5. Neural Network

LING-581-Natural Language Processing 1

Instructor: Hakyung Sung

September 9, 2025

*Acknowledgment: These course slides are based on materials from CS224N @ Stanford University; Dr. Kilho Shin @ Kyocera

Table of contents

Review

Word vectors

- One-hot encoding (Bag-of-words representation)

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model
    - Word2Vec

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model
    - Word2Vec
    - GloVe

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model
    - Word2Vec
    - GloVe
- Evaluation (External, Internal)

Word vectors

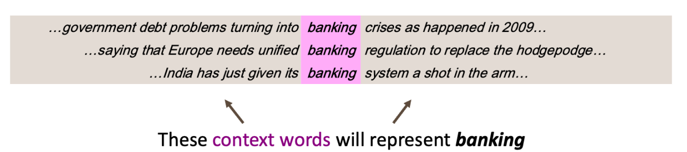
- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model
  - Word2Vec
  - GloVe
- Evaluation (External, Internal)
- Word meanings can be represented well by a high-dimensional vector of real numbers

Word vectors

- One-hot encoding (Bag-of-words representation)
- Count-based model
- Neural network-based model
    - Word2Vec
    - GloVe
- Evaluation (External, Internal)
- Word meanings can be represented well by a high-dimensional vector of real numbers
- **Linguistic idea:** A word's meaning is given by the words that frequently appear close-by

Distributional semantics

- "You shall know a word by the company it keeps" (Firth, 1957) - One of the most successful ideas of modern statistical NLP.



Lesson plan

Lesson plan

- ~~Review~~

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)
- Artificial neural network (10 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)
- Artificial neural network (10 mins)
- Perceptrons (10 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)
- Artificial neural network (10 mins)
- Perceptrons (10 mins)
- Multi-layer perceptrons (10 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)
- Artificial neural network (10 mins)
- Perceptrons (10 mins)
- Multi-layer perceptrons (10 mins)
- Gradient descendant and loss function (10 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

Lesson plan

- ~~Review~~
- GloVe (5 mins)
- Artificial neural network (10 mins)
- Perceptrons (10 mins)
- Multi-layer perceptrons (10 mins)
- Gradient descendant and loss function (10 mins)
- Backpropagation (15 mins)

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.

GloVe

Revisit: Count-based & Neural-based models

- Count-based
  - Fast training
  - Efficient usage of statistics
  - Primarily used to capture word similarity
- Neural-based
  - Scales with corpus size
  - Inefficient usage of statistics (e.g., random sampling)

Motivation: Encoding meaning via co-occurrence ratios

- Idea: Meaning differences between words can be reflected in the **ratios** of their co-occurrence probabilities with other words.
- GloVe leverages these ratios to learn word vectors where *vector differences encode semantic components*.

Motivation: Encoding meaning via co-occurrence ratios

Example: *ice* vs. *steam*

- x = a context word (e.g., *solid*, *gas*, *water*, *random*)
- Compare $P(x \mid \text{ice})$ and $P(x \mid \text{steam})$

	x = solid	x = gas	x = water	x = random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$	large	small	≈ 1	≈ 1

Interpretation?

- “solid” with “ice”, “gas” with ”steam” → strong contrast

Motivation: Encoding meaning via co-occurrence ratios

Example: *ice* vs. *steam*

- x = a context word (e.g., *solid*, *gas*, *water*, *random*)
- Compare $P(x \mid \text{ice})$ and $P(x \mid \text{steam})$

	x = solid	x = gas	x = water	x = random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$	large	small	≈ 1	≈ 1

Interpretation?

- “solid” with “ice”, “gas” with ”steam” → strong contrast
- For neutral words (e.g., “water”, “random”), both words co-occur similarly → ratio ≈ 1

Motivation: Encoding meaning via co-occurrence ratios

Example: *ice* vs. *steam*

- x = a context word (e.g., *solid*, *gas*, *water*, *random*)
- Compare $P(x \mid \text{ice})$ and $P(x \mid \text{steam})$

	x = solid	x = gas	x = water	x = random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$	large	small	≈ 1	≈ 1

Interpretation?

- "solid" with "ice", "gas" with "steam" → strong contrast
- For neutral words (e.g., "water", "random"), both words co-occur similarly → ratio ≈ 1
- These ratio patterns can encode **semantic differences**.

GloVe's goal:

Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

How can we capture *ratios of co-occurrence probabilities* as **linear meaning components** in a word vector space?

Represent word meaning differences with vector differences:

$$\vec{w}_x \cdot (\vec{w}_a - \vec{w}_b) = \log \frac{P(x \mid a)}{P(x \mid b)}$$

GloVe's goal:

Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

How can we capture *ratios of co-occurrence probabilities* as **linear meaning components** in a word vector space?

Represent word meaning differences with vector differences:

$$\vec{w}_x \cdot (\vec{w}_a - \vec{w}_b) = \log \frac{P(x \mid a)}{P(x \mid b)}$$

- Example: $x = \text{solid}$ → appears much more often with "ice" than with "steam" ⇒ inner product is positive

GloVe's goal:

Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

How can we capture *ratios of co-occurrence probabilities* as **linear meaning components** in a word vector space?

Represent word meaning differences with vector differences:

$$\vec{w}_x \cdot (\vec{w}_a - \vec{w}_b) = \log \frac{P(x \mid a)}{P(x \mid b)}$$

- Example: $x =$ solid → appears much more often with "ice" than with "steam" ⇒ inner product is positive
- Example: $x =$ gas → appears more often with "steam" than with "ice" ⇒ inner product is negative

GloVe's goal:

Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

How can we capture *ratios of co-occurrence probabilities* as **linear meaning components** in a word vector space?

Represent word meaning differences with vector differences:

$$\vec{w}_x \cdot (\vec{w}_a - \vec{w}_b) = \log \frac{P(x \mid a)}{P(x \mid b)}$$

- Example: $x =$ solid → appears much more often with "ice" than with "steam" ⇒ inner product is positive
- Example: $x =$ gas → appears more often with "steam" than with "ice" ⇒ inner product is negative
- Meaning difference (ice vs steam) is captured as a **vector difference**, and other words (x) can be placed accordingly.

More on GloVe

Loss Function:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(\vec{w}_i^\top \tilde{\vec{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

- X_{ij}: Co-occurrence count of word i with context word j
- $f(X_{ij})$: Weighting function to discount rare and overly frequent co-occurrences
- $b_i, \tilde{b}_j$: Bias terms for words and contexts

More on GloVe

Interpretation (Regression view):

- *Target*: $\log X_{ij}$
- *Prediction*: $\vec{w}_i^\top \vec{\tilde{w}}_j + b_i + \tilde{b}_j$
- *Error*: squared difference between prediction and target
- *Weighting*: $f(X_{ij})$ adjusts importance (rare vs. frequent pairs)

$\Rightarrow$ GloVe solves a **weighted least squares regression** problem, where word–context vectors approximate the **log co-occurrence counts**.

More on GloVe: Advantages

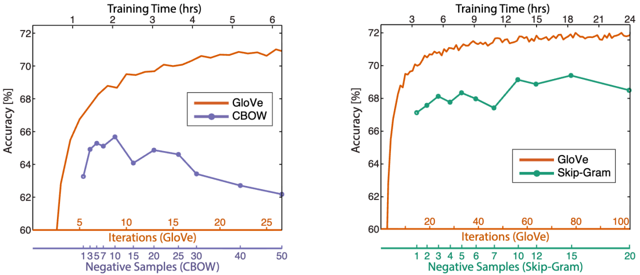
- Fast training

More on GloVe: Advantages

- Fast training
- Scales well to large corpora

More on GloVe: Advantages

- Fast training
- Scales well to large corpora
- Good performance even with small corpus / small vector size



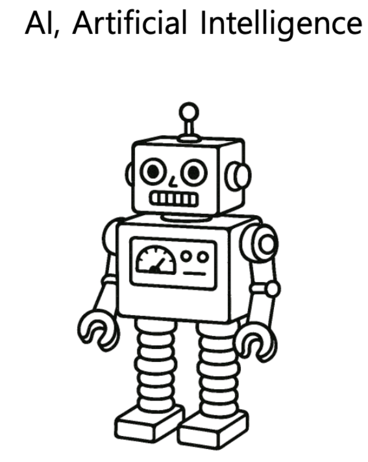
(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

Figure 1: Pennington et al. (2014)

Artificial neural network

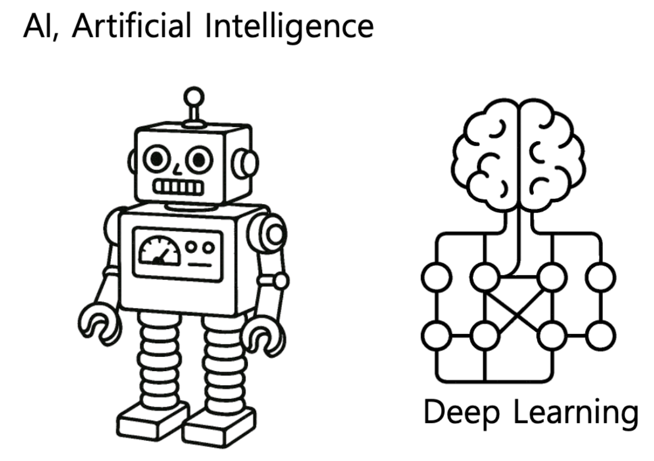
Artificial neural network

Creating machines that can think and communicate like humans has been a long-standing dream of humanity.



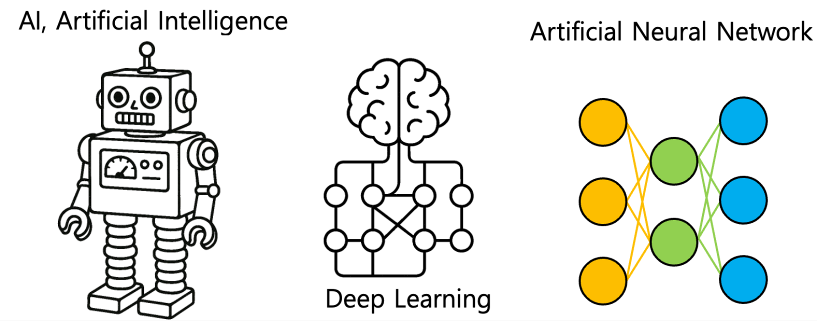
Artificial neural network

Today, artificial intelligence is largely based on machine learning, especially deep learning technologies.



Artificial neural network

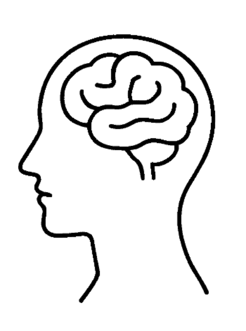
At the foundation of deep learning lies the artificial neural network, which serves as the starting point for understanding deep learning.



NLP: neural networks involve in word embeddings, recurrent neural networks, Transformer models

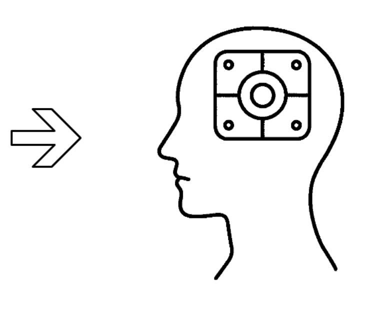
Understanding human brain

Artificial neural networks are computer programs designed to mimic the human brain.



Understanding human brain

Artificial neural networks are computer programs designed to mimic the human brain.

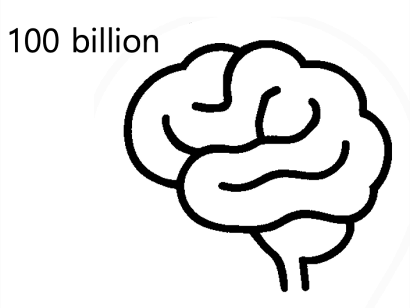
Therefore, understanding how the human brain works is the very first step.

