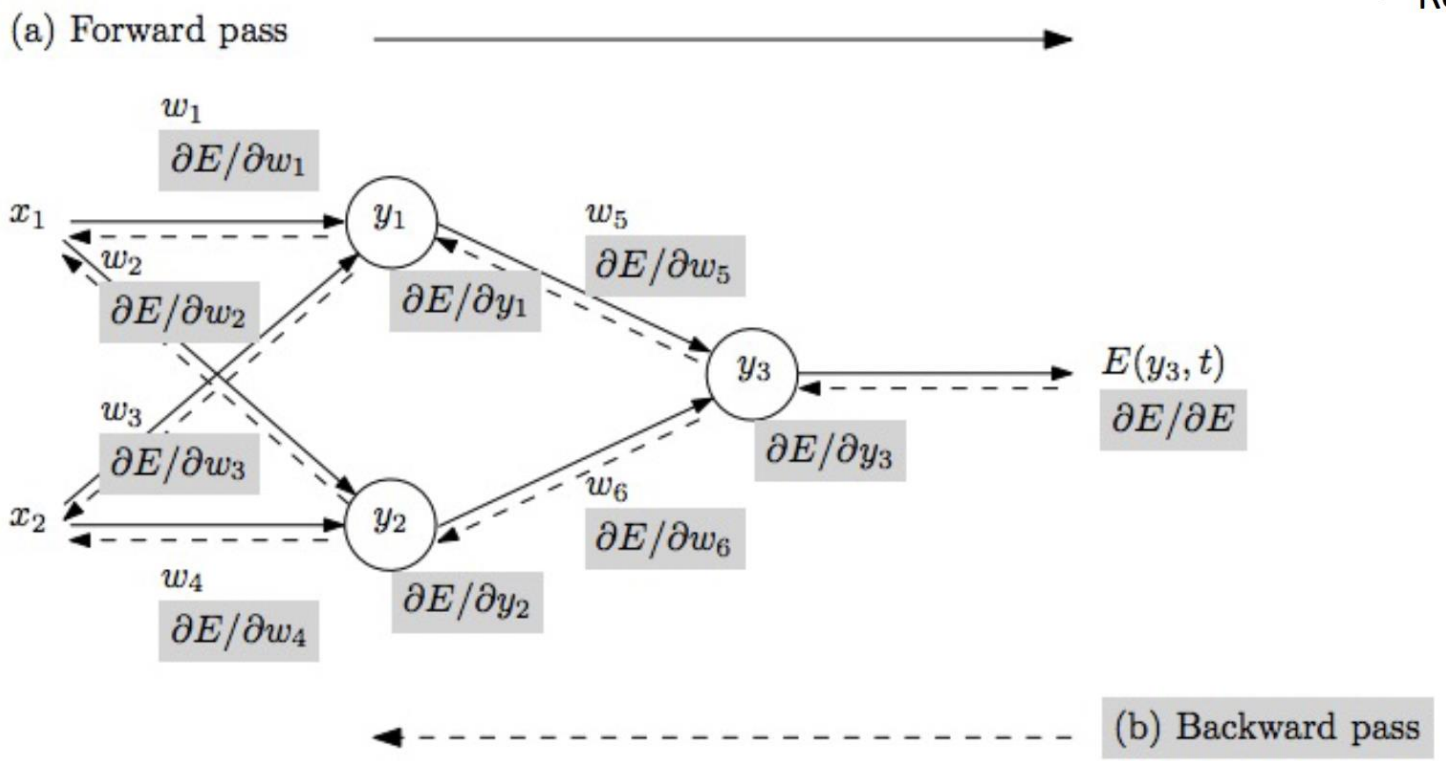
$e_4 = -0.0087$

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

Neural Network Training: Backpropagation

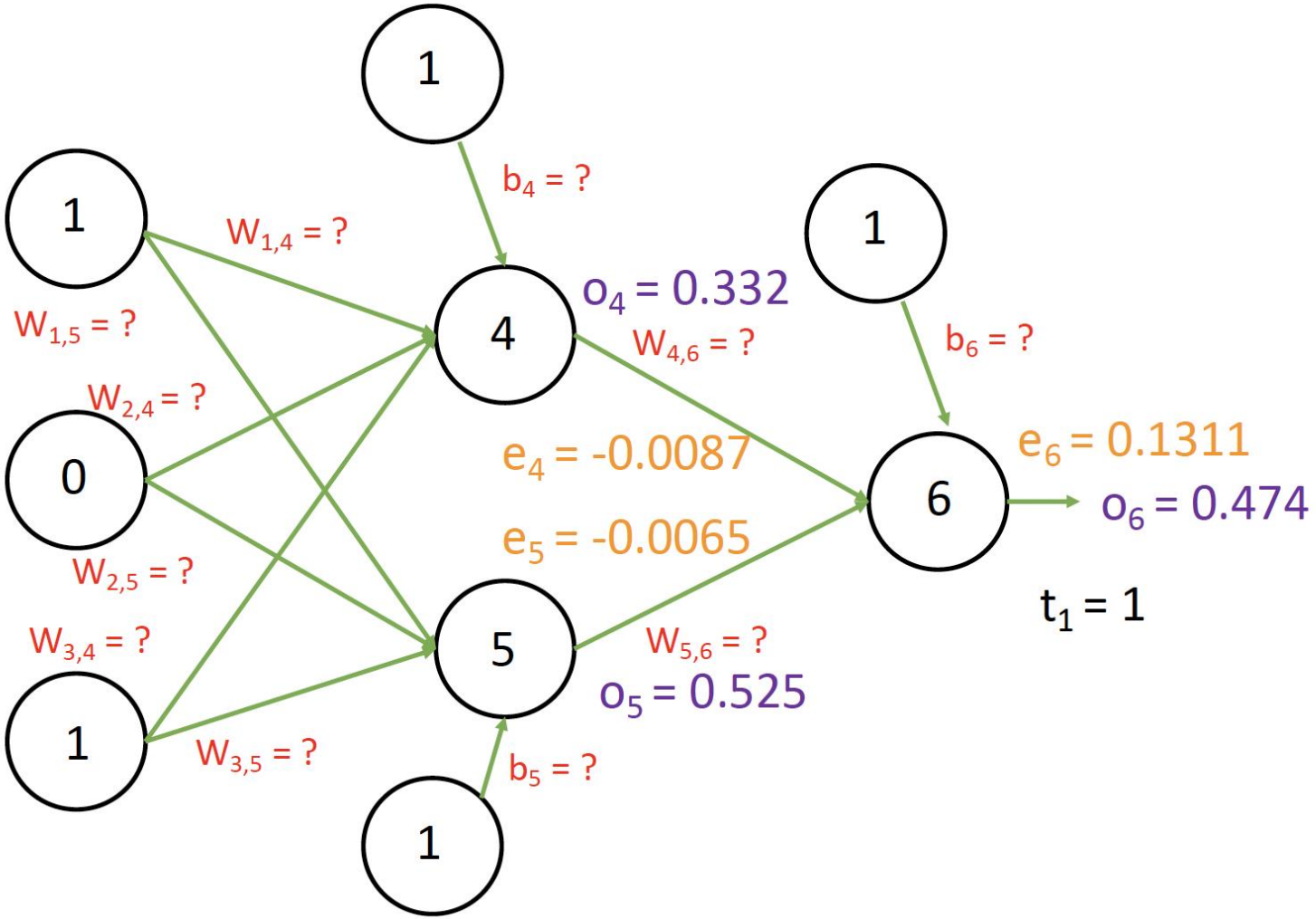


Neural Network Training: Backpropagation

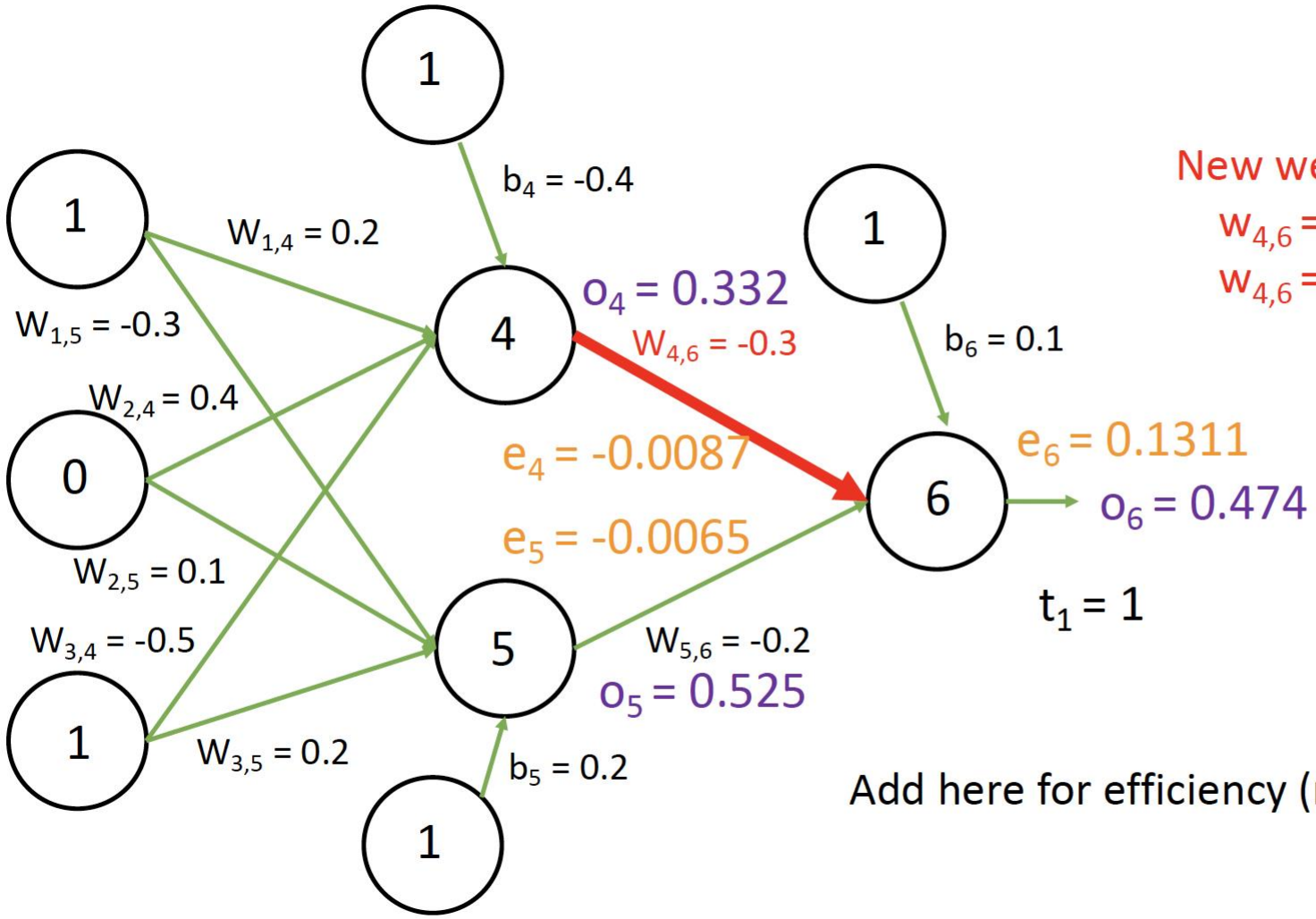


- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction
  2. Quantify the dissatisfaction with a model's results on the training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

