Overview of Deep Learning

Tianxiang (Adam) Gao

September 9, 2024

Outline

Recap: What is AI?

- **Artificial Intelligence (AI)** is a broad field that focuses on creating systems capable of performing tasks that typically require human intelligence.

Russell, Stuart J., and Peter Norvig. 2016. "Artificial Intelligence: A Modern Approach."

Recap: What is AI?

- **Artificial Intelligence (AI)** is a broad field that focuses on creating systems capable of performing tasks that typically require human intelligence.
- To pass the *(total) Turing Test*, it needs:
  - *Natural language processing (NLP)* to enable it to communicate effectively;
  - *Knowledge representation* to store and retrieve information;
  - *Automated reasoning* to use the stored information to answer questions and to draw new conclusions;
  - *Machine learning* to recognize patterns from data and adapt to new situations;
  - *Computer vision (CV)* to perceive objects;
  - *Robotics* to manipulate and interact with the physical world.

Russell, Stuart J., and Peter Norvig. 2016. "Artificial Intelligence: A Modern Approach."

Recap: What is ML?

- **Machine learning (ML)** is a subset of AI that focuses on developing algorithms and (statistical) models to learn from (training) data and generalize to unseen data.
- Commonly used algorithms and models are categorized into:
  - *Supervised learning*: linear/logistic regression, supper vector machines (SVM), decision trees, and neural networks
  - *Unsupervised learning*: k-means clustering, dimensionality reduction, Gaussian mixture models, generative models
  - *Reinforcement learning*: Q-Learning, policy gradient, deep Q-networks (DQN)

What is DL?

- **Deep learning (DL)** is a subset of ML that focuses on using *deep neural networks (DNN)* with many layers to learn representation in large datasets.

What is DL?

- **Deep learning (DL)** is a subset of ML that focuses on using *deep neural networks (DNN)* with many layers to learn representation in large datasets.
    - It is capable of **automatically** learning features from raw data, unlike other machine learning models that rely on manually crafted features.

What is DL?

- **Deep learning (DL)** is a subset of ML that focuses on using *deep neural networks (DNN)* with many layers to learn representation in large datasets.
  - It is capable of **automatically** learning features from raw data, unlike other machine learning models that rely on manually crafted features.
  - It learns the intricate structures from data by using *backpropogation* to update the parameters in DNN.

What is DL?

- **Deep learning (DL)** is a subset of ML that focuses on using *deep neural networks (DNN)* with many layers to learn representation in large datasets.
  - It is capable of **automatically** learning features from raw data, unlike other machine learning models that rely on manually crafted features.
  - It learns the intricate structures from data by using *backpropogation* to update the parameters in DNN.
  - It encompasses various neural network architectures, including *convolutional neural Networks (CNNs)*, *recurrent neural networks (RNNs)*, *transformers*, and *graph neural networks (GNNs)*, each tailored to specific tasks.

What is DL?

- **Deep learning (DL)** is a subset of ML that focuses on using *deep neural networks (DNN)* with many layers to learn representation in large datasets.
    - It is capable of **automatically** learning features from raw data, unlike other machine learning models that rely on manually crafted features.
    - It learns the intricate structures from data by using *backpropogation* to update the parameters in DNN.
    - It encompasses various neural network architectures, including *convolutional neural Networks (CNNs)*, *recurrent neural networks (RNNs)*, *transformers*, and *graph neural networks (GNNs)*, each tailored to specific tasks.
    - It has led to significant breakthroughs in various applications, such as CV, NLP, speech recognition, and biomedical science.

Early Beginnings and "AI Winters"

1940s: Early Beginnings

1. In 1943, Warren McCulloch and Walter Pitts introduced the first mathematical model of a neuron, the **McCulloch-Pitts Neuron Model**.
2. In 1949, Donald Hebb proposed **Hebbian learning** in his book *The Organization of Behavior*, summarized as "cells that fire together wire together."

Early Beginnings and "AI Winters"

1940s: Early Beginnings

1. In 1943, Warren McCulloch and Walter Pitts introduced the first mathematical model of a neuron, the **McCulloch-Pitts Neuron Model**.
2. In 1949, Donald Hebb proposed **Hebbian learning** in his book *The Organization of Behavior*, summarized as "cells that fire together wire together."

1950s-1960s: Perceptron and the First "AI Winter"

1. In 1958, Frank Rosenblatt proposed the **perceptron**, an early neural network model.
2. In 1969, Marvin Minsky and Seymour Papert demonstrated that the perceptron could not solve **non-linear** problems, such as the **XOR problem**, leading to the first "AI winter."

Early Beginnings and "AI Winters"

1940s: Early Beginnings

1. In 1943, Warren McCulloch and Walter Pitts introduced the first mathematical model of a neuron, the **McCulloch-Pitts Neuron Model**.
2. In 1949, Donald Hebb proposed **Hebbian learning** in his book *The Organization of Behavior*, summarized as "cells that fire together wire together."

1950s-1960s: Perceptron and the First "AI Winter"

1. In 1958, Frank Rosenblatt proposed the **perceptron**, an early neural network model.
2. In 1969, Marvin Minsky and Seymour Papert demonstrated that the perceptron could not solve **non-linear** problems, such as the **XOR problem**, leading to the first "AI winter."

1980s-1990s: Revival with Expert Systems and Backpropagation

1. In 1980, **XCON** became one of the first commercially successful expert systems, marking a turning point for AI in the industry.
2. In 1986, Geoffrey Hinton, David Rumelhart, and Ronald Williams popularized **backpropagation** in their *Nature* paper, "Learning Representations by Back-Propagating Errors."

