Tools of Artificial Intelligence

Introduction to AI and Feed-forward Neural Network

Smith K. Khare

The Mærsk Mc Kinney Møller Institute
University of Southern Denmark

February 2, 2026

Outline

Introduction

- Artificial Intelligence (AI) has evolved over decades.
- It includes multiple breakthroughs in computing, machine learning, and deep learning.
- AI has gone through cycles of optimism, setbacks (AI winters), and resurgence.

Early AI (1950s - 1970s)

- 1950: Alan Turing proposes the Turing Test and the concept of machine intelligence.
- 1956: Dartmouth Conference, organized by John McCarthy, officially coins the term "Artificial Intelligence."
- 1958: John McCarthy develops the Lisp programming language, instrumental in AI research.
- 1960s-70s: Development of early AI programs such as ELIZA (Weizenbaum) and SHRDLU (Winograd).
- 1973: First AI winter due to reduced funding and skepticism about AI's potential.

AI Winters and Resurgence (1980s - 1990s)

- 1980s: Expert systems, such as MYCIN and XCON, gain traction in business applications.
- 1987-1993: Second AI winter due to limitations in expert systems and declining interest.
- 1990s: Revival of AI through probabilistic models, neural networks, and early machine learning approaches.
- 1997: IBM's Deep Blue defeats world chess champion Garry Kasparov.

Modern AI (2000s - 2019)

- 2006: Geoffrey Hinton and colleagues revive deep learning with backpropagation and neural networks.
- 2010s: Advances in GPU computing accelerate deep learning progress.
- 2011: IBM Watson wins Jeopardy! against human champions.
- 2012: AlexNet revolutionizes image classification using deep convolutional neural networks.
- 2016: AlphaGo, developed by DeepMind, defeats world champion Lee Sedol in Go.
- 2019: AI applications expand into various domains including healthcare, autonomous systems, and finance.

AI Developments (2020 - Present)

- 2020: GPT-3, a powerful language model, is released by OpenAI.
- 2021: AI is increasingly used in drug discovery and climate modeling.
- 2022: Generative AI, such as DALL·E and Stable Diffusion, advances image synthesis.
- 2023: AI legislation and regulation debates intensify worldwide.
- 2024: AI-driven robotics and autonomous agents continue to evolve in real-world applications.

Future of AI

- Ethical considerations and AI safety, including bias and interpretability.
- Advances in artificial general intelligence (AGI) and reinforcement learning.
- Increasing AI integration in daily life, smart cities, and industries.
- Potential regulatory frameworks to ensure responsible AI development.

Data (Observations)

- **Data** is information collected, measured, or observed, often in raw form, that can be used for analysis, decision-making, or understanding phenomena.
- Data is being produced and stored continuously.

Data (Observations)

- **Data** is information collected, measured, or observed, often in raw form, that can be used for analysis, decision-making, or understanding phenomena.
- Data is being produced and stored continuously.
- **Science**: Genome sequences, Climate data, Telescope images, Chemical compositions, Seismic activity records

Data (Observations)

- **Data** is information collected, measured, or observed, often in raw form, that can be used for analysis, decision-making, or understanding phenomena.
- Data is being produced and stored continuously.
- **Science**: Genome sequences, Climate data, Telescope images, Chemical compositions, Seismic activity records
- **Sensor Networks**: Temperature readings, Motion sensor data, Pressure sensor measurements, Air quality levels, Acoustic signals

Data (Observations)

- **Data** is information collected, measured, or observed, often in raw form, that can be used for analysis, decision-making, or understanding phenomena.
- Data is being produced and stored continuously.
- **Science**: Genome sequences, Climate data, Telescope images, Chemical compositions, Seismic activity records
- **Sensor Networks**: Temperature readings, Motion sensor data, Pressure sensor measurements, Air quality levels, Acoustic signals
- **Sensor Networks**: Social media posts, Survey responses, Online reviews GPS location data, Biometric data

Data (Observations)