Neural Network Training: Backpropagation



Neural Network Training: Backpropagation



New weights (learning rate = 0.9):

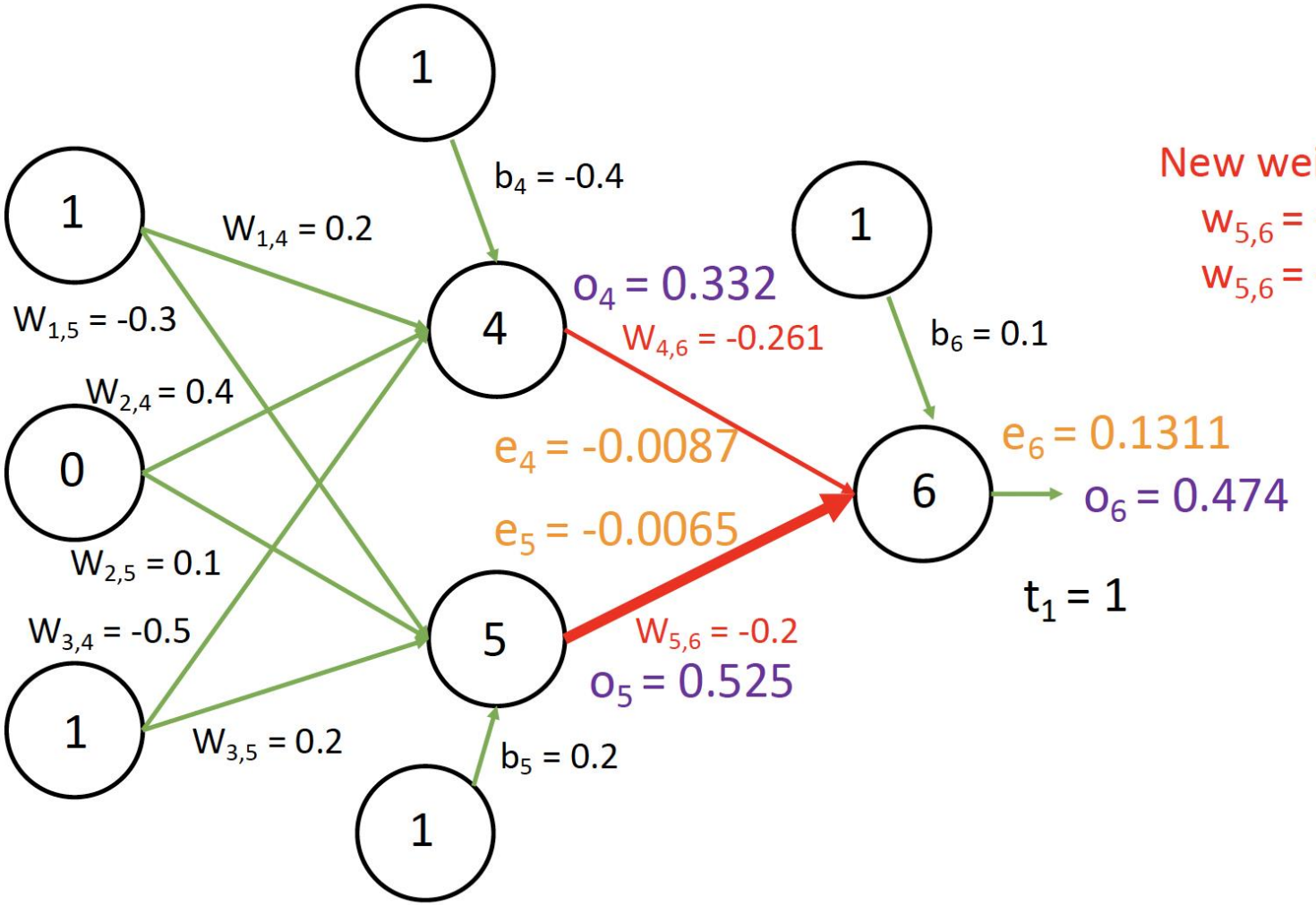
$w_{4,6} = -0.3 + (0.9)(0.1311)(0.332)$

$w_{4,6} = -0.261$

Add here for efficiency (removed from earlier equation)

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

Neural Network Training: Backpropagation



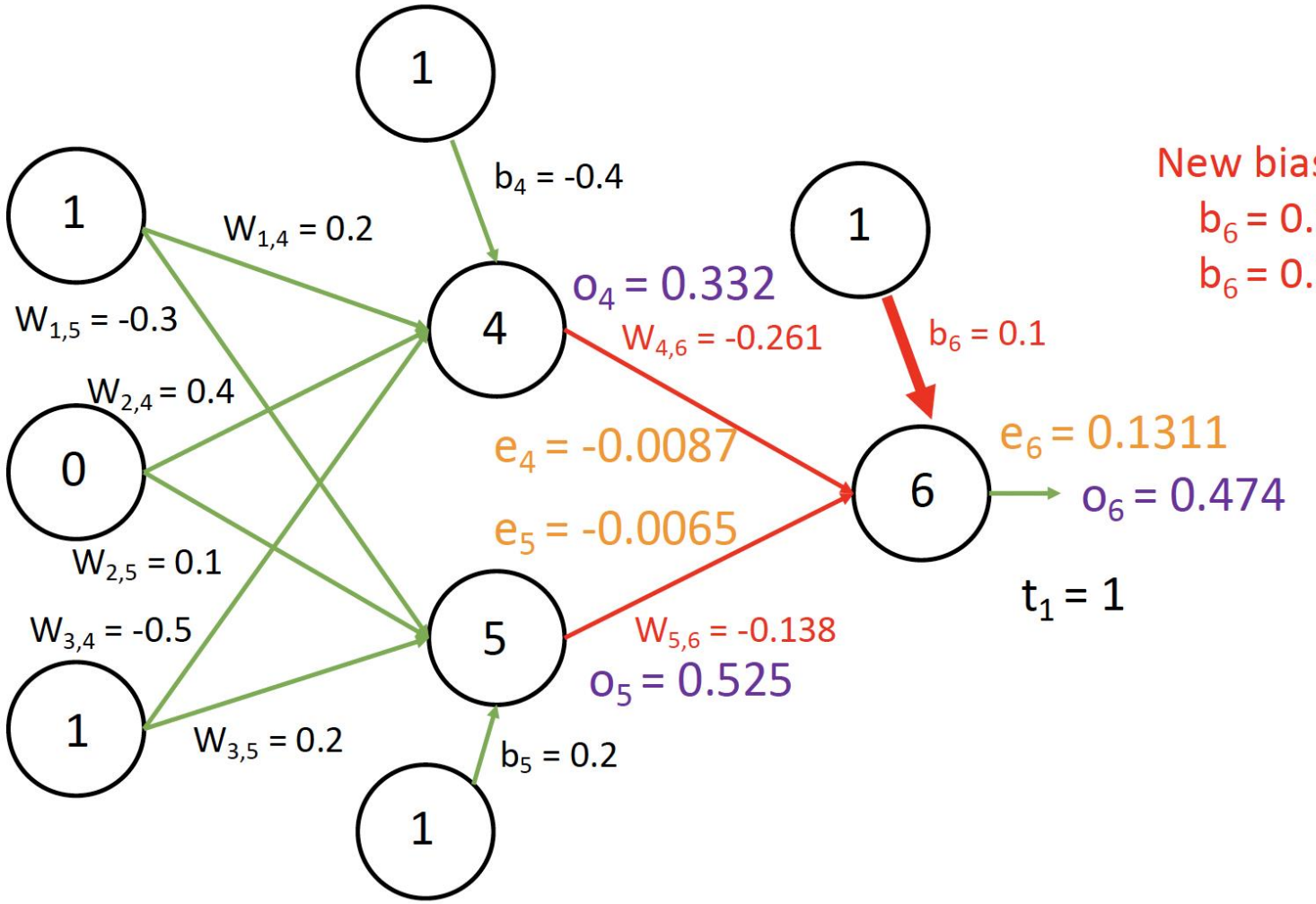
New weights (learning rate = 0.9):