Neuron and artificial neuron

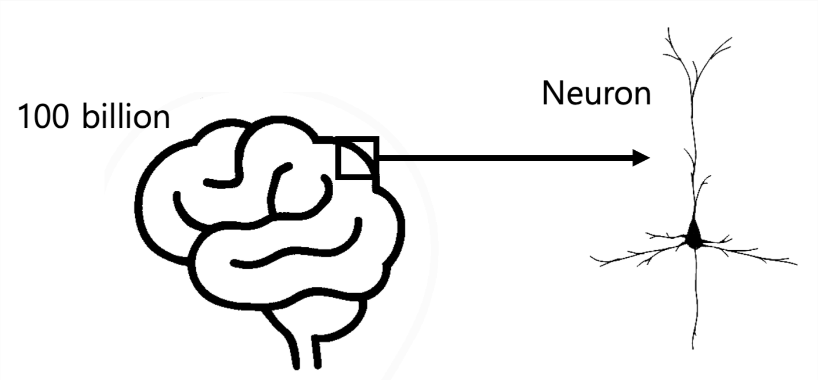
The human brain is made up of about one hundred billion neurons,

Neuron and artificial neuron

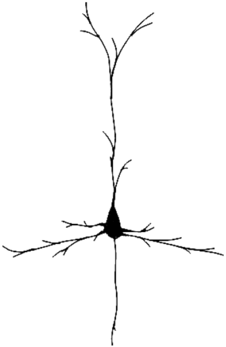
The human brain is made up of about one hundred billion neurons, and while its structure and functions are highly complex



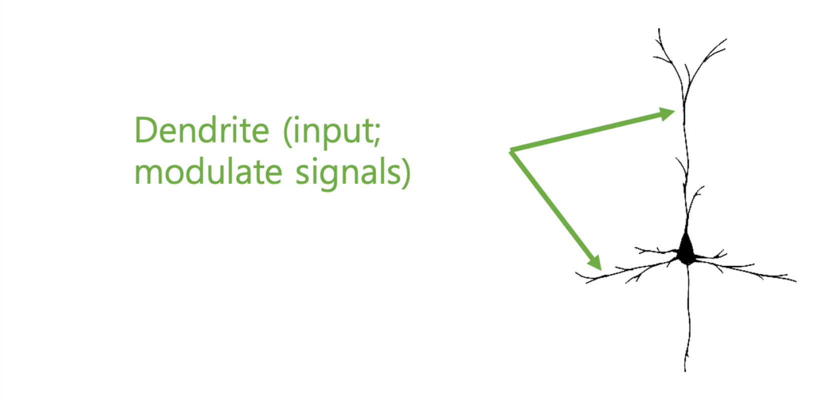
The basic unit that composes the brain is relatively simple.



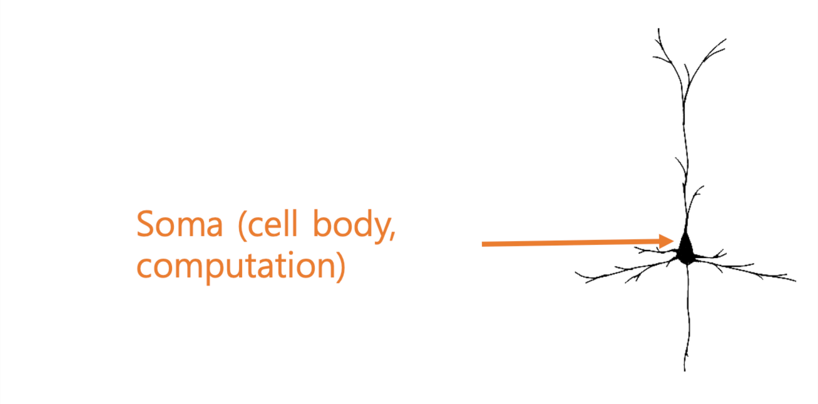
Neuron



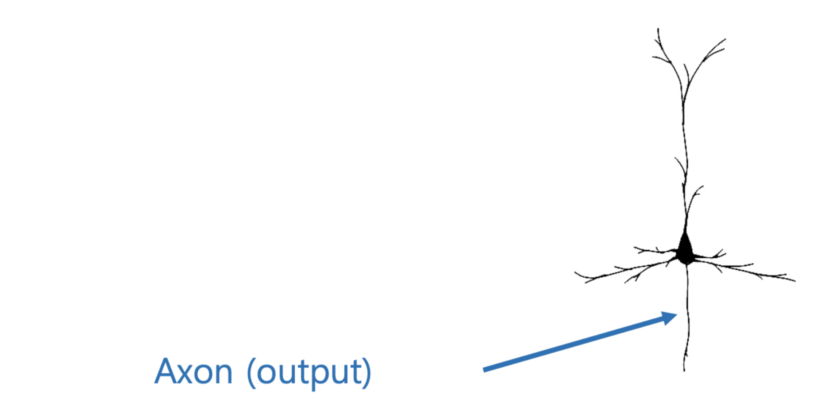
Neuron



Neuron

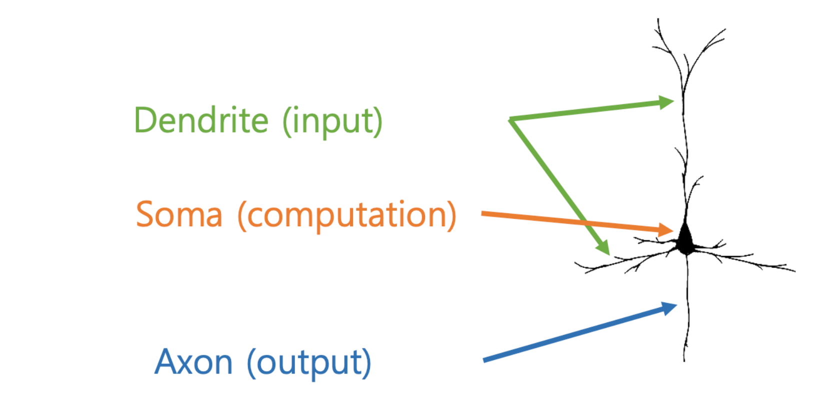


Neuron



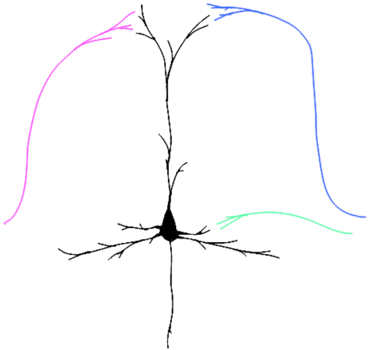
Neuron as an information processor

We can think of a neuron as an information-processing unit with three main functions: (1) input, (2) computation, and (3) output.



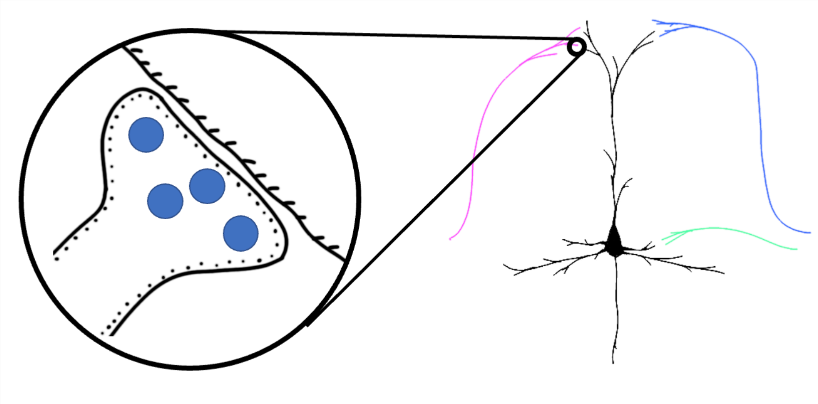
Connections between neurons

It connects to another neuron's axon



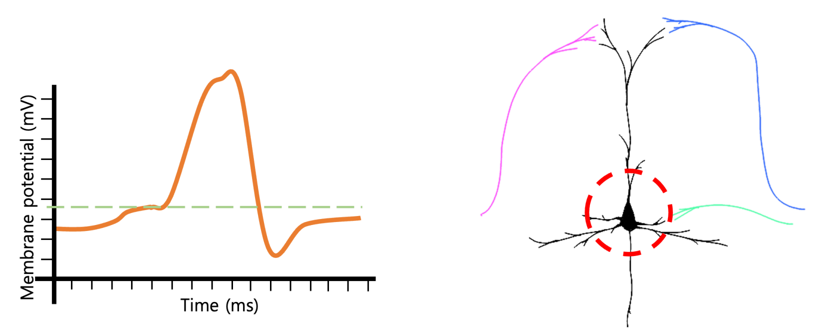
Connections between neurons

It connects to another neuron's axon through a **synapse**

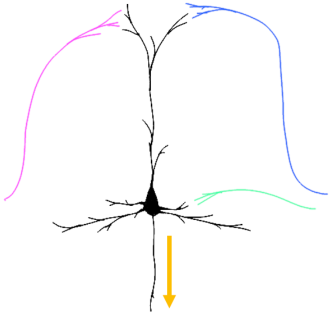


Firing of a neuron

In the soma (cell body), if the incoming signals exceed a certain threshold, the neuron fires an action potential.

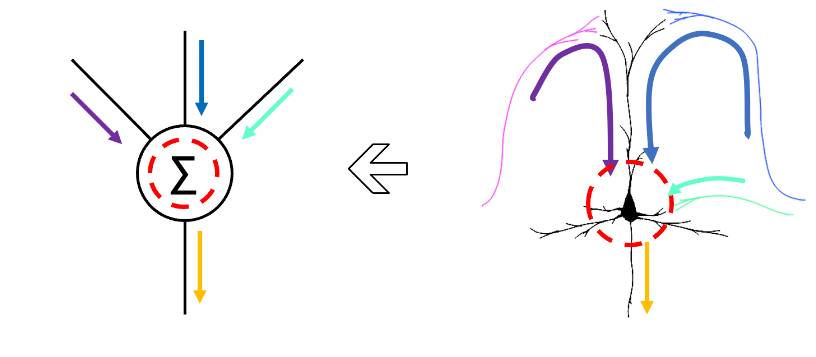


It allows the neuron to transfer information to the next neuron.



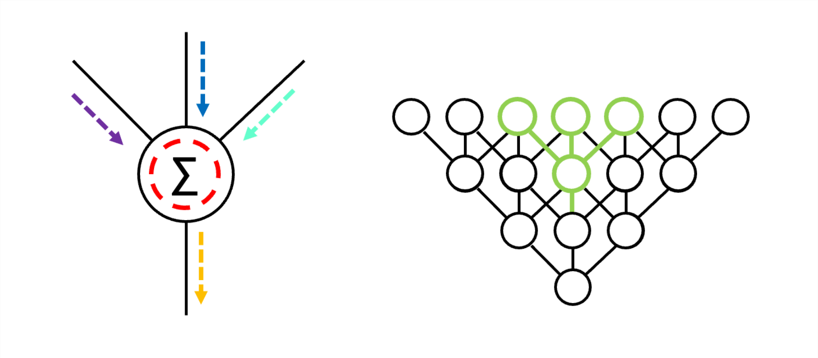
Neurons to artificial neurons

Artificial neurons are designed to mimic the information-processing mechanisms of biological neurons.



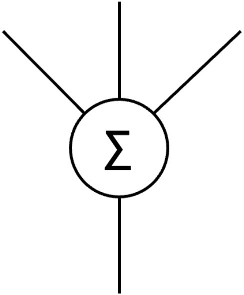
Neurons to artificial neurons

When combined, they form artificial neural networks. Then, let's try to understand how artificial neurons work.

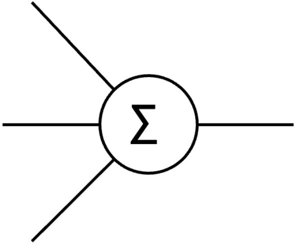


Perceptron

Artificial neurons

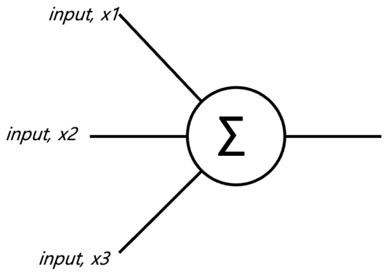


Artificial neurons



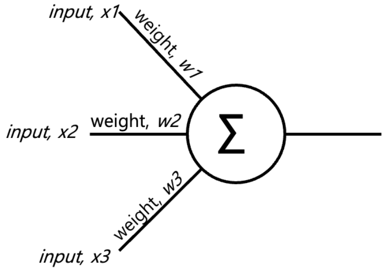
Artificial neurons

This cell receives **inputs** from three neurons, **calculates** whether the total input exceeds the **threshold**, and then produces an **output**.



