Neural network

Jan 27, 2026

Hakyung Sung (PSYC681)

RIT | Rochester Institute of Technology

Outline

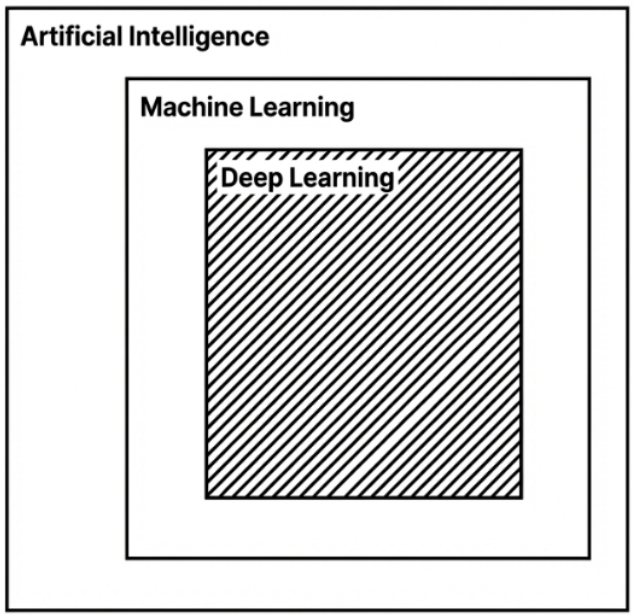
Outline

Last week

- The sift from count-based statistical models to neural representation learning marked a major turning point in NLP
  - *Word embeddings* were a key early example of this shift.
- So, what is a neural network and how it works?

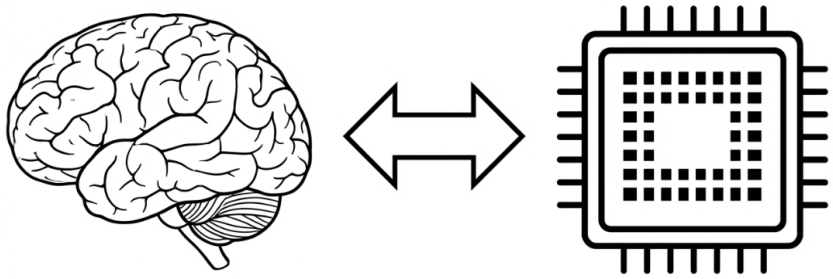
Terms

- Artificial intelligence: The broad goal of creating thinking machines.
- Machine learning: Systems that learn from data.
- Deep learning: The subset built on *artificial neural network*.



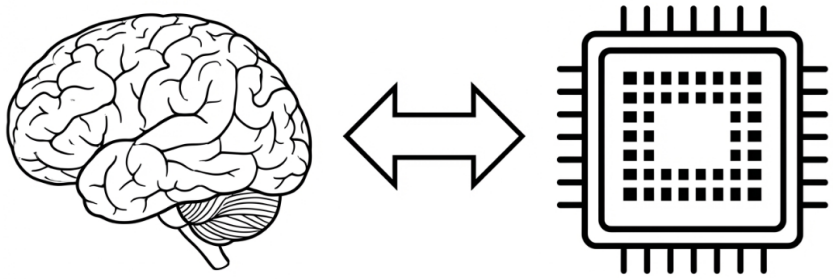
Understanding human brain

Artificial neural networks are explicitly designed to mimic the structure and function of biological processors (i.e., brain).



Understanding human brain

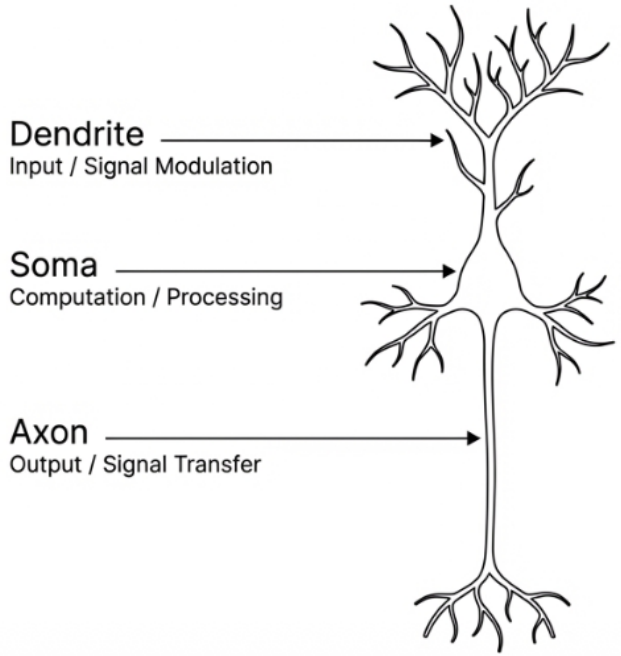
Artificial neural networks are explicitly designed to mimic the structure and function of biological processors (i.e., brain).



- Therefore, understanding how human brains work is the very first step; Human brains are quite complex.
- But, its basic unit is relatively simple.

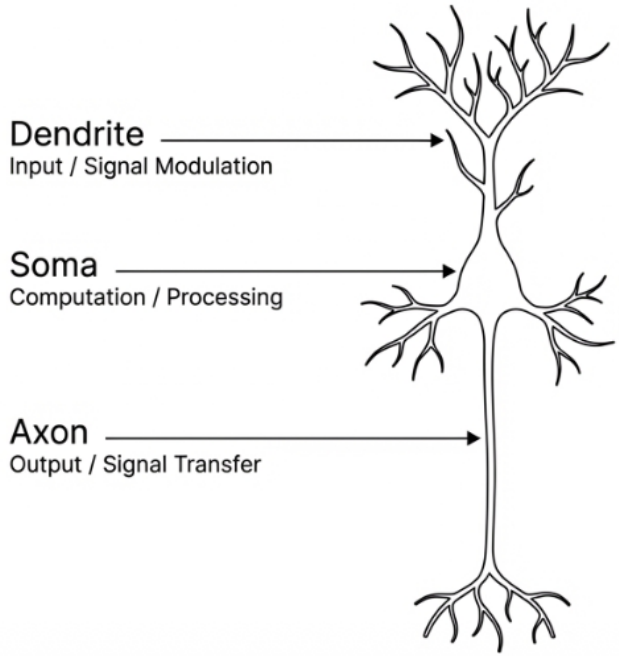
The biological processor: Neuron

A single neuron acts as an information-processing unit with three functions: input, computation, output



The biological processor: Neuron

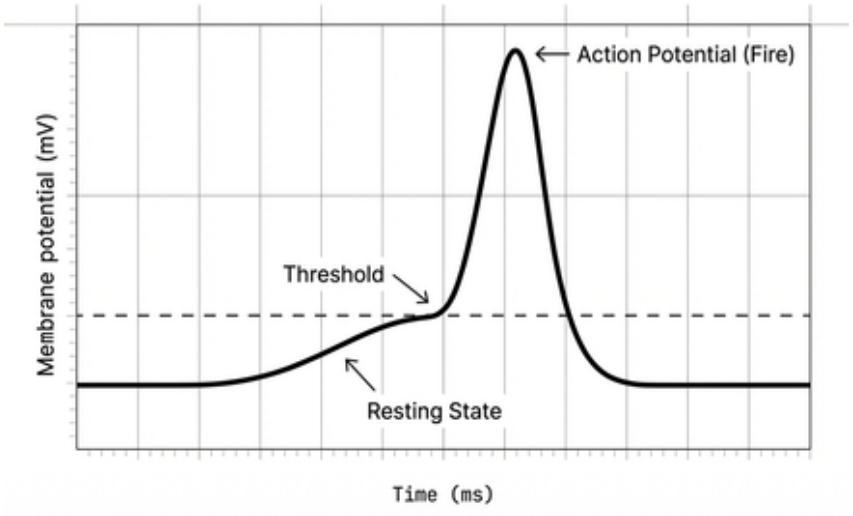
A single neuron acts as an information-processing unit with three functions: input, computation, output



cf. connect to other neurons, *synapse*

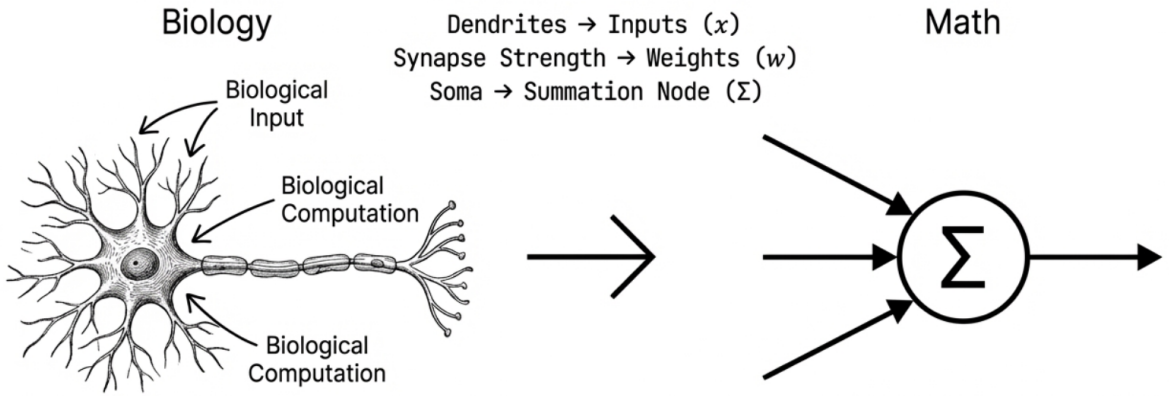
Firing

- Neurons do not simply pass signals along; they make decisions.
- In the soma, incoming signals are computed.
  - If this total signal exceeds a specific threshold, the neuron files an "action potential" (i.e., an electrical spike that travels down the axon).
  - if not, nothing happens.



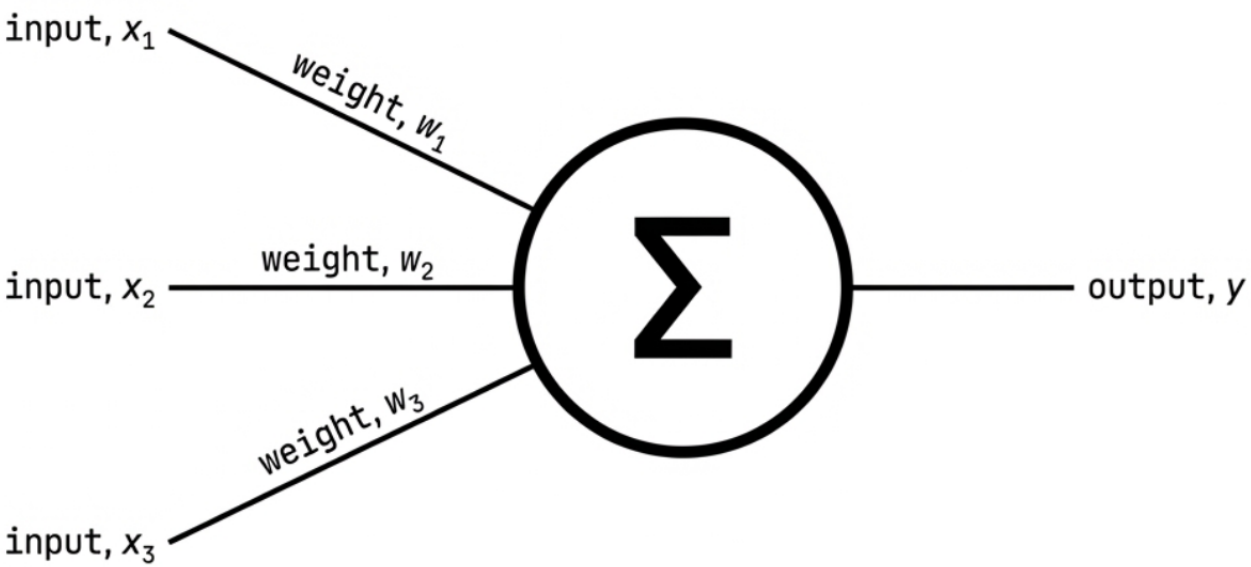
From biology to code

- Artificial neurons are designed to mimic the information-processing mechanisms of these neurons.
- We abstract the biology into a mathematical model called *perceptron*.



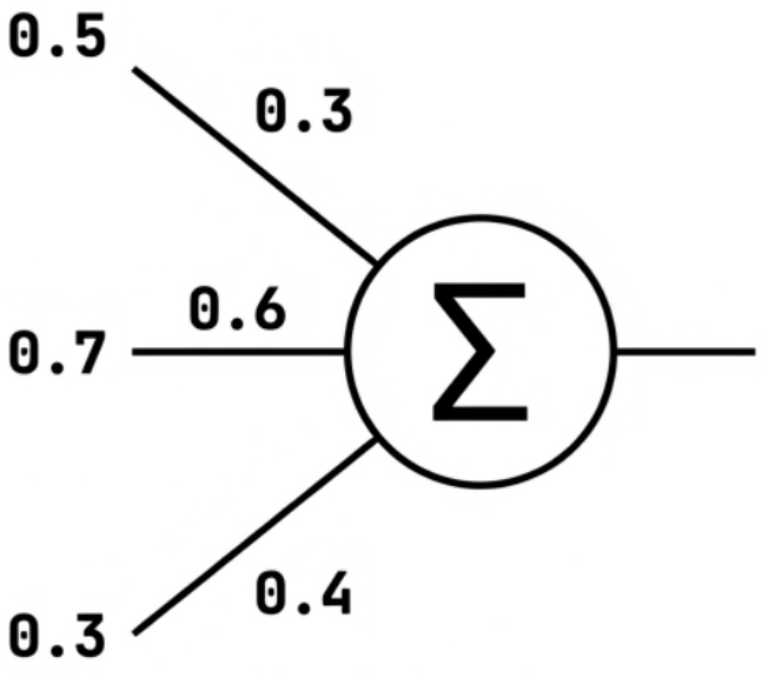
Perceptron

- The artificial node receives multiple inputs (x).
- Each input is multiplied by a corresponding weight (w).
- The circular node ($\sum$) sums these values.
- The result is passed to an activation function to determine the final output (y).



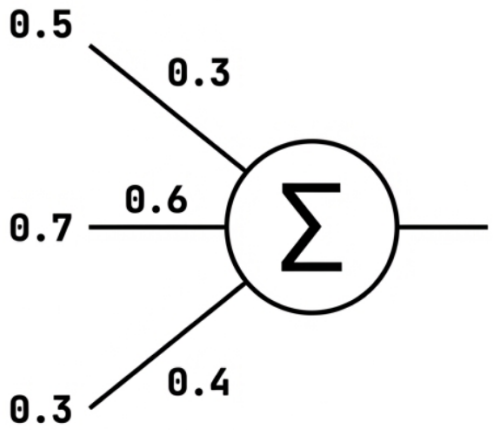
Perceptron: Example

- Let's calculate the value for a single node using specific inputs and weights.

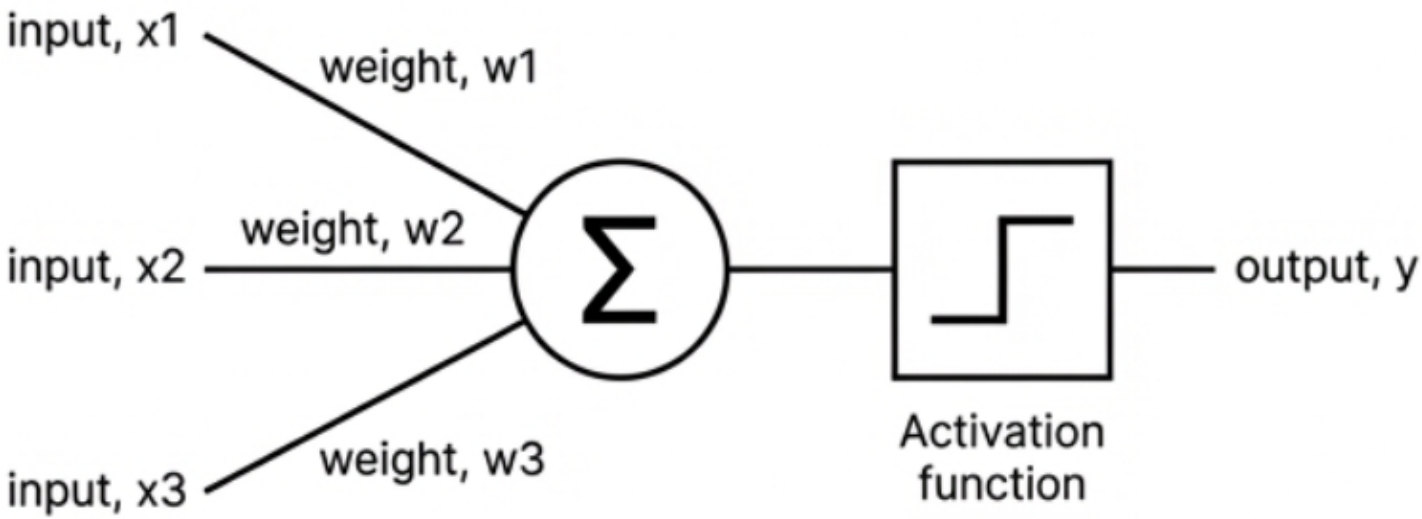


Perceptron: Example

- Let's calculate the value for a single node using specific inputs and weights.



- Weighted sum $= 0.69 \Rightarrow$ The neuron decides whether to fire.
- Apply a simple *step function* with a threshold of 0.5.

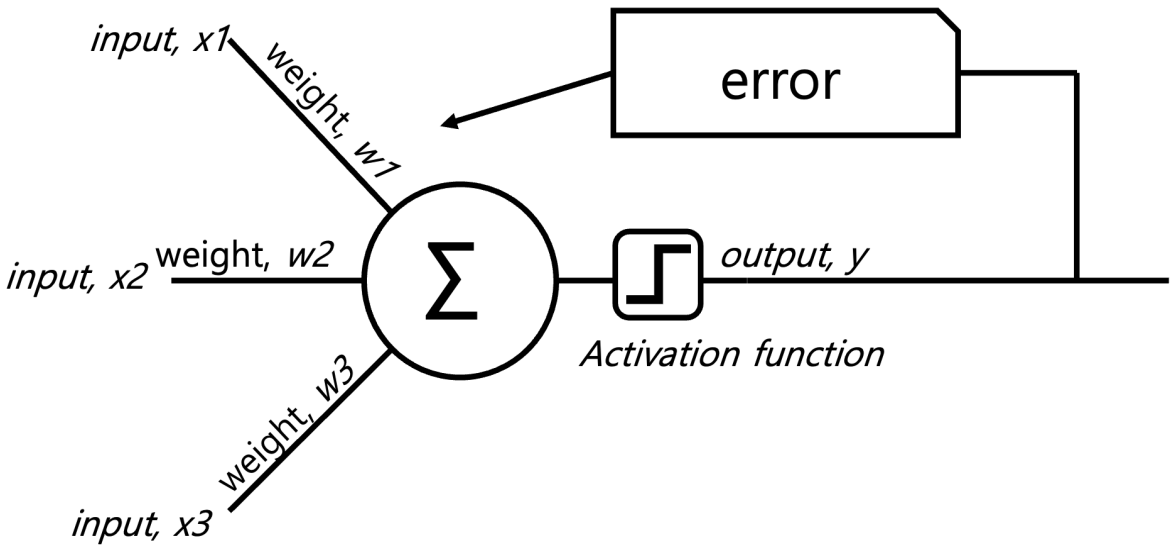

input, x1
weight, w1
input, x2
weight, w2
input, x3
weight, w3
Σ
Activation function
output, y

```
IF (Sum >= Threshold):
    Output = 1
ELSE:
    Output = 0

Calculated Sum: 0.69
Threshold: 0.5
Result: 1 (The Neuron Fires)
```

Training perceptron

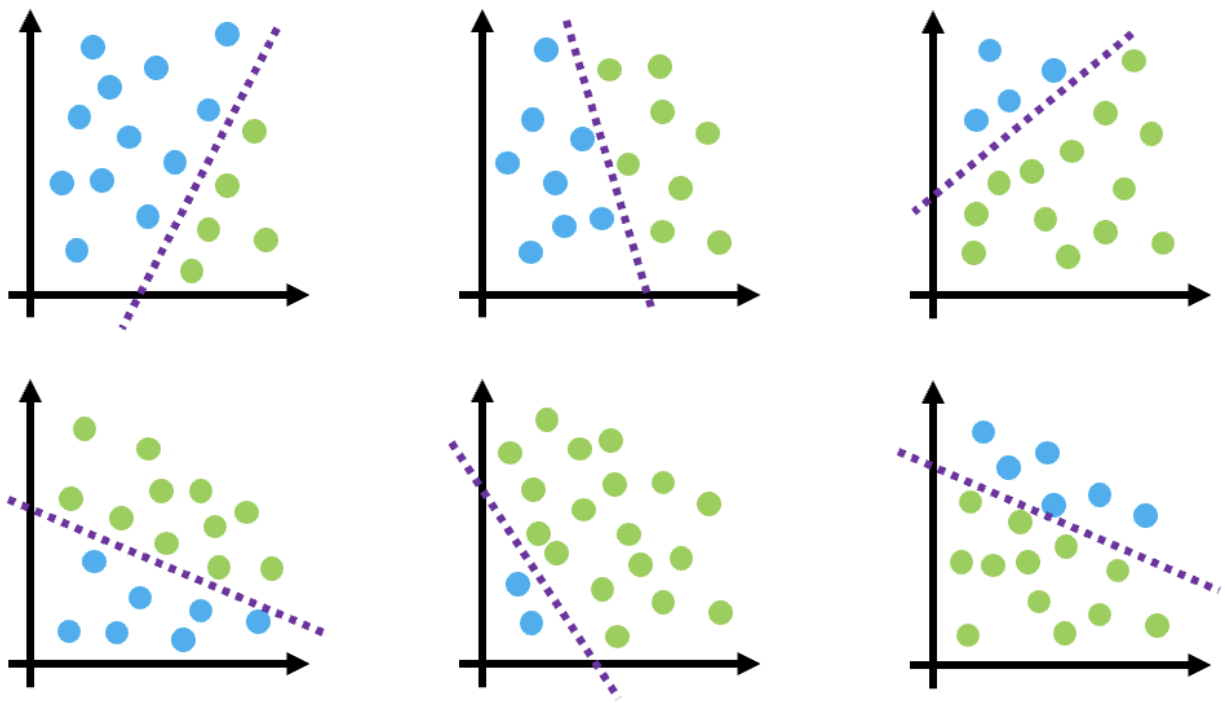
To train perceptron, we compare the predicted output with the actual output. This difference is the *error*, which is then used to adjust the weights so that the model improves over time.