- **Data** is information collected, measured, or observed, often in raw form, that can be used for analysis, decision-making, or understanding phenomena.
- Data is being produced and stored continuously.
- **Science**: Genome sequences, Climate data, Telescope images, Chemical compositions, Seismic activity records
- **Sensor Networks**: Temperature readings, Motion sensor data, Pressure sensor measurements, Air quality levels, Acoustic signals
- **Sensor Networks**: Social media posts, Survey responses, Online reviews GPS location data, Biometric data
- **Maths**: Matrices, Probability distributions, Graphs (networks), Differential equations, Statistical datasets

Approach to tackle data

Traditional Programming

Approach to tackle data

Traditional Programming

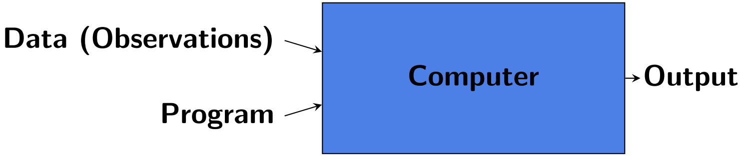


Figure: Traditional Programming

Approach to tackle data

Traditional Programming

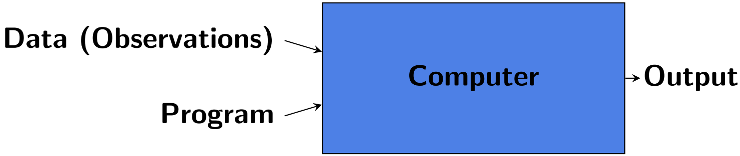


Figure: Traditional Programming

- In traditional programming, humans write explicit rules (code) to process input and produce output.

Approach to tackle data

Traditional Programming

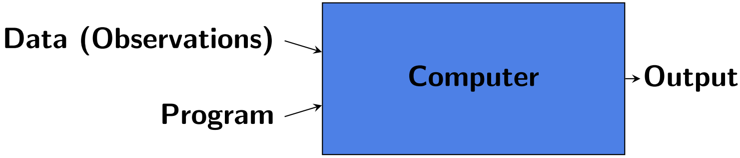


Figure: Traditional Programming

- In traditional programming, humans write explicit rules (code) to process input and produce output.
- Example: You create rules like, "If an email contains 'free money' or 'win now,' mark it as spam." This approach is rigid and can't handle the complexity of diverse spam content.

Approach to tackle data

Machine learning

Approach to tackle data

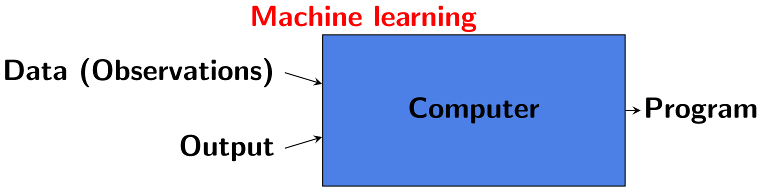


Figure: Machine learning

Approach to tackle data

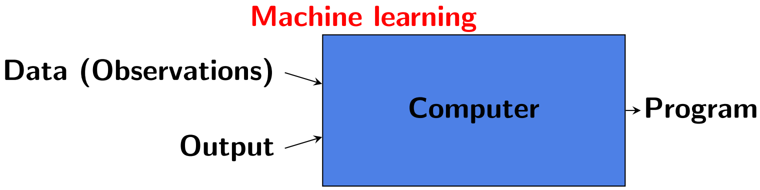


Figure: Machine learning

- In machine learning, the system learns patterns from data to make predictions or decisions without being explicitly programmed.
- Example: A model is trained on a dataset of labeled emails (spam or not) to identify patterns, such as word frequency or sender behavior, and classify emails.

Machine learning

- Human learning is gaining knowledge, skills, or understanding through experience, study, or teaching.
- **Can machines adapt their behavior based on experience?**
- **Yes, using machine learning**

Machine learning

- Human learning is gaining knowledge, skills, or understanding through experience, study, or teaching.
- **Can machines adapt their behavior based on experience?**
- **Yes, using machine learning**
- **Machine learning (ML)** is a branch of artificial intelligence (AI) that focuses on enabling systems to learn and improve based on experience rather than explicit programming. It involves creating models to analyze data, make predictions, and support decision-making.