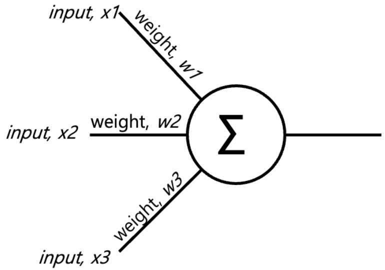
Artificial neurons

Each neuron provides an **input**, denoted as x_1, x_2, x_3.



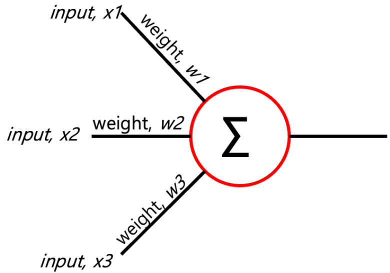
Artificial neurons

Each input x_1, x_2, x_3 is multiplied by a corresponding **weight** w_1, w_2, w_3 before being combined.



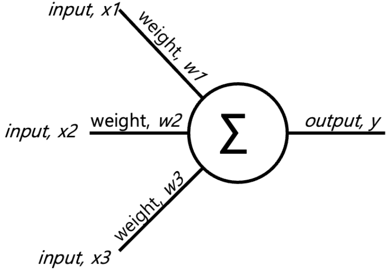
Artificial neurons

The circular unit is called a **node**. It receives the inputs from neurons, combines them with their weights, and calculates the node value.



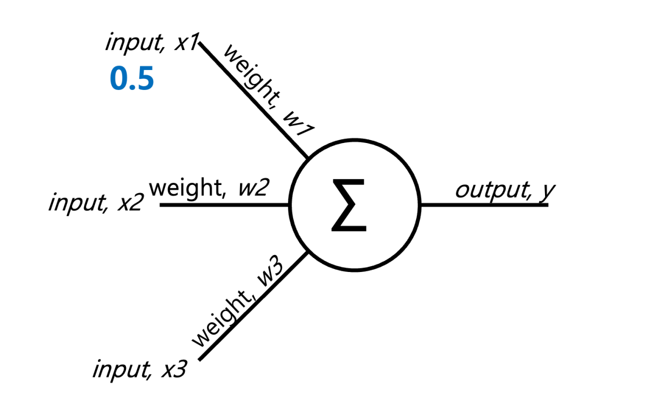
Artificial neurons

This computed output is then passed on to the next neuron.

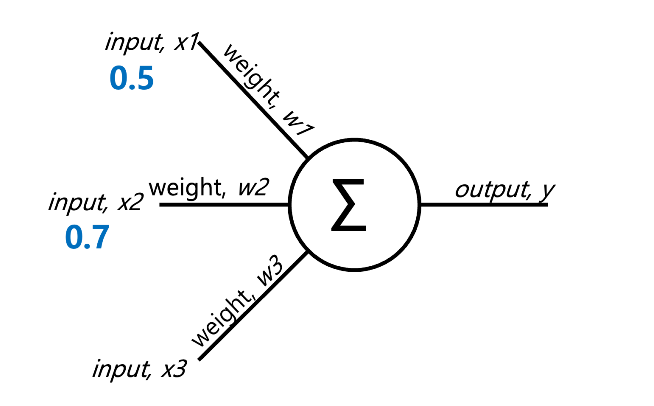


Artificial neurons

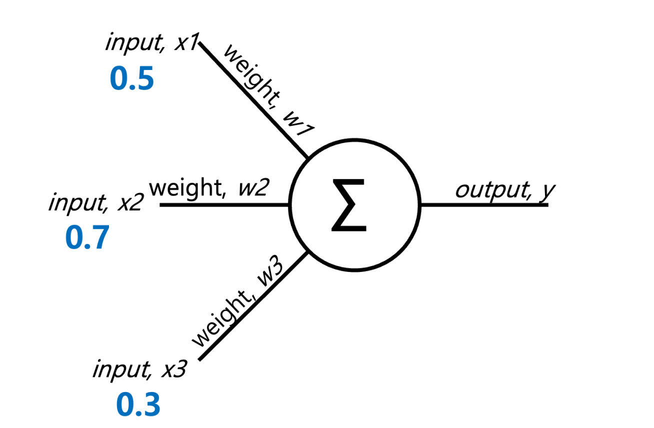
Now, let's put in actual values and calculate the output.



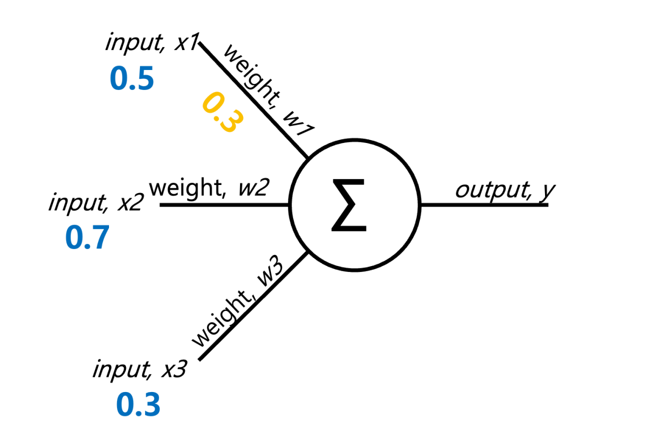
Now, let’s put in actual values and calculate the output.



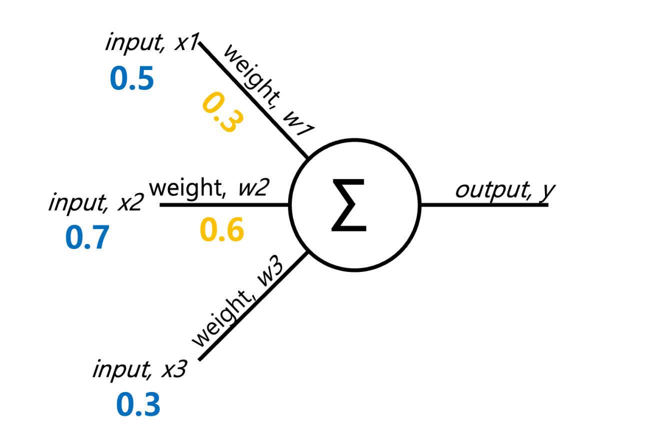
Now, let's put in actual values and calculate the output.



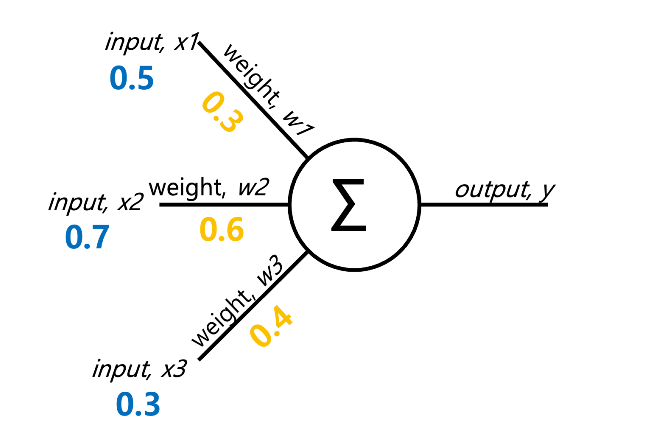
Now, let's put in actual values and calculate the output.



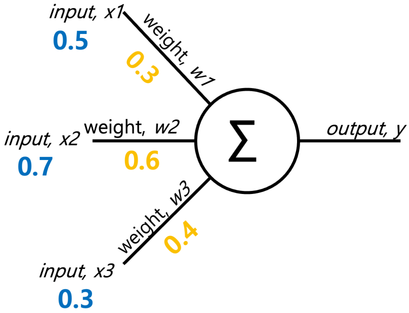
Now, let's put in actual values and calculate the output.



Now, let's put in actual values and calculate the output.



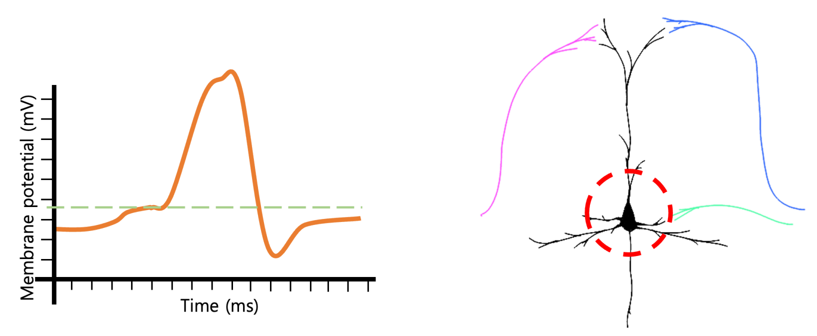
Now, let's put in actual values and calculate the output.



0.5 x 0.3 = 0.15
0.7 x 0.6 = 0.42
0.3 x 0.4 = 0.12

Artificial neurons: activation function

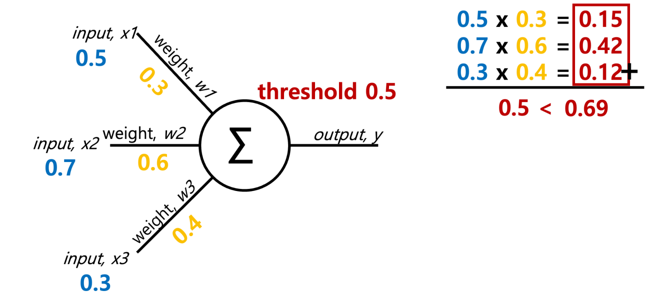
Just like the soma decides whether to fire based on the threshold, an **artificial neuron** computes a weighted sum of inputs and applies an **activation function**.



Artificial neurons: activation function

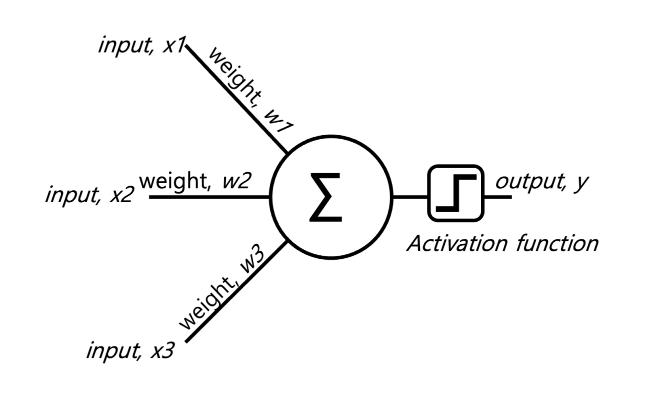
An **activation function** is applied to the weighted sum of inputs to determine the output. For simplicity, let's assume a step function as the activation function:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0.5 \\ 0 & \text{if } z < 0.5 \end{cases}$$



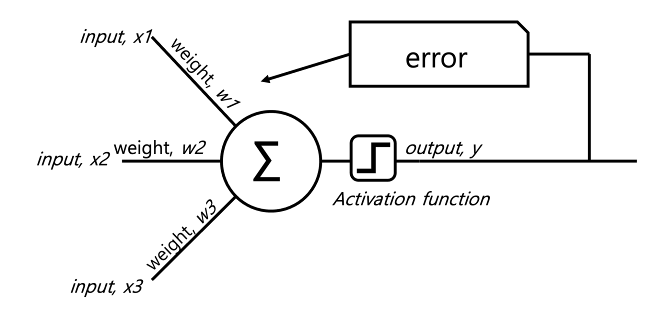
Perceptron

This is a basic structure of the **perceptron**.