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

$w_{5,6}$ = -0.138

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

Neural Network Training: Backpropagation



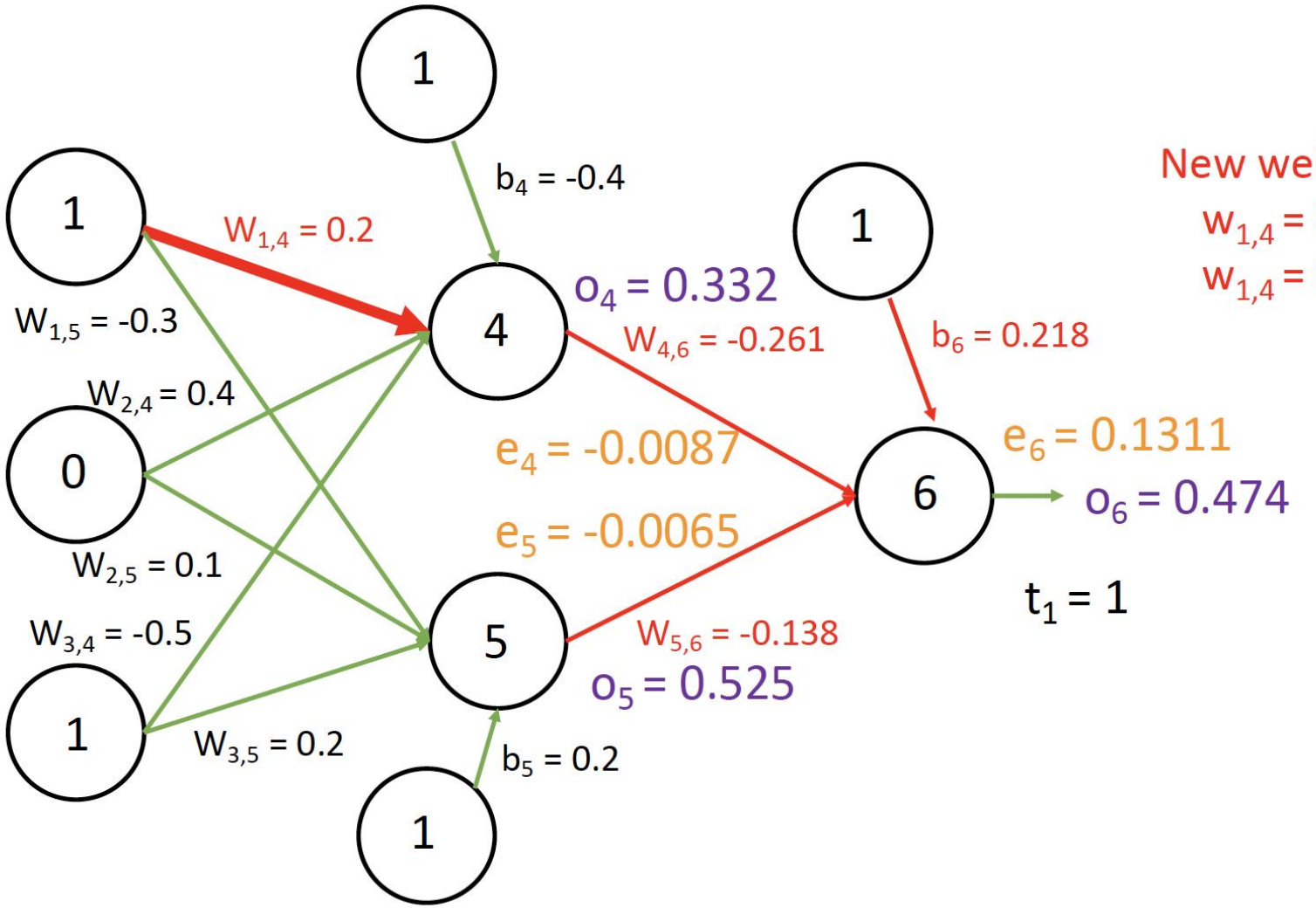
New bias (learning rate = 0.9):

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

$b_6 = 0.218$

$$b_k = b_k + \eta Err_k$$

Neural Network Training: Backpropagation



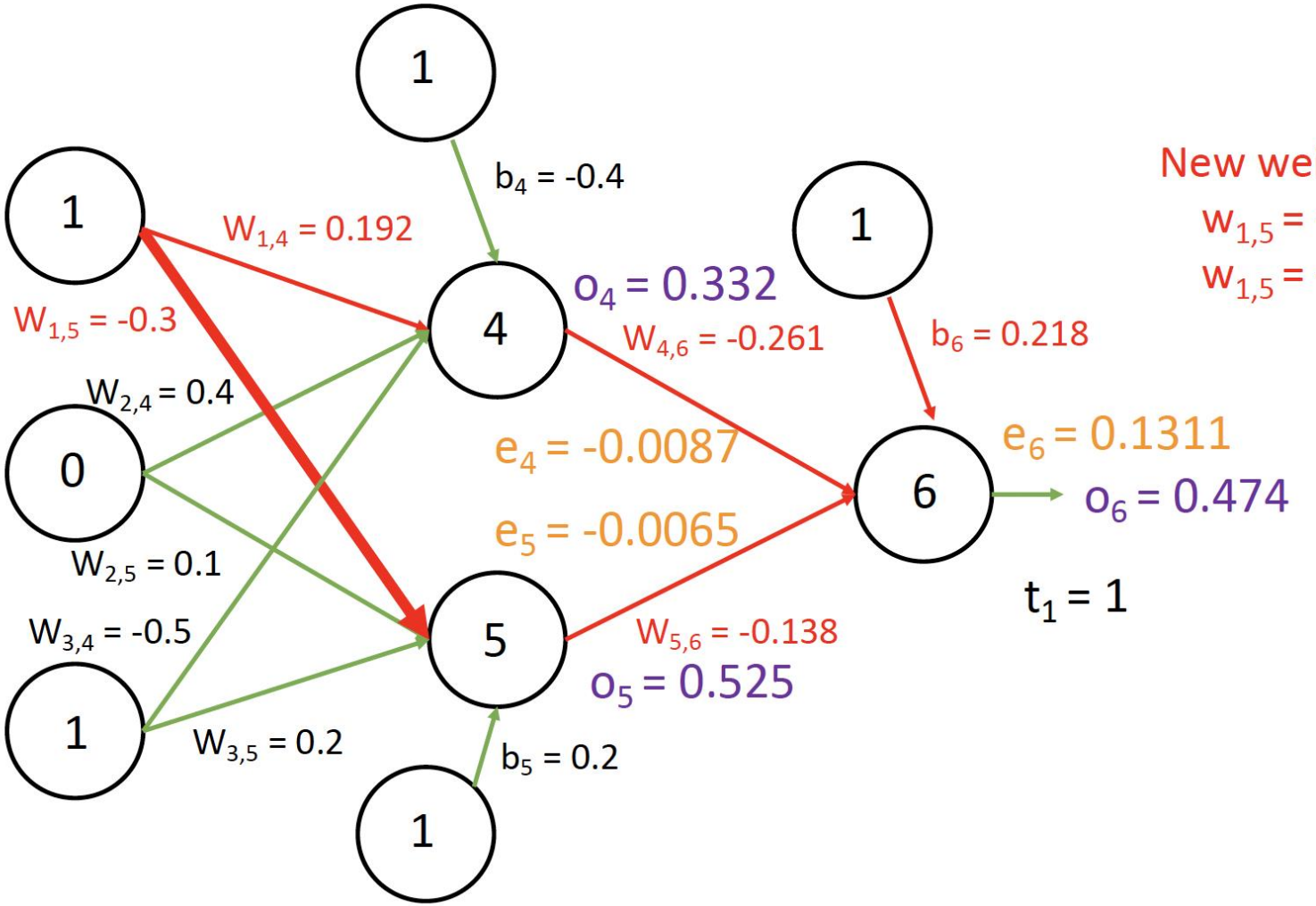
New weights (learning rate = 0.9):

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

$w_{1,4} = 0.192$

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

Neural Network Training: Backpropagation



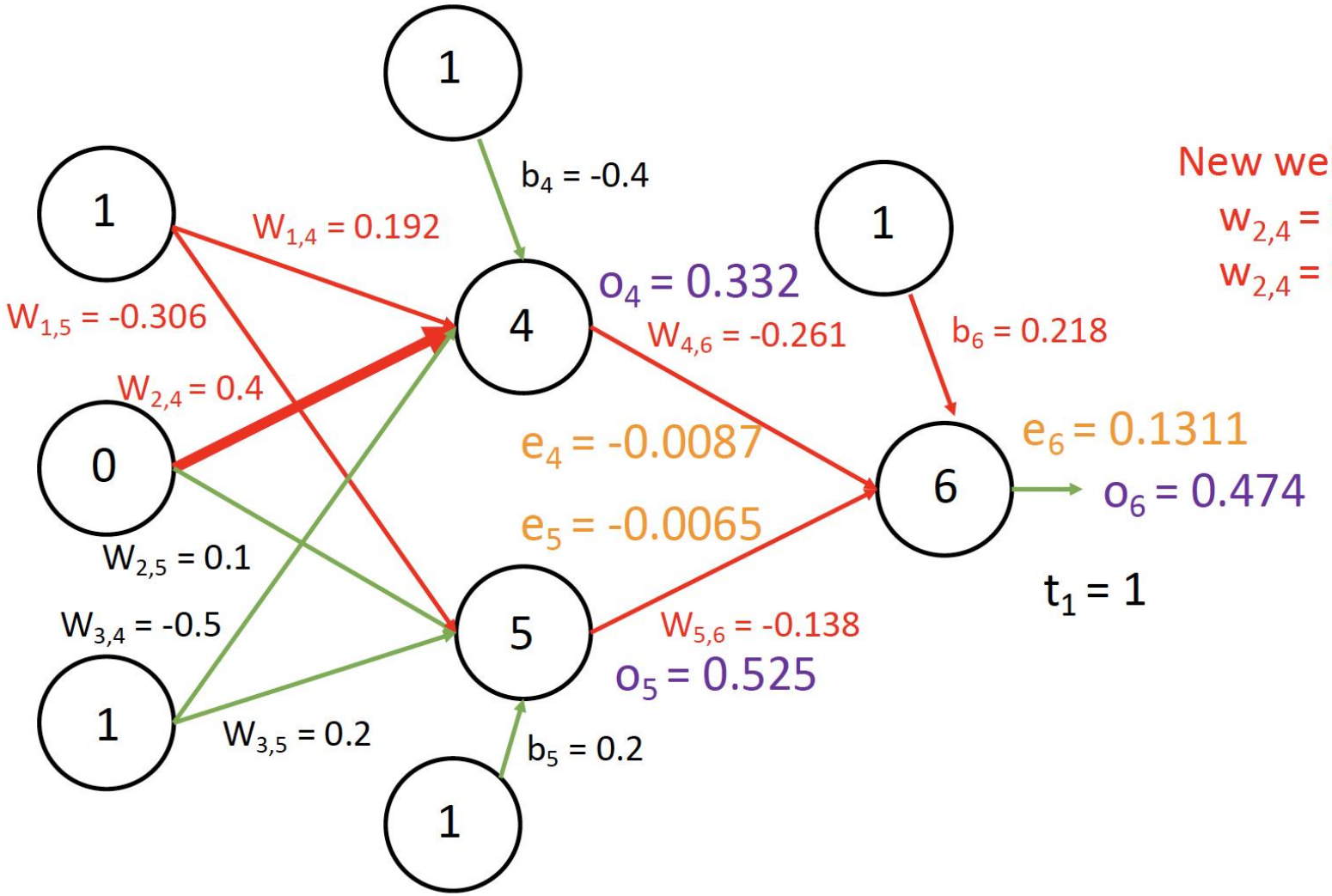
New weights (learning rate = 0.9):