AI, ML, and deep learning (DL)

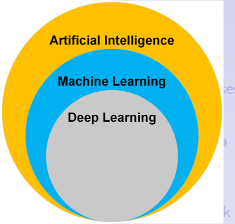
- **AI (Artificial Intelligence)**: The broad field of creating machines or systems that can perform tasks that would normally require human intelligence, like reasoning, learning, and problem-solving.

AI, ML, and deep learning (DL)

- **AI (Artificial Intelligence)**: The broad field of creating machines or systems that can perform tasks that would normally require human intelligence, like reasoning, learning, and problem-solving.
- **ML (Machine Learning)**: A subset of AI focused on creating algorithms that allow computers to learn from data and improve their performance without being explicitly programmed.

AI, ML, and deep learning (DL)

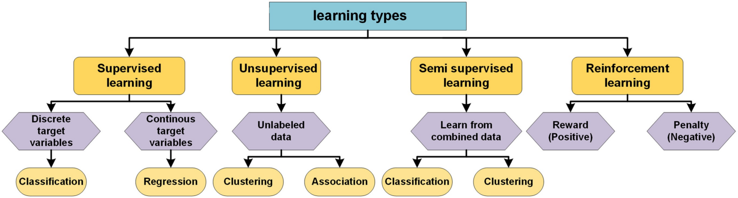
- **AI (Artificial Intelligence)**: The broad field of creating machines or systems that can perform tasks that would normally require human intelligence, like reasoning, learning, and problem-solving.
- **ML (Machine Learning)**: A subset of AI focused on creating algorithms that allow computers to learn from data and improve their performance without being explicitly programmed.
- **DL (Deep Learning)**: A subset of ML that uses neural networks with many layers (hence "**deep**") to analyze large amounts of data. It's particularly good at tasks like image and speech recognition.



Source: Khare, Smith K., et al. "Introduction to smart healthcare and the role of cognitive sensors." Cognitive Sensors, Volume 2: Applications in smart healthcare. Bristol, UK: IOP Publishing, 2023. 1-1.

Types of learning

- Supervised (inductive) learning
- Semi-supervised learning
- Unsupervised learning
- Reinforcement learning



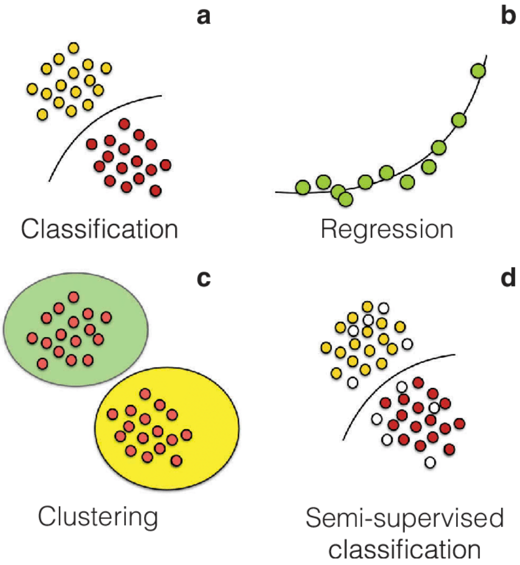
Source: Khare, Smith K., et al. "Introduction to smart healthcare and the role of cognitive sensors." Cognitive Sensors, Volume 2: Applications in smart healthcare. Bristol, UK: IOP Publishing, 2023. 1-1.