Outline

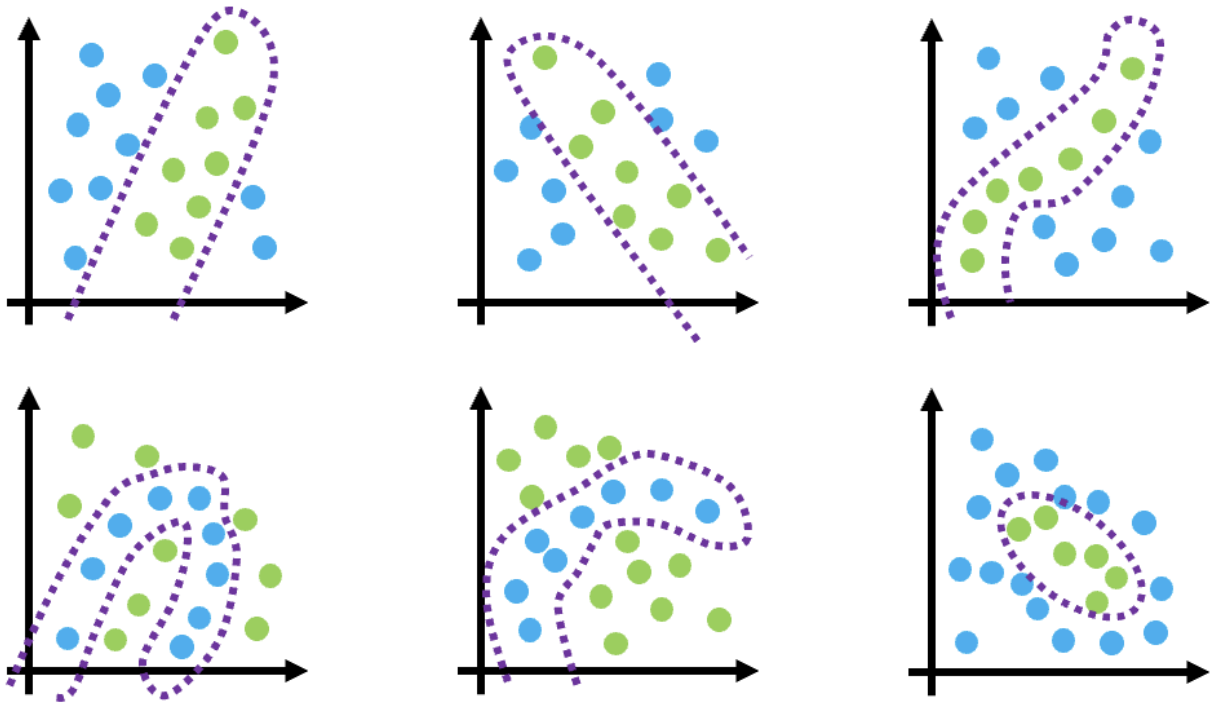
Limitation of a single-layer perceptron

- A single-layer perceptron works well for linearly separable data.
- If the data points can be divided by a single straight line in a 2D plane, the perceptron can learn to adjust its weights to find that line and separate the classes.



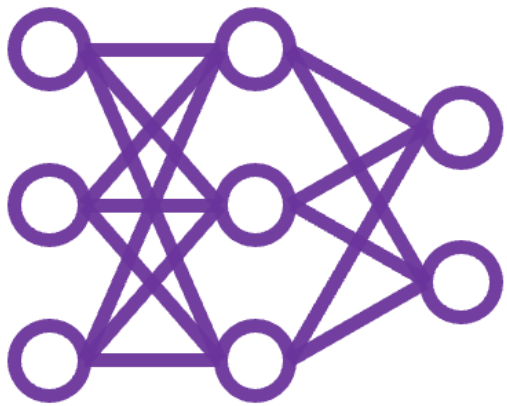
Limitation of a single-layer perceptron

However, a single-layer perceptron cannot solve problems where the data is **not linearly separable**.

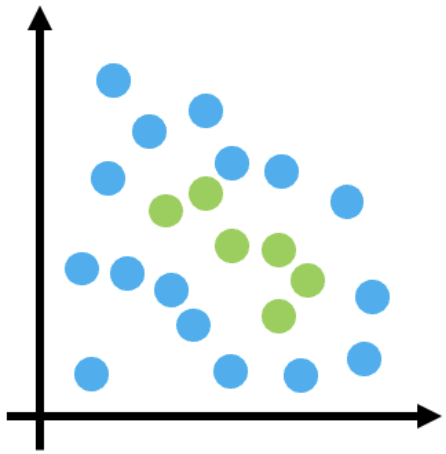


Multi-layer perceptron (MLP)

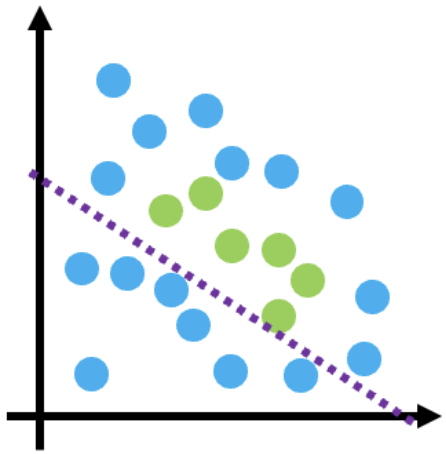
But if we allow multiple lines, there is a possibility to separate even non-linear data. This idea leads us to the **multi-layer perceptron (MLP)**.

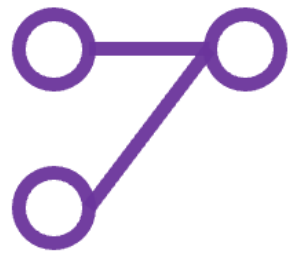


Let's assume we are given data in a complex form like this.

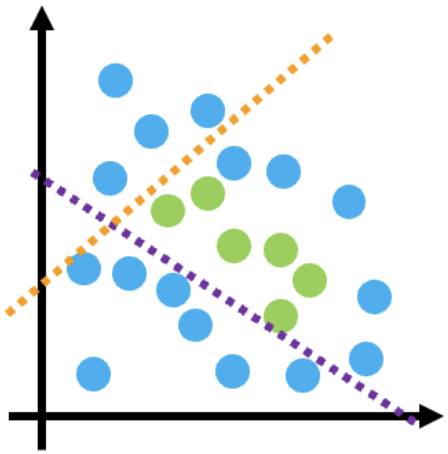


With a single perceptron, linear separation is not possible.

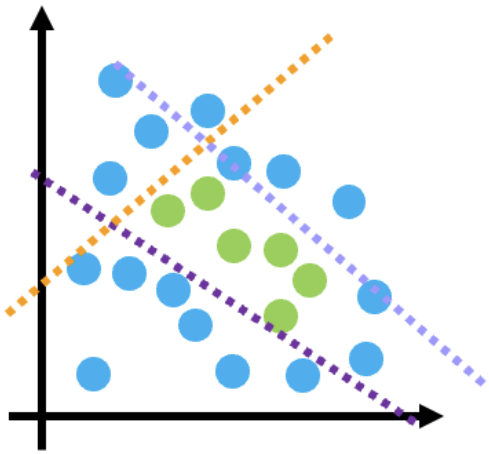




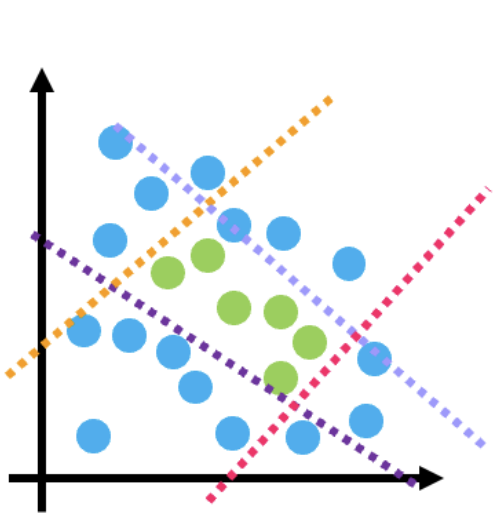
But if we add more lines, it becomes possible to separate further.



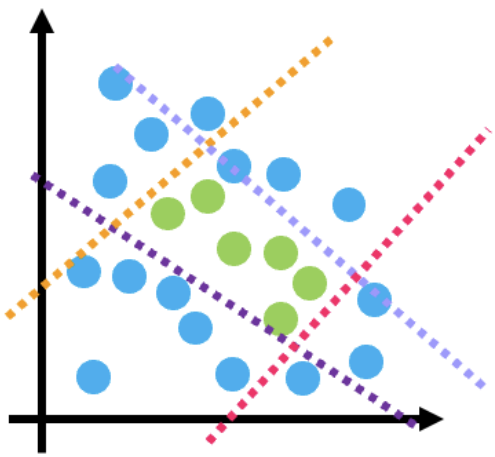
By adding several lines, the separation becomes more feasible.

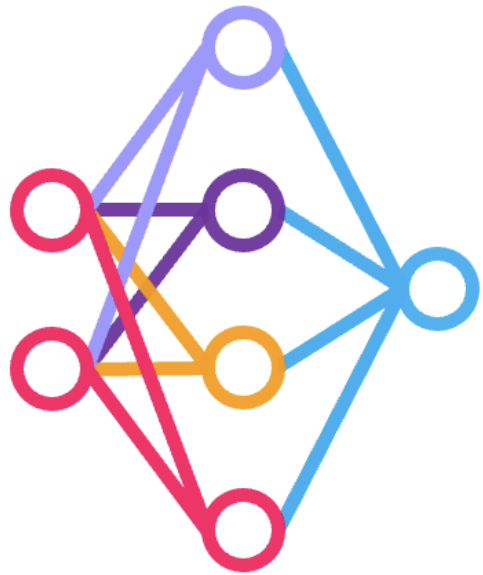


Four lines can be thought of as the outputs of four perceptrons.



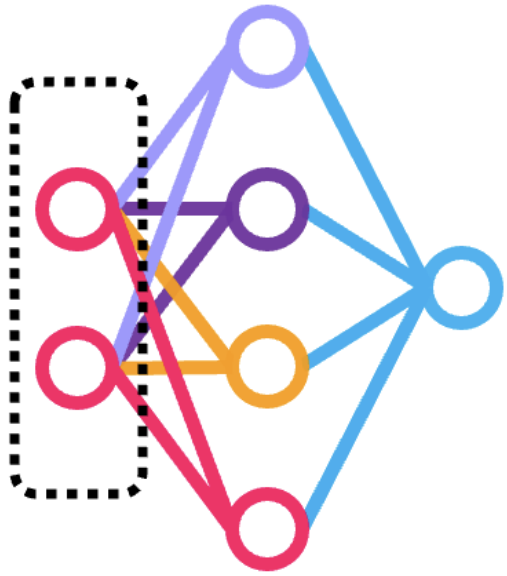
If we then connect another perceptron that takes these four outputs as its inputs, we can construct a **multi-layer neural network** capable of non-linear separation.





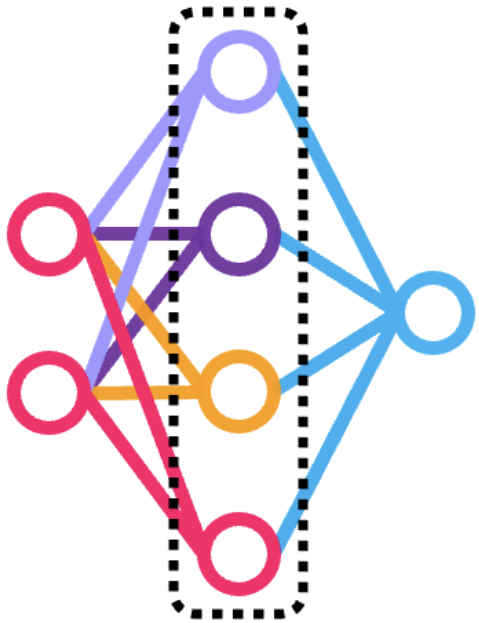
Structure of an MLP

So the MLP we build here consists of an input layer



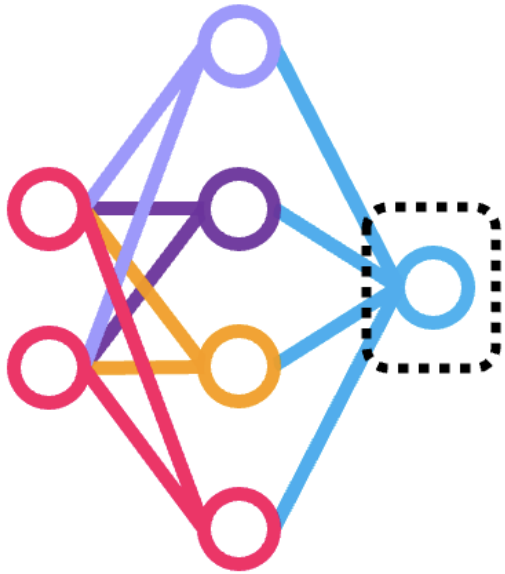
Structure of an MLP

a hidden layer



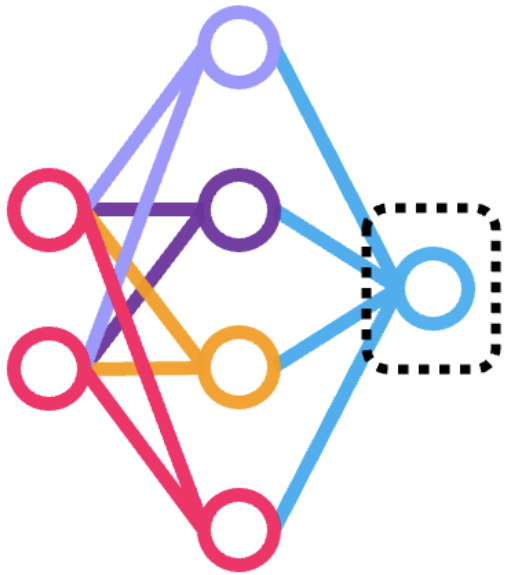
Structure of an MLP

and an output layer



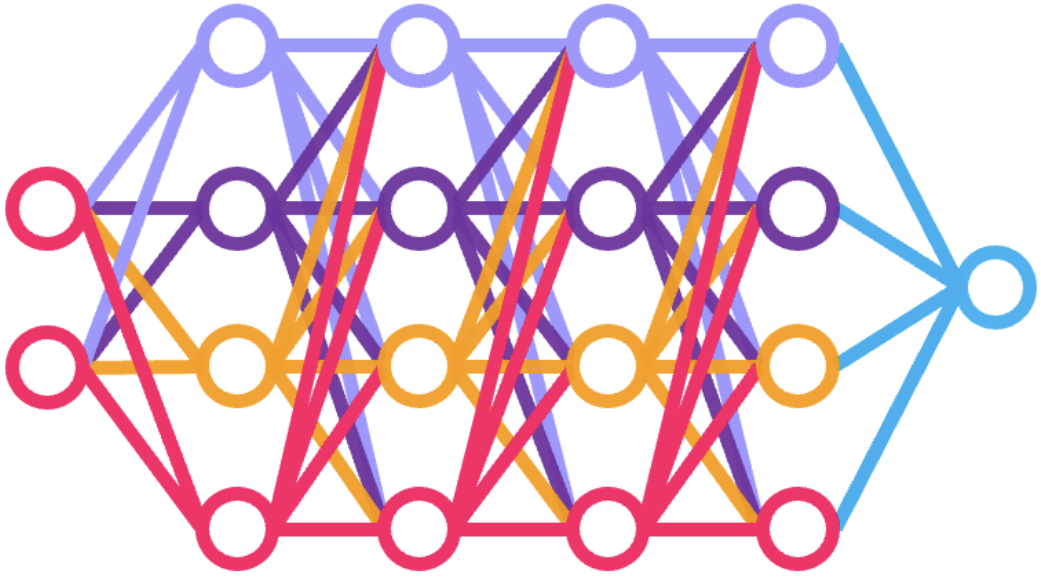
Structure of an MLP

As the number of layers increases, the model can handle more complex data.



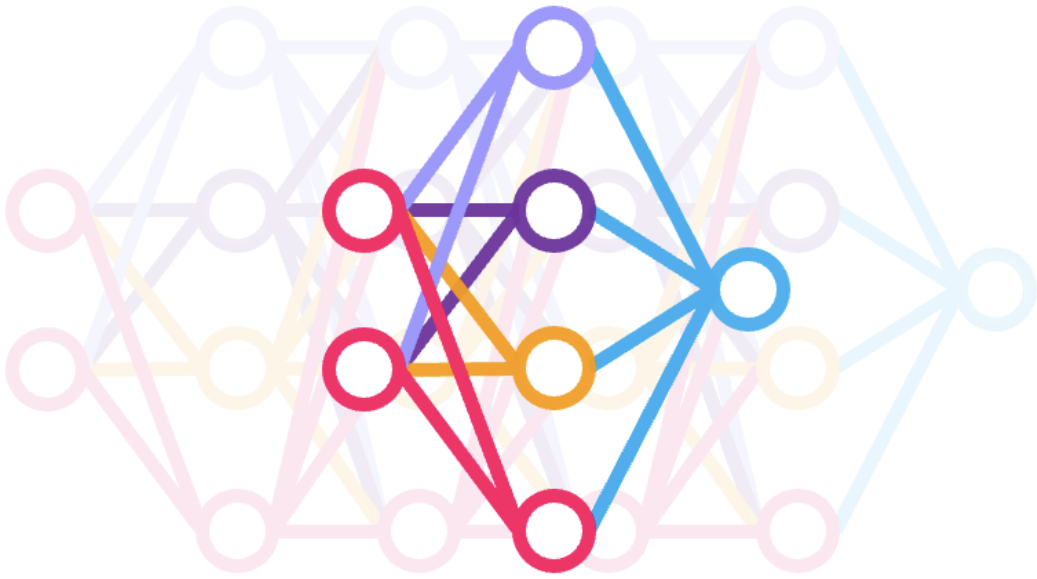
Structure of an MLP

When a network has many layers, we call it "deep." This is where the term deep learning comes from.