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

$w_{1,5}$ = -0.306

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

Neural Network Training: Backpropagation



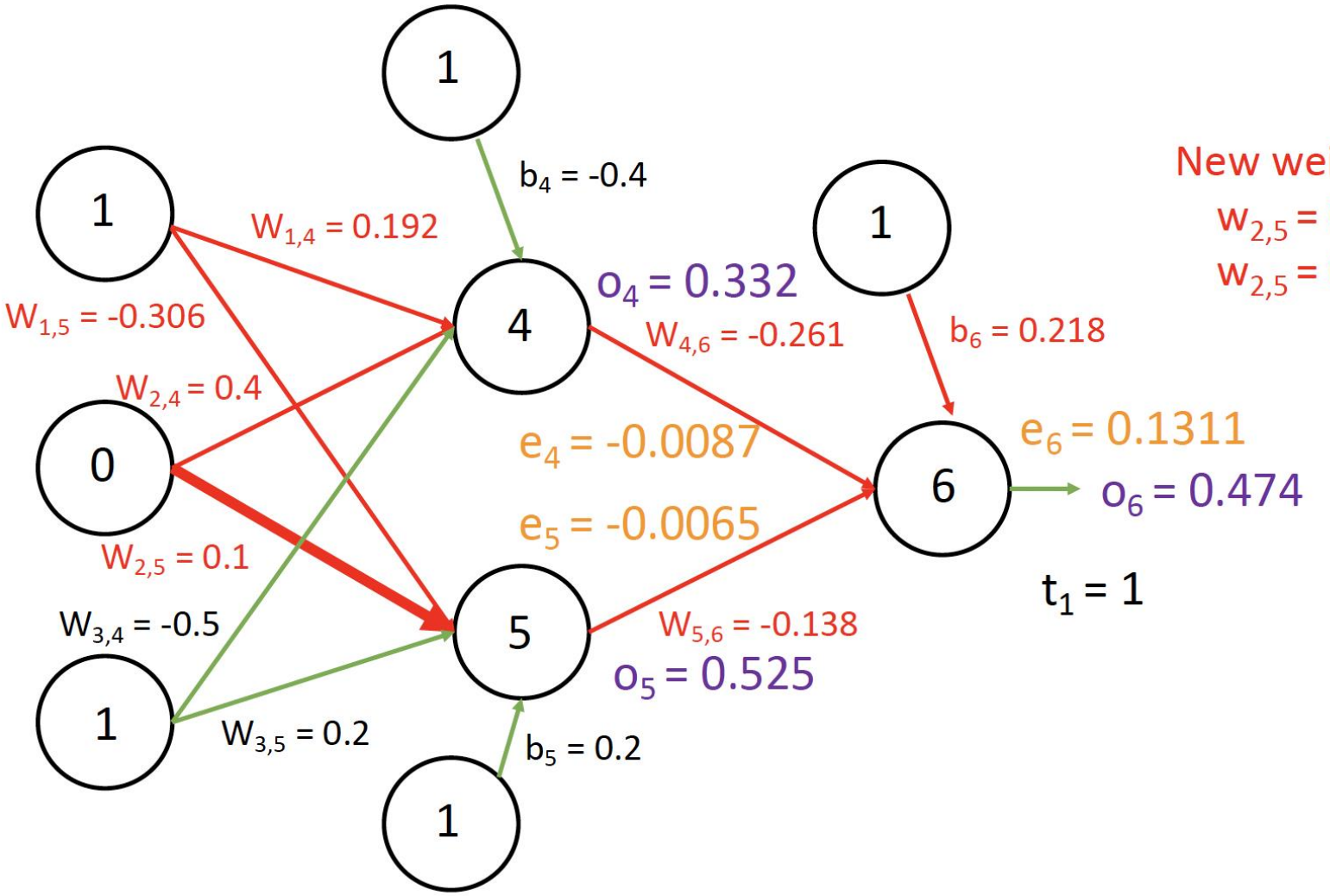
New weights (learning rate = 0.9):

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

$w_{2,4} = 0.4$

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

Neural Network Training: Backpropagation



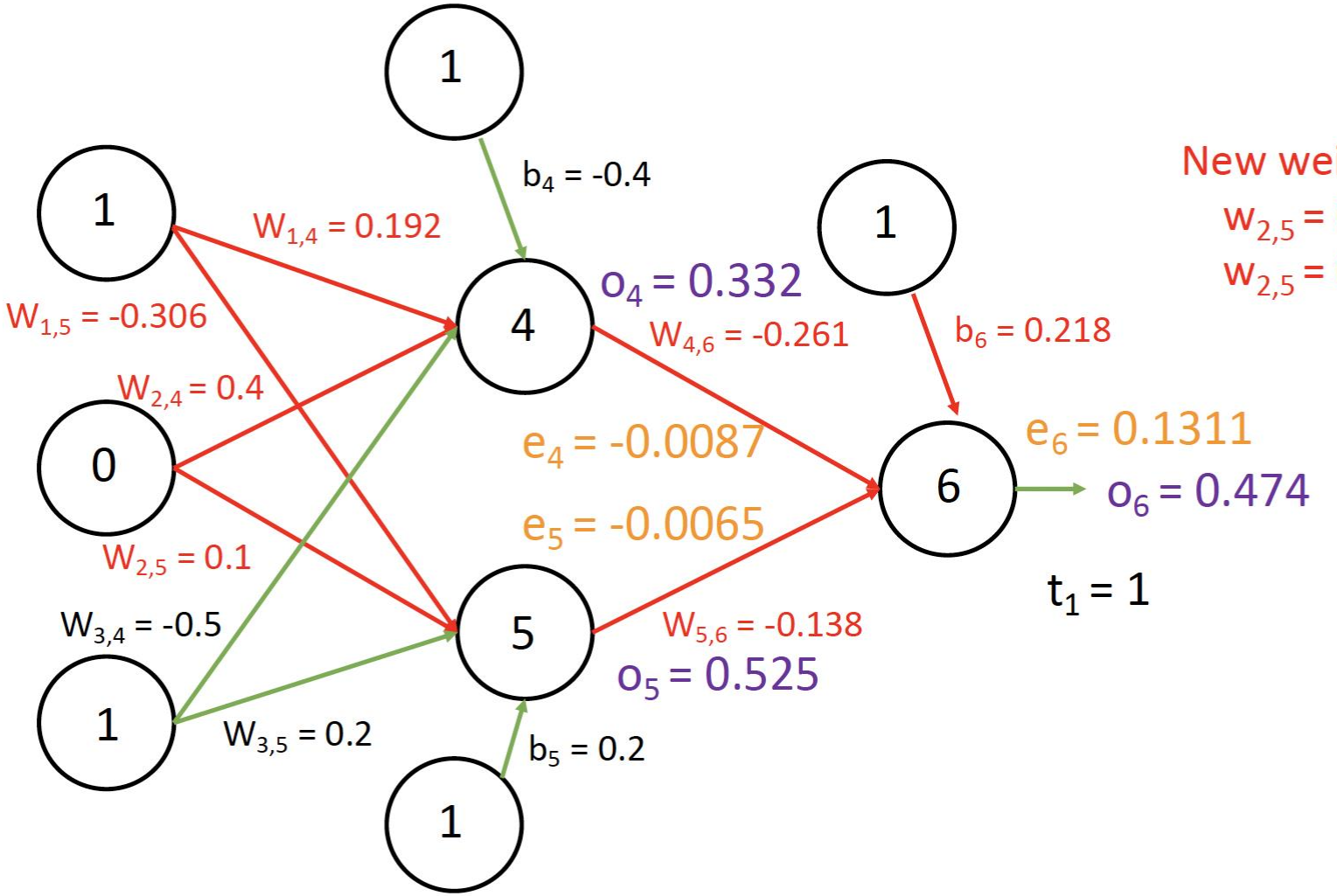
New weights (learning rate = 0.9):