Types of learning

Type of Learning	Task in Relation to Labels	Examples
Supervised Learning	Learn from labeled data to predict labels for new data.	▶ **Classification:** Spam detection, image recognition. ▶ **Regression:** House price prediction, stock forecasting.
Unsupervised Learning	Find patterns or structure in unlabeled data.	▶ **Clustering:** Customer segmentation, document grouping. ▶ **Dimensionality Reduction:** PCA, t-SNE.
Semi-Supervised Learning	Use a small amount of labeled data and a large amount of unlabeled data.	▶ **Combination:** Image classification with few labels. ▶ **Self-Training:** Predicting labels for unlabeled data.
Reinforcement Learning	Learn from interactions to maximize reward.	▶ **Game Playing:** AlphaGo, video games. ▶ **Robotics:** Robot navigation, grasping objects.

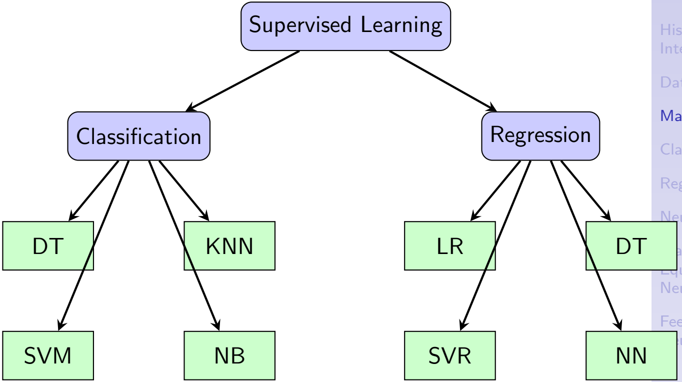
Table: Types of Machine Learning with Examples and Tasks

Examples



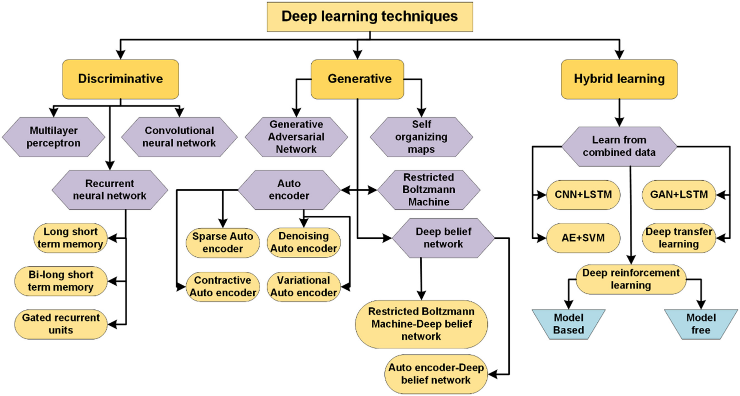
Carrasquilla, Juan. "Machine learning for quantum matter." Advances in Physics: X 5.1 (2020): 1797528.

Supervised ML techniques



Legend:
DT = Decision Trees
KNN = k-Nearest Neighbors
SVM = Support Vector Machines
NB = Naive Bayes
LR = Linear Regression
SVR = Support Vector Regression
NN = Neural Networks

Deep learning techniques



Source: Khare, Smith K., et al. "Introduction to smart healthcare and the role of cognitive sensors." Cognitive Sensors, Volume 2: Applications in smart healthcare. Bristol, UK: IOP Publishing, 2023. 1-1.

Supervised ML: Classification

- **Goal:** Assign a discrete label y_i to each input $\mathbf{x}_i$.
- **Input:**

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}] \quad i = 1, 2, \ldots, N$$

  where d is the number of features, and N is the total number of samples.
- **Output:**

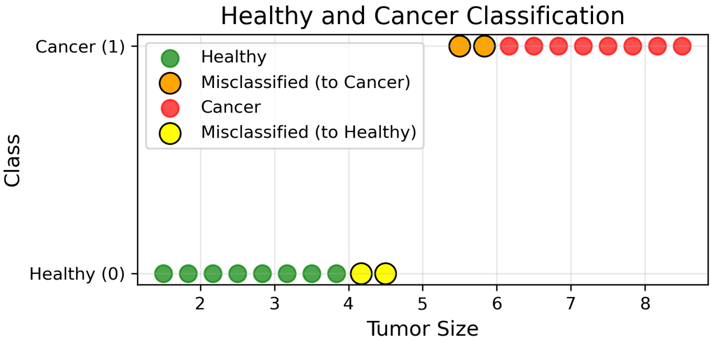
$$y_i \in \{1, 2, \ldots, K\}, \quad K = \text{Number of Classes.}$$

- **Type of y_i: –** y is categorical $==$ classification (e.g., categories or classes).