Training perceptron

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



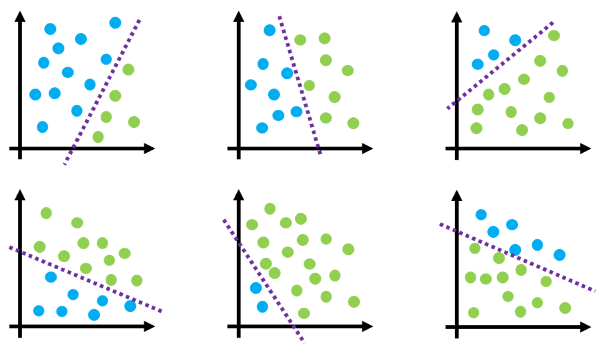
Training perceptron: Demo

Hands-on practice in Lab 3

Multi-layer perceptron

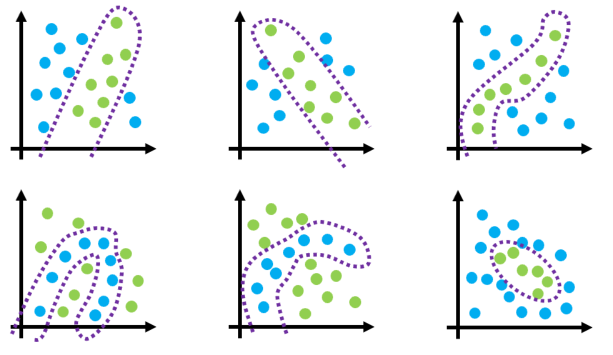
Limitation of a single-layer perceptron

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.



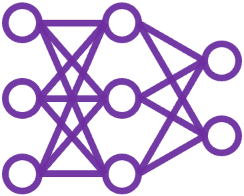
Limitation of a single-layer perceptron

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



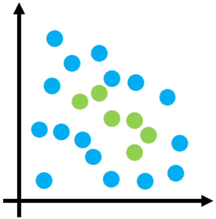
Introduction of an MLP

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)**.

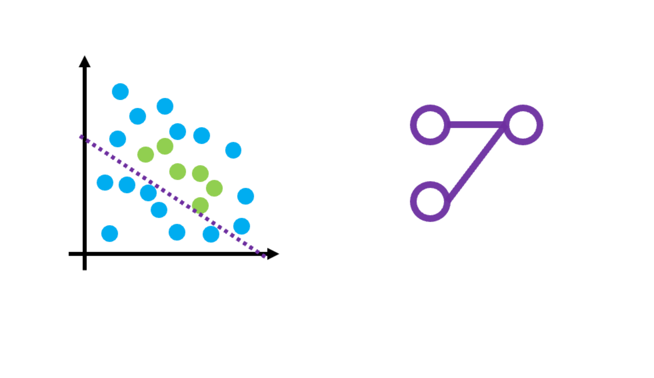


Introduction to an MLP

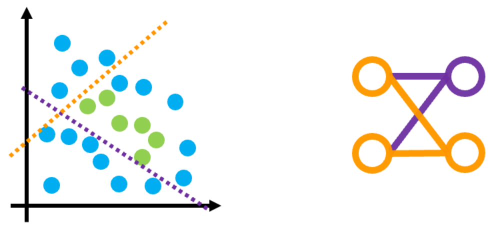
Let's assume we are given data in a complex form like this.



With a single perceptron, linear separation is not possible.

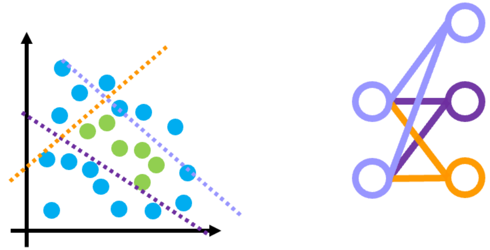


But if we add more lines, it becomes possible to separate further.



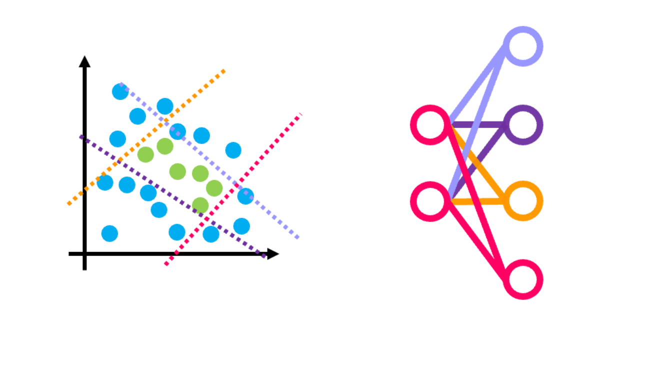
Introduction to an MLP

By adding several lines, the separation becomes more feasible.

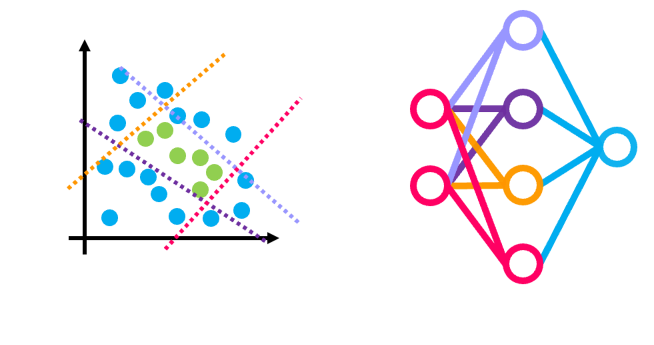


Introduction of an MLP

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

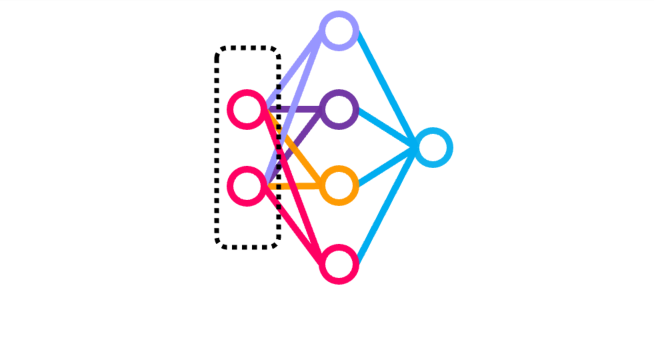


If we then connect another perceptron that takes these four outputs as its inputs, we can construct a **multi-layer neural network** capable of non-linear separation.



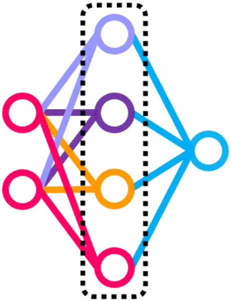
Structure of an MLP

So the MLP we build here consists of an input layer



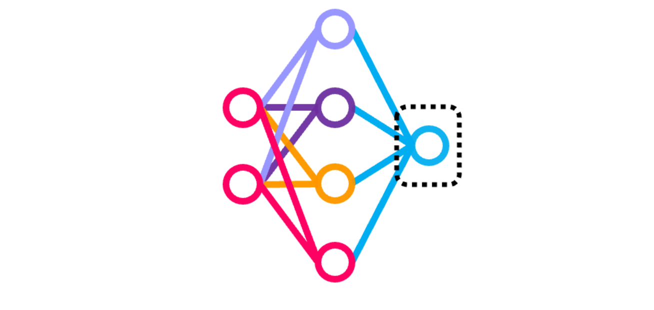
Structure of an MLP

a hidden layer



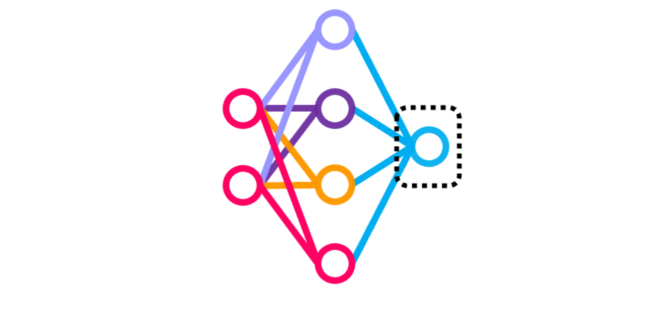
Structure of an MLP

and an output layer.



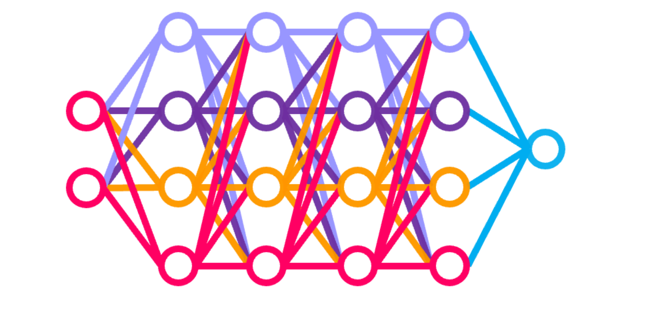
Structure of an MLP

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



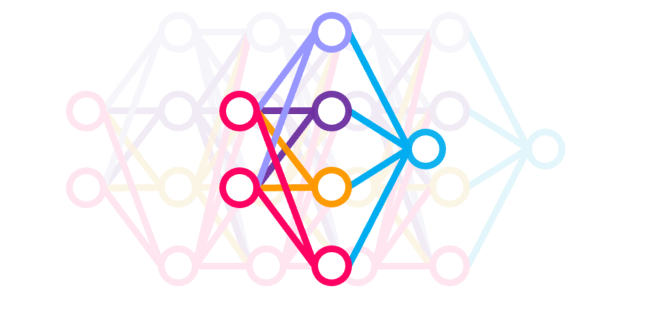
Structure of an MLP

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



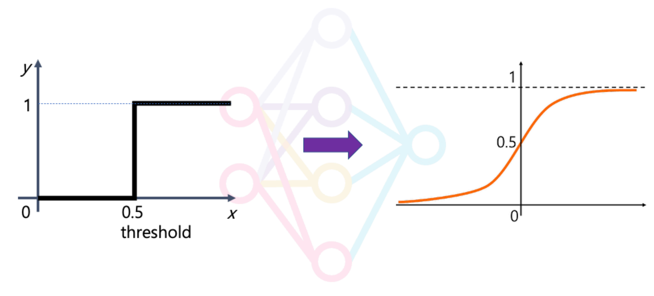
Structure of an MLP – Deep Learning

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* we learned last time.



More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic

More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic
- tanh

More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic
- tanh
- ReLU

More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic
- tanh
- ReLU
- Leaky/Parametric ReLU

More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic
- tanh
- ReLU
- Leaky/Parametric ReLU
- Swish

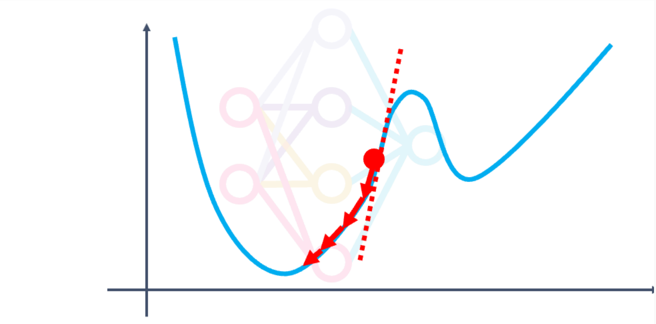
More: At-a-Glance Comparison

In fact, many more activation functions have been introduced.

- Sigmoid/Logistic
- tanh
- ReLU
- Leaky/Parametric ReLU
- Swish
- GELU – frequently used with Transformers (BERT, RoBERTa)

More changes: Optimization

To reduce errors in multilayer networks, methods like gradient descent (also introduced last time) are used.

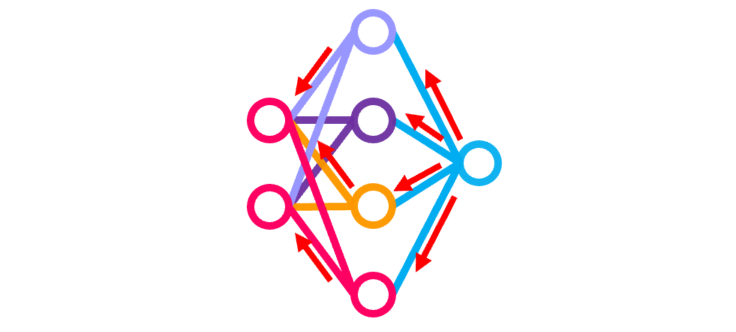


More changes: Optimization (Review from last class)

- **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 changes: Backpropagation algorithm

A key algorithm in training neural networks is backpropagation.