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

$w_{2,5} = 0.1$

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

Neural Network Training: Backpropagation



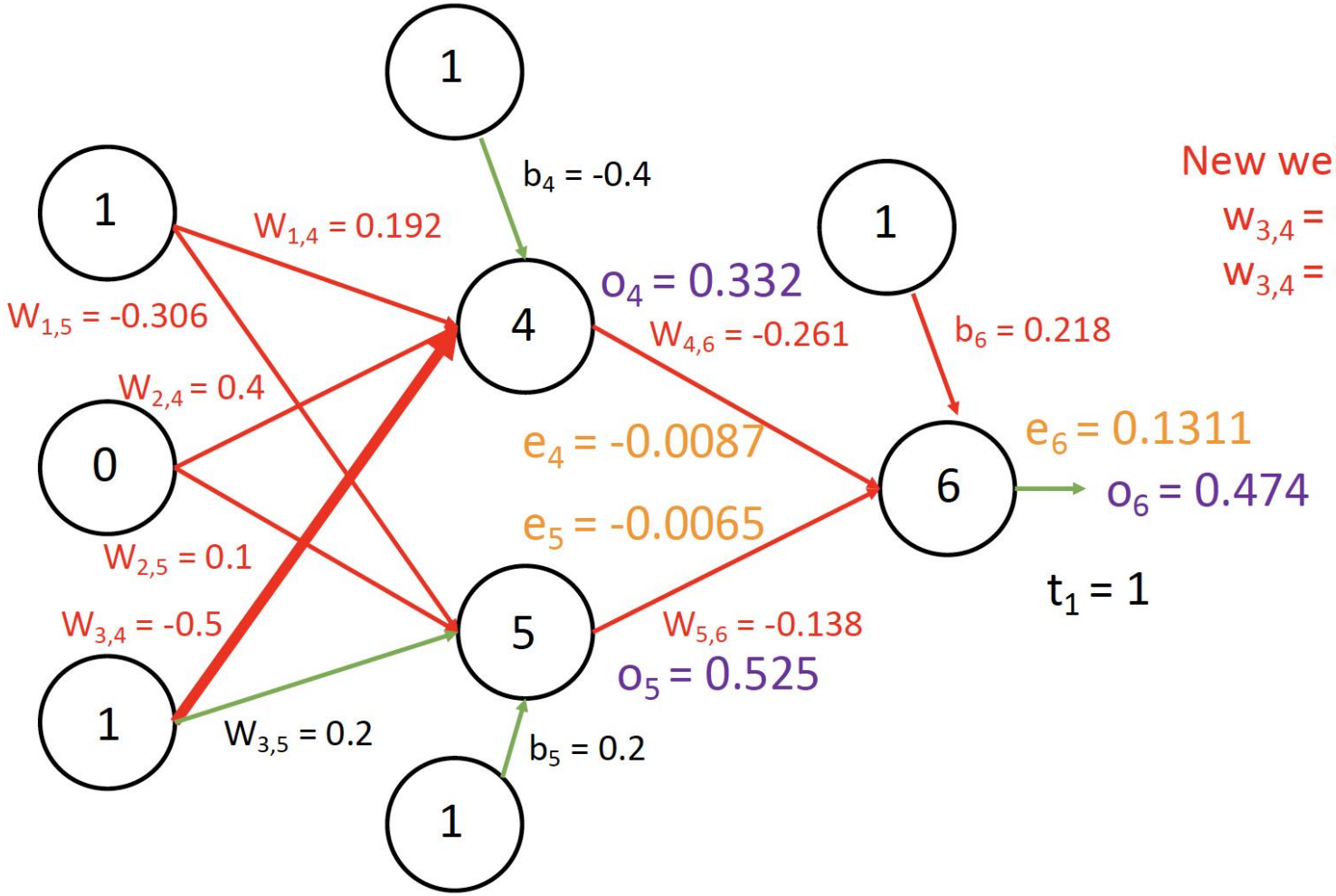
New weights (learning rate = 0.9):

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

$w_{2,5} = 0.1$

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

Neural Network Training: Backpropagation



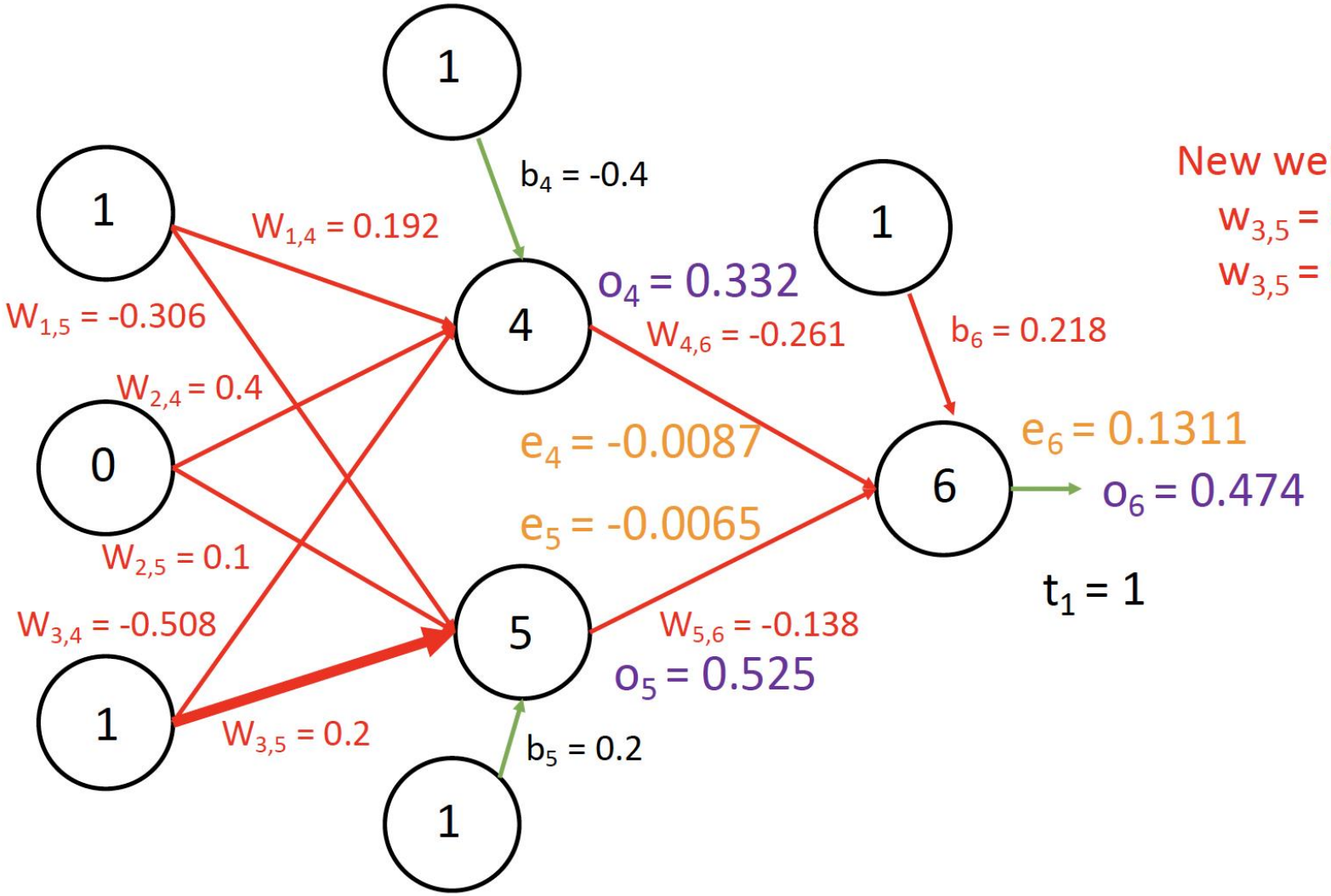
New weights (learning rate = 0.9):

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

$w_{3,4}$ = -0.508

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

Neural Network Training: Backpropagation



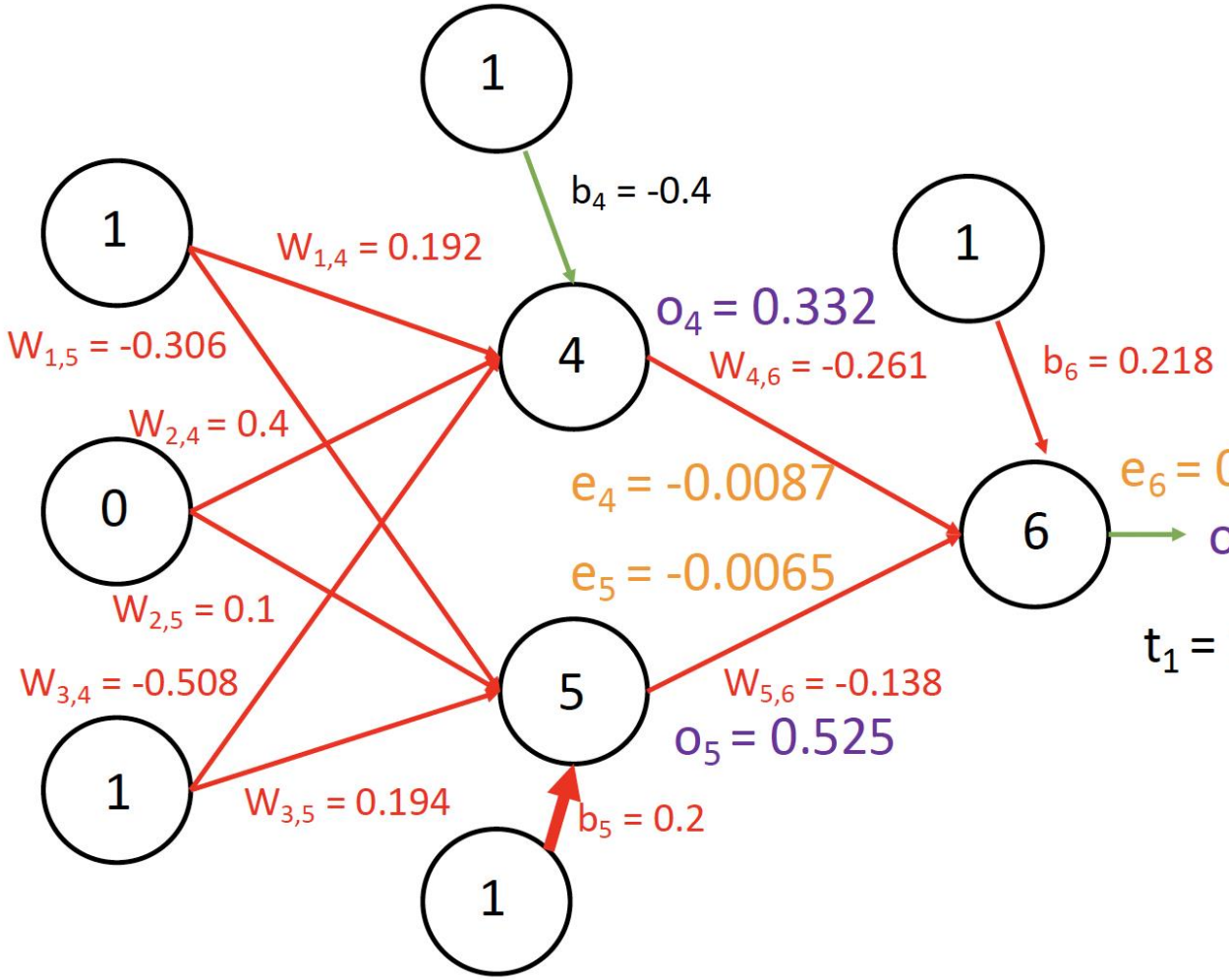
New weights (learning rate = 0.9):