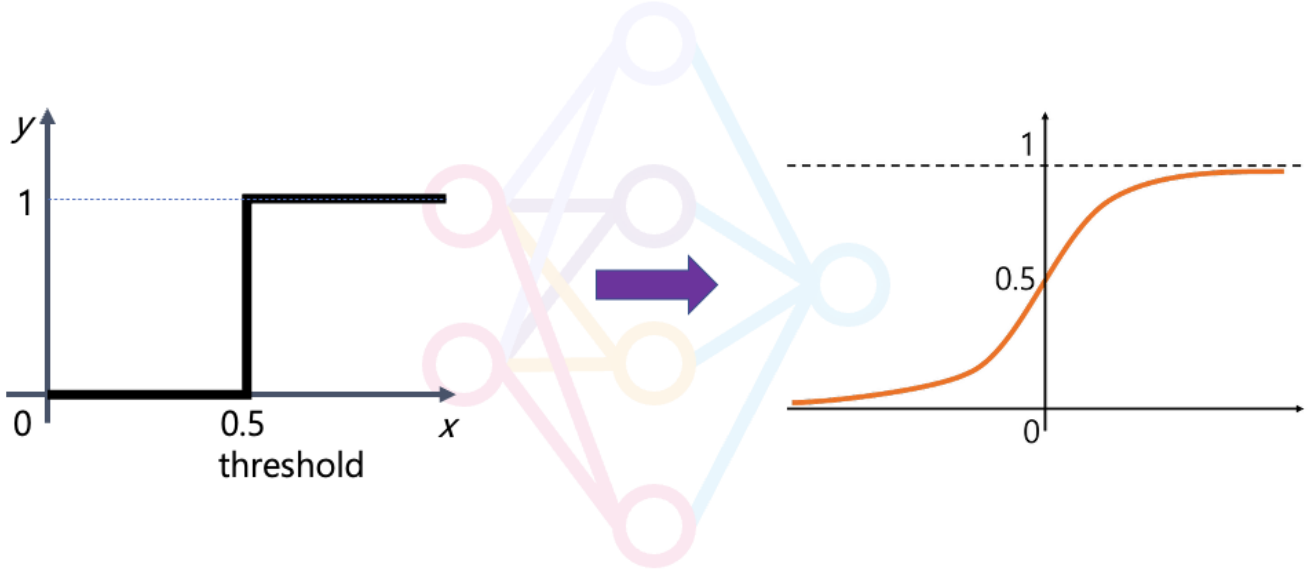
Structure of an MLP – Deep Learning

To understand how multilayer networks work, we need to look at a few more changes.



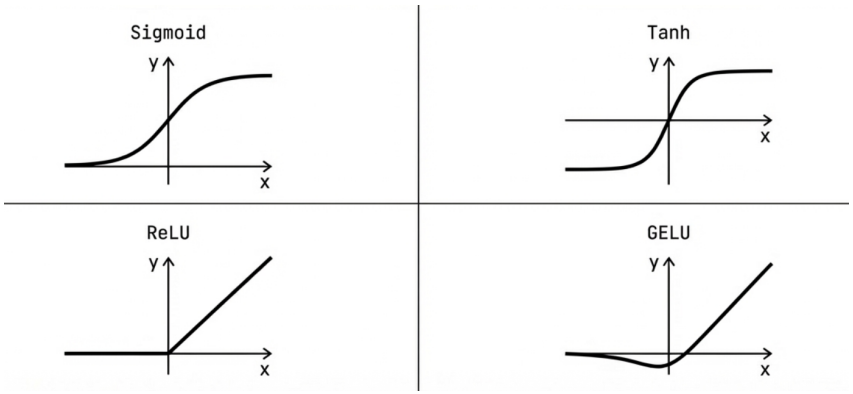
More changes: Activation function

More complex activation functions are used. For example, the *sigmoid function* (which we briefly skimmed) last class.



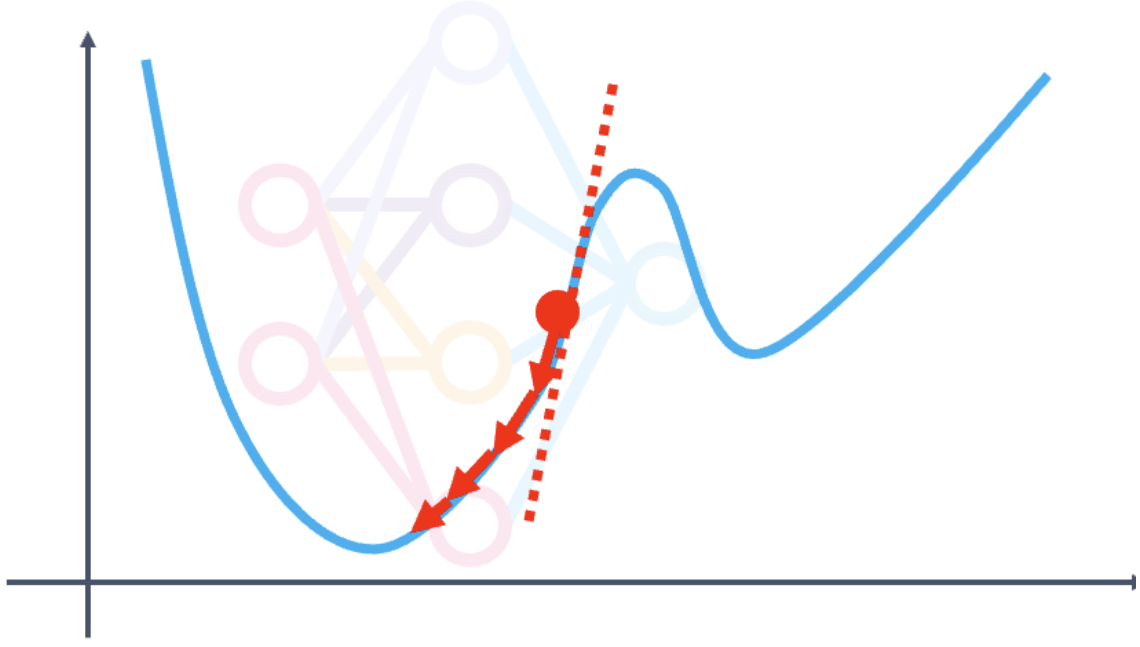
More: At-a-Glance Comparison

In fact, many more activation functions have been introduced:



More changes: Optimization

To reduce errors in multilayer networks, methods like gradient descent are used.

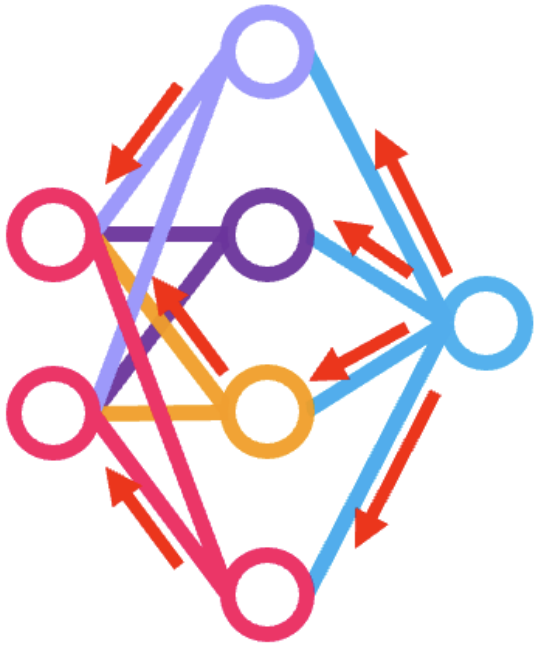


More changes: Optimization

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values
  - Compute the gradient of $J(\theta)$ (which tells us the slope)
  - Move a small step in the **opposite direction** of the gradient
  - Repeat many times until the loss becomes small
  - *more in ml class*

More changes: Backpropagation algorithm

A key algorithm in training neural networks is backpropagation.



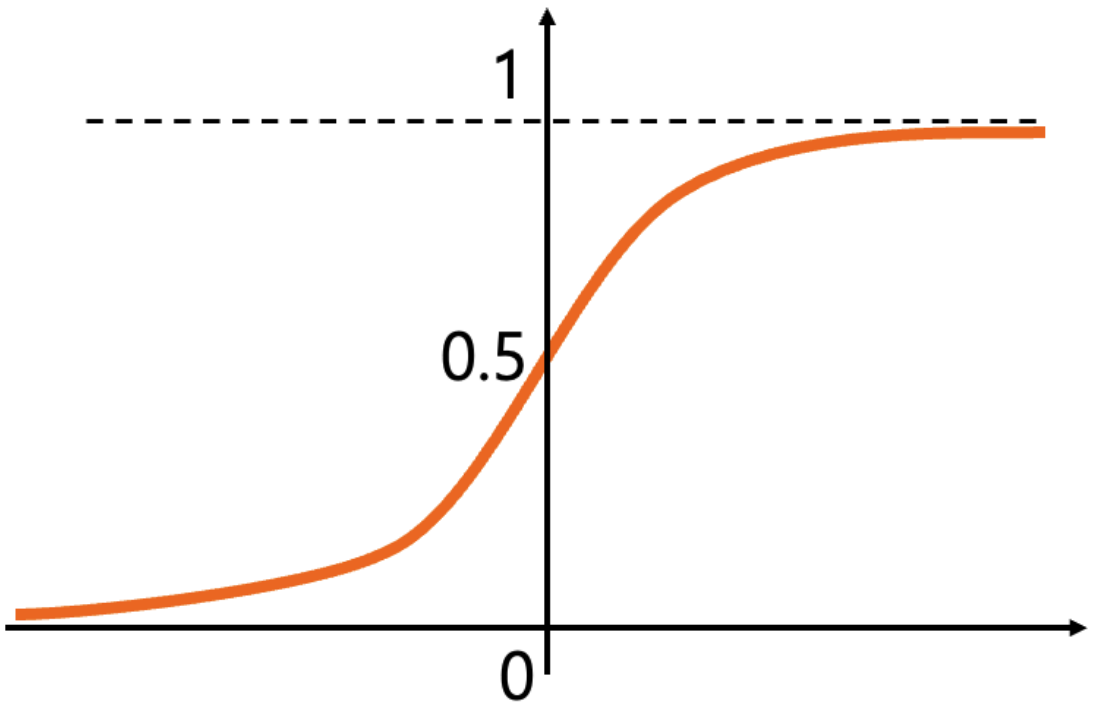
Outline

Gradient Descent: Definition

- Gradient descent is an optimization algorithm used in deep learning.
- It minimizes a given loss function by updating model parameters.
- Model parameters include:
  - Weights – connection strengths between neurons
  - Bias – shifts the activation function left or right to speed up learning

Gradient Descent: Definition

- Model parameters include:
  - Weights – connection strengths between neurons
  - Bias – shifts the activation function left or right to speed up learning

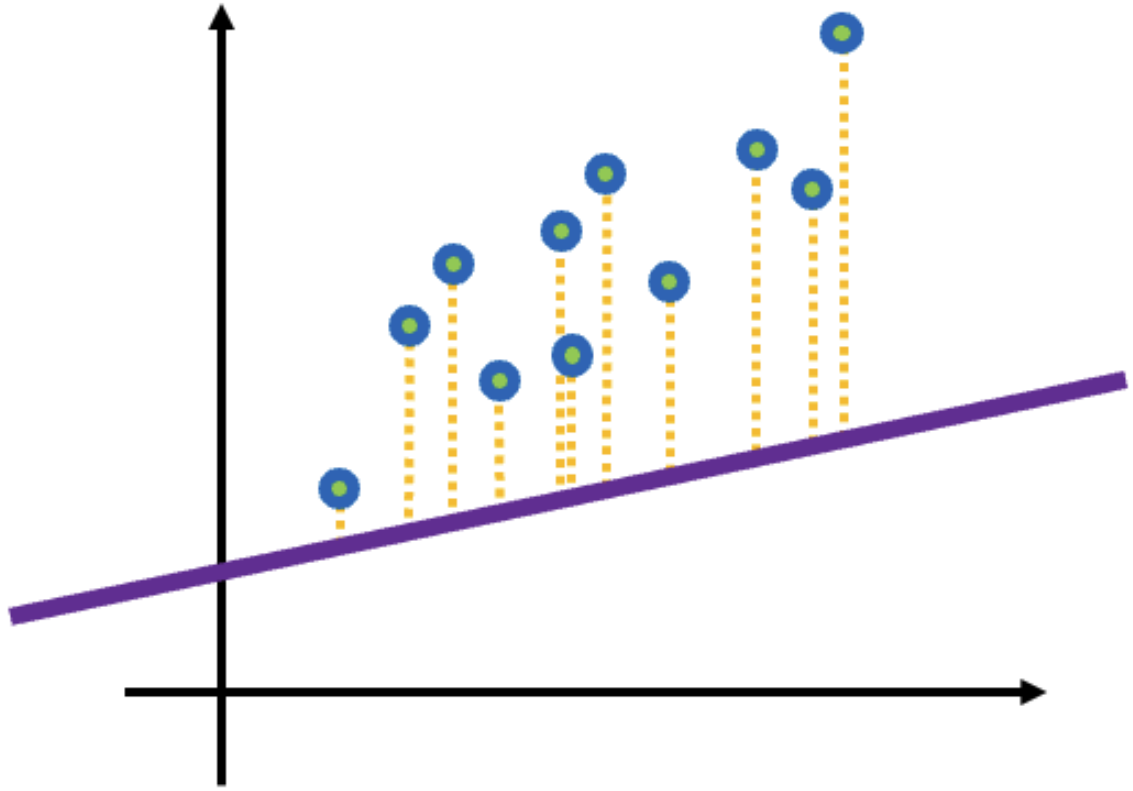


Gradient Descent: Definition

- Gradient descent minimizes a given loss function by updating model parameters.

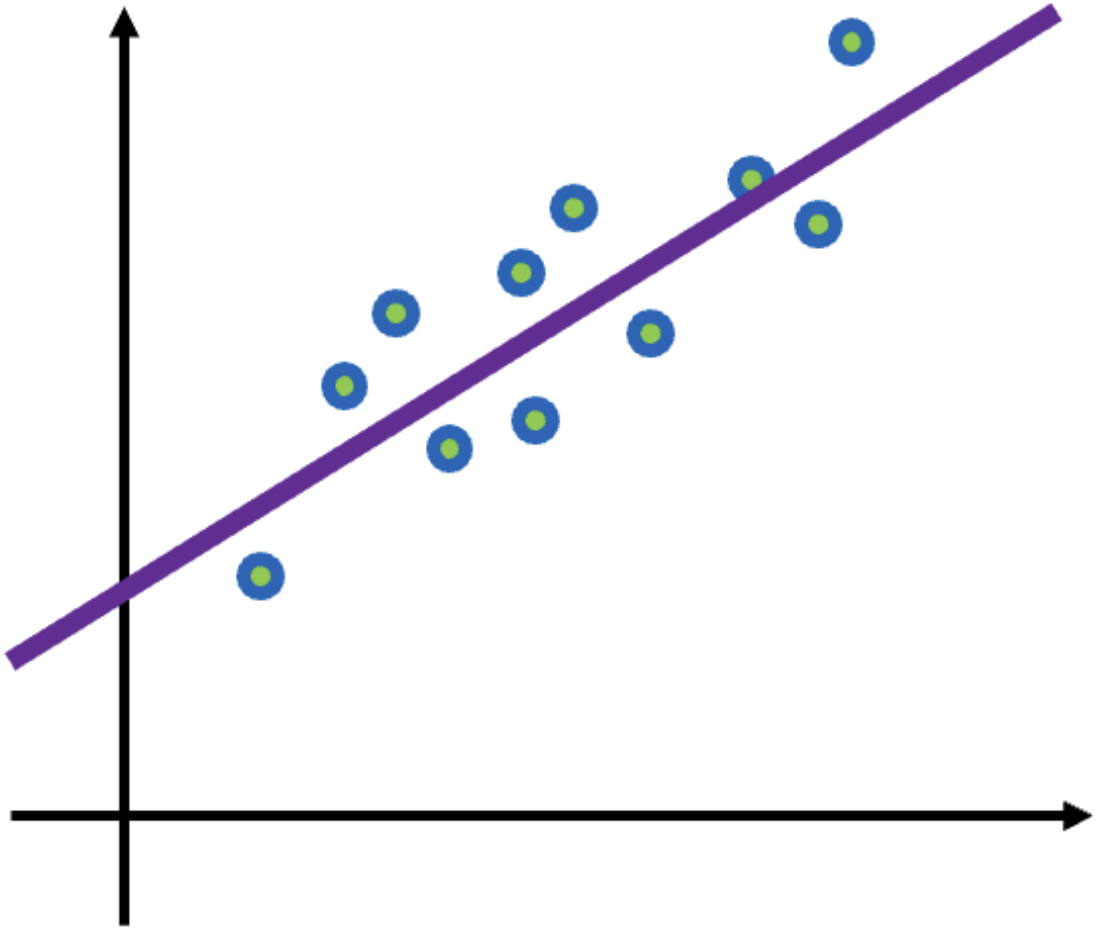
Loss Function

- A loss function measures the difference between predicted values and actual values.



Loss Function

- After training, if the relationship between the predictions and the actual values changes like this, the error will likely decrease.



Loss Function

- A loss function measures the difference between predicted values and actual values.
- Training a neural network means reducing this error step by step.
- Example: **Mean Squared Error (MSE)**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Loss Function: MSE

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

error

Loss Function: MSE

To remove the effect of the sign, square the difference.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Loss Function: MSE

Add up the errors for all the data points.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Loss Function: MSE

Divide by the number of data points.

$$MSE = \frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2$$

Loss Function

- Other common loss functions: *Cross-Entropy* (faster for classification tasks, which also briefly mentioned in the last class).
- **Smaller** loss values indicate better model performance.

Gradient Descent: Core Idea

Now that we understand the loss function, let's think about how to apply the gradient descent algorithm using MSE.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Gradient Descent: Core Idea

To make things simple, let's first consider the case where we only have one data point.

$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

Gradient Descent: Core Idea

In this case, it reduces to a familiar quadratic equation.

$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

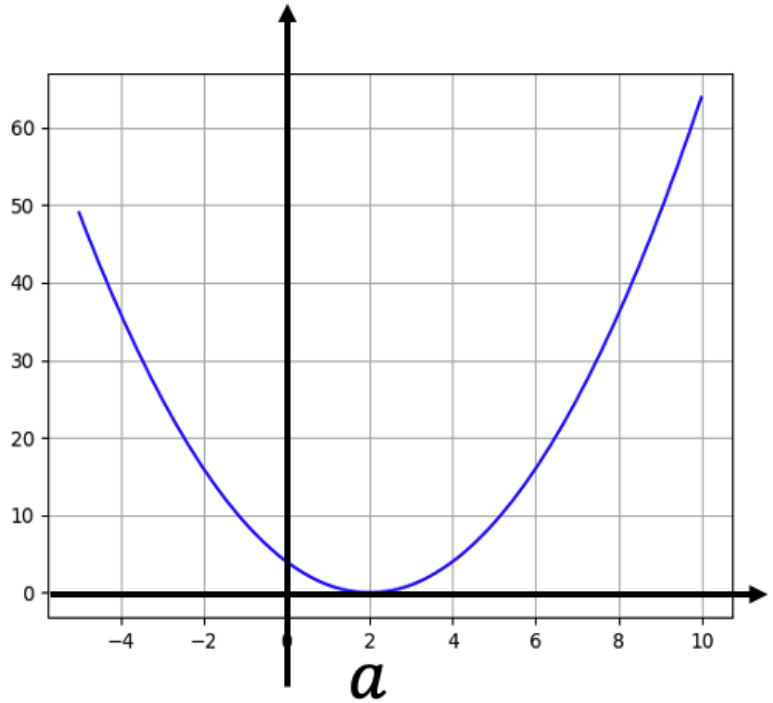
$$y = \frac{1}{1}\sum_{1}^{1}(x_{\square} - a)^2$$

Gradient Descent: Core Idea

If we plot this as a graph, it looks like this.

$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

$$y = \frac{1}{1}\sum_{1}^{1}(x_{\square} - a)^2$$

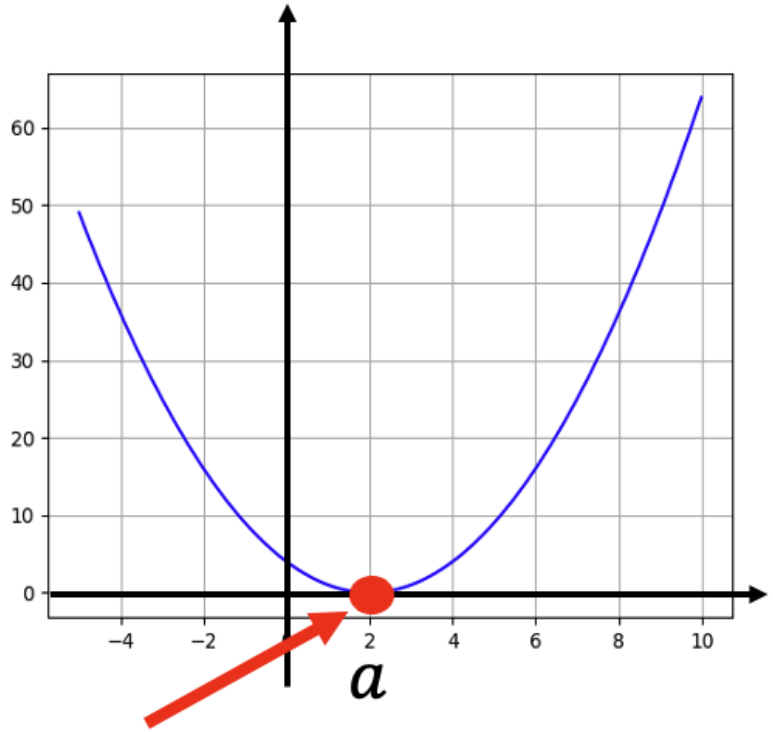


Gradient Descent: Core Idea

The point where the error is minimized is here.