Early Beginnings and “AI Winters”

1940s: Early Beginnings

1. In 1943, Warren McCulloch and Walter Pitts introduced the first mathematical model of a neuron, the **McCulloch-Pitts Neuron Model**.
2. In 1949, Donald Hebb proposed **Hebbian learning** in his book *The Organization of Behavior*, summarized as “cells that fire together wire together.”

1950s-1960s: Perceptron and the First “AI Winter”

1. In 1958, Frank Rosenblatt proposed the **perceptron**, an early neural network model.
2. In 1969, Marvin Minsky and Seymour Papert demonstrated that the perceptron could not solve **non-linear** problems, such as the **XOR problem**, leading to the first “AI winter.”

1980s-1990s: Revival with Expert Systems and Backpropagation

1. In 1980, **XCON** became one of the first commercially successful expert systems, marking a turning point for AI in the industry.
2. In 1986, Geoffrey Hinton, David Rumelhart, and Ronald Williams popularized **backpropagation** in their *Nature* paper, “Learning Representations by Back-Propagating Errors.”

1990s-2000s: The Second ”AI Winter”

- The second “AI winter” occurred mainly due to the failure of **expert systems** to scale and generalize beyond narrow, rule-based tasks.
- Neural networks, despite the promise shown after the introduction of backpropagation, were still constrained by limited **computational power**, **scalability issues**, and **insufficient data**.

Emergence of Deep Learning

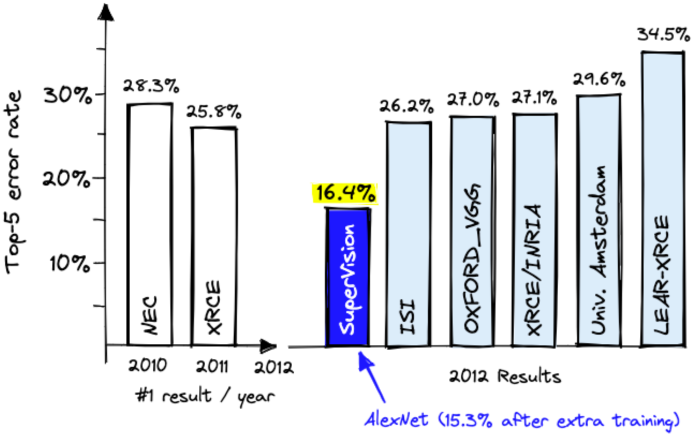
1990s-2000s: Advancements and the Emergence of Deep Learning

- In 1989, Yann LeCun et al. created the LetNet as an early example of a **CNN**, which became critical for image processing tasks.
- In 1997, the **Long Short-Term Memory (LSTM) network** was proposed by Sepp Hochreiter and Jürgen Schmidhuber that overcome the problem of vanishing gradients in RNNs

Modern Era: AlexNet 2012

2010s-Present: Deep Learning Revolution

- In 2012, AlexNet trained using **GPUs** dominated the ImageNet competition

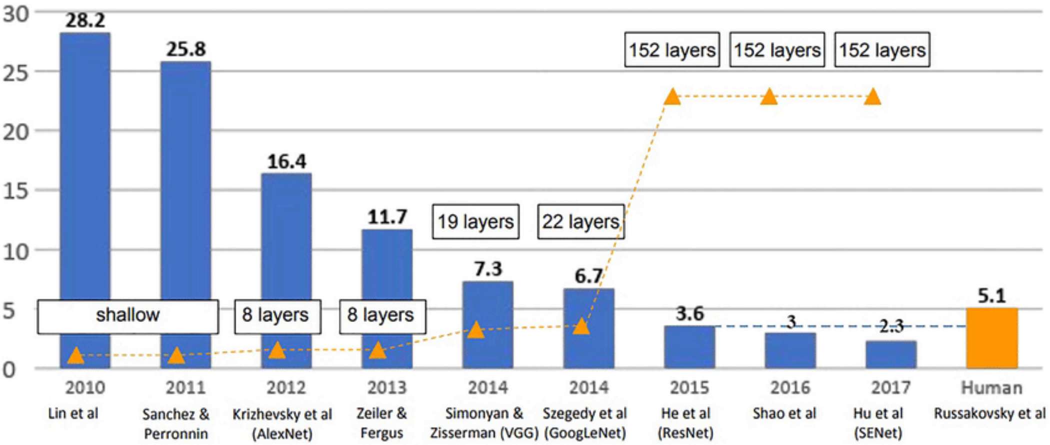


ImageNet is a large-scale image dataset consisting of 1,000 classes, 1 million training samples, and 100,000 test samples.

Modern Era: ResNet 2015

2010s-Present: Deep Learning Revolution

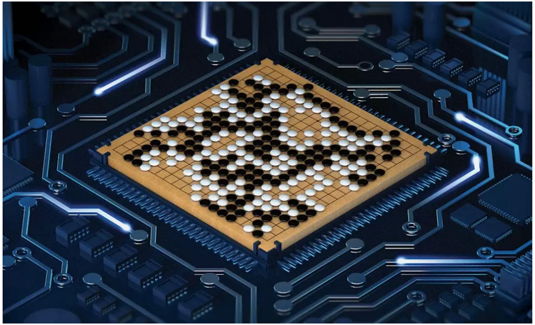
- In 2015, the introduction of skip connection in **ResNet** allowed the training of extremely ResNet, *e.g.*, 152 layers, achieving **better-than-human** performance on ImageNet