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

$w_{3,5} = 01.94$

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

Neural Network Training: Backpropagation



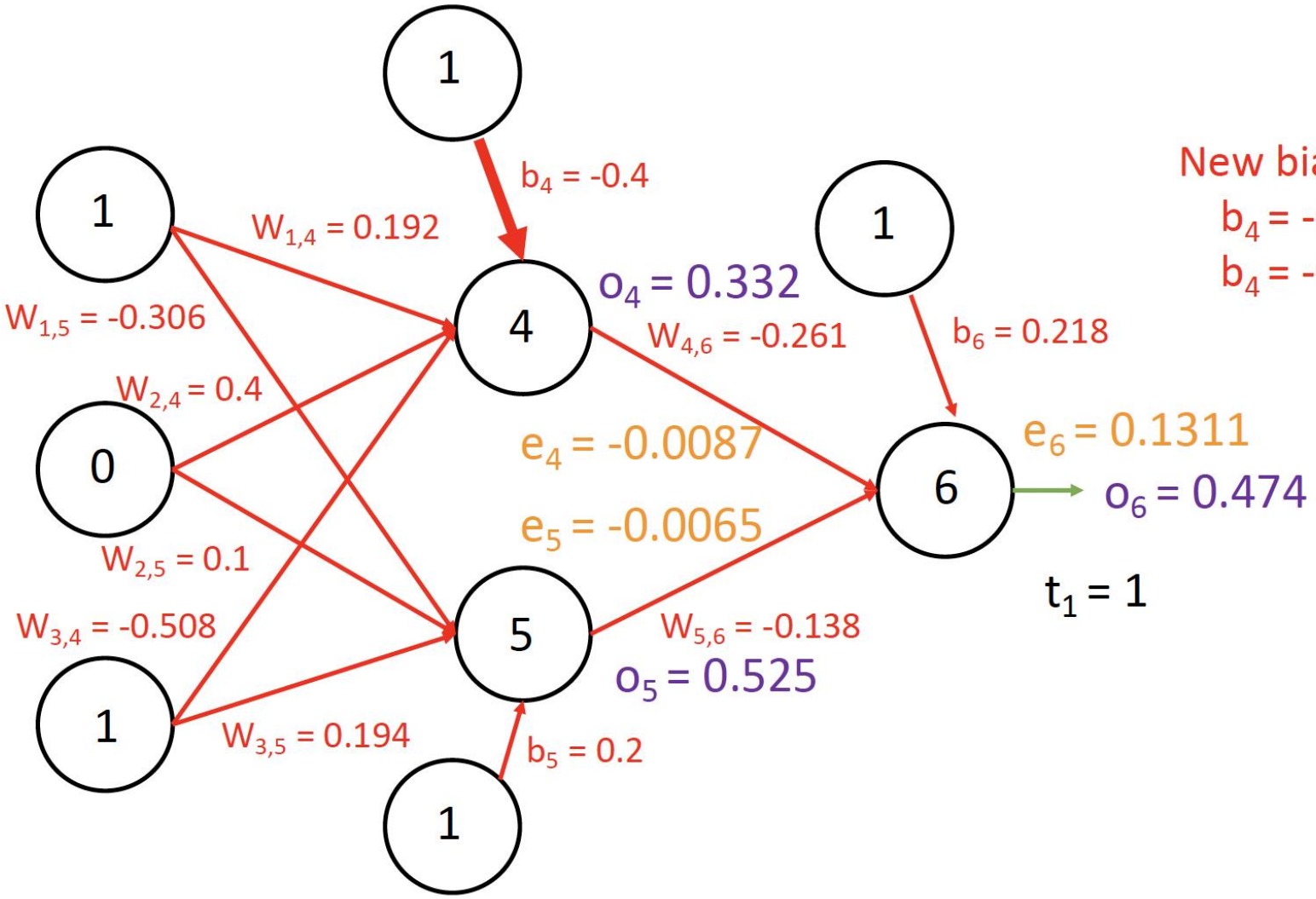
New bias (learning rate = 0.9):

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

$b_5 = 0.194$

$$b_k = b_k + \eta Err_k$$

Neural Network Training: Backpropagation



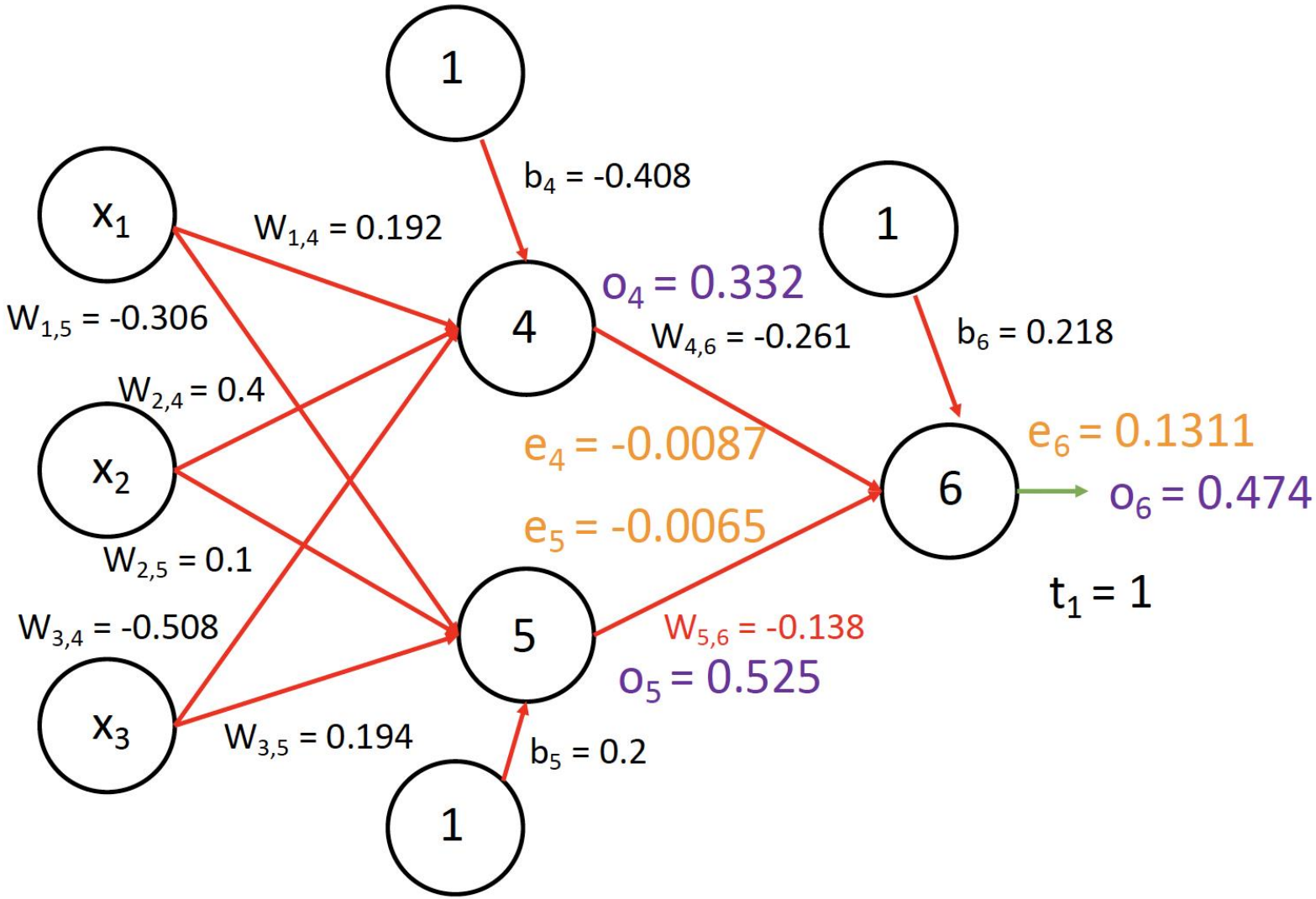
New bias (learning rate = 0.9):