Gradient descendant and loss function

Gradient Descent: Definition

- Gradient descent is an optimization algorithm used in deep learning.

Gradient Descent: Definition

- Gradient descent is an optimization algorithm used in deep learning.
- It minimizes a given loss function by updating model parameters.

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:

Gradient Descent: Definition

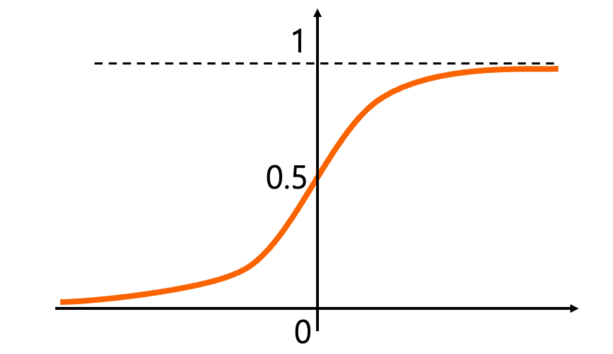
- Gradient descent is an optimization algorithm used in deep learning.
- It minimizes a given loss function by updating model parameters.
- Model parameters include:
    - Weights – connection strengths between neurons

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

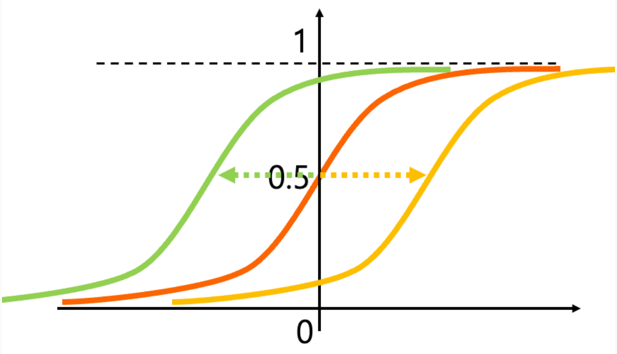
Gradient Descent: Definition

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



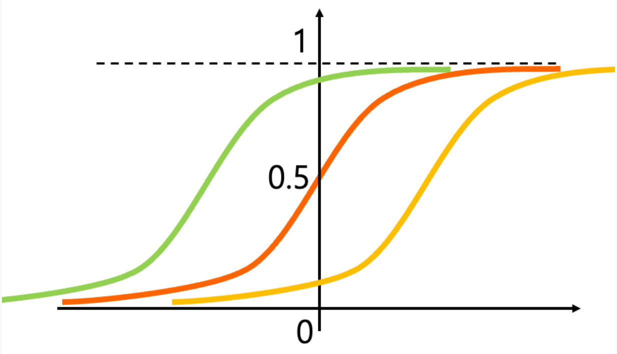
Gradient Descent: Definition

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



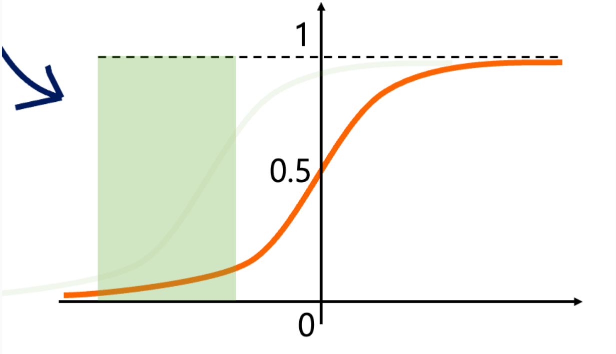
Gradient Descent: Definition

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



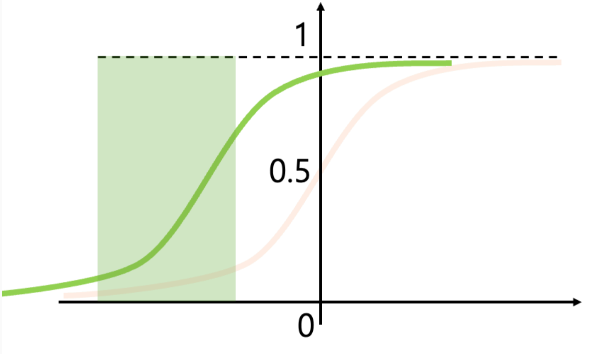
Gradient Descent: Definition

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



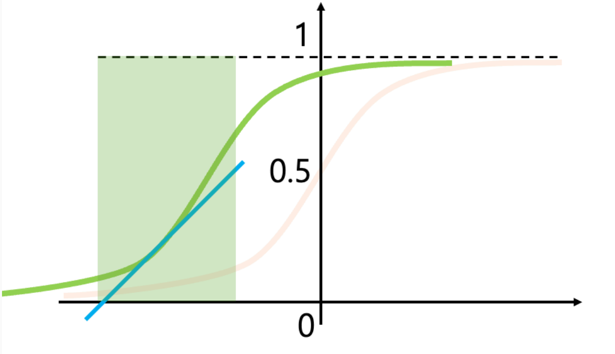
Gradient Descent: Definition

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



Gradient Descent: Definition

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

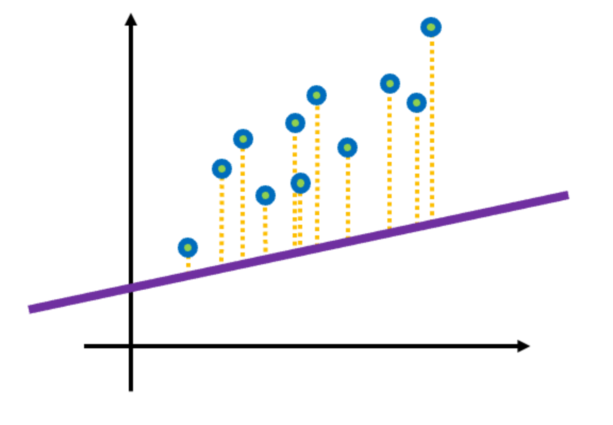


Gradient Descent: Definition

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

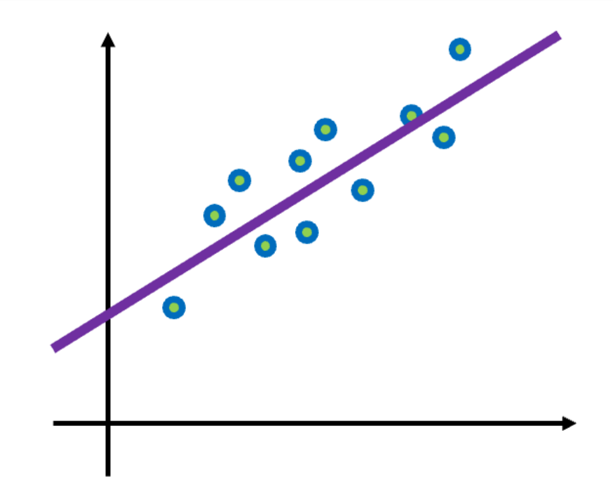
Loss Function

- 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

Even though it looks like a formula, the idea is very simple: the difference between the actual value and the prediction.

$$MSE = \frac{1}{n}\sum_{i}^{n}\left(\underset{\text{error}}{\boxed{y_i - \hat{y}_i}}\right)^2$$

Loss Function: MSE

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

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

Loss Function: MSE

Then, we add up the errors for all the data points.

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

Loss Function: MSE

Finally, we divide by the number of data points.

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

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.

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

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

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

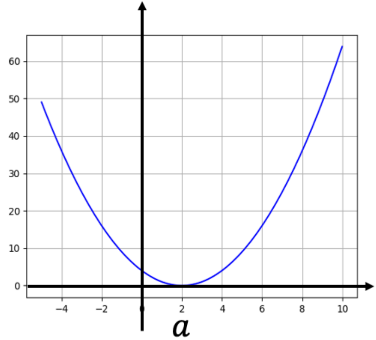
$$y = \frac{1}{1}\sum_{1}^{1}(x_{\square} - a)^2$$

Gradient Descent: Core Idea

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

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

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

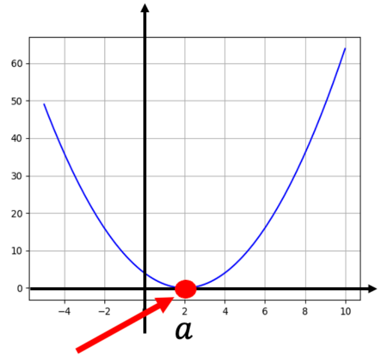


Gradient Descent: Core Idea

The point where the error is minimized is here.

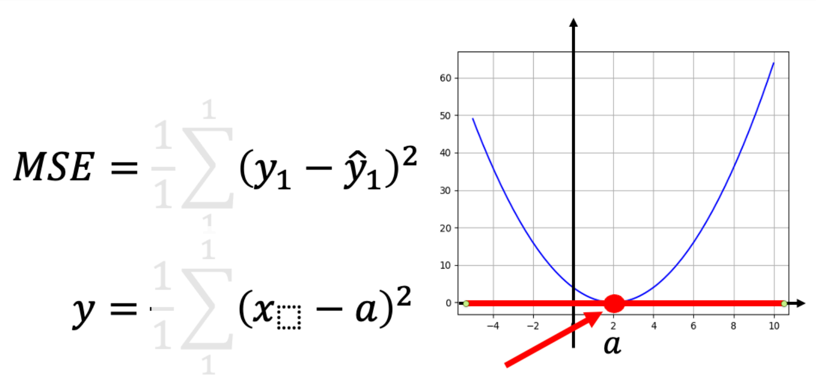
$$MSE = \frac{1}{1}\sum_{1}^{1}(y_1 - \hat{y}_1)^2$$