Modern Era: AlphaGo 2016-2017

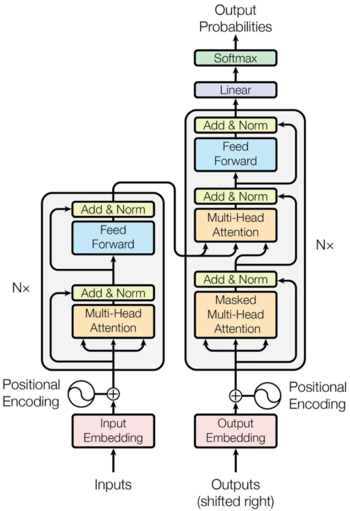
2010s-Present: Deep Learning Revolution

- In 2016, **AlphaGo** gained worldwide attention by defeating South Korean professional Go player Lee Sedol, one of the best players in the world, showing that AI can master complex and strategic games.
- In 2017, AlphaGo further cemented its dominance by defeating the world's number one Go player, Ke Jie of China, in a best-of-three match, **winning all three games**.



Modern Era: The Transformer and Attention 2017-2022

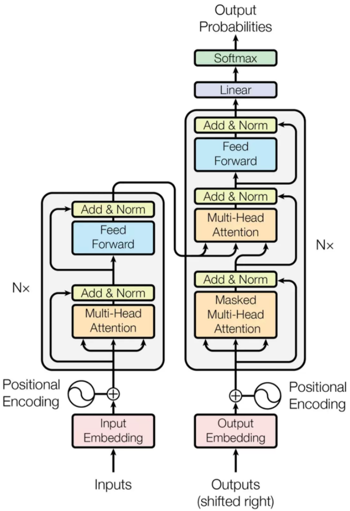
2010s-Present: Deep Learning Revolution



- In 2017, the **Transformer architecture** and **attention mechanism** were introduced, revolutionizing NLP with superior performance in sequence-based tasks.

Modern Era: The Transformer and Attention 2017-2022

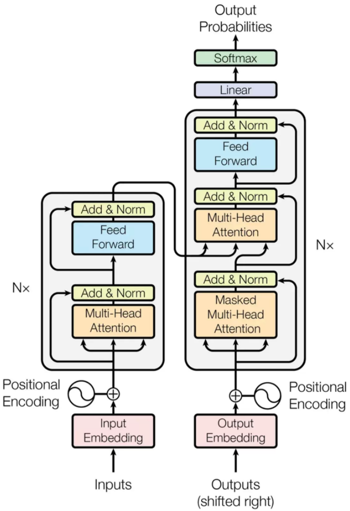
2010s-Present: Deep Learning Revolution



- In 2017, the **Transformer architecture** and **attention mechanism** were introduced, revolutionizing NLP with superior performance in sequence-based tasks.
- In 2018, **BERT** and **GPT** emerged as foundational pre-trained models in NLP, significantly improving performance across a wide range of downstream tasks.

Modern Era: The Transformer and Attention 2017-2022

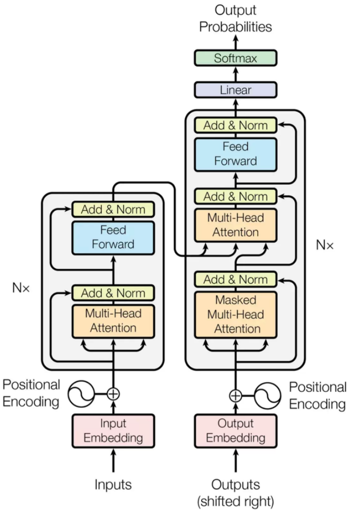
2010s-Present: Deep Learning Revolution



- In 2017, the **Transformer architecture** and **attention mechanism** were introduced, revolutionizing NLP with superior performance in sequence-based tasks.
- In 2018, **BERT** and **GPT** emerged as foundational pre-trained models in NLP, significantly improving performance across a wide range of downstream tasks.
- In 2020, OpenAI developed **GPT-3** as the largest language model at time with **175 billion parameters**, trained on **499 billion tokens** (approximately **570 GB of text**) using around **10,000 GPUs** over several months.

Modern Era: The Transformer and Attention 2017-2022

2010s-Present: Deep Learning Revolution



- In 2017, the **Transformer architecture** and **attention mechanism** were introduced, revolutionizing NLP with superior performance in sequence-based tasks.
- In 2018, **BERT** and **GPT** emerged as foundational pre-trained models in NLP, significantly improving performance across a wide range of downstream tasks.
- In 2020, OpenAI developed **GPT-3** as the largest language model at time with **175 billion parameters**, trained on **499 billion tokens** (approximately **570 GB of text**) using around **10,000 GPUs** over several months.
- In 2022, **ChatGPT** was released by OpenAI, based on the GPT-3.5 model. It quickly gained widespread attention for its ability to generate human-like text.

Modern Era: Generative AI 2021-Present

2010s-Present: Deep Learning Revolution

- In 2021, **Diffusion models** like Denoising Diffusion Probabilistic Models (DDPM) became prominent for generating high-quality images, challenging the dominance of GANs.

Modern Era: Generative AI 2021-Present

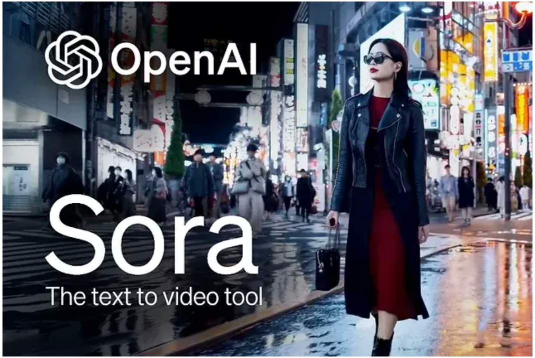
2010s-Present: Deep Learning Revolution

- In 2021, **Diffusion models** like Denoising Diffusion Probabilistic Models (DDPM) became prominent for generating high-quality images, challenging the dominance of GANs.
- In 2022, Stability AI released **Stable Diffusion**, allowing users to generate images from text prompts based on diffusion models.