Visualization of classification

Example: Cancer and healthy subjects classification

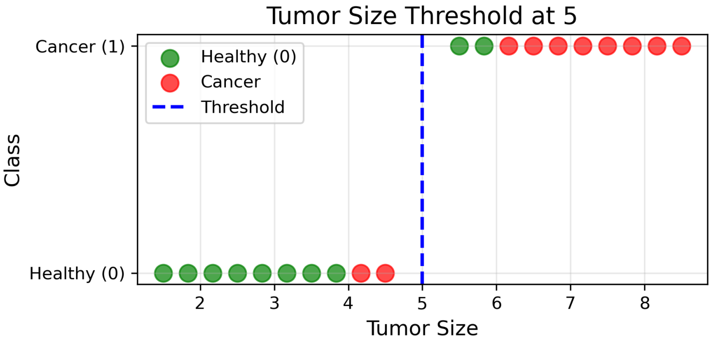
- Input: size of the tumor ($\mathbf{x}_1$).
- Output: $y_i \in \{0, 1\}$, where $0 =$ **Healthy** and $1 =$ **Cancer**.



Visualization of classification

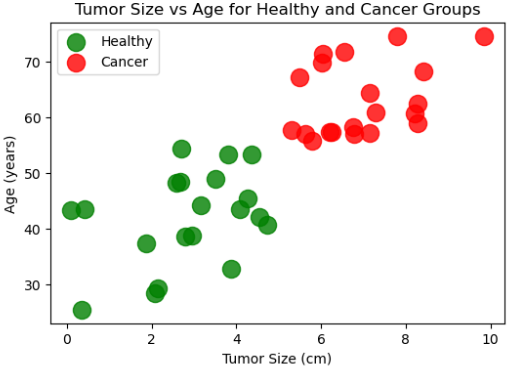
Example: Cancer and healthy subjects classification

- Input: size of the tumor ($\mathbf{x}_1$).
- Output: $y_i \in \{0, 1\}$, where $0 =$ **Healthy** and $1 =$ **Cancer**.



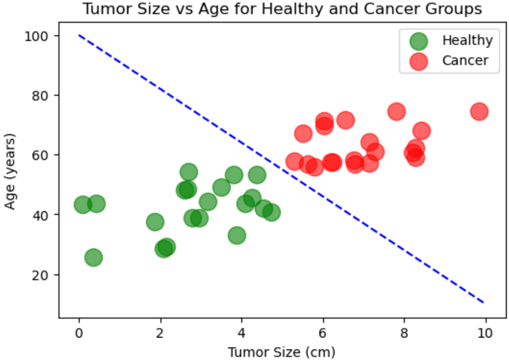
Visualization of classification

- The input x can be multi-dimensional such that $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}] \quad i = 1, 2, \ldots, N$.
- In the same example, if we add one more attribute (feature), namely "age", the dimensions of features are transformed from 1-D to 2-D.



Visualization of classification

- Now, the decision boundary is not as straight-forward like drawing a **vertical line**, since the data points are not separable.
- Similarly, drawing a decision boundaries can be tricky with an addition of attributes.



Supervised ML: Regression

- **Goal:** Predict continuous values y_i based on the input $\mathbf{x}_i$.
- **Input:**

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}], \quad i = 1, 2, \ldots, N.$$

- **Output:**

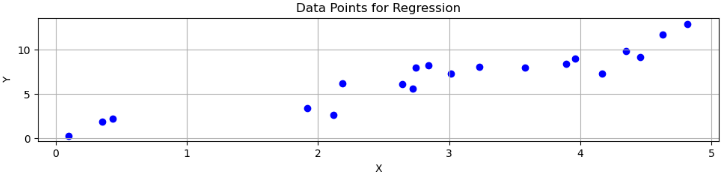
$$y_i \in \mathbb{R}, \quad \text{a real-valued target.}$$

- **Type of y_i:** y is Continuous (real) values $==$ regression(e.g., numerical values).