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



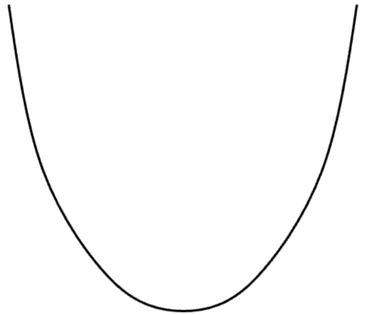
Gradient Descent: Core Idea

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



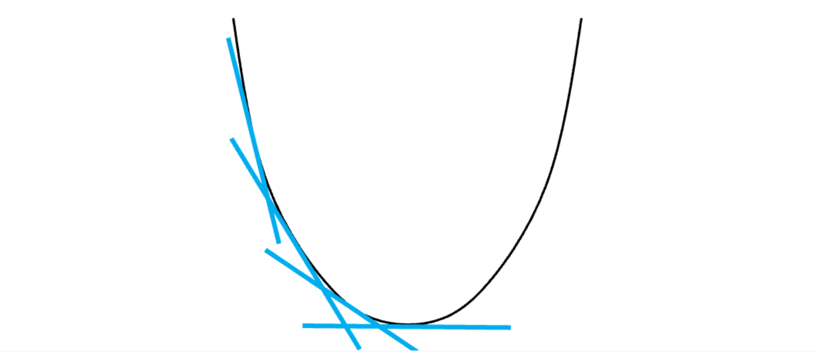
Gradient Descent: Core Idea

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



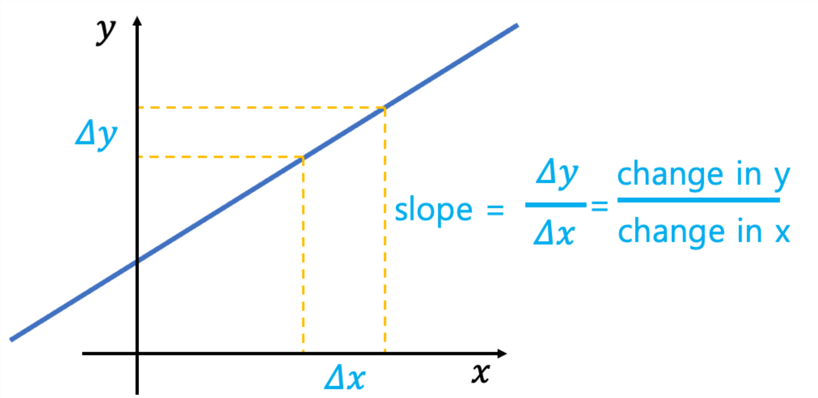
Gradient Descent: Core Idea

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

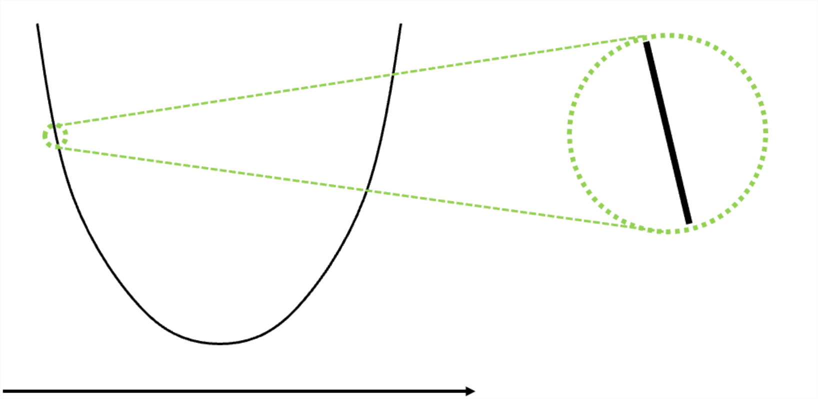


Gradient Descent: Core Idea

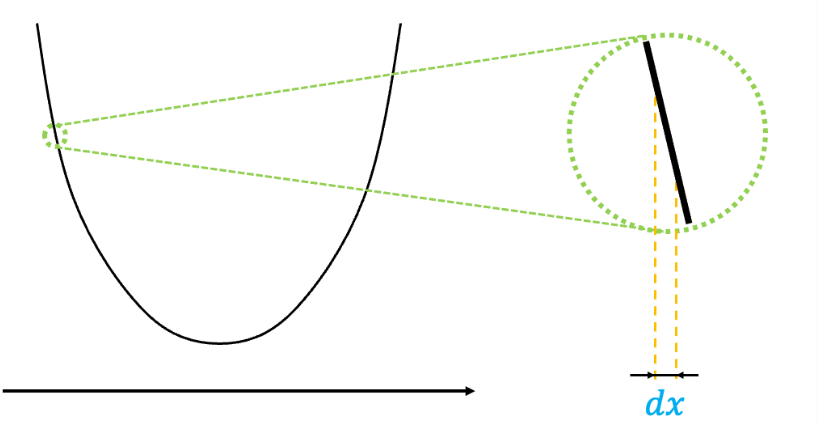
For a linear function, we can express the slope like this.



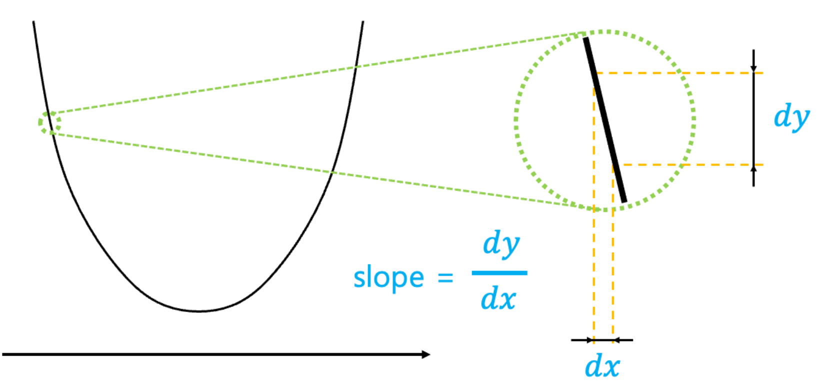
And by using derivatives, we can compute slopes even when the curve is not a straight line.



With an infinitesimally small change in x, dx,

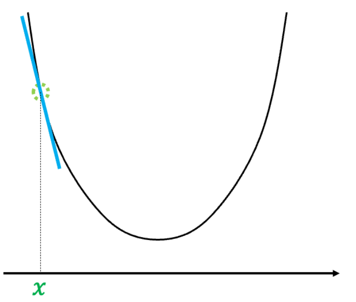


and the corresponding infinitesimal change in y, dy, we can calculate the slope.



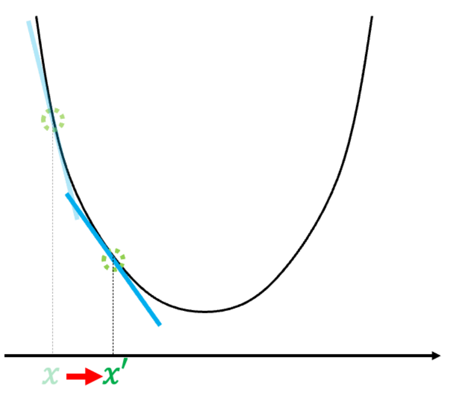
Gradient Descent: Core Idea

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



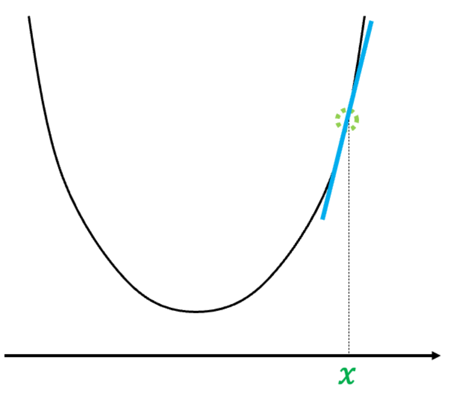
Gradient Descent: Core Idea

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



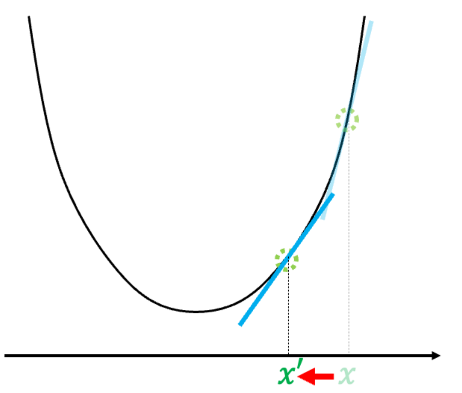
Gradient Descent: Core Idea

- The algorithm updates parameters by moving in the **opposite direction of the gradient**.
  - If slope is positive → decrease the parameter value.



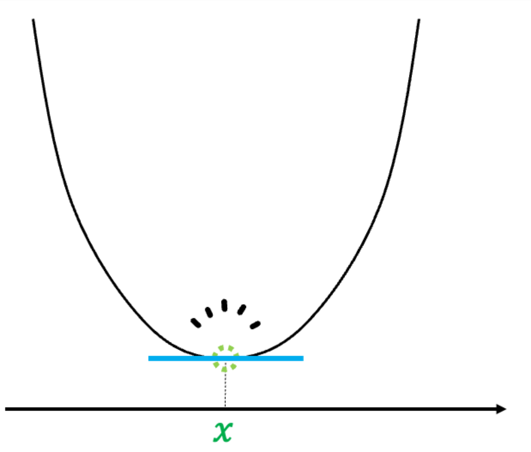
Gradient Descent: Core Idea

- The algorithm updates parameters by moving in the **opposite direction of the gradient**.
  - If slope is positive → decrease the parameter value.



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

Gradient descent learning rule:

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

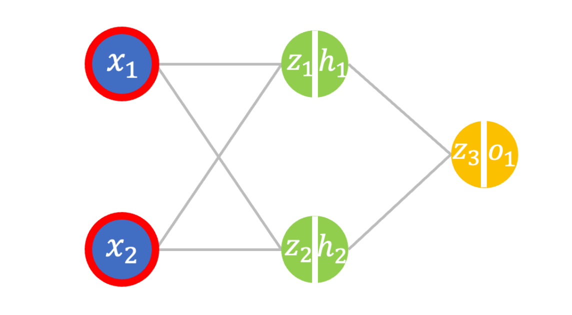
- L: loss function
- $\frac{\partial L}{\partial w}$: gradient of the loss function with respect to weight

Backpropagation

Now, we have learned everything we need to understand how backpropagation works—something that can seem almost magical in the way deep learning encodes languages into vector spaces.

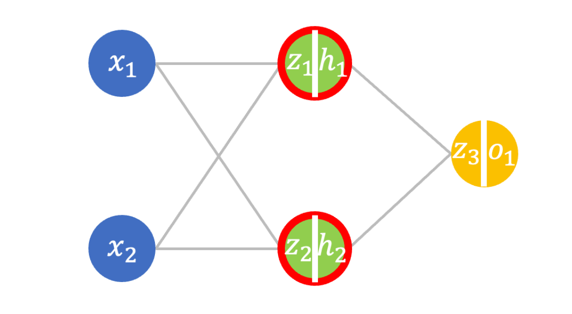
Setup: Input Layer

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

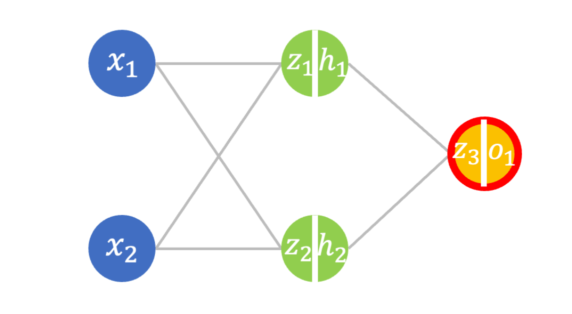


Setup: Hidden Layer

The hidden layer contains two neurons.

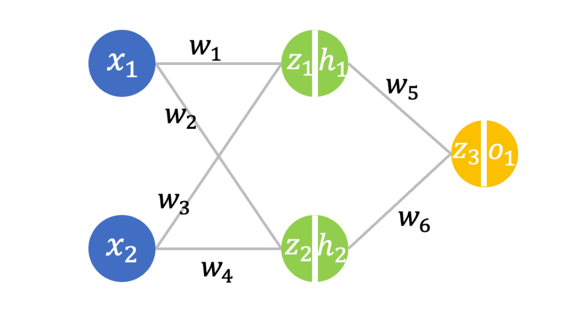


The output layer has one neuron.



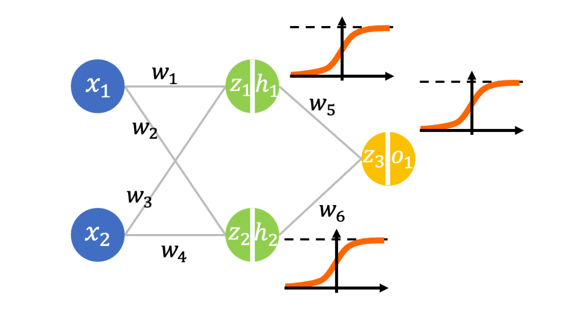
Simple Multilayer Neural Network

There are some weights.



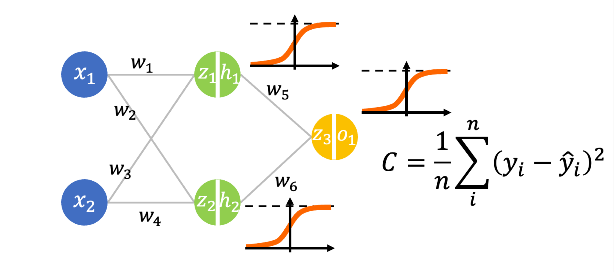
Activation Function

The activation function is the sigmoid.



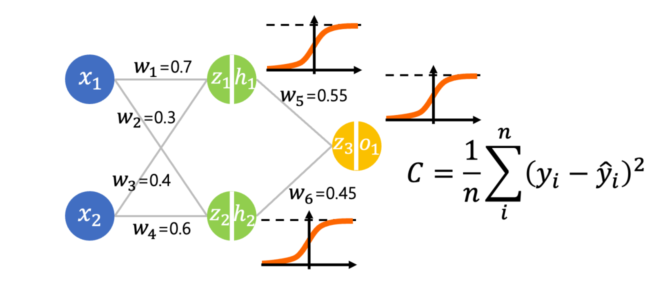
Loss Function

The loss function is Mean Squared Error (MSE).



Weight Initialization and Learning Rate

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



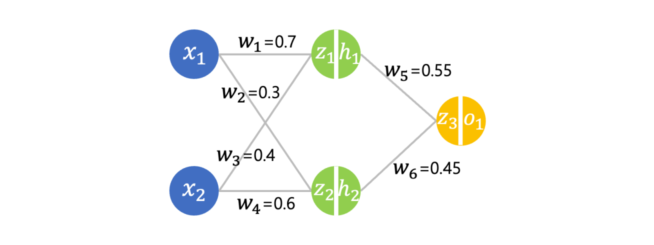
Learning Process

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)**.