Modern Era: Generative AI 2021-Present

2010s-Present: Deep Learning Revolution

- In 2021, **Diffusion models** like Denoising Diffusion Probabilistic Models (DDPM) became prominent for generating high-quality images, challenging the dominance of GANs.
- In 2022, Stability AI released **Stable Diffusion**, allowing users to generate images from text prompts based on diffusion models.
- In 2024, OpenAI introduced **Sora**, a text-to-video model based on diffusion models that can generate about 1 minute of high-quality video from text prompts.



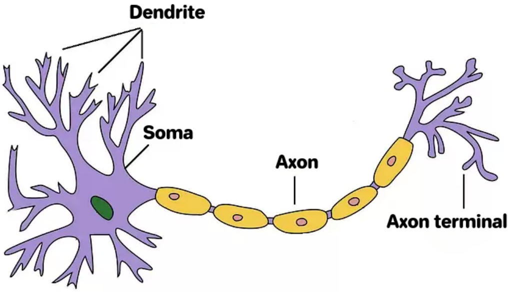
Applications of Deep Learning

- **Computer Vision**
- **Natural Language Processing (NLP)**
- **Speech Recognition and Generation**
- **Biomedical Science and Healthcare**
- Self-Driving Vehicles
- Recommendation Systems
- Finance and Fraud Detection
- Medical Imaging and Diagnostics
- Robotics and Automation
- Music Technology and Audio Processing
- Climate Science and Environmental Monitoring
- Biological Sciences and Bioinformatics

Applications of Deep Learning

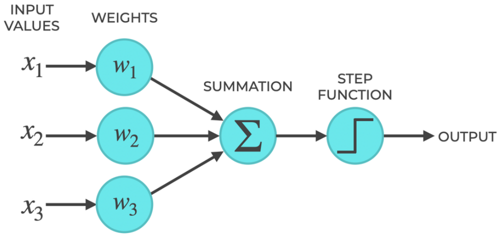
- **Computer Vision**
- **Natural Language Processing (NLP)**
- **Speech Recognition and Generation**
- **Biomedical Science and Healthcare**
- Self-Driving Vehicles
- Recommendation Systems
- Finance and Fraud Detection
- Medical Imaging and Diagnostics
- Robotics and Automation
- Music Technology and Audio Processing
- Climate Science and Environmental Monitoring
- Biological Sciences and Bioinformatics
- Agricultural Technology
- Meteorology and Weather Prediction
- Cloud Computing and Data Centers
- Smart Manufacturing and Industry 4.0
- Logistics and Supply Chain Optimization
- Food Security and Sustainable Agriculture
- Cybersecurity and Threat Detection
- Software Engineering and DevOps
- Materials Science and Engineering
- Mechanical Engineering and System Design
- Digital Media, Filmmaking, and Animation
- User Interface and User Experience (UI/UX) Development

Biological Neuron



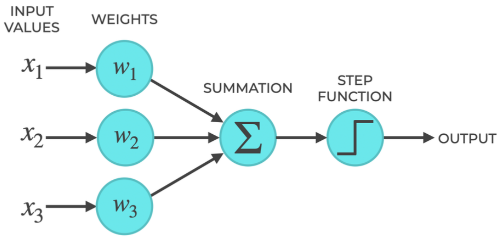
- **Dendrite**: Receives signals from other neurons
- **Soma**: Processes the information
- **Axon**: Transmits signals away from Soma
- **Axon terminal**: Send signals to other neurons

Perceptron



- Each input x_i is multiplied by its corresponding weight w_i
- The weighted inputs are summed together (along with the bias b).
- The sum is passed through a step function to produce the estimated output.

Mathematical Form of Perceptron



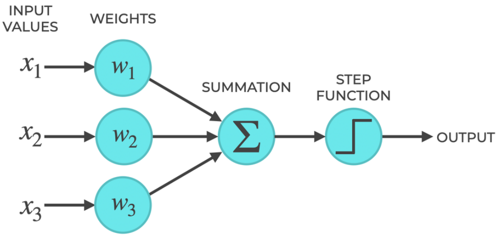
Mathematical Form of Perceptron:

$$\hat{y} = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

- x_i are the **input**, w_i are the **weights**, b is the **bias**, $\hat{y}$ is the **prediction**, ϕ is the **step function**:

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

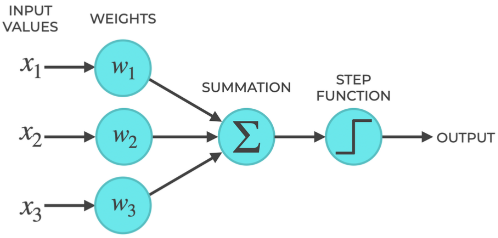
Perceptron Example



Consider

- Inputs: x_1, $x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1$, $w_2 = 1$
- Bias: $b = -1.5$

Perceptron Example



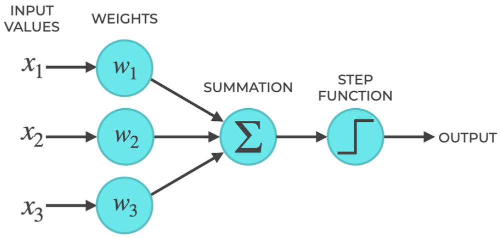
Consider

- Inputs: $x_1,\ x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1,\ w_2 = 1$
- Bias: $b = -1.5$