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

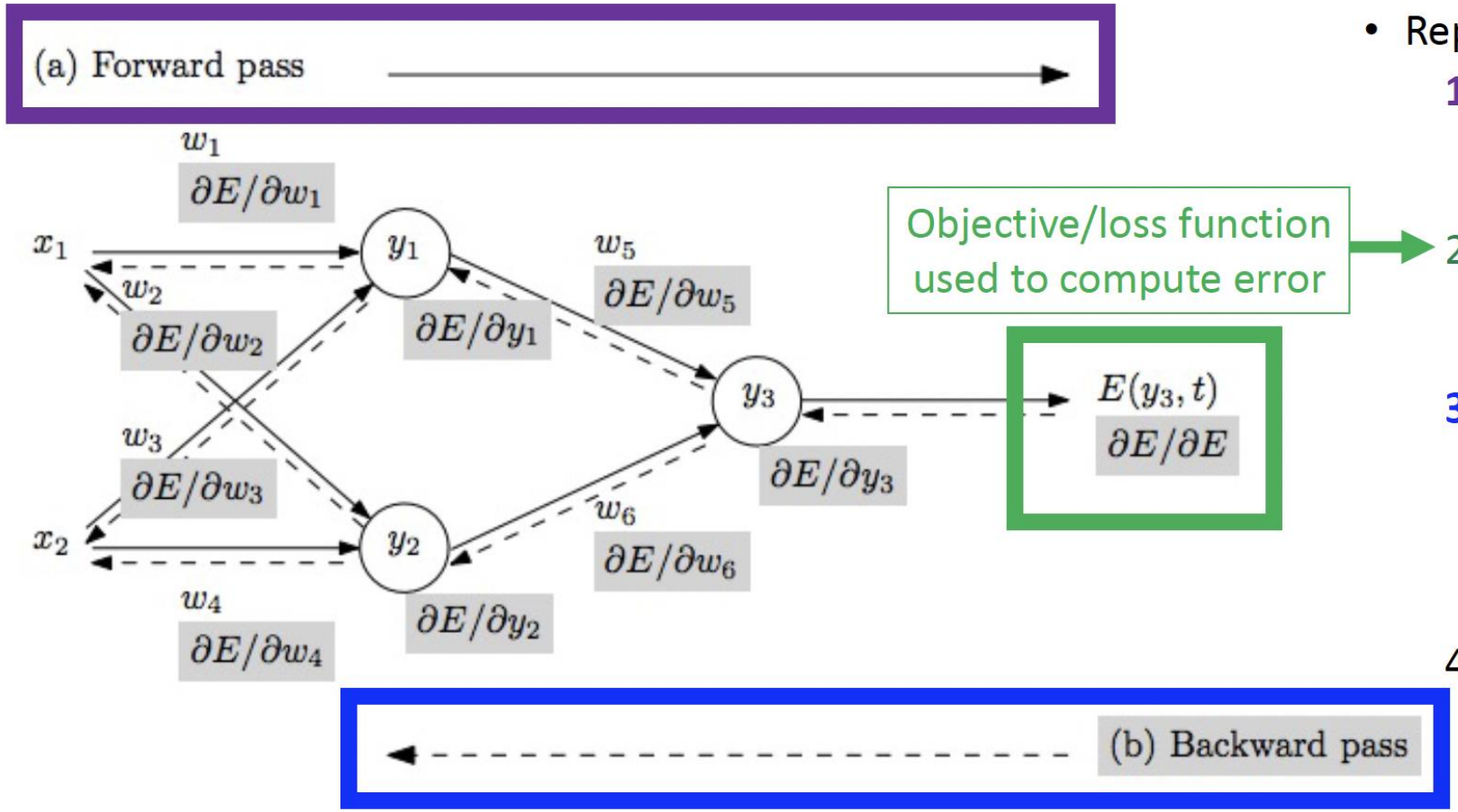
$b_4 = -0.408$

$$b_k = b_k + \eta Err_k$$

Neural Network Training: Backpropagation



Neural Network Training: Backpropagation



Objective/loss function used to compute error

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction
  2. Quantify the dissatisfaction with a model's results on the training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

Neural Network Training: Backpropagation

What type of gradient descent was used in the toy example?

a. Batch gradient descent
b. Stochastic gradient descent
c. Mini-batch gradient descent

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make prediction
  2. Quantify the dissatisfaction with a model's results on the training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

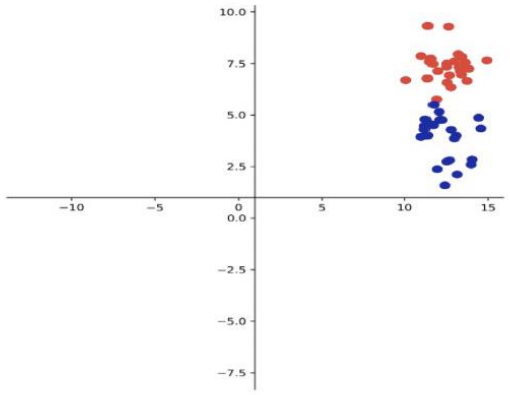
Initialization and Normalization Techniques

Initialization means choosing the starting values of the weights and biases of a neural network before training begins. Good initialization helps the model learn faster and avoids unstable training.

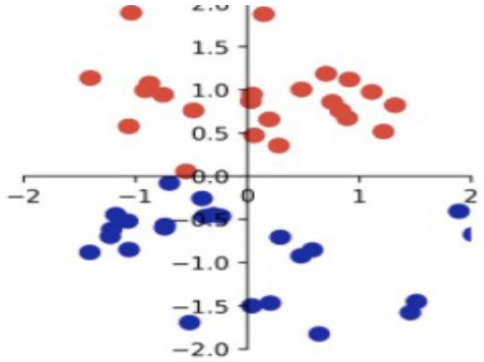
Basic data initialization approach:

- standardize so mean is 0 and standard deviation 1
- simplifies learning

Original data:



Standardized data:



Basic model parameter initialization:

- set weights to random values drawn from Gaussian or uniform distribution
- set biases to 0

Initialization and Normalization Techniques

Normalization techniques are used to make training faster, more stable, and often more accurate. They reduce internal distribution changes during training.

Technique	Purpose	When Used	Best For
Zero Initialization	Simple starting point	Before training	Not suitable for hidden weights
Random Initialization	Break symmetry	Before training	Basic models
Xavier Initialization	Stable variance	Before training	Sigmoid, tanh
He Initialization	Better for ReLU	Before training	ReLU networks
Data Normalization	Scale inputs	Before training	All models
Batch Normalization	Normalize batch activations	During training	CNNs, deep networks
Layer Normalization	Normalize features per sample	During training	RNNs, transformers
Group Normalization	Normalize groups of channels	During training	Small-batch CNNs

Vanishing and Exploding Gradients

In deep networks, gradients are computed using the **chain rule**:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial W_n} \cdot \frac{\partial W_n}{\partial W_{n-1}} \cdot \ldots \cdot \frac{\partial W_2}{\partial W_1}$$

This means gradients are **multiplied layer by layer**.

Vanishing Gradient : When gradients become **very small (close to zero)** as they move backward through layers, it is called the **vanishing gradient problem**.

Why it happens

If values are repeatedly multiplied by numbers less than 1:

$$0.5 \times 0.5 \times 0.5 \times \cdots \to 0$$

In deep networks, this happens due to:

- small weight values
- activation functions like **sigmoid** and **tanh**

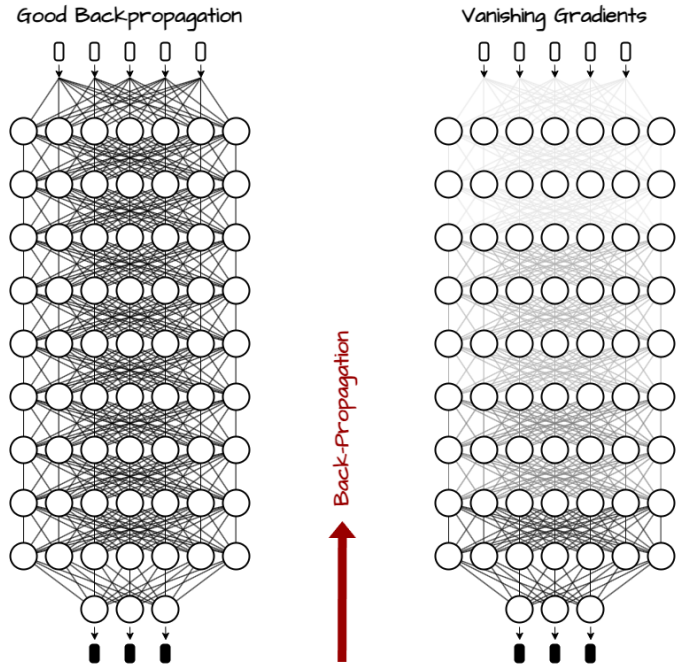
Example (sigmoid derivative):

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

Maximum value is **0.25**, so gradients shrink quickly.

Effects

- Early layers learn **very slowly**
- Network fails to capture important features
- Training becomes ineffective



Vanishing and Exploding Gradients

In deep networks, gradients are computed using the **chain rule**:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial W_n} \cdot \frac{\partial W_n}{\partial W_{n-1}} \cdot \ldots \cdot \frac{\partial W_2}{\partial W_1}$$

This means gradients are **multiplied layer by layer**.

Exploding Gradient : When gradients become **very large**, it is called the **exploding gradient problem**.