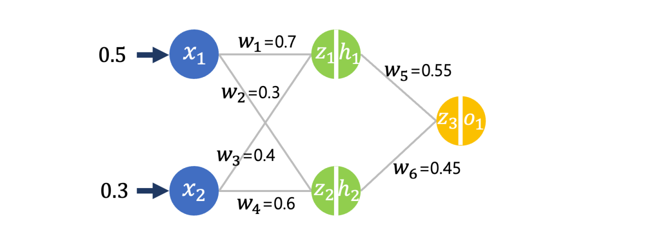
Step 1: Feedforward

The first step is the feedforward stage.



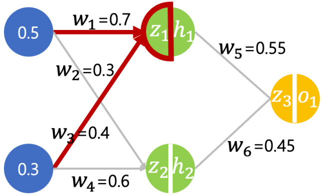
Input Values

The inputs are given as follows.



Weighted Sum to Hidden Node (1)

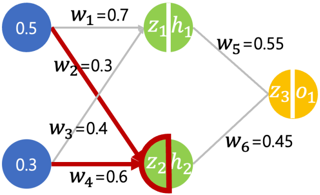
Each input is multiplied by its connection weight, and the results are summed into the hidden layer node.



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

Weighted Sum to Hidden Node (2)

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

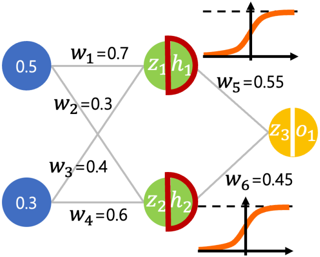


$$z_1 = x_1 w_1 + x_2 w_3 = 0.5\text{x}0.7 + 0.3\text{x}0.4 = \mathbf{0.47}$$

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

Activation at Hidden Node

The activation function is applied to the hidden layer node.



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

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

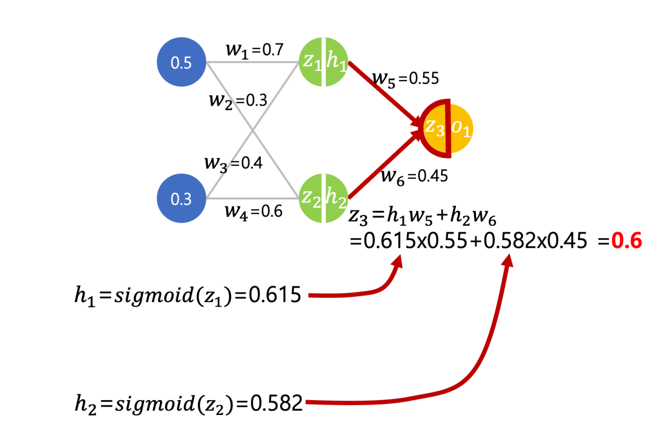
$$z_2 = x_1 w_2 + x_2 w_4 = 0.5\text{x}0.3 + 0.3\text{x}0.6 = 0.33$$

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

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

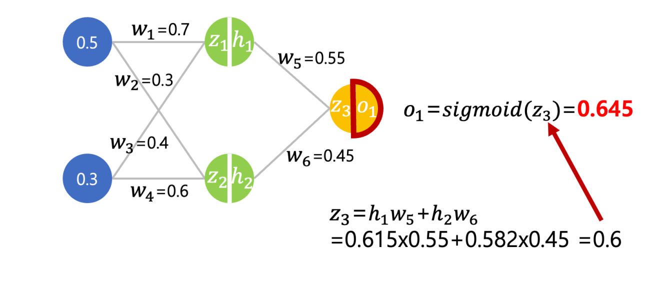
Weighted Sum to Output Neuron

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



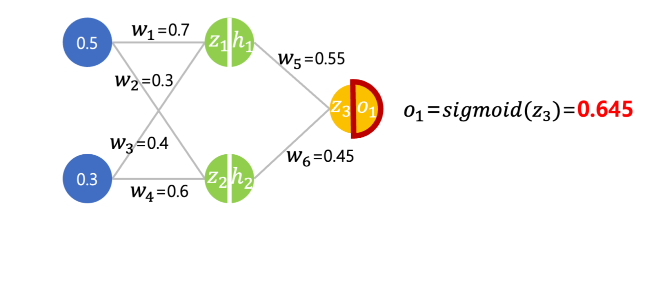
Final Output

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



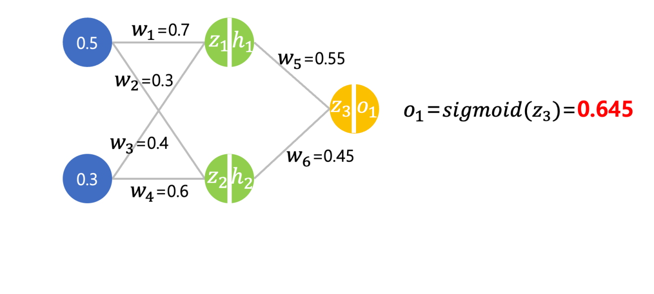
Feedforward Completed

The feedforward stage is now complete.



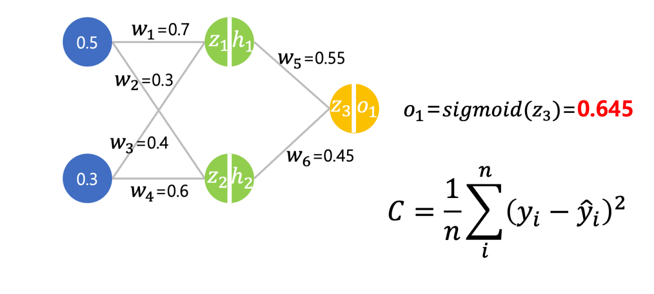
Step 2: Loss Calculation

The second step is the loss calculation stage.



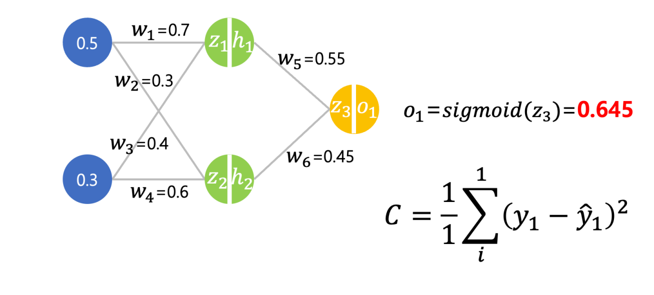
Applying the Loss Function

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



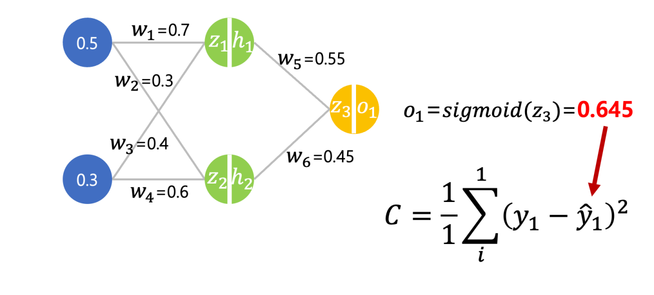
Single Output Neuron

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



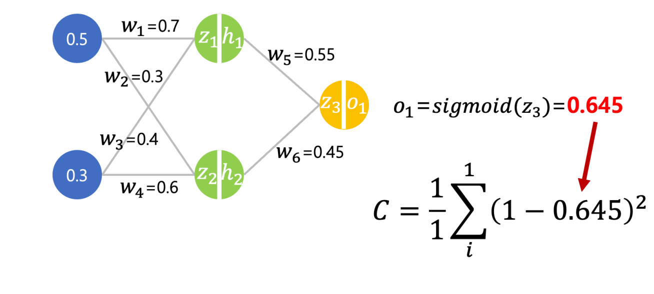
Substituting the Output Value

The output value 0.645 is substituted into the MSE.



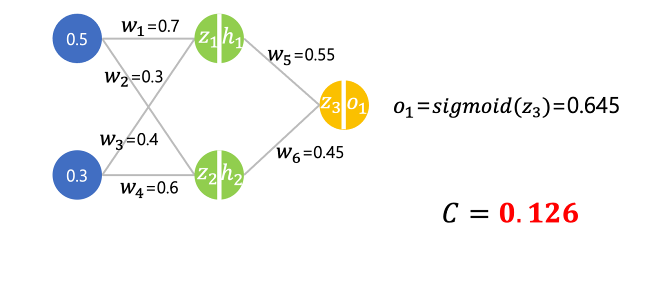
Assumed Target Value

Suppose the actual target value is 1.



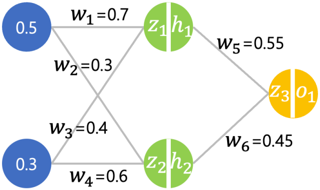
Error Calculation

The error (C) is then calculated.



Step 3: Backpropagation

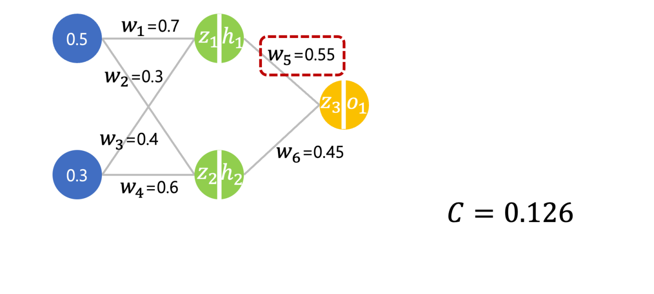
The third step is the backpropagation stage.



$$C = 0.126$$

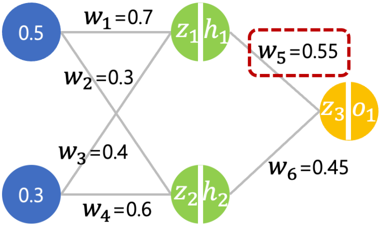
Updating Weight w_5

Using backpropagation, we update the weight w_5.



Weight Update Rule

Recall the weight update formula in gradient descent.



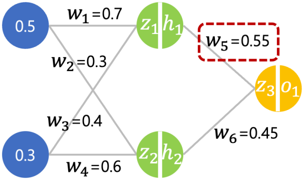
Gradient descent learning rule:

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

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

Derivative for w_5

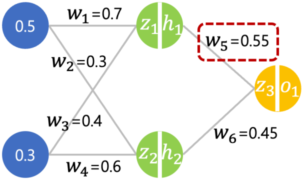
Therefore, to update w_5, we need to compute the following derivative.



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

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.

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

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.

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

Chain Rule: An Analogy

How many times faster is the cheetah than the human?

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...

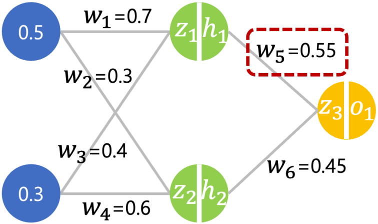
Now, you know:

- 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

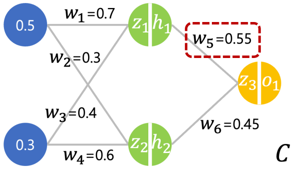
Therefore, we calculate the value by computing each part step by step.



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

First Derivative

First, we compute the first derivative.