The perceptron output is computed as:

$$\hat{y} = \phi(w_1 x_1 + w_2 x_2 + b) = \phi(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

Perceptron Example



Consider

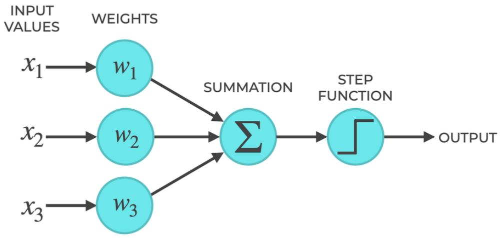
- Inputs: x_1, $x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1$, $w_2 = 1$
- Bias: $b = -1.5$

The perceptron output is computed as:

$$\hat{y} = \phi(w_1 x_1 + w_2 x_2 + b) = \phi(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

- **Input:** $x_1 = 0, x_2 = 0$; **Output**: $z = 1 \cdot 0 + 1 \cdot 0 - 1.5 = -1.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$

Perceptron Example



Consider

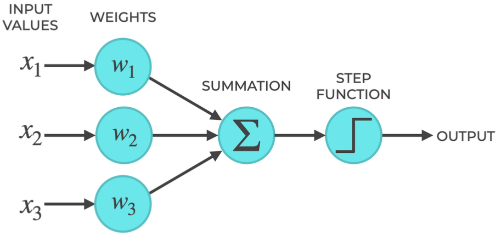
- Inputs: $x_1, x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1$, $w_2 = 1$
- Bias: $b = -1.5$

The perceptron output is computed as:

$$\hat{y} = \phi(w_1 x_1 + w_2 x_2 + b) = \phi(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

- **Input:** $x_1 = 0, x_2 = 0$; **Output**: $z = 1 \cdot 0 + 1 \cdot 0 - 1.5 = -1.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 0, x_2 = 1$; **Output**: $z = 1 \cdot 0 + 1 \cdot 1 - 1.5 = -0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$

Perceptron Example



Consider

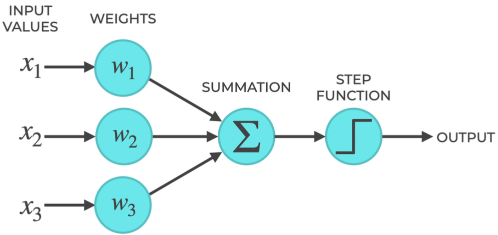
- Inputs: x_1, $x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1$, $w_2 = 1$
- Bias: $b = -1.5$

The perceptron output is computed as:

$$\hat{y} = \phi(w_1 x_1 + w_2 x_2 + b) = \phi(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

- **Input:** $x_1 = 0, x_2 = 0$; **Output**: $z = 1 \cdot 0 + 1 \cdot 0 - 1.5 = -1.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 0, x_2 = 1$; **Output**: $z = 1 \cdot 0 + 1 \cdot 1 - 1.5 = -0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 1, x_2 = 0$; **Output**: $z = 1 \cdot 1 + 1 \cdot 0 - 1.5 = -0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 1, x_2 = 1$; **Output**: $z = 1 \cdot 1 + 1 \cdot 1 - 1.5 = 0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 1.$

Perceptron Example



Consider

- Inputs: $x_1, x_2 \in \{0, 1\}$ are **binary** values
- Weights: $w_1 = 1,\ w_2 = 1$
- Bias: $b = -1.5$

The perceptron output is computed as:

$$\hat{y} = \phi(w_1 x_1 + w_2 x_2 + b) = \phi(1 \cdot x_1 + 1 \cdot x_2 - 1.5)$$

- **Input:** $x_1 = 0, x_2 = 0$; **Output**: $z = 1 \cdot 0 + 1 \cdot 0 - 1.5 = -1.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 0, x_2 = 1$; **Output**: $z = 1 \cdot 0 + 1 \cdot 1 - 1.5 = -0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 1, x_2 = 0$; **Output**: $z = 1 \cdot 1 + 1 \cdot 0 - 1.5 = -0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 0.$
- **Input:** $x_1 = 1, x_2 = 1$; **Output**: $z = 1 \cdot 1 + 1 \cdot 1 - 1.5 = 0.5 \quad \Rightarrow \quad \hat{y} = \phi(z) = 1.$

Conclusion

The perceptron correctly implements the **AND** operator.

Matrix-Vector Representation of Perceptron

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Matrix-Vector Representation of Perceptron

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

- Define vectors $\boldsymbol{x}$, $\boldsymbol{w} \in \mathbb{R}^n$:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Matrix-Vector Representation of Perceptron

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

- Define vectors $\boldsymbol{x}$, $\boldsymbol{w} \in \mathbb{R}^n$:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

- The perceptron can be defined in vector form:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

where the inner or dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$ is given by

$$\boldsymbol{w}^\top \boldsymbol{x} = \sum_{i=1}^{n} w_i x_i$$

Using Perceptrons to Implement Logical Operators

Logical operators such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$):

- $\wedge$: the result is true if both x_1 and x_2 are
- $\vee$: the result is true if either x_1 and x_2 is
- $\neg$: the result is true if the input is not

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$\neg x_1$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Table: Boolean table illustrating $\wedge$, $\vee$, and $\neg$ operations with x_1 and x_2

Using Perceptrons to Implement Logical Operators

Logical operators such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$):

- $\wedge$: the result is true if both x_1 and x_2 are
- $\vee$: the result is true if either x_1 and x_2 is
- $\neg$: the result is true if the input is not