$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

$$y = \frac{1}{1}\sum_{1}^{1}(x_{\square} - a)^2$$

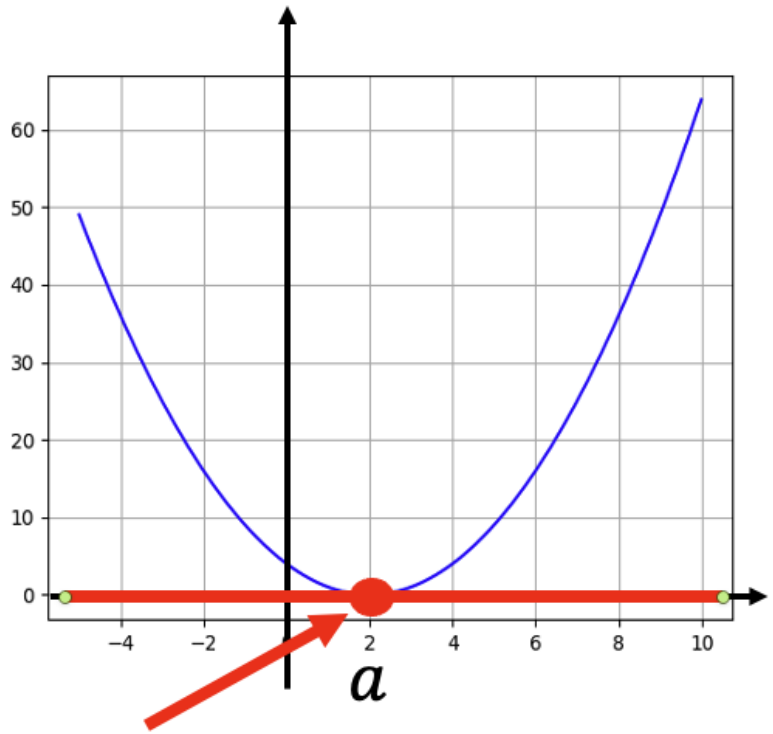


Gradient Descent: Core Idea

At the minimum error, the slope of the curve is zero.

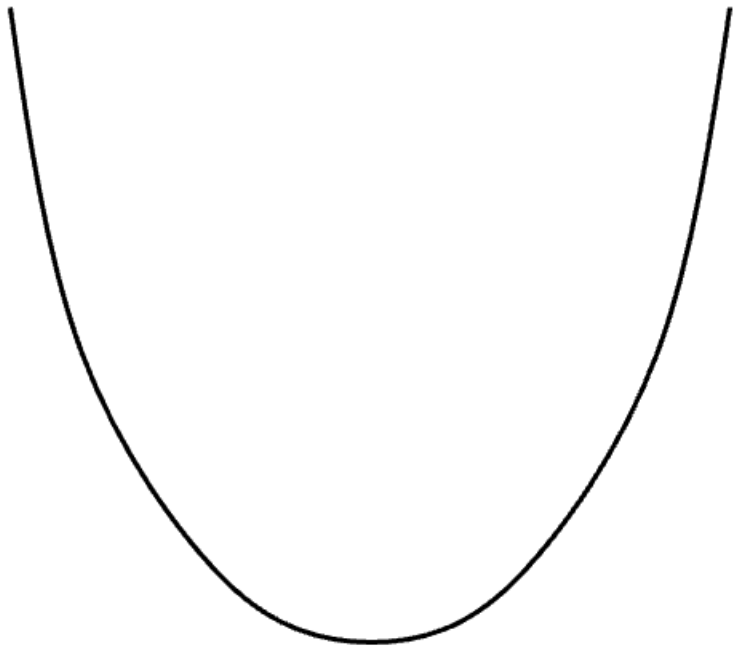
$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

$$y = \frac{1}{1}\sum_{1}^{1}(x_{\square} - a)^2$$



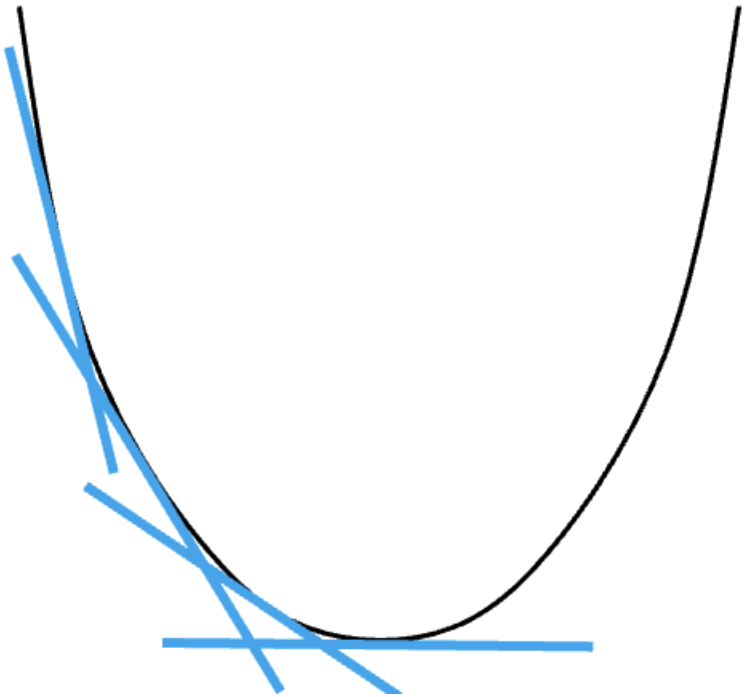
Gradient Descent: Core Idea

So the goal is to find the point where the tangent slope becomes zero.



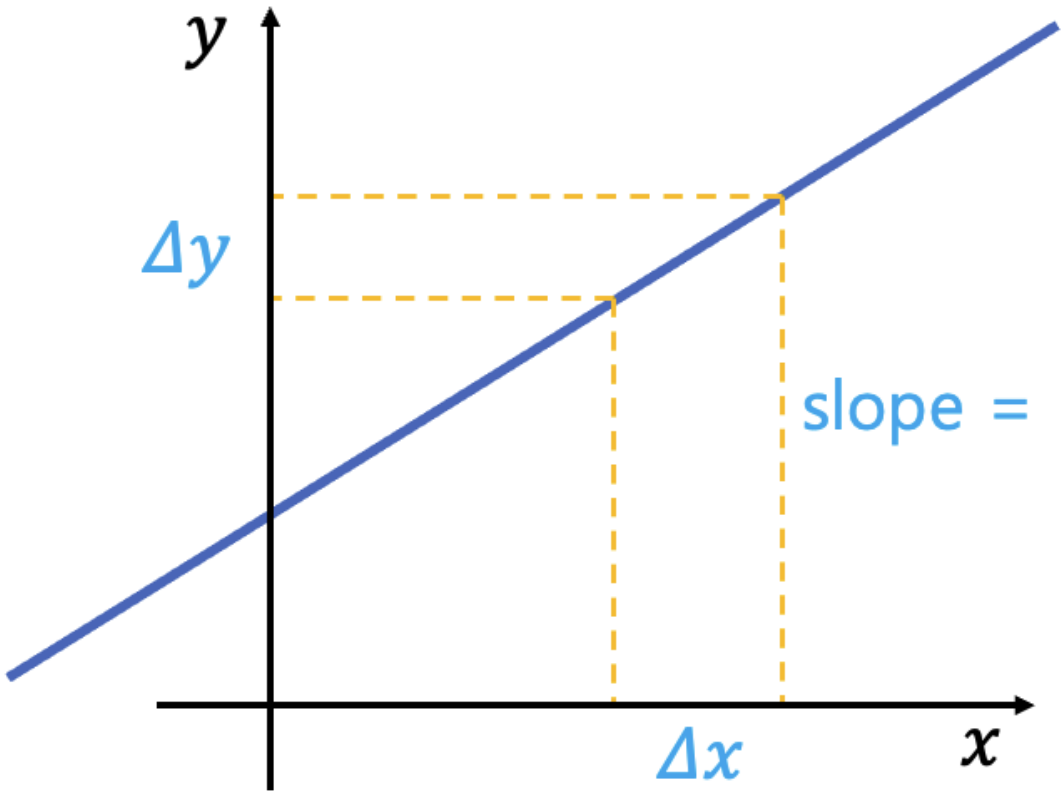
Gradient Descent: Core Idea

The method of gradually decreasing the slope of the tangent is what we call gradient descent.



Gradient Descent: Core Idea

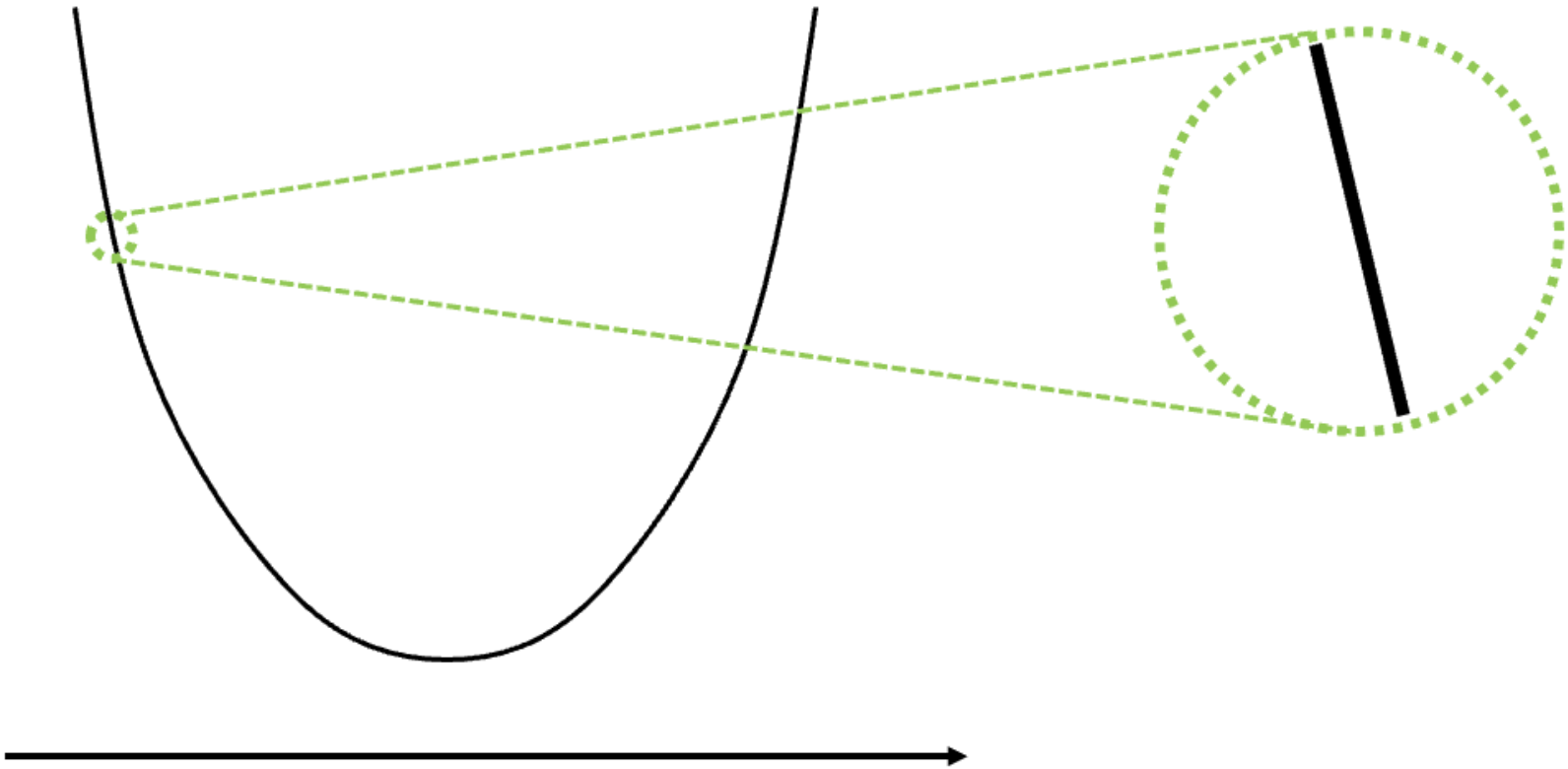
For a linear function, we can express the slope like this.



$$\text{slope} = \frac{\Delta y}{\Delta x} = \frac{\text{change in y}}{\text{change in x}}$$

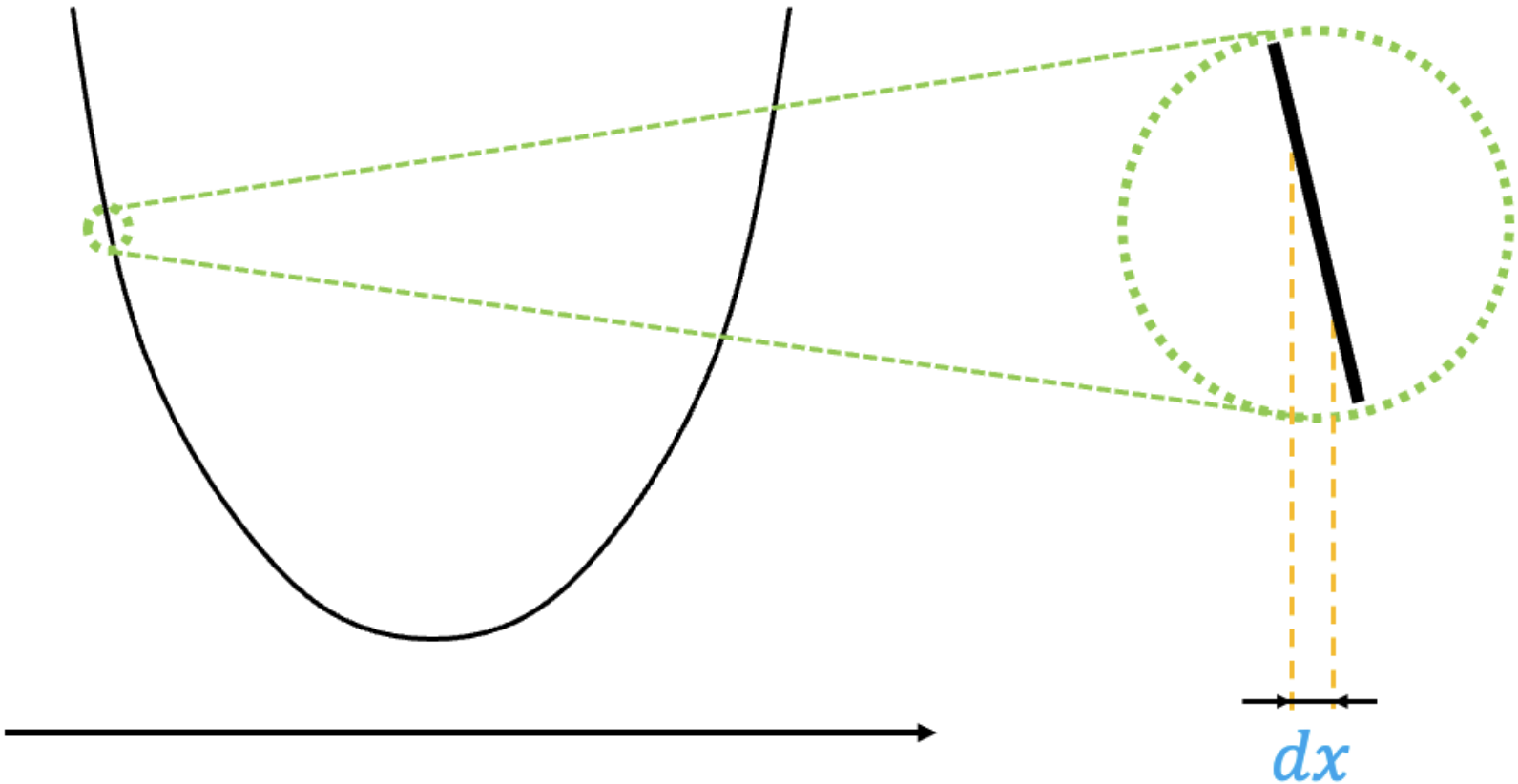
Gradient Descent: Core Idea

And by using derivatives, we can compute slopes even when the curve is not a straight line.



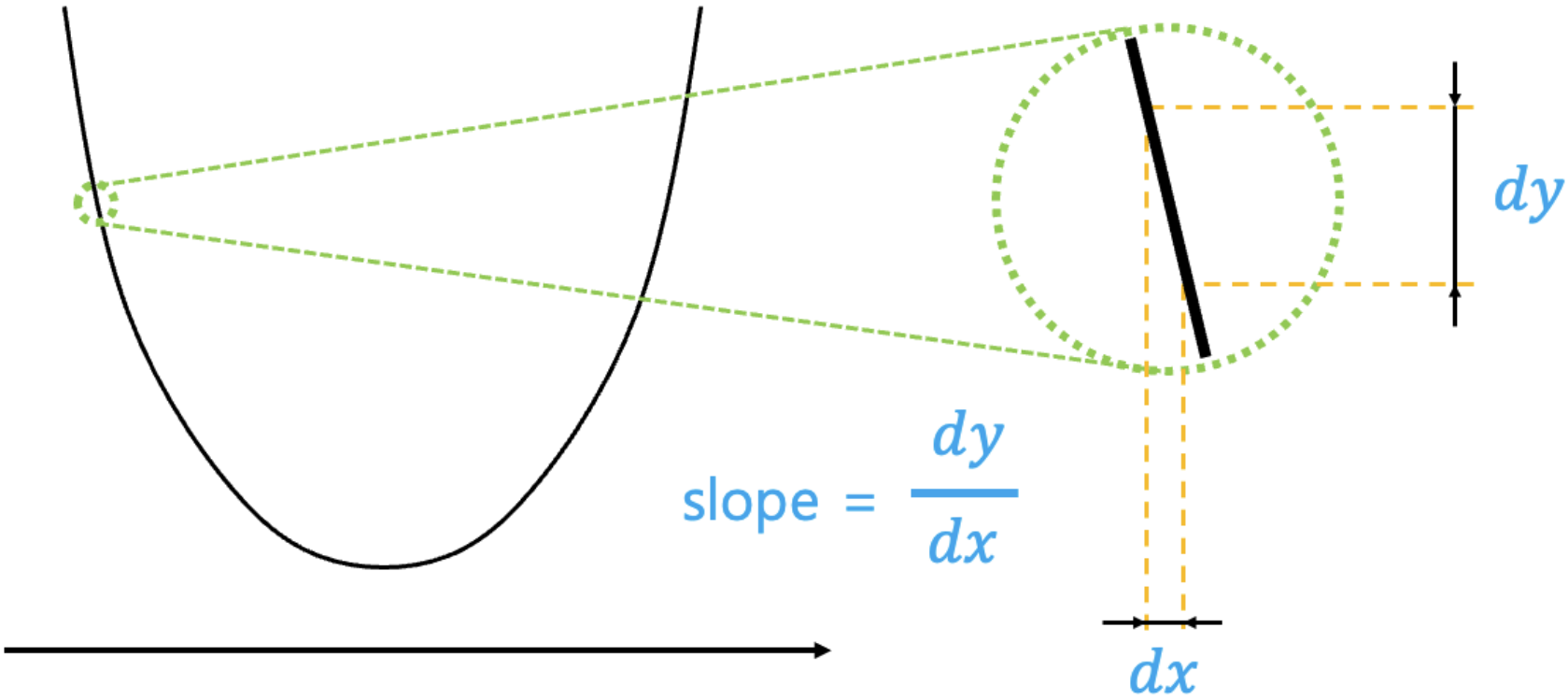
Gradient Descent: Core Idea

With an infinitesimally small change in x, dx,



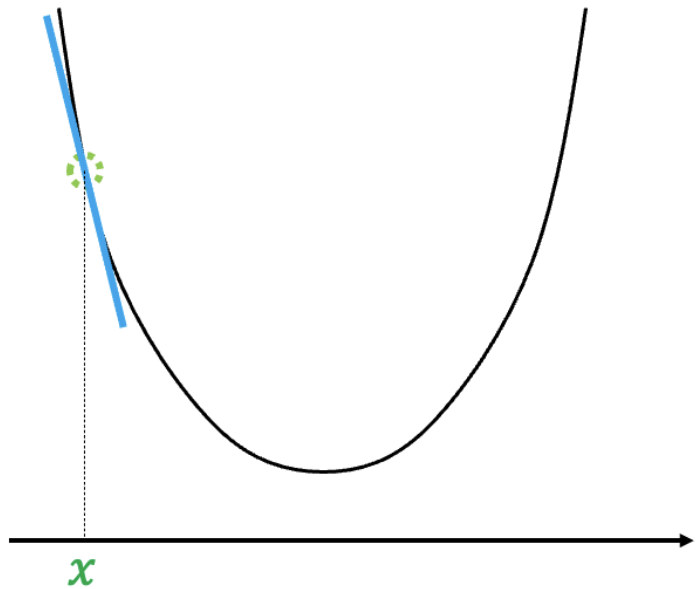
Gradient Descent: Core Idea

and the corresponding infinitesimal change in y, dy, we can calculate the slope.