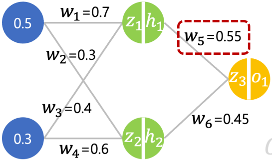


$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

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

Case of $n = 1$

Since $n = 1$,



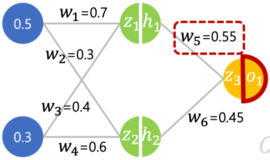
$$C = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2$$

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

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

Step-by-Step Calculation (1)

Proceeding step by step, we obtain:



$o_1 = sigmoid(z_3) =$ **0.645**

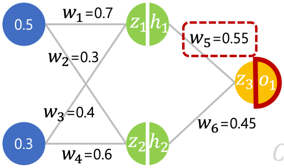
$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

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

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

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

Step-by-Step Calculation (2)



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

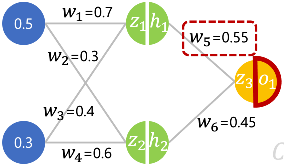
$$C = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

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

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

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

Step-by-Step Calculation (3)



$o_1 = sigmoid(z_3) =$ **0.645**

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

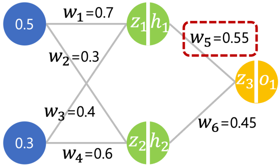
$$C = (y - o_1)^2$$

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

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

Second Derivative

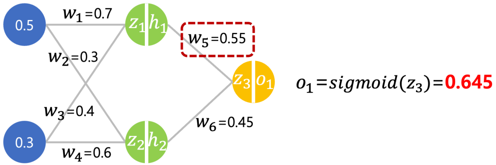
To compute the second derivative,



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

Sigmoid in Feedforward

Recall that we used the sigmoid function during feedforward.



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

Sigmoid Formula

The mathematical formula of the sigmoid function is as follows:

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

$o_1 = sigmoid(z_3) =$ **0.645**

Using O and Z Variables

When expressed with O and Z variables, we obtain:

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

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

$o_1 = sigmoid(z_3) =$ **0.645**

Derivative of Sigmoid

The derivative of the sigmoid function is:

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

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

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

$o_1 = sigmoid(z_3) =$ **0.645**

Simplified Expression

It can also be expressed as:

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

$o_1 = sigmoid(z_3) =$ **0.645**

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

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

Computing the Sigmoid Derivative

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

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

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

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

$o_1 = sigmoid(z_3) =$ **0.645**

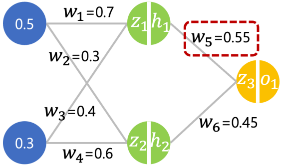
Step-by-Step Expansion (1)

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

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

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

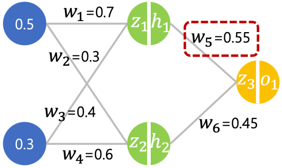
Step-by-Step Expansion (2)



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

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

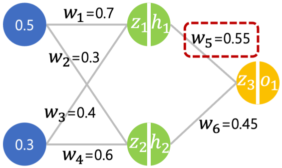
Step-by-Step Expansion (3)



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

Third Term

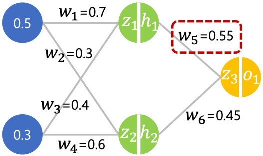
For the third term, we compute:



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

Formula for z

Using the formula for z,

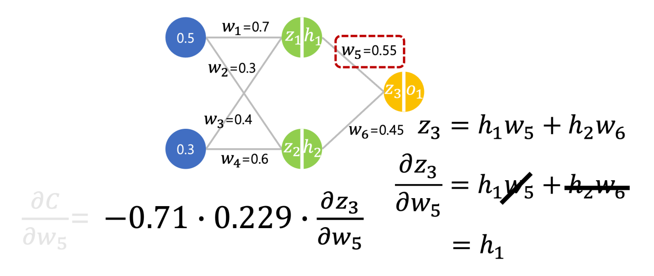


$$z_3 = h_1 w_5 + h_2 w_6$$

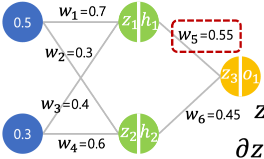
$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot \frac{\partial z_3}{\partial w_5}$$

Partial Derivative of z_3

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



From the feedforward calculation, $h_1 = 0.615$.



$$z_3 = h_1 w_5 + h_2 w_6$$

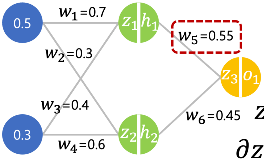
$$\frac{\partial z_3}{\partial w_5} = h_1 \cancel{w_5} + \cancel{h_2 w_6}$$

$$= h_1 = 0.615$$

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

Substituting Values

Now, substituting the values, we obtain:



$$z_3 = h_1 w_5 + h_2 w_6$$

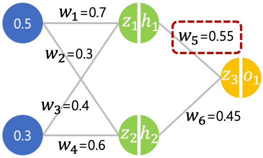
$$\frac{\partial z_3}{\partial w_5} = h_1 \cancel{w_5} + \cancel{h_2 w_6}$$

$$= h_1 = 0.615$$

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

Gradient of the Loss Function

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



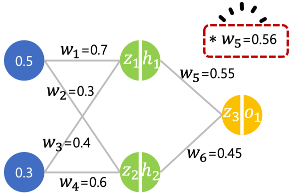
$$\frac{\partial C}{\partial w_5} = -0.71 \cdot 0.229 \cdot 0.615 = -0.1$$

Weight Update with Gradient Descent

According to the gradient descent learning rule:

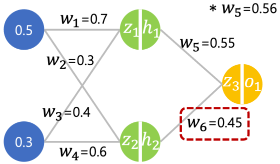
$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

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



Updating Weight w_6

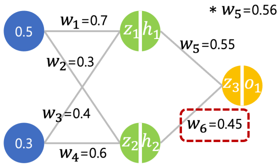
Now, let us update w_6 as well.



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

Using the Chain Rule Again

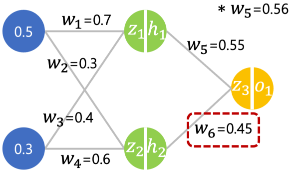
By the chain rule, we can similarly derive the expression.



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

Same as w_5 Formula

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

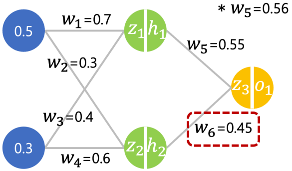


$$\frac{\partial C}{\partial w_6} = \boxed{\frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3}} \cdot \frac{\partial z_3}{\partial w_6}$$

Calculated equally as the previous equation

Substituting Values

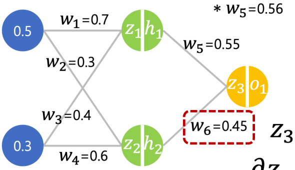
Now, substituting the values, we obtain:



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

Third Term

The third term once again becomes h_2.



$$z_3 = h_1 w_5 + h_2 w_6$$

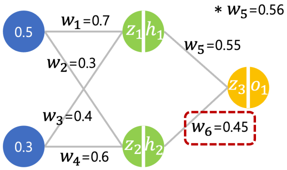
$$\frac{\partial z_3}{\partial w_6} = \cancel{h_1 w_5} + h_2 \cancel{w}_6$$

$$= h_2 = 0.582$$

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

Gradient for w_6

The gradient of the loss function is -0.095.



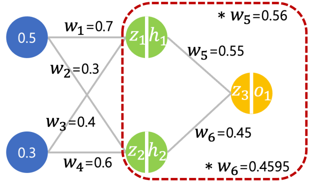
$$\frac{\partial C}{\partial w_6} = -0.71 \cdot 0.229 \cdot 0.582 = -0.095$$

Weight Update with Gradient Descent

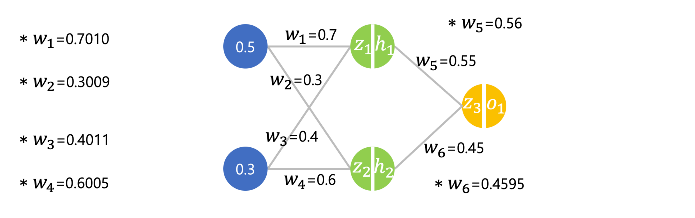
According to the gradient descent learning rule:

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

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

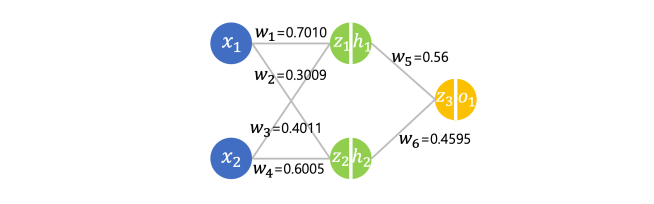


The remaining weights in the first layer are updated in the same way.



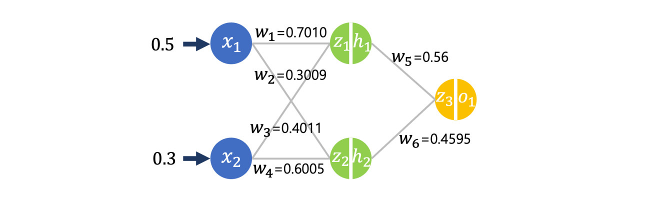
Checking Error After Backpropagation

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



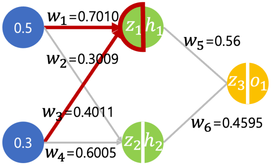
Same Input Values

We feed in the same input values again.



Hidden Node z_1

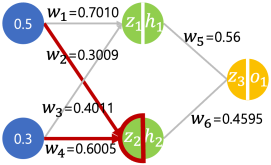
Weighted sums are passed into hidden node z_1.



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

Hidden Node z_2

Weighted sums are passed into hidden node z_2.

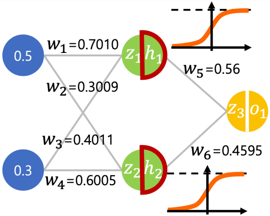


$z_1 = x_1 w_1 + x_2 w_3 = 0.5 \times 0.7010 + 0.3 \times 0.4011 = \mathbf{0.4708}$

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

Hidden Layer Activation

The activation function is applied at the hidden nodes.



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

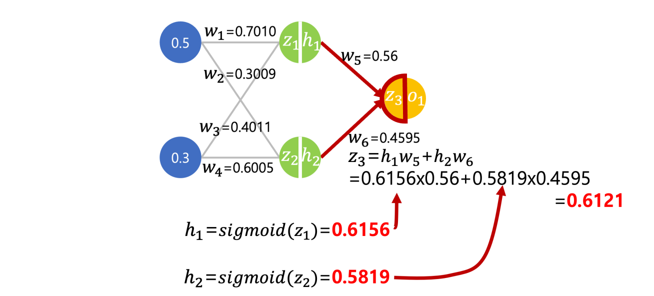
$h_1 = sigmoid(z_1) = \mathbf{0.6156}$

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

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

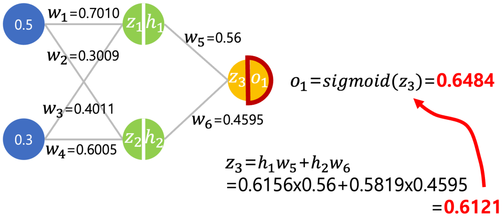
Output Node Weighted Sum

The weighted inputs are summed at the output node.



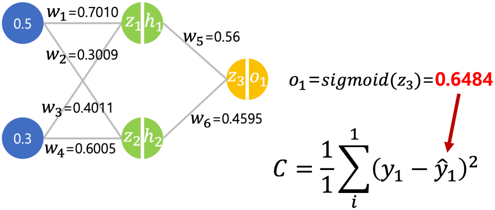
Final Output

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

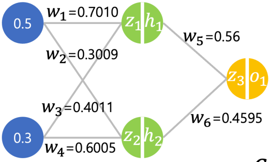


Loss Calculation (1)

The output value is substituted into the loss function.



Loss Calculation (2)

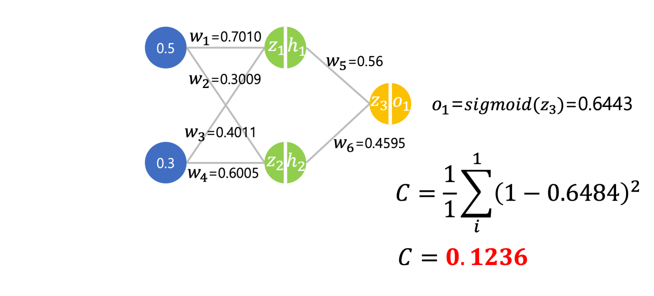


$o_1 = sigmoid(z_3) = \mathbf{0.6484}$

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

Error Comparison

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



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 TensorFlow 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? *Pytorch Tutorial*

Wrap-up

Conclusion

- GloVe
- Artificial neural network
- Perceptrons
- MLP
- Gradient descendant and loss function
- Backpropagation

Key idea: Modern NLP systems are built on deep learning; deep learning algorithm is not magic.