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$\neg x_1$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Table: Boolean table illustrating $\wedge$, $\vee$, and $\neg$ operations with x_1 and x_2

Perceptron can be used to implement them:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b) = \phi(w_1 x_1 + w_2 x_2 + b)$$

- $\wedge$: $\boldsymbol{w} = [1, 1]$ and $b = -1.5$
- $\vee$: $\boldsymbol{w} = [1, 1]$ and $b - 0.5$
- $\neg$: $w = -1$ and $b = 0.5$

Limitations of Perceptron

The perceptron cannot solve **nonlinear** problems such as the logical operator XOR ($\oplus$):

- $\oplus$: the result is true if x_1 and x_2 are different

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$\neg x_1$	$x_1 \oplus x_2$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Table: Boolean table illustrating $\wedge$, $\vee$, and $\oplus$ operations with x_1 and x_2

Limitations of Perceptron

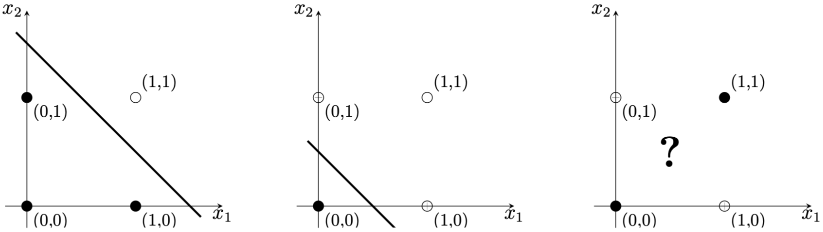
The perceptron cannot solve **nonlinear** problems such as the logical operator XOR ($\oplus$):

- $\oplus$: the result is true if x_1 and x_2 are different

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$\neg x_1$	$x_1 \oplus x_2$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

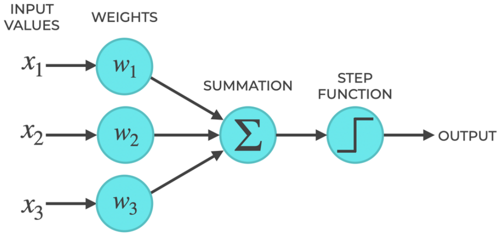
Table: Boolean table illustrating $\wedge$, $\vee$, and $\oplus$ operations with x_1 and x_2

The perceptron can only solve **linearly** separable data:



(a) AND ($\wedge$) operator (b) OR ($\vee$) operator (c) XOR ($\oplus$) operator

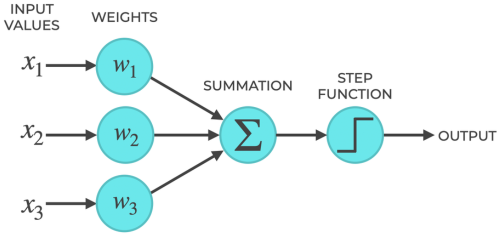
Summary of Perceptron



The perceptron is defined as:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

Summary of Perceptron

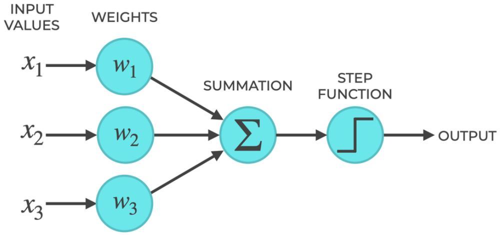


The perceptron is defined as:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

- **Mathematical Model**: It computes the weighted sum of the inputs along with the bias term

Summary of Perceptron

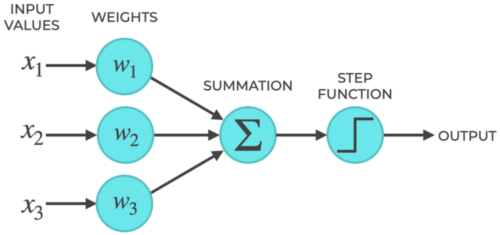


The perceptron is defined as:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

- **Mathematical Model**: It computes the weighted sum of the inputs along with the bias term
- **Activation**: The perceptron (or neuron) is activated by the step (activation) function if the weighted sum exceeds a certain threshold.

Summary of Perceptron

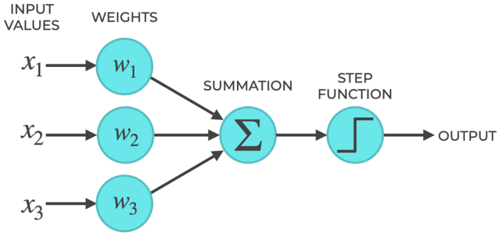


The perceptron is defined as:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

- **Mathematical Model**: It computes the weighted sum of the inputs along with the bias term
- **Activation**: The perceptron (or neuron) is activated by the step (activation) function if the weighted sum exceeds a certain threshold.
- **Linear Separability**: The perceptron can classify linearly separable data, *e.g.*, $\wedge$, $\vee$, and $\neg$.

Summary of Perceptron



The perceptron is defined as:

$$\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b),$$

- **Mathematical Model**: It computes the weighted sum of the inputs along with the bias term
- **Activation**: The perceptron (or neuron) is activated by the step (activation) function if the weighted sum exceeds a certain threshold.
- **Linear Separability**: The perceptron can classify linearly separable data, *e.g.*, $\wedge$, $\vee$, and $\neg$.
- **Limitation**: It cannot solve **nonlinear** problems, *e.g.*, $\oplus$.

Why Multilayer Perceptrons (MLP)?

Recap: A single perceptron can implement AND, OR, and NOT, but **not** XOR

Why Multilayer Perceptrons (MLP)?