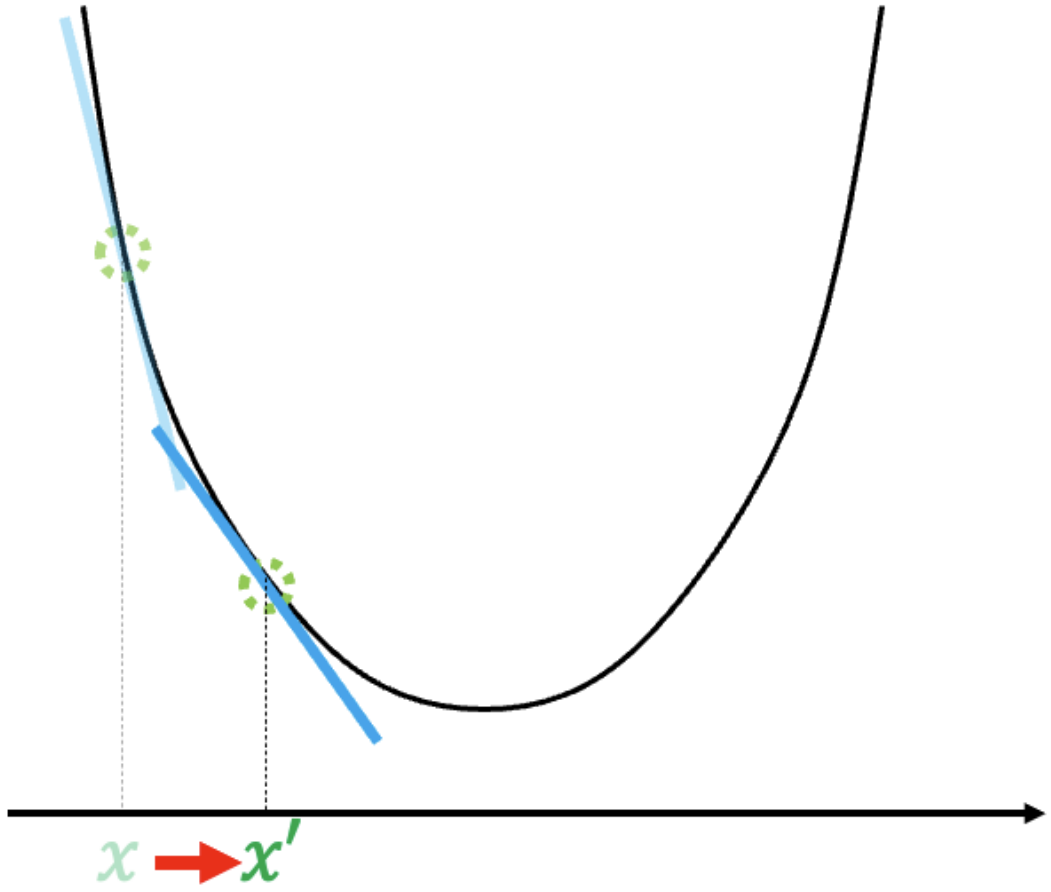


Gradient Descent: Core Idea

- The algorithm updates parameters by moving in the **opposite direction of the gradient**.
  - If slope is negative $\rightarrow$ increase the parameter value.

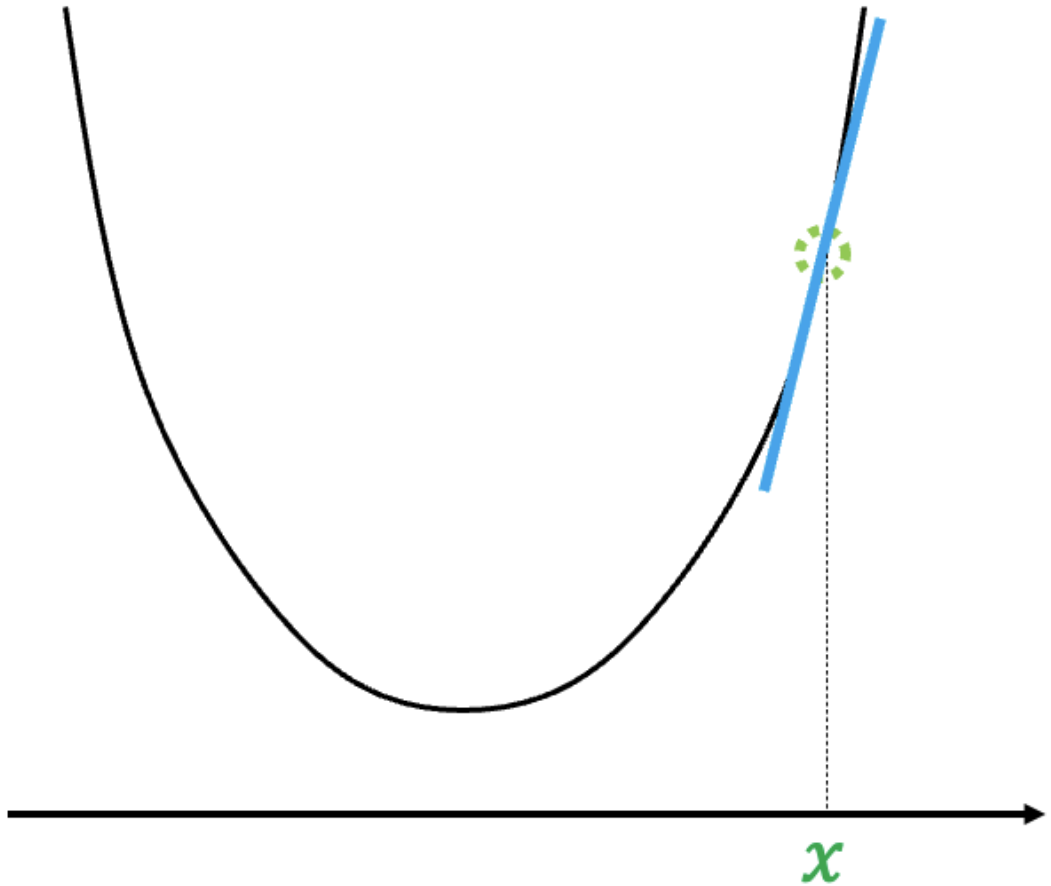


Gradient Descent: Core Idea

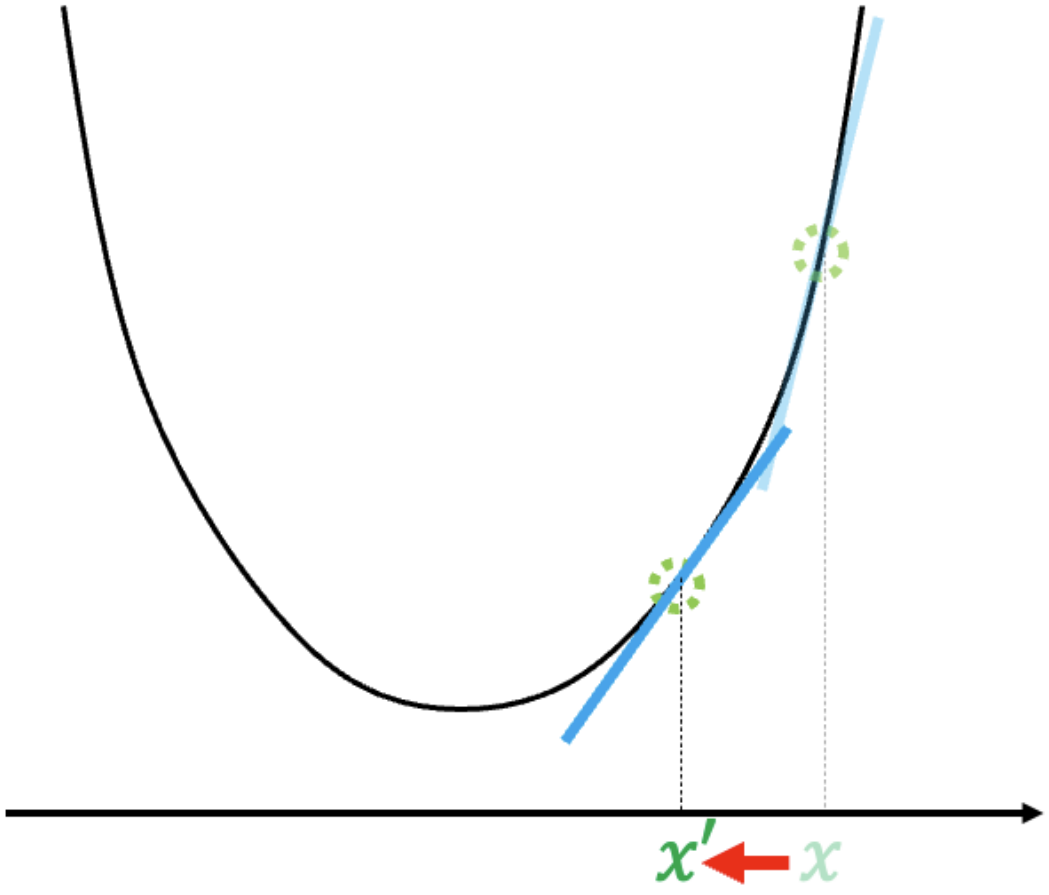


Gradient Descent: Core Idea

If slope is positive $\rightarrow$ decrease the parameter value.

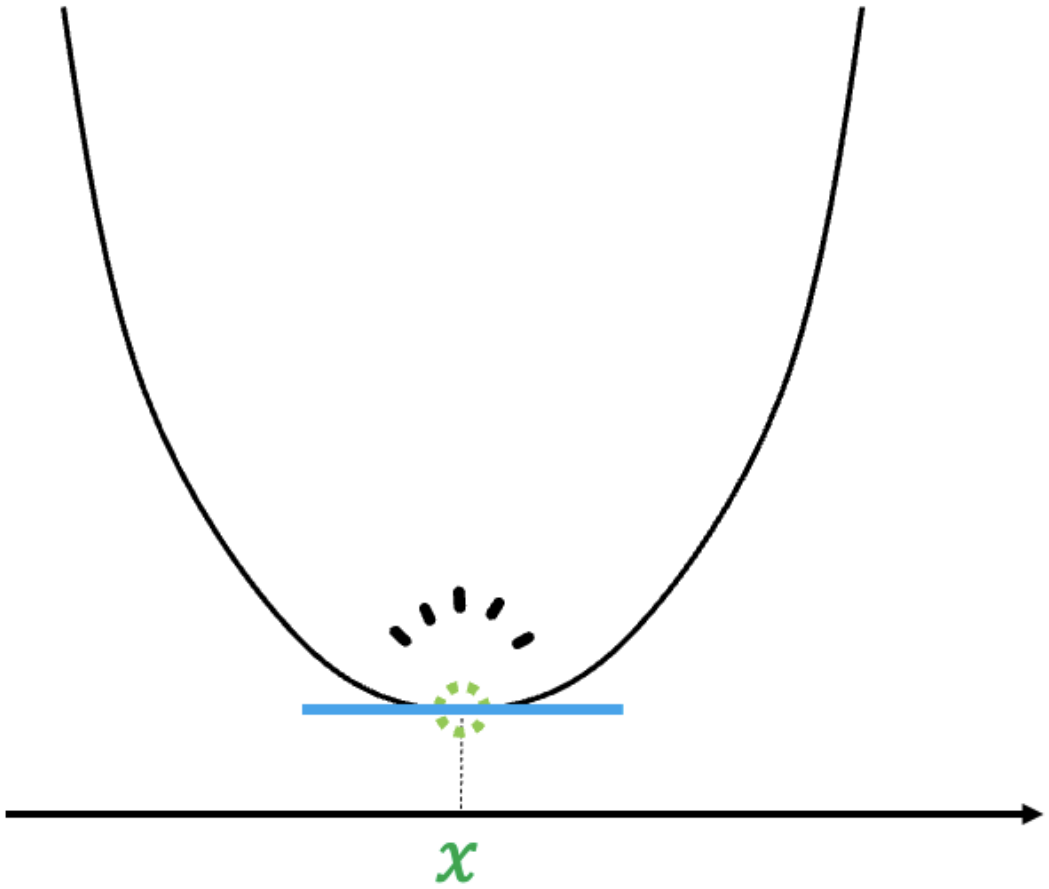


Gradient Descent: Core Idea



Gradient Descent: Core Idea

- Iteration continues until the slope converges to zero (minimum loss).



Learning Rules

Perceptron learning rule:

$$w_{\text{new}} = w_{\text{current}} + \eta \cdot x \cdot (y - \hat{y})$$

- w_{new}: updated weight
- w_{current}: current weight
- η: learning rate
- x: input value
- y: target (actual value)
- $\hat{y}$: predicted value
- $(y - \hat{y})$: error

Learning Rules

Gradient descent learning rule:

$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

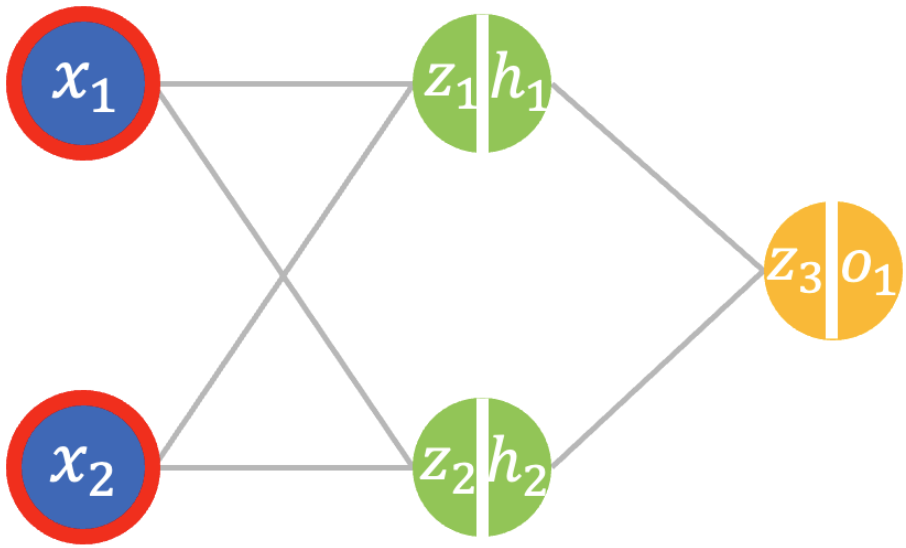
- L: loss function
- $\frac{\partial L}{\partial w}$: gradient of the loss function with respect to weight

Outline

Now, we have learned everything we need to understand how backpropagation.

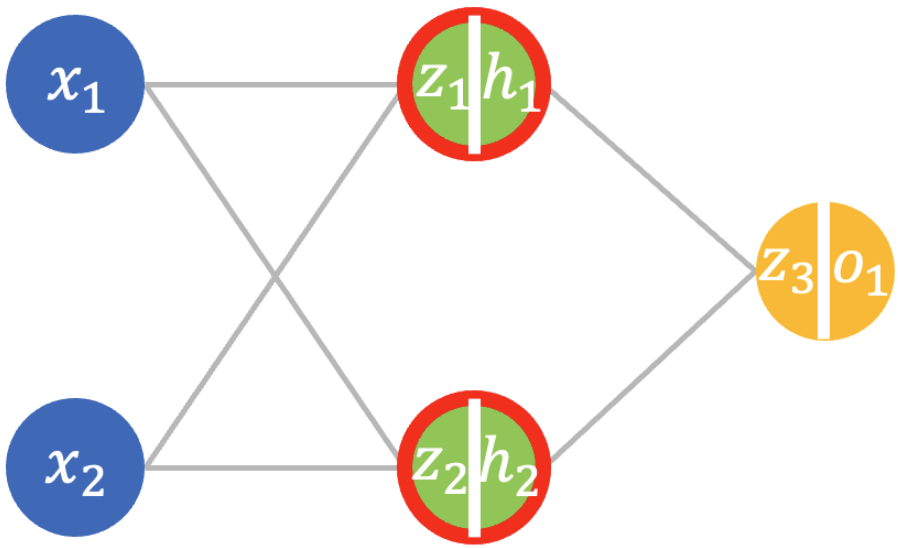
Setup: Input Layer

To understand the core of the backpropagation algorithm, we assume a simple multilayer neural network with two neurons in the input layer.



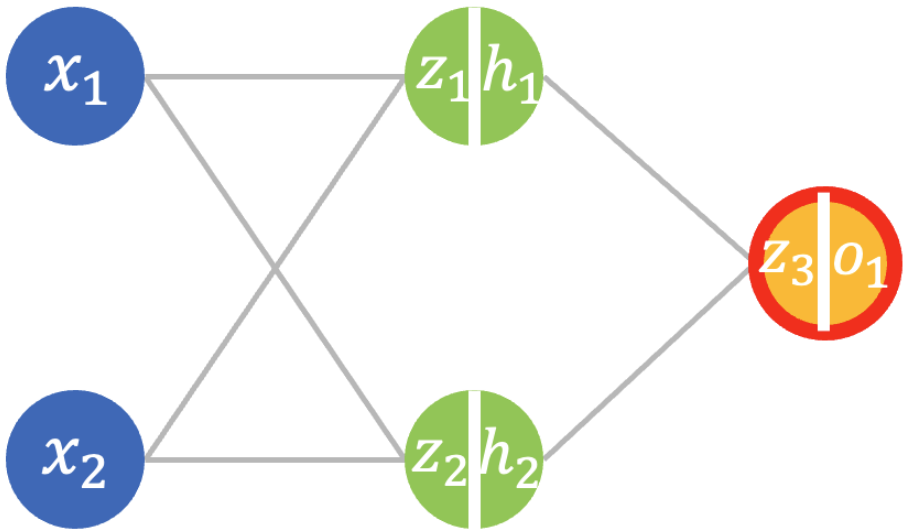
Setup: Hidden Layer

The hidden layer contains two neurons.



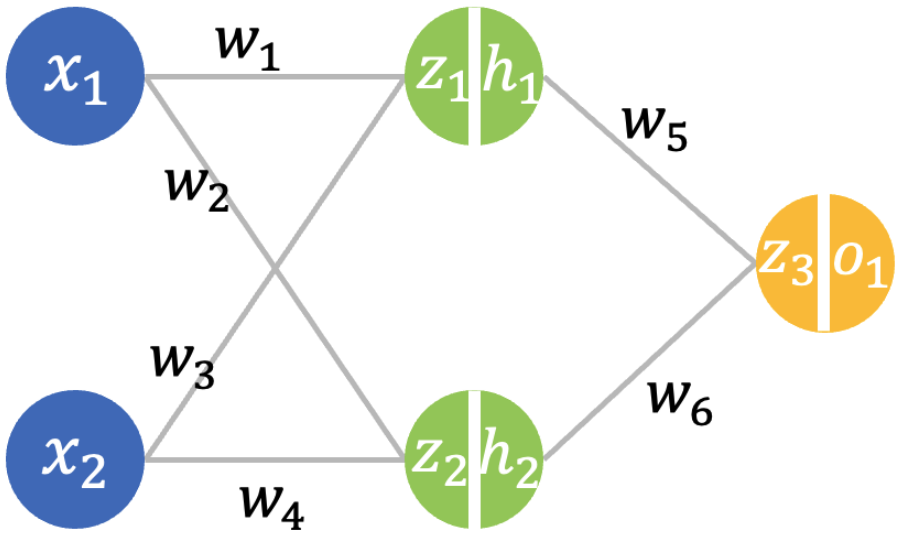
Setup: Output Layer

The output layer has one neuron.