Why it happens

If values are repeatedly multiplied by numbers greater than 1:

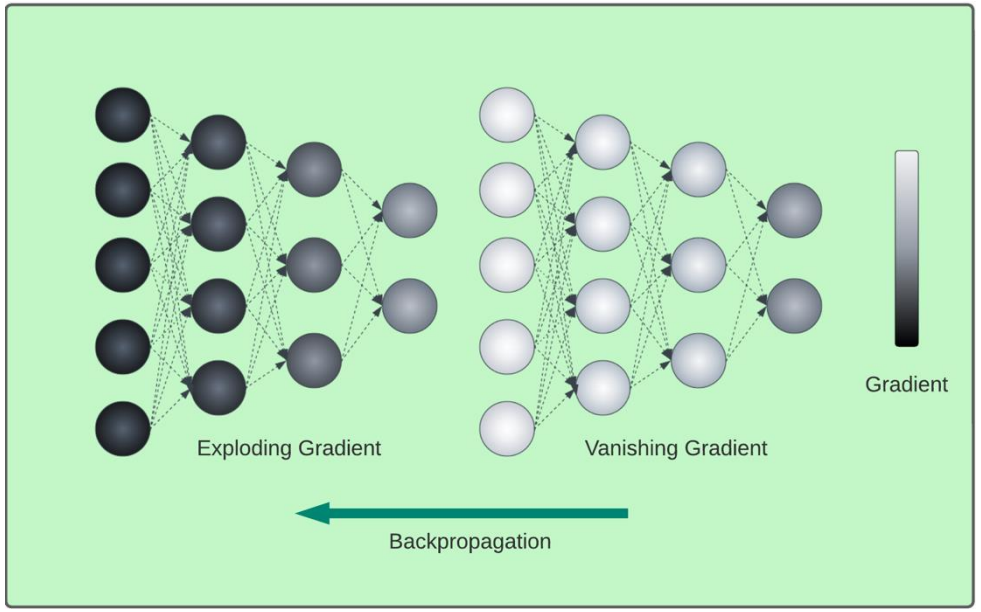
$$2 \times 2 \times 2 \times \cdots \rightarrow \infty$$

This occurs due to:

- large weight values
- deep networks
- poor initialization

Effects

- very large weight updates
- unstable training
- loss becomes NaN or diverges



Regularization in Neural Networks

Regularization is used to **prevent overfitting**, where a model performs well on training data but poorly on unseen data.

What is Overfitting?

- Model memorizes training data
- Learns noise instead of patterns
- High training accuracy but low-test accuracy

Regularization helps the model **generalize better**.

Weight Decay (L2 Regularization)
Penalize large weights so the model stays simple.

Modified Loss Function
Instead of minimizing only loss L, we add a penalty term:

$$L' = L + \lambda \sum W^2$$

Where:

λ= regularization parameter
W= weights

Effect on Weight Update

$$W = W - \eta \left(\frac{\partial L}{\partial W} + 2\lambda W \right)$$

This forces weights to become **smaller over time**.

Intuition

- Large weights → complex model → overfitting
- Small weights → smoother model → better generalization

Key Properties

- Reduces model complexity
- Keeps all features but with smaller influence
- Works well in most neural networks

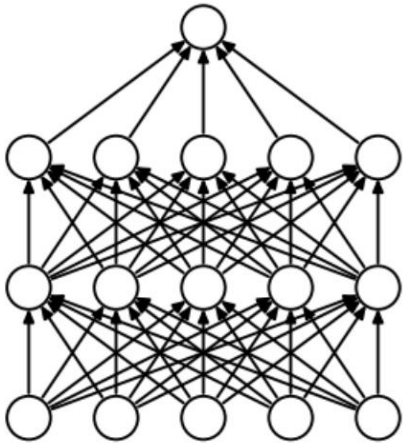
Regularization in Neural Networks

Regularization is used to **prevent overfitting**, where a model performs well on training data but poorly on unseen data.

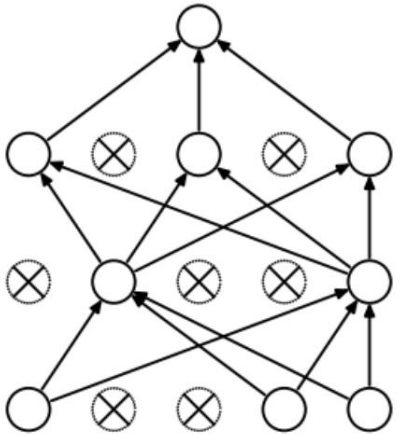
What is Overfitting?

- Model memorizes training data
- Learns noise instead of patterns
- High training accuracy but low-test accuracy

Regularization helps the model **generalize better**.



(a) Standard Neural Net



(b) After applying dropout.

Dropout

Randomly **turn off neurons** during training.

How it Works

During each training step:

- Each neuron is kept with probability p
- Otherwise, it is temporarily removed

Example:

If $p = 0.5$, half of neurons are randomly dropped

Mathematical View

$$h' = h \cdot r$$

Where:

$$r \sim Bernoulli(p)$$

h =neuron output

During Testing

- No neurons are dropped
- Outputs are scaled to match training behavior

Intuition

Dropout forces the network to:

- not depend on specific neurons
- learn **redundant, robust features**

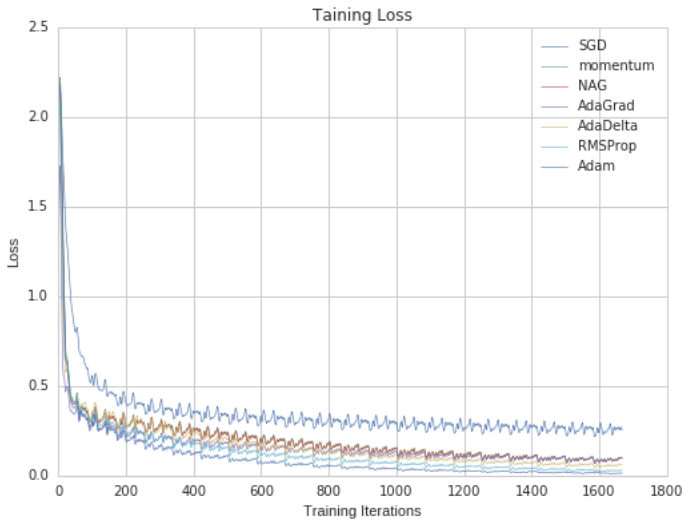
It acts like training **many smaller networks** and averaging them.

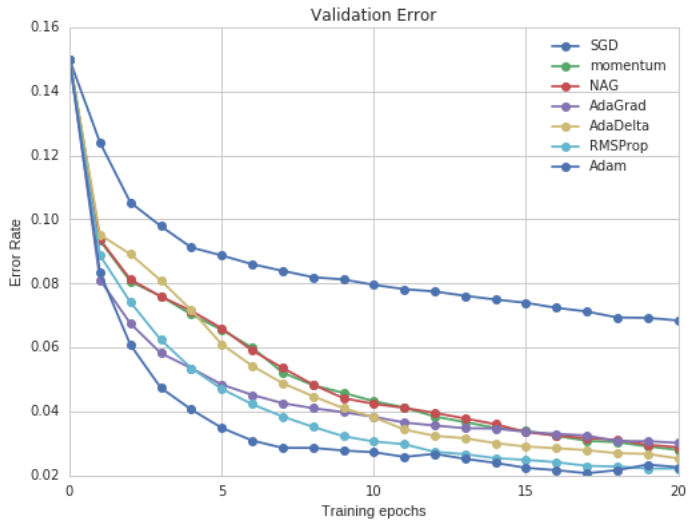
Key Benefits

- Reduces overfitting significantly
- Improves generalization
- Works especially well in deep networks

Optimization Algorithms

Algorithm	Main Idea	Advantages	Disadvantages
Gradient Descent	General method that updates parameters by moving in the negative direction of the gradient of the loss function	Simple concept, foundation of all optimizers	Can be slow, may get stuck in local minima
Batch Gradient Descent	Computes gradient using the entire dataset before updating parameters	Stable and accurate updates	Very slow for large datasets, high memory usage
Stochastic Gradient Descent (SGD)	Updates parameters using one training sample at a time	Fast, efficient for large datasets	Noisy updates, oscillation near minimum
SGD with Momentum	Adds past gradients to smooth updates and accelerate learning	Faster convergence, reduces oscillation	Extra hyperparameter (momentum)
RMSprop	Adapts learning rate using moving average of squared gradients	Faster convergence, reduces oscillation	Requires tuning, not always best generalization
Adam	Combines momentum and RMSprop using first and second moment estimates	Fast, robust, widely used, less tuning required	More memory usage, may generalize worse than SGD





Hyperparameter Tuning

Hyperparameters are configuration settings set before training a model.
Examples:

- Learning rate
- Batch size
- Number of epochs
- Number of layers

Hyperparameter Tuning is the process of finding the best combination of these values to improve model performance.

Why is it Important?
Choosing good hyperparameters can:

- Improve accuracy and performance
- Speed up training
- Prevent overfitting or underfitting

Poor choices can lead to:

- Slow convergence
- Low accuracy
- Unstable training

Hyperparameter	Description
Learning Rate	Controls how much weights are updated
Batch Size	Number of samples per update
Epochs	Number of complete passes through data
Number of Layers	Depth of the neural network
Number of Neurons	Units in each layer
Dropout Rate	Fraction of neurons randomly turned off
Regularization	Controls overfitting (L1/L2)

Method	Main Idea	Advantages	Disadvantages
Manual Search	Try values based on experience	Simple	Time-consuming, not optimal
Grid Search	Try all combinations of parameters	Exhaustive, reliable	Very slow, expensive
Random Search	Randomly sample parameter combinations	Faster than grid, good coverage	May miss best combination
Bayesian Optimization	Uses past results to choose next parameters	Efficient, intelligent search	Complex to implement
Hyperband	Uses early stopping to test many configs quickly	Very fast, resource efficient	May discard good models early

Model Interpretation

Model interpretation explains:

- Why the model made a prediction
- Which features influenced the decision

Importance:

- Trust and transparency
- Debugging models
- Compliance (e.g., healthcare, finance)

Type	Description
Global Interpretation	Understand overall model behavior
Local Interpretation	Explain individual predictions

Interpretation Techniques

Method	Idea	Use Case
Feature Importance	Measures influence of each feature	Tree-based models
SHAP	Uses game theory to explain predictions	Deep learning & ML
LIME	Explains predictions locally	Any black-box model
Partial Dependence Plot	Shows effect of a feature on prediction	Global analysis

Term	Main Idea	Explanation	Example
Epoch	One full pass through the entire dataset	The model sees all training samples once	Dataset = 1000 samples → 1 epoch = all 1000 used
Batch (Batch Size)	Number of samples processed before updating weights	Data is divided into smaller groups	Batch size = 100 → model processes 100 samples at a time
Iteration	One update step of the model	Each batch processed = 1 iteration	1000 samples / batch size 100 → 10 iterations per epoch