Recap: A single perceptron can implement AND, OR, and NOT, but **not** XOR
However, XOR can be implemented by **multiple** percetrons.

Why Multilayer Perceptrons (MLP)?

Recap: A single perceptron can implement AND, OR, and NOT, but **not** XOR
However, XOR can be implemented by **multiple** percetrons.

- **NAND** $\uparrow$: the result is false if both inputs are true

x_1	x_2	$x_1 \wedge x_2$	$x_1 \uparrow x_2$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

- A single perceptron can implement NAND with $\boldsymbol{w} = [-1, -1]$, and $b = 1.5$

Why Multilayer Perceptrons (MLP)?

Recap: A single perceptron can implement AND, OR, and NOT, but **not** XOR
However, XOR can be implemented by **multiple** percetrons.

- **NAND** $\uparrow$: the result is false if both inputs are true

x_1	x_2	$x_1 \wedge x_2$	$x_1 \uparrow x_2$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

- A single perceptron can implement NAND with $\boldsymbol{w} = [-1, -1]$, and $b = 1.5$
- We can express XOR in terms of AND, OR, and NAND as follows

$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge (x_1 \uparrow x_2).$$

Why Multilayer Perceptrons (MLP)?

Recap: A single perceptron can implement AND, OR, and NOT, but **not** XOR
However, XOR can be implemented by **multiple** percetrons.

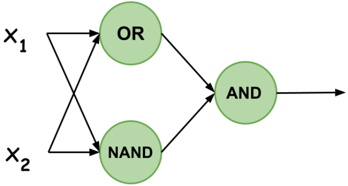
- **NAND** $\uparrow$: the result is false if both inputs are true

x_1	x_2	$x_1 \wedge x_2$	$x_1 \uparrow x_2$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

- A single perceptron can implement NAND with $\boldsymbol{w} = [-1, -1]$, and $b = 1.5$
- We can express XOR in terms of AND, OR, and NAND as follows

$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge (x_1 \uparrow x_2).$$

This forms a **2-layer network**:



MLP for XOR

The XOR function can be computed using MLP as follows:

- Define logical operators using single perceptrons:

$$h_i(\boldsymbol{x}) = \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i), \quad \forall i \in \{1, 2, 3\},$$

  where the weight and biases are:

$$\begin{aligned} \textbf{OR}: &\quad \boldsymbol{w}_1 = [1, 1], \quad b_1 = -0.5 \\ \textbf{NAND}: &\quad \boldsymbol{w}_2 = [-1, -1], \quad b_2 = 1.5 \\ \textbf{AND}: &\quad \boldsymbol{w}_3 = [1, 1], \quad b_3 = -1.5 \end{aligned}$$

MLP for XOR

The XOR function can be computed using MLP as follows:

- Define logical operators using single perceptrons:

$$h_i(\boldsymbol{x}) = \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i), \quad \forall i \in \{1, 2, 3\},$$

  where the weight and biases are:

$$\begin{aligned} \textbf{OR}: &\quad \boldsymbol{w}_1 = [1, 1], \quad b_1 = -0.5 \\ \textbf{NAND}: &\quad \boldsymbol{w}_2 = [-1, -1], \quad b_2 = 1.5 \\ \textbf{AND}: &\quad \boldsymbol{w}_3 = [1, 1], \quad b_3 = -1.5 \end{aligned}$$

- Define the **activation vector** $\boldsymbol{a} \in \mathbb{R}^2$ as the intermediate results from the first layer:

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

  where $a_1 = h_1(\boldsymbol{x})$ from the OR output and $a_2 = h_2(\boldsymbol{x})$ from the NAND output.

MLP for XOR

The XOR function can be computed using MLP as follows:

- Define logical operators using single perceptrons:

$$h_i(\boldsymbol{x}) = \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i), \quad \forall i \in \{1, 2, 3\},$$

  where the weight and biases are:

$$\begin{aligned} \textbf{OR}: &\quad \boldsymbol{w}_1 = [1, 1], \quad b_1 = -0.5 \\ \textbf{NAND}: &\quad \boldsymbol{w}_2 = [-1, -1], \quad b_2 = 1.5 \\ \textbf{AND}: &\quad \boldsymbol{w}_3 = [1, 1], \quad b_3 = -1.5 \end{aligned}$$

- Define the **activation vector** $\boldsymbol{a} \in \mathbb{R}^2$ as the intermediate results from the first layer:

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

  where $a_1 = h_1(\boldsymbol{x})$ from the OR output and $a_2 = h_2(\boldsymbol{x})$ from the NAND output.
- Compute the estimated output $\hat{y}$ using the activation vector $\boldsymbol{a}$ from the first layer:

$$\hat{y} = h_3(\boldsymbol{a}) = \phi(\boldsymbol{w}_3^\top \boldsymbol{a} + b_3)$$

  where h_3 represents the AND operation on the outputs of OR and NAND.

Structure of MLP

For each **hidden layer** in a multi-layer perceptron:

- The **input vector** $\boldsymbol{x} \in \mathbb{R}^d$ is obtained from the previous layer (or from the input data for the first layer).

Structure of MLP

For each **hidden layer** in a multi-layer perceptron:

- The **input vector** $\boldsymbol{x} \in \mathbb{R}^d$ is obtained from the previous layer (or from the input data for the first layer).
- We define n perceptrons, each with independent **weights** $\boldsymbol{w}_i \in \mathbb{R}^d$ and a **bias** $b_i \in \mathbb{R}$ for $i \in [n] := \{1, 2, \ldots, n\}$:

$$h_i(\boldsymbol{x}) = \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i)$$

where ϕ is the **activation function**.