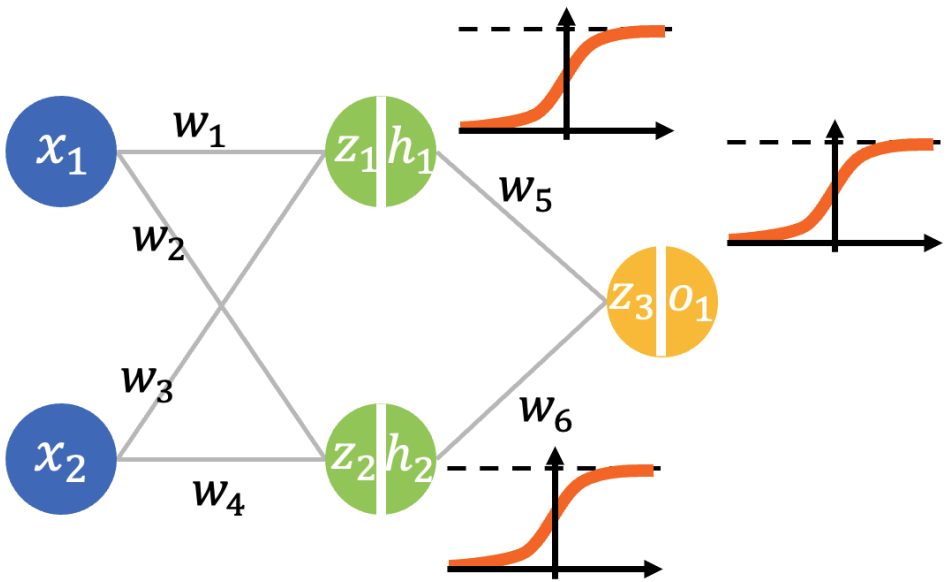
Simple Multilayer Neural Network

There are some weights.



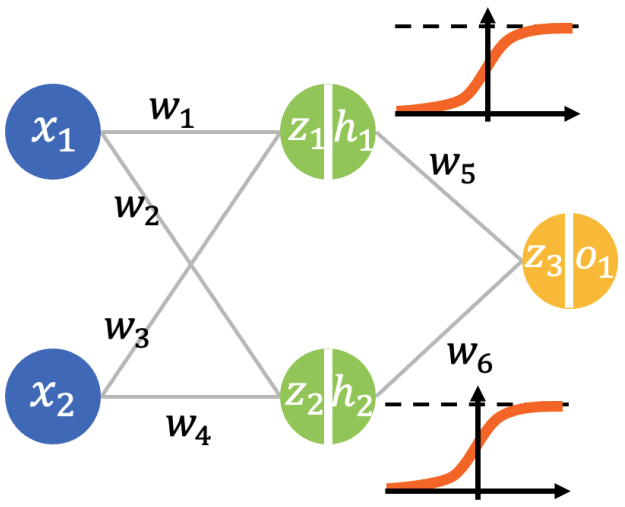
Activation Function

The activation function is the sigmoid.



Loss Function

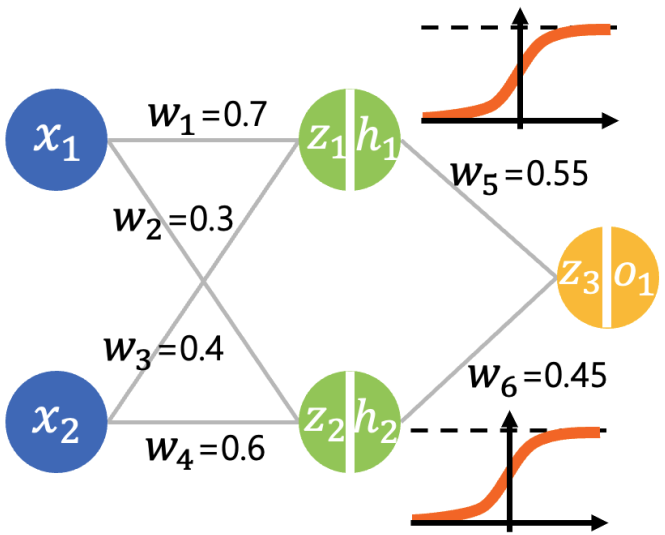
The loss function is Mean Squared Error (MSE).



$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Weight Initialization and Learning Rate

At the beginning, we suppose that all weights are initialized randomly. The learning rate is set to 0.1.



$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Learning Process

1. **Feedforward**: compute outputs
2. **Loss calculation**: evaluate error
3. **Backpropagation**: propagate errors backward

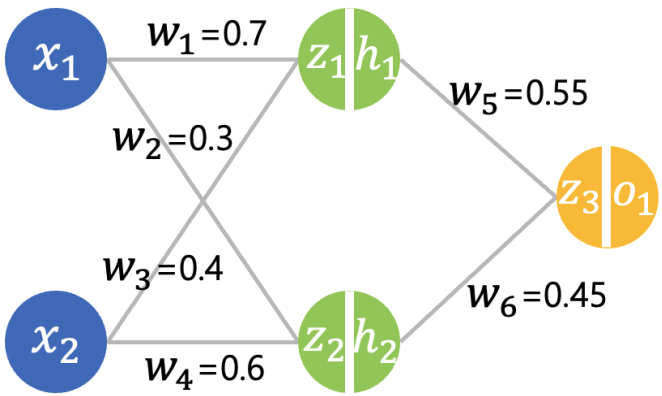
Learning Process

1. **Feedforward**: compute outputs
2. **Loss calculation**: evaluate error
3. **Backpropagation**: propagate errors backward

The process of finding the best parameters is called **learning (optimization)**.

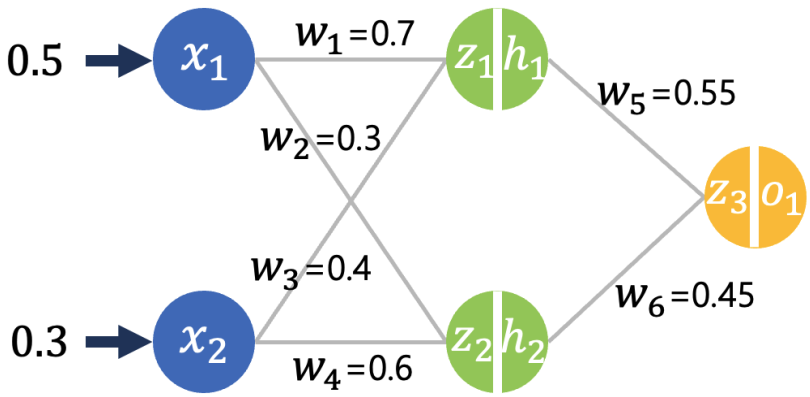
Step 1: Feedforward

The first step is the feedforward stage.



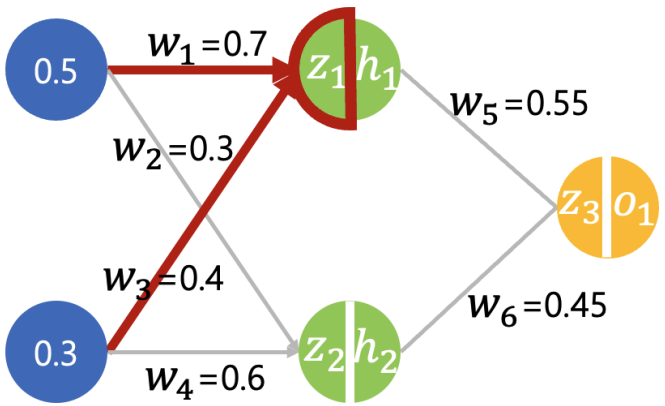
Input Values

The inputs are given as follows.



Weighted Sum to Hidden Node (1)

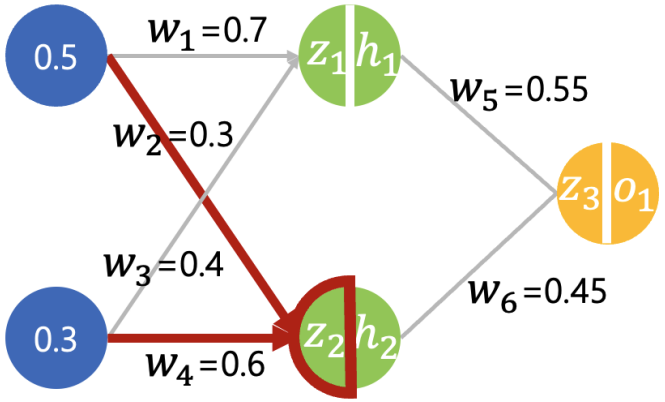
Each input is multiplied by its connection weight, and the results are summed into the hidden layer node.



$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7 + 0.3\text{x}0.4 = \mathbf{0.47}$

Weighted Sum to Hidden Node (2)

Each input is multiplied by its connection weight, and the results are summed into the hidden layer node.

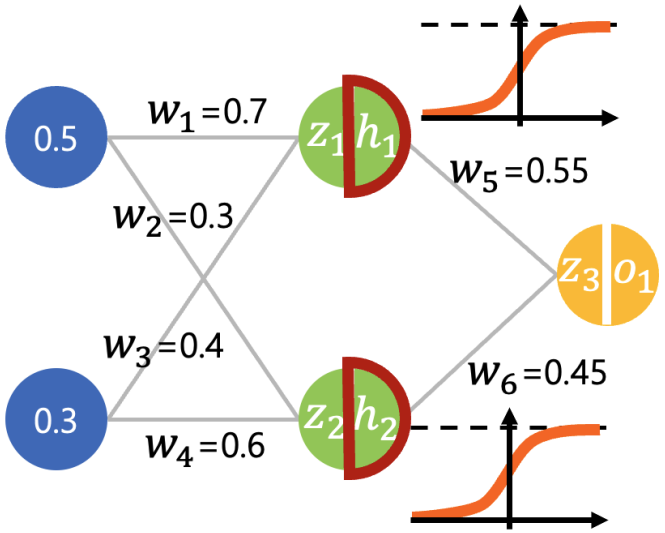


$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7 + 0.3\text{x}0.4 = \mathbf{0.47}$

$z_2 = x_1 w_2 + x_2 w_4 = 0.5\text{x}0.3 + 0.3\text{x}0.6 = \mathbf{0.33}$

Activation at Hidden Node

The activation function is applied to the hidden layer node.



$$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7 + 0.3\text{x}0.4 = 0.47$$

$$h_1 = sigmoid(z_1) = \mathbf{0.615}$$

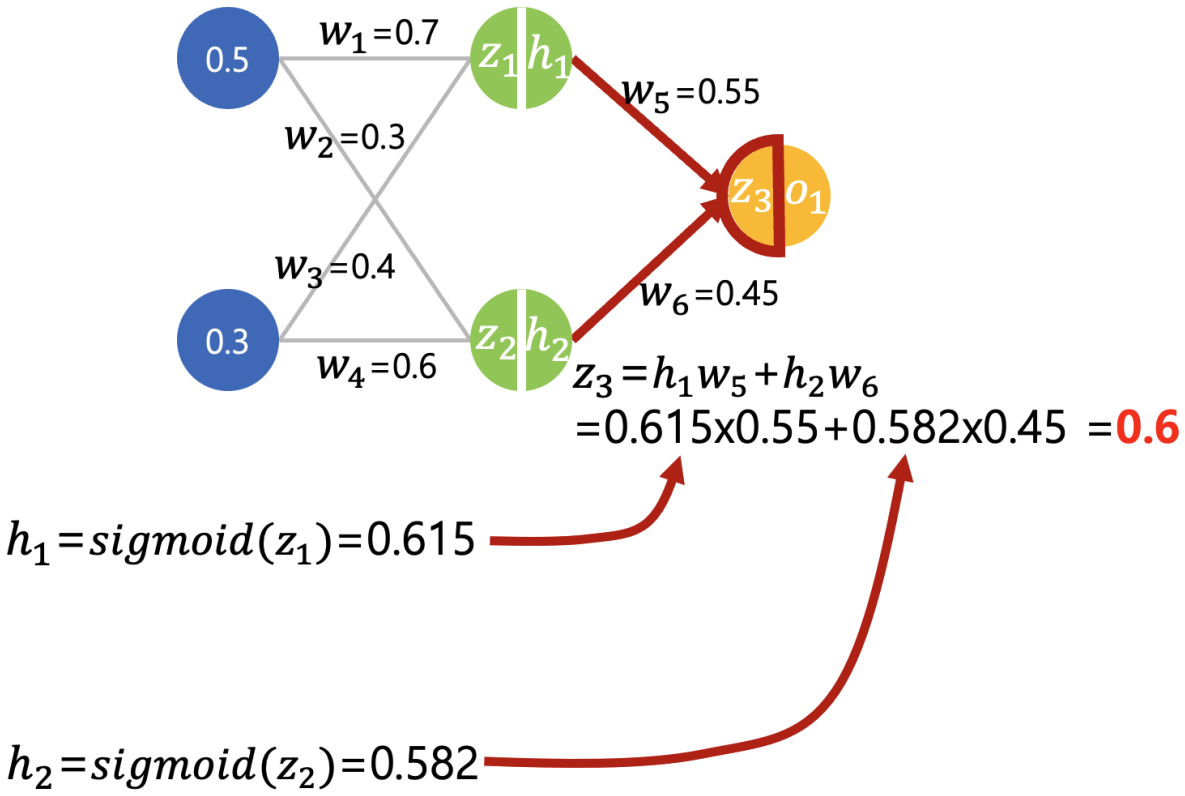
$$z_2 = x_1 w_2 + x_2 w_4 = 0.5\text{x}0.3 + 0.3\text{x}0.6 = 0.33$$

$$h_2 = sigmoid(z_2) = \mathbf{0.582}$$

calculator: https://www.tinkershop.net/ml/sigmoid_calculator.html

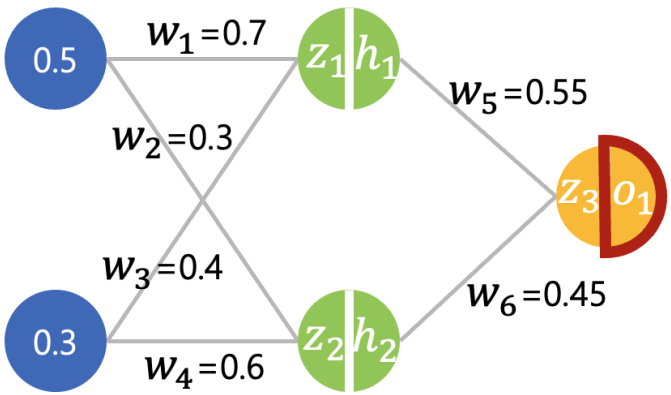
Weighted Sum to Output Neuron

The weighted inputs are summed and passed into the output layer neuron.



Final Output

The sigmoid function at the output layer produces the final output value.

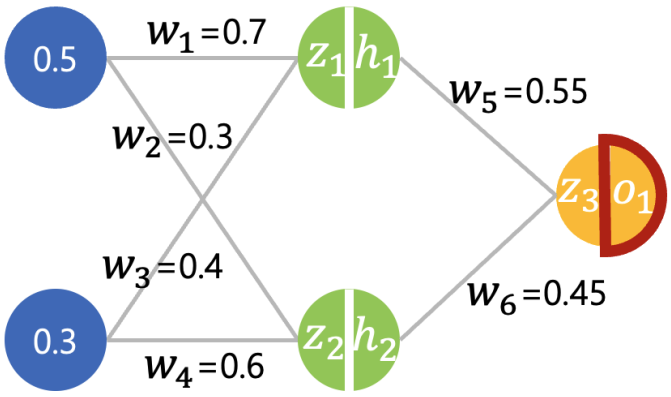


$o_1 = sigmoid(z_3) = \mathbf{0.645}$

$z_3 = h_1 w_5 + h_2 w_6$
$= 0.615 \times 0.55 + 0.582 \times 0.45 = 0.6$

Feedforward Completed

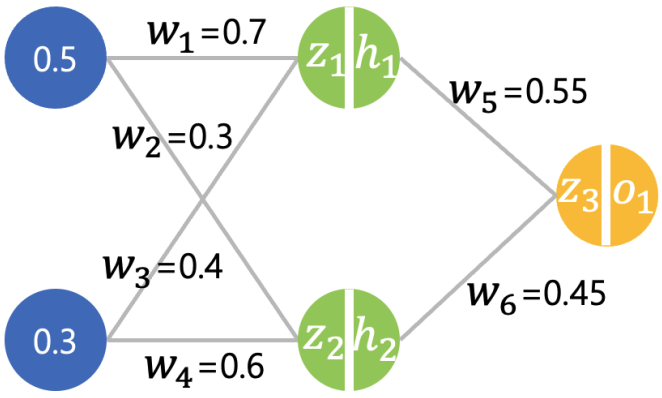
The feedforward stage is now complete.



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Step 2: Loss Calculation

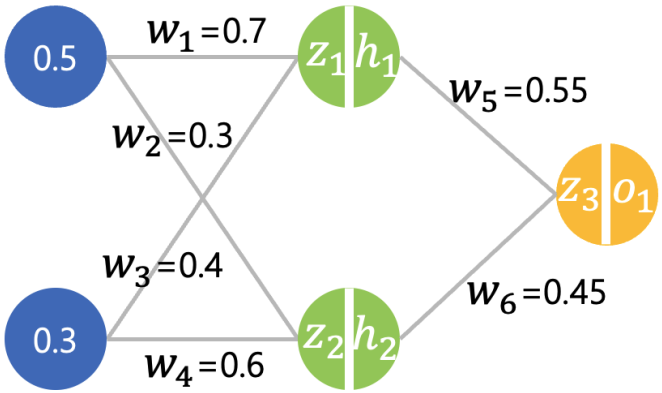
The second step is the loss calculation stage.



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Applying the Loss Function

Since we use MSE as the loss function, the output is substituted into the MSE formula.

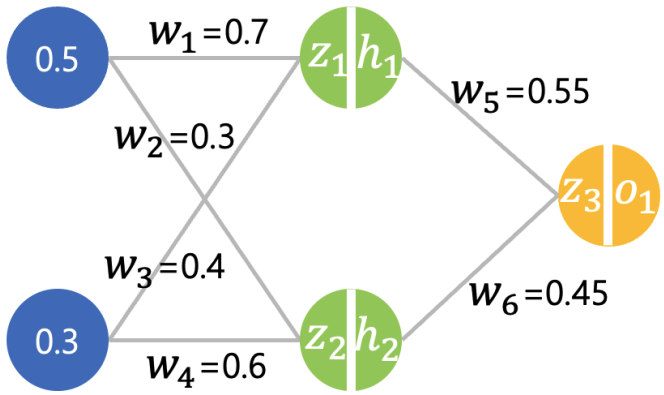


$o_1 = sigmoid(z_3) = \mathbf{0.645}$

$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

Single Output Neuron

Because there is only one actual output neuron, $n = 1$.

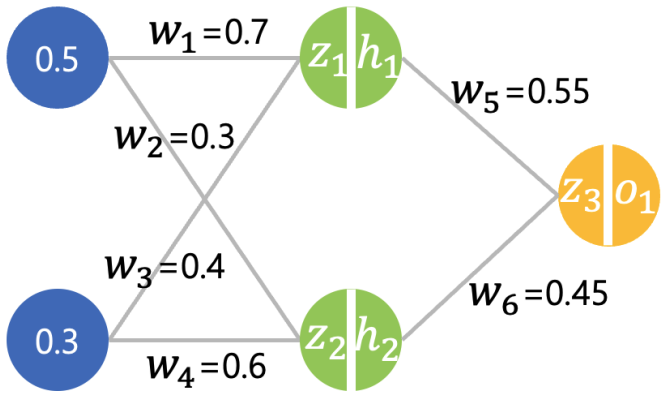


$o_1 = sigmoid(z_3) = \mathbf{0.645}$

$$C = \frac{1}{1}\sum_{i}^{1}(y_1 - \hat{y}_1)^2$$

Substituting the Output Value

The predicted value 0.645 is substituted into the MSE.

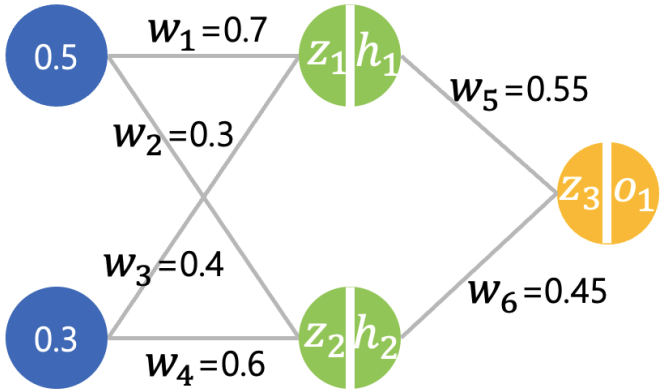


$$o_1 = sigmoid(z_3) = \mathbf{0.645}$$

$$C = \frac{1}{1}\sum_{i}^{1}(y_1 - \hat{y}_1)^2$$

Assumed Target Value

Suppose the actual target value is 1.