Structure of MLP

For each **hidden layer** in a multi-layer perceptron:

- The **input vector** $\boldsymbol{x} \in \mathbb{R}^d$ is obtained from the previous layer (or from the input data for the first layer).
- We define n perceptrons, each with independent **weights** $\boldsymbol{w}_i \in \mathbb{R}^d$ and a **bias** $b_i \in \mathbb{R}$ for $i \in [n] := \{1, 2, \dots, n\}$:

$$h_i(\boldsymbol{x}) = \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i)$$

  where ϕ is the **activation function**.
- The computed outputs $h_i(\boldsymbol{x})$ are stacked into an **activation vector** $\boldsymbol{a} \in \mathbb{R}^n$, representing the output of this layer:

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

  where $a_i = h_i(\boldsymbol{x})$ for each i.

Matrix-Vector Representation of MLP

To rewrite the MLP in matrix-vector form for each layer ℓ:

- Define the **weight matrix** $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ and the **bias vector** $\boldsymbol{b} \in \mathbb{R}^{n}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_n^\top \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

  where each $\boldsymbol{w}_i \in \mathbb{R}^d$ and b_i represents the weights and bias for the i-th perceptron.

Matrix-Vector Representation of MLP

To rewrite the MLP in matrix-vector form for each layer ℓ:

- Define the **weight matrix** $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ and the **bias vector** $\boldsymbol{b} \in \mathbb{R}^{n}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_n^\top \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

  where each $\boldsymbol{w}_i \in \mathbb{R}^d$ and b_i represents the weights and bias for the i-th perceptron.
- Define the **pre-activation vector** $\boldsymbol{z} \in \mathbb{R}^n$ as:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} \boldsymbol{w}_1^\top \boldsymbol{x} + b_1 \\ \vdots \\ \boldsymbol{w}_n^\top \boldsymbol{x} + b_n \end{bmatrix}$$

  This combines the weighted sum of the inputs from the previous layer.

Matrix-Vector Representation of MLP

To rewrite the MLP in matrix-vector form for each layer ℓ:

- Define the **weight matrix** $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ and the **bias vector** $\boldsymbol{b} \in \mathbb{R}^{n}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_n^\top \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

  where each $\boldsymbol{w}_i \in \mathbb{R}^d$ and b_i represents the weights and bias for the i-th perceptron.

- Define the **pre-activation vector** $\boldsymbol{z} \in \mathbb{R}^n$ as:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} \boldsymbol{w}_1^\top \boldsymbol{x} + b_1 \\ \vdots \\ \boldsymbol{w}_n^\top \boldsymbol{x} + b_n \end{bmatrix}$$

  This combines the weighted sum of the inputs from the previous layer.

- Apply the activation function ϕ **element-wise** to $\boldsymbol{z}$ to obtain the activation vector $\boldsymbol{a} \in \mathbb{R}^n$:

$$\boldsymbol{a} = \phi(\boldsymbol{z}) = \begin{bmatrix} \phi(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1) \\ \vdots \\ \phi(\boldsymbol{w}_n^\top \boldsymbol{x} + b_n) \end{bmatrix}$$

  where ϕ is applied to each element of the pre-activation vector $\boldsymbol{z}$.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell} \boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.
3. The total number of layers L defines the **depth** of the network.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.
3. The total number of layers L defines the **depth** of the network.
4. The final estimated output $\hat{\boldsymbol{y}} = \boldsymbol{x}^{L}$ can have **multiple dimensions**, depending on the task (e.g., classification or regression).

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.
3. The total number of layers L defines the **depth** of the network.
4. The final estimated output $\hat{\boldsymbol{y}} = \boldsymbol{x}^{L}$ can have **multiple dimensions**, depending on the task (e.g., classification or regression).
5. The **width** and **depth** are hyperparameters chosen by the network designer.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\begin{aligned} \boldsymbol{z}^\ell &= \boldsymbol{W}^\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}^\ell, \\ \boldsymbol{x}^\ell &= \phi(\boldsymbol{z}^\ell), \end{aligned}$$

where $\boldsymbol{x}^\ell = \boldsymbol{a}^\ell$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^0 = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.
3. The total number of layers L defines the **depth** of the network.
4. The final estimated output $\hat{\boldsymbol{y}} = \boldsymbol{x}^L$ can have **multiple dimensions**, depending on the task (e.g., classification or regression).
5. The **width** and **depth** are hyperparameters chosen by the network designer.
6. An MLP is capable of solving **nonlinear problems** that a single perceptron cannot handle.

Summary of MLP

An MLP with L layers can be defined in a recurrent manner: for each layer $\ell \in [L]$,

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell},$$
$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{x}^{\ell} = \boldsymbol{a}^{\ell}$ serves as the input to the next layer, and the initial input is $\boldsymbol{x}^{0} = \boldsymbol{x}$.

1. The MLP is also called a **feed-forward network** because the data flows from the input layer to the output layer through hidden layers without any feedback loops.
2. Each hidden layer consists of n_ℓ perceptrons (or neurons), where n_ℓ is referred to as the **width** of the network at layer ℓ.
3. The total number of layers L defines the **depth** of the network.
4. The final estimated output $\hat{\boldsymbol{y}} = \boldsymbol{x}^{L}$ can have **multiple dimensions**, depending on the task (e.g., classification or regression).
5. The **width** and **depth** are hyperparameters chosen by the network designer.
6. An MLP is capable of solving **nonlinear problems** that a single perceptron cannot handle.

Question

How do we effectively select the weights $\boldsymbol{W}^{\ell}$ and biases $\boldsymbol{b}^{\ell}$?