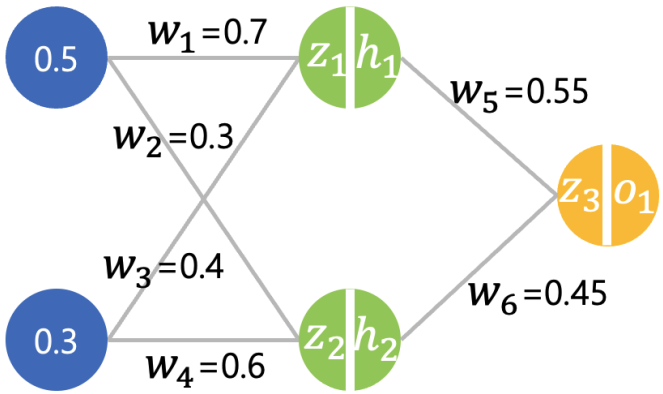


$$o_1 = sigmoid(z_3) = \mathbf{0.645}$$

$$C = \frac{1}{1}\sum_{i}^{1}(1 - 0.645)^2$$

Error Calculation

The error (C) is then calculated.

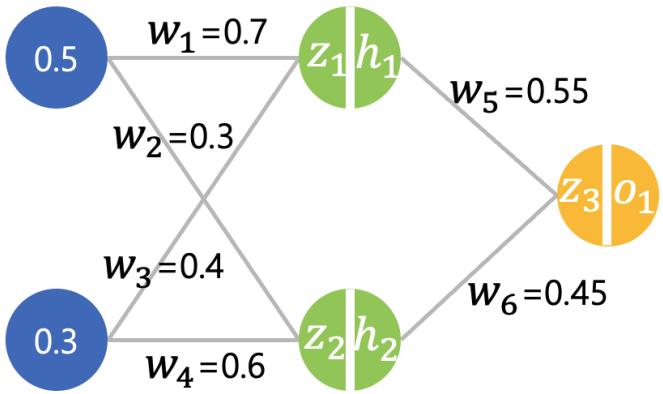


$o_1 = sigmoid(z_3) = 0.645$

$C = \mathbf{0.126}$

Step 3: Backpropagation

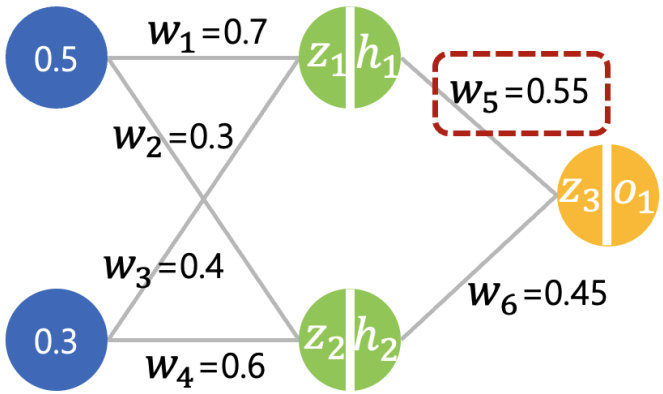
The third step is the backpropagation stage.



$$C = 0.126$$

Updating Weight w_5

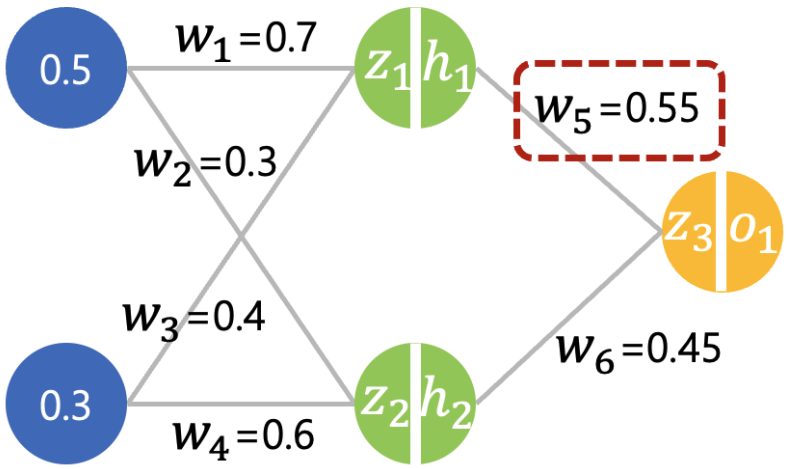
Using backpropagation, we update the weight w_5.



$$C = 0.126$$

Weight Update Rule

Recall the weight update formula in gradient descent.



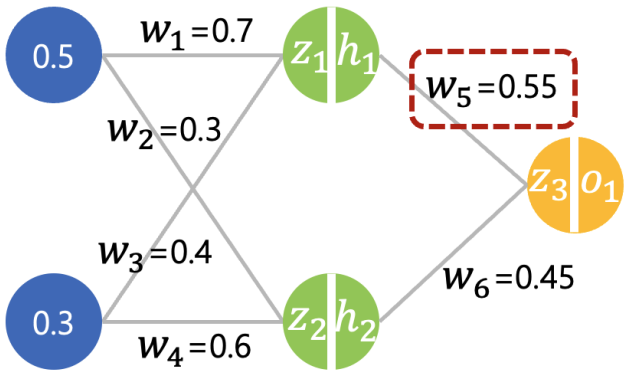
Gradient descent learning rule:

$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

- L: loss function
- $\frac{\partial L}{\partial w}$: gradient of the loss function w.r.t. weight

Derivative for w_5

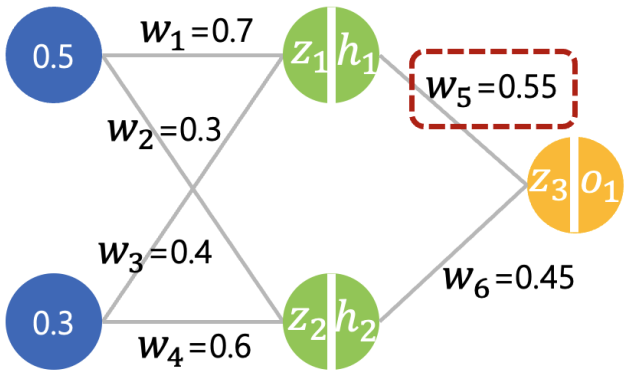
Therefore, to update w_5, we need to compute the following derivative.



$$\frac{\partial C}{\partial w_5}$$

Using the Chain Rule

Since this derivative cannot be computed directly, we apply the chain rule.



$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Chain Rule

The chain rule is the core of the backpropagation algorithm.

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Unknown Relationship

When we want to compute the derivative of two variables but do not know their direct relationship,

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Expanding with Known Derivatives

we can expand the expression step by step using known partial derivatives, solving the parts to obtain the overall derivative.

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Eliminating Intermediate Variables

In this way, intermediate variables are eliminated, leaving only the relationship we want to compute.

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Chain Rule: An Analogy

How many times faster is the cheetah than the human?

Chain Rule: An Analogy

How many times faster is the cheetah than the human?
I don't know...

Chain Rule: An Analogy

How many times faster is the cheetah than the human?
I don't know...
Now, you know:

- a cheetah is twice as fast as a lion,
- a lion is twice as fast as a bear,
- and a bear is 1.5 times faster than a *human*

Chain Rule: An Analogy

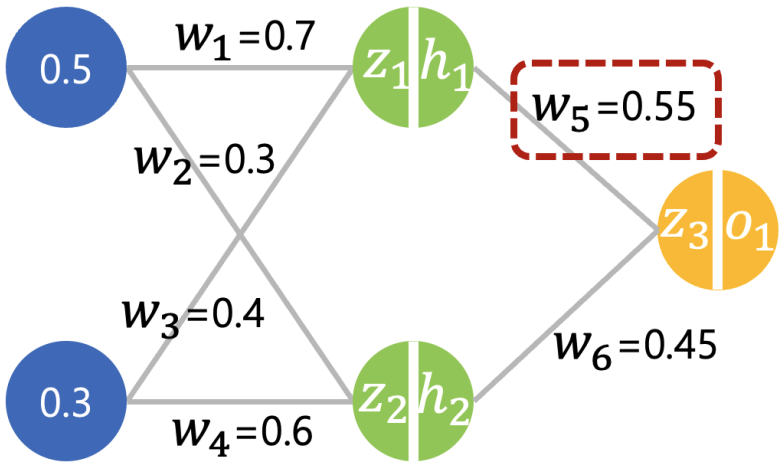
How many times faster is the cheetah than the human?
I don't know...
Now, you know:

- a cheetah is twice as fast as a lion,
- a lion is twice as fast as a bear,
- and a bear is 1.5 times faster than a *human*

How many times faster is the cheetah than the human?

Breaking Down the Parts

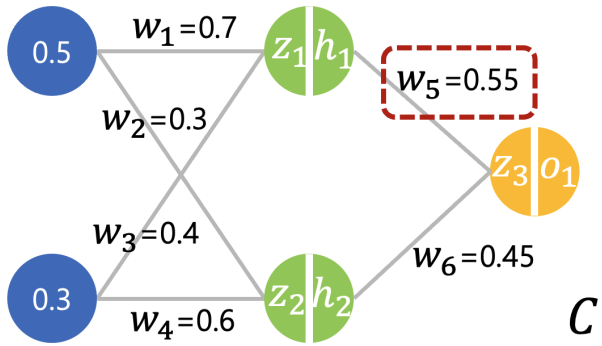
Therefore, we calculate the value by computing each part step by step.



$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

First Derivative

First, we compute the first derivative.

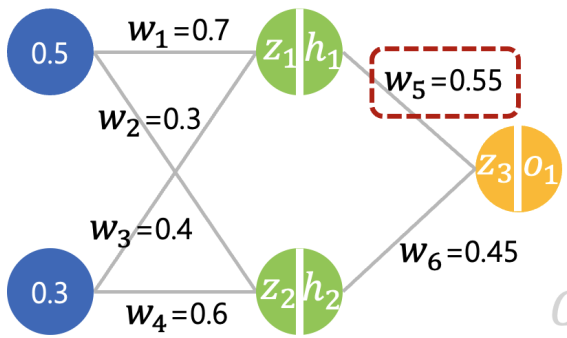


$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Case of $n = 1$

Since $n = 1$,



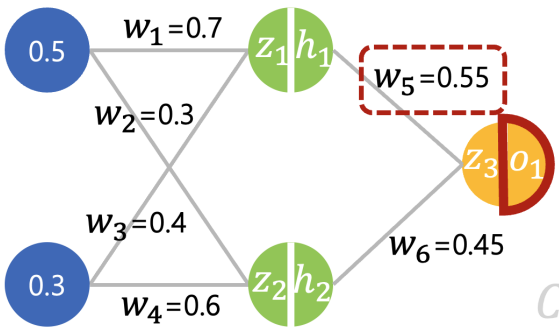
$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$C = (y - o_1)^2$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Step-by-Step Calculation (1)

Proceeding step by step, we obtain:



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

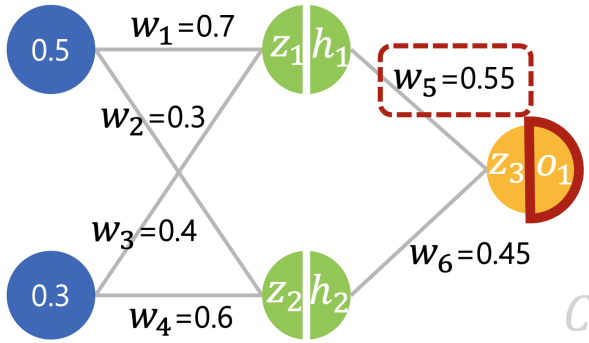
$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

$$C = (y - o_1)^2$$

$$\frac{\partial C}{\partial o_1} = 2(y - o_1)^{2-1} * (-1)$$

Step-by-Step Calculation (2)



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

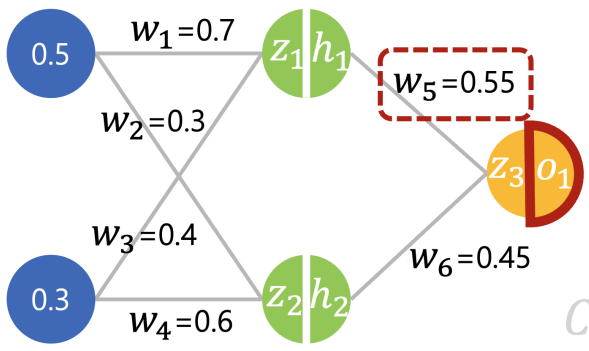
$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

$$C = (y - o_1)^2$$

$$\frac{\partial C}{\partial o_1} = -2(1 - 0.645)$$

Step-by-Step Calculation (3)



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

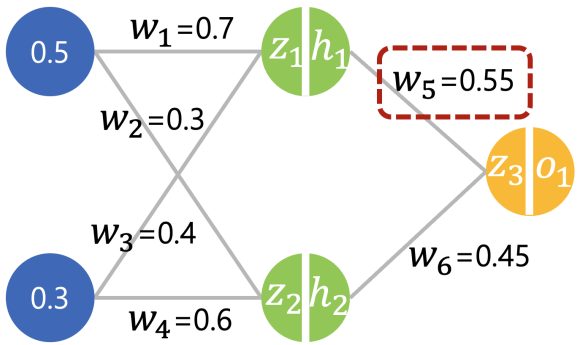
$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

$$C = (y - o_1)^2$$

$$\frac{\partial C}{\partial o_1} = -2(1 - 0.645) = -0.71$$

Second Derivative

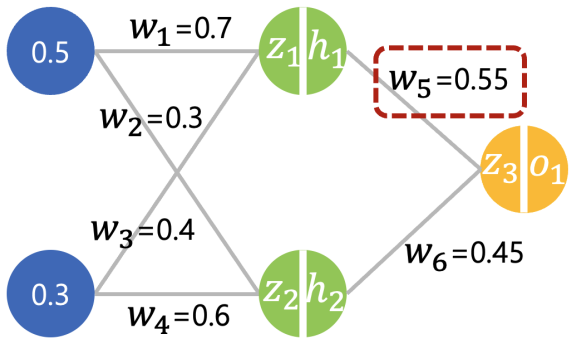
To compute the second derivative,



$$\frac{\partial C}{\partial w_5} = -0.71 \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Sigmoid in Feedforward

Recall that we used the sigmoid function during feedforward.



$o_1 = sigmoid(z_3) = \mathbf{0.645}$

$$\frac{\partial C}{\partial w_5} = -0.71 \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Sigmoid Formula

The mathematical formula of the sigmoid function is as follows:

$$S(x) = \frac{1}{1 + e^{-x}}$$

$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Using O and Z Variables

When expressed with O and Z variables, we obtain:

$$S(x) = \frac{1}{1 + e^{-x}}$$

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

$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Derivative of Sigmoid

The derivative of the sigmoid function is:

$$S(x) = \frac{1}{1+e^{-x}}$$

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

$$\frac{\partial O}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Simplified Expression

It can also be expressed as:

$$S(x) = \frac{1}{1 + e^{-x}}$$

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

$$\frac{\partial O}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1 + e^{-z}}\right) = O(z)(1 - O(z))$$

$o_1 = sigmoid(z_3) = \mathbf{0.645}$

Computing the Sigmoid Derivative

Since we already know the value of O_1, we can compute the derivative of the sigmoid.

$$S(x) = \frac{1}{1 + e^{-x}}$$

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

$$\frac{\partial O}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1 + e^{-z}}\right) = O(z)(1 - O(z))$$

$o_1 = sigmoid(z_3) = \mathbf{0.645}$

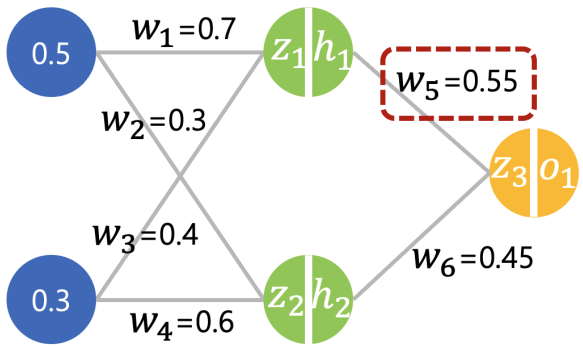
Step-by-Step Expansion (1)

$$S(x) = \frac{1}{1 + e^{-x}}$$

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

$$\frac{\partial O}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1 + e^{-z}}\right) = 0.645\,(1 - 0.645) = \mathbf{0.229}$$

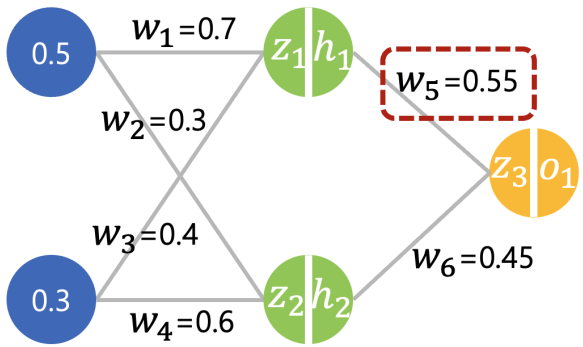
Step-by-Step Expansion (2)



$$\frac{\partial C}{\partial w_5} = -0.71 \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

$$0.645(1 - 0.645) = \mathbf{0.229}$$

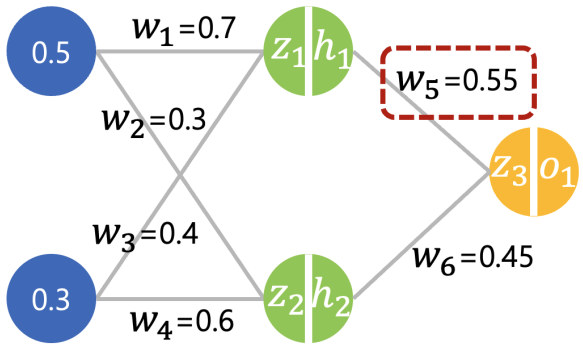
Step-by-Step Expansion (3)



$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Third Term

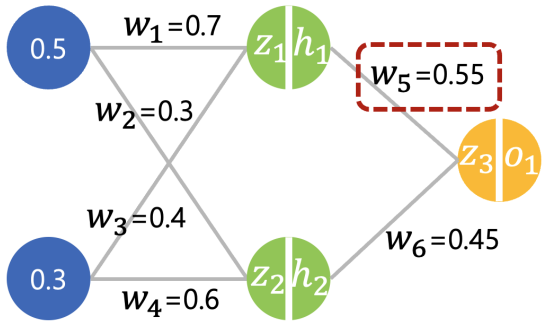
For the third term, we compute:



$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Formula for z

Using the formula for z,

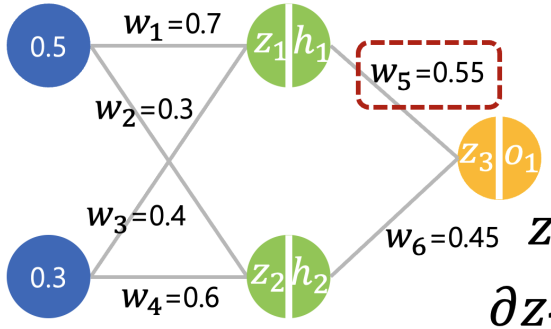


$$z_3 = h_1 w_5 + h_2 w_6$$

$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Partial Derivative of z_3

Taking the partial derivative of z_3 with respect to w_5 directly gives h_1.



$$z_3 = h_1 w_5 + h_2 w_6$$

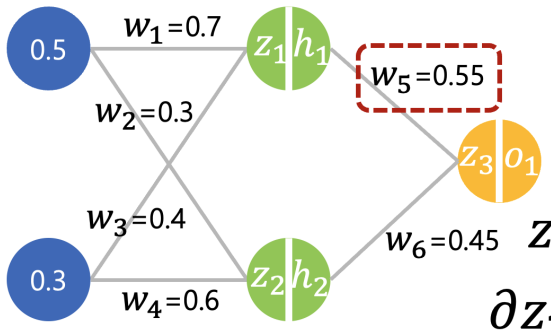
$$\frac{\partial z_3}{\partial w_5} = h_1 \cancel{w_5} + \cancel{h_2 w_6}$$

$$= h_1$$

$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Value of h_1

From the feedforward calculation, $h_1 = 0.615$.



$$z_3 = h_1 w_5 + h_2 w_6$$

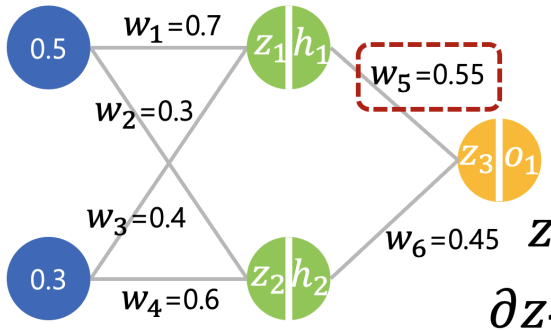
$$\frac{\partial z_3}{\partial w_5} = h_1 \cancel{w_5} + \cancel{h_2 w_6}$$

$$= h_1 = 0.615$$

$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Substituting Values

Now, substituting the values, we obtain:



$$z_3 = h_1 w_5 + h_2 w_6$$

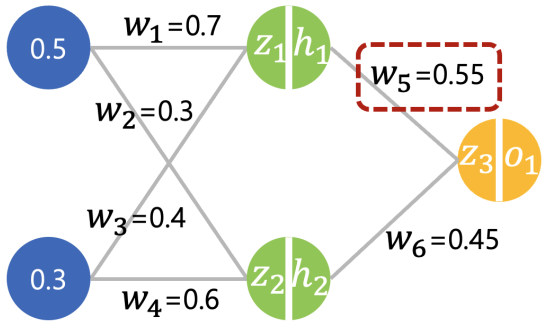
$$\frac{\partial z_3}{\partial w_5} = h_1 \cancel{w_5} + \cancel{h_2 w_6}$$

$$= h_1 = 0.615$$

$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Gradient of the Loss Function

Finally, we can compute the gradient of the loss function.



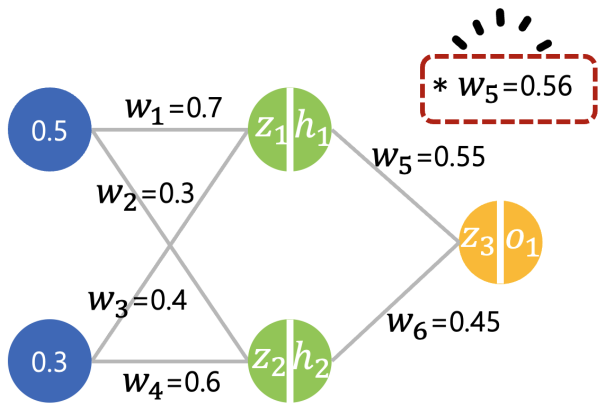
$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot 0.615 = \text{-0.1}$$

Weight Update with Gradient Descent

According to the gradient descent learning rule:

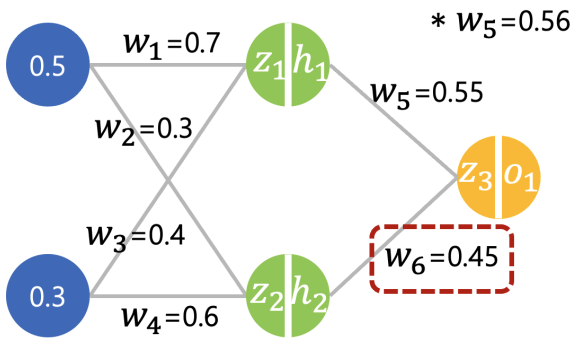
$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

The new weight $= 0.55 - (-0.1) \cdot 0.1 = 0.56$



Updating Weight w_6

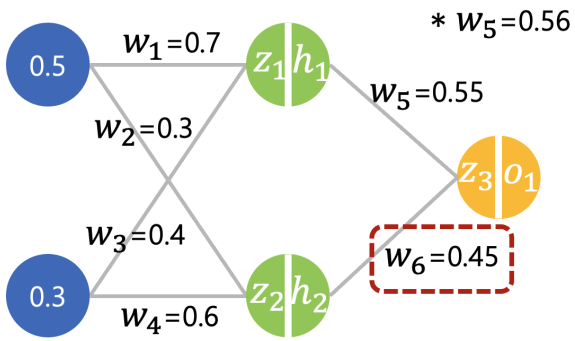
Now, let us update w_6 as well.



$$\frac{\partial C}{\partial w_6}$$

Using the Chain Rule Again

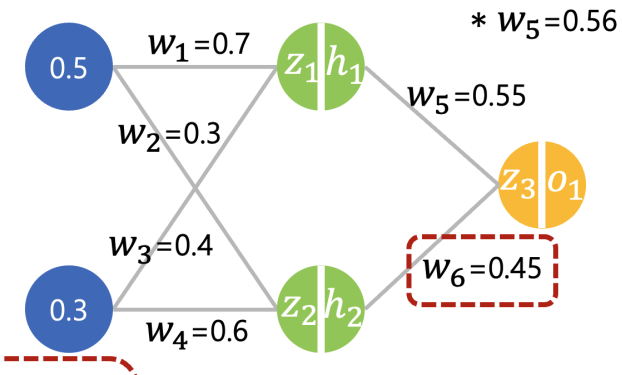
By the chain rule, we can similarly derive the expression.



$$\frac{\partial C}{\partial w_6} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_6}$$

Same as w_5 Formula

The previous steps are the same as for the w_5 calculation.

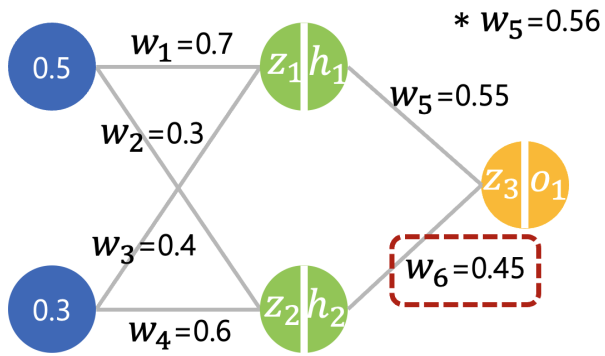


$$\frac{\partial C}{\partial w_6} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_6}$$

Calculated equally as the previous equation

Substituting Values

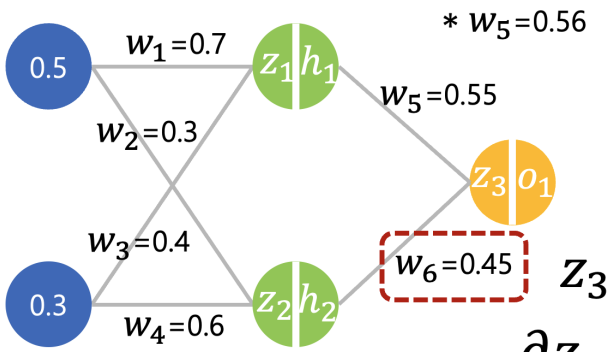
Now, substituting the values, we obtain:



$$\frac{\partial C}{\partial w_6} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_6}$$

Third Term

The third term once again becomes h_2.



$$z_3 = h_1 w_5 + h_2 w_6$$

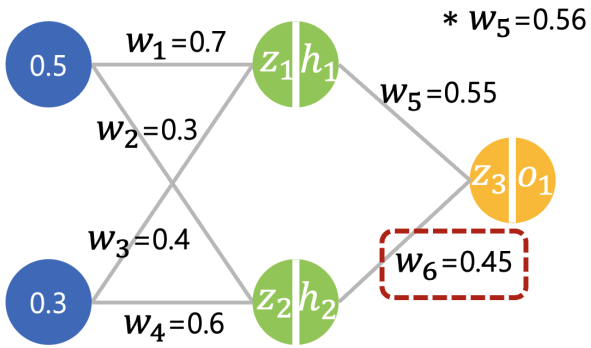
$$\frac{\partial z_3}{\partial w_6} = \cancel{h_1 w_5} + h_2 \cancel{w_6}$$

$$= h_2 = 0.582$$

$$\frac{\partial C}{\partial w_6} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_6}$$

Gradient for w_6

The gradient of the loss function is -0.095.



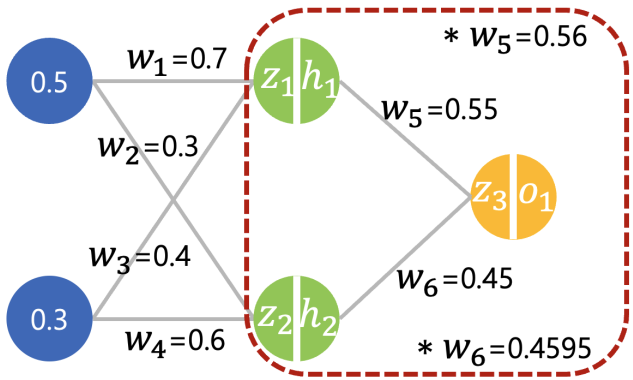
$$\frac{\partial C}{\partial w_6} = -0.71 \cdot 0.229 \cdot 0.582 = -0.095$$

Weight Update with Gradient Descent

According to the gradient descent learning rule:

$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

The new weight $= 0.45 - (-0.095) \cdot 0.1 = 0.4595$



Other Weights

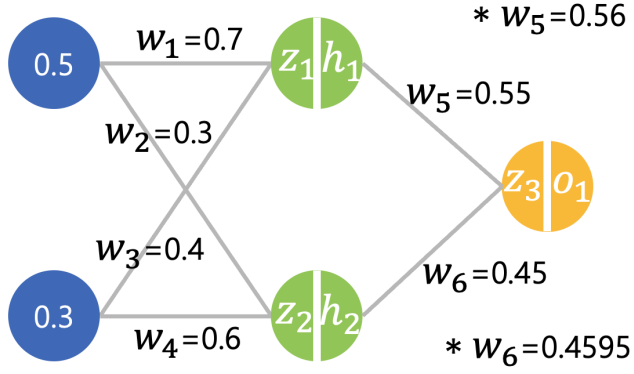
The remaining weights in the first layer are updated in the same way.

* w_1=0.7010

* w_2=0.3009

* w_3=0.4011

* w_4=0.6005

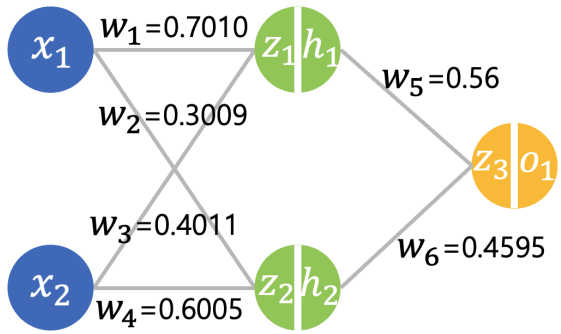


* w_5=0.56

* w_6=0.4595

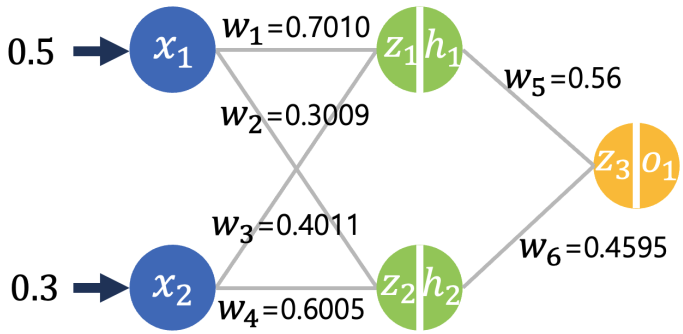
Checking Error After Backpropagation

Now, let us check whether the network error has decreased after backpropagation.



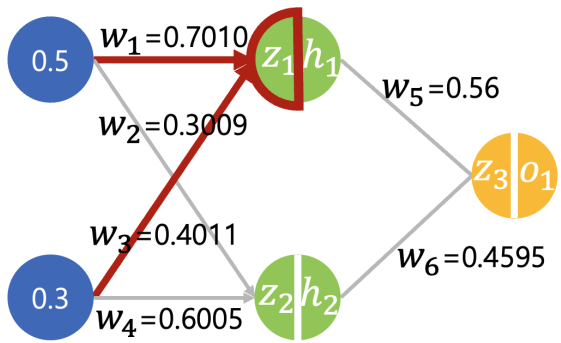
Same Input Values

We feed in the same input values again.



Hidden Node z_1

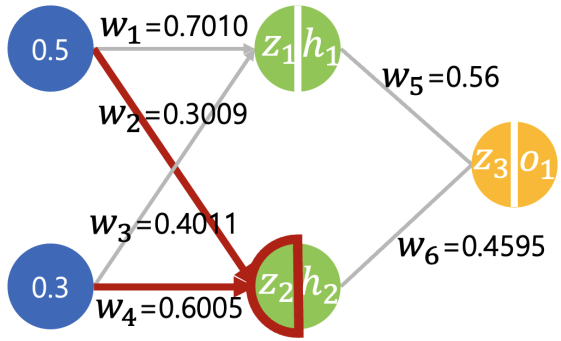
Weighted sums are passed into hidden node z_1.



$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7010 + 0.3\text{x}0.4011 = \mathbf{0.4708}$

Hidden Node z_2

Weighted sums are passed into hidden node z_2.

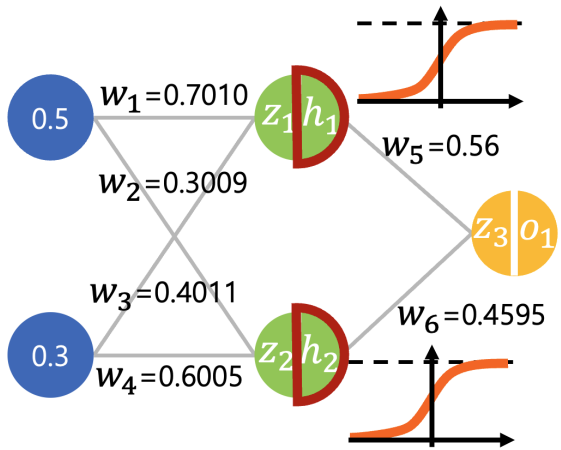


$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7010 + 0.3\text{x}0.4011 = \mathbf{0.4708}$

$z_2 = x_1 w_2 + x_2 w_4 = 0.5\text{x}0.3009 + 0.3\text{x}0.6005 = \mathbf{0.3306}$

Hidden Layer Activation

The activation function is applied at the hidden nodes.



$z_1 = x_1 w_1 + x_2 w_3 = 0.5 \text{x} 0.7010 + 0.3 \text{x} 0.4011 = \mathbf{0.4708}$

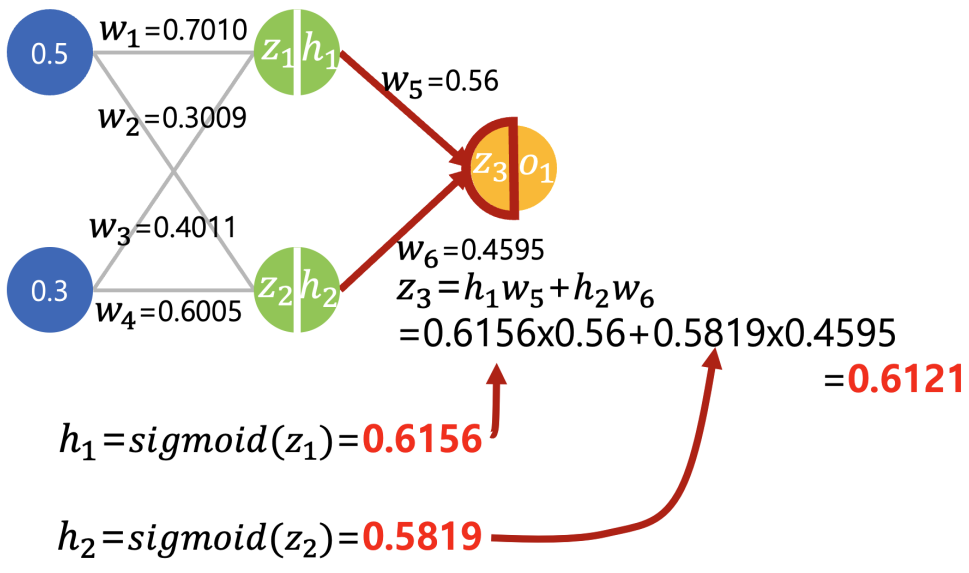
$h_1 = sigmoid(z_1) = \mathbf{0.6156}$

$z_2 = x_1 w_2 + x_2 w_4 = 0.5 \text{x} 0.3009 + 0.3 \text{x} 0.6005 = \mathbf{0.3306}$

$h_2 = sigmoid(z_2) = \mathbf{0.5819}$

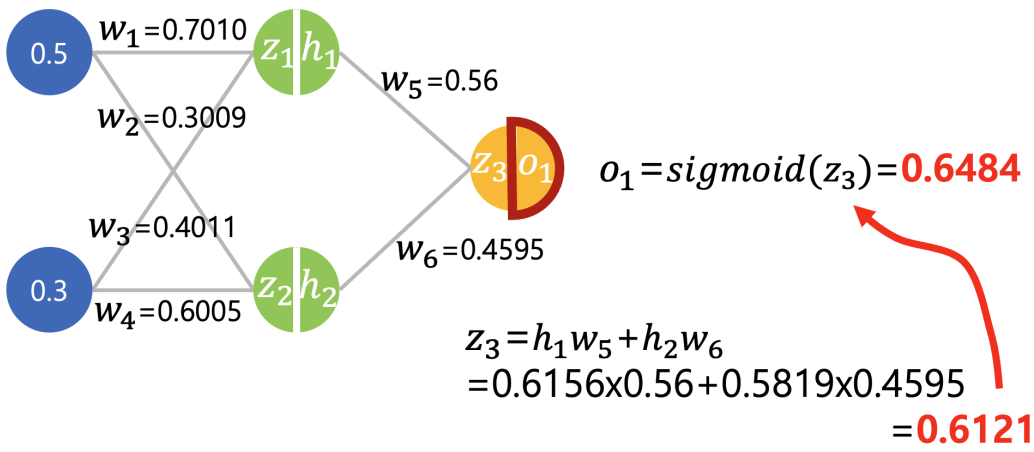
Output Node Weighted Sum

The weighted inputs are summed at the output node.



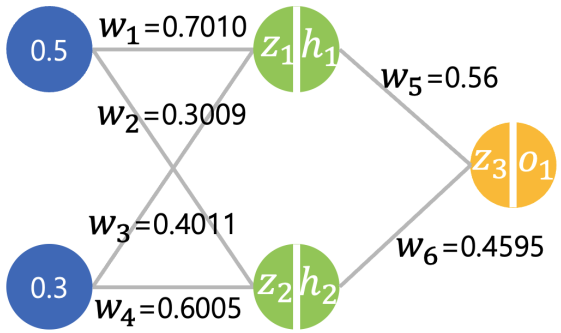
Final Output

Finally, the sigmoid function at the output layer produces the final output.



Loss Calculation (1)

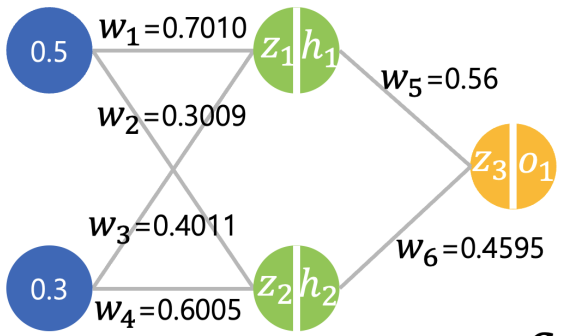
The output value is substituted into the loss function.



$$o_1 = sigmoid(z_3) = \mathbf{0.6484}$$

$$C = \frac{1}{1}\sum_{i}^{1}(y_1 - \hat{y}_1)^2$$

Loss Calculation (2)

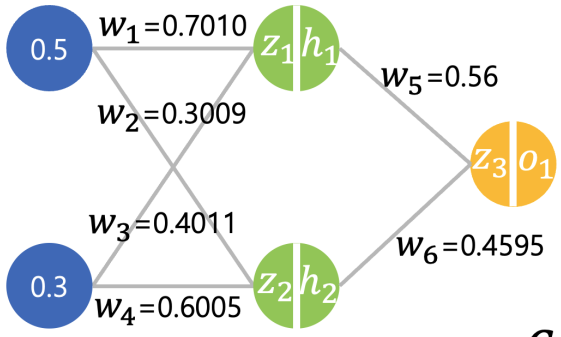


$o_1 = sigmoid(z_3) = \mathbf{0.6484}$

$$C = \frac{1}{1}\sum_{i}^{1}(1 - 0.6484)^2$$

Error Comparison

Comparing with the previous error $C = 0.126$, the error is reduced.



$o_1 = sigmoid(z_3) = 0.6443$

$$C = \frac{1}{1}\sum_{i}^{1}(1 - 0.6484)^2$$

$$C = \mathbf{0.1236}$$

Training Process

After this, the same cycle of feedforward, loss calculation, and backpropagation is repeated until the error reaches a minimum, at which point training stops.

From Scalar Derivatives to Matrices

So far, we have considered the case of a **single output**, where the gradient with respect to the weights is just a scalar derivative.

But when the function has **multiple outputs**, the derivative generalizes to a **matrix**.

Jacobian Matrix (Just the Idea!)

- When the function has **multiple outputs**, the gradient becomes a **matrix**.
- **Jacobian** matrix:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Why Learn About Gradients?

- Modern deep learning frameworks like **PyTorch** and compute gradients for you!
- But knowing how gradients work helps you:
  - Understand what's going on under the hood
  - Debug unexpected behavior
  - Design better models and training routines
- Want to see this in action? Lab 3: *Pytorch*