Course structures

Example: House Price Prediction

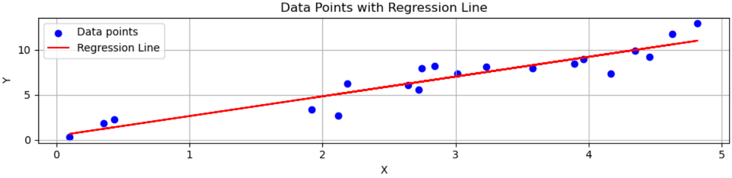
- Input: Features like size, location, and age of the house ($\mathbf{x}_i$).
- Output: y_i = predicted price (e.g., \$250,000).



Course structures

Example: House Price Prediction

- Input: Features like size, location, and age of the house ($\mathbf{x}_i$).
- Output: y_i = predicted price (e.g., \$250,000).



What is a Neural Process?

- Inspired by biological neural networks in the brain.
- Involves processing information through interconnected units called **neurons**.
- Each neuron performs a simple computation and passes the result to the next layer.

Key Idea

The goal is to learn patterns or functions from data to make predictions or decisions.

Biological neuron

- **Dendrites**: Receive input signals from other neurons.
- **Soma**: Processes inputs and combines them.
- **Axon**: Carries the processed signal to the next neuron.
- **Conduction**: Transmits signals across the network.

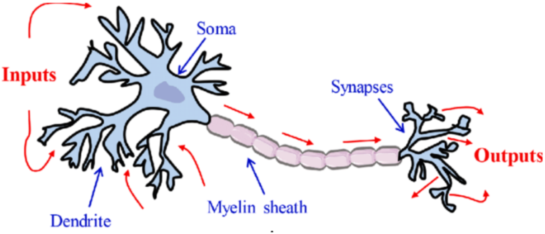


Figure: Biological neuron

Source: https://www.sciencedirect.com/science/article/pii/S2352012421003179?via%3Dihub

Artificial Neuron

- **Inputs (x_i)**: Correspond to signals received by dendrites.
- **Weighted Sum ($z = \sum w_i x_i + b$)**: Mimics the processing in the soma.
- **Activation Output ($y = f(z)$)**: Equivalent to the signal carried by the axon.
- **Forward Propagation**: Analogous to conduction.

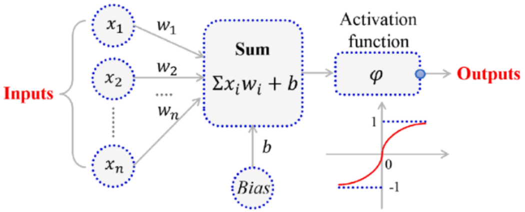


Figure: Artificial neuron

Source: https://www.sciencedirect.com/science/article/pii/S2352012421003179?via%3Dihub

Biological Neuron vs Artificial Neuron

Biological Neuron

- **Dendrites**: Receive input signals from other neurons.
- **Soma**: Processes and integrates inputs.
- **Axon**: Transmits the processed signal.
- **Conduction**: Signal flow along the axon.

Artificial Neuron

- **Inputs (x_i)**: Correspond to dendritic signals.
- **Weighted Sum ($z = \sum w_i x_i + b$)**: Combines and processes inputs, like the soma.
- **Activation Output ($y = f(z)$)**: Acts as the signal passed along the axon.
- **Forward Propagation**: Mimics conduction along neural pathways.

The Mathematical Model of a Neuron

- A neuron receives inputs $x_1, x_2, \ldots, x_n$.
- Each input is associated with a weight $w_1, w_2, \ldots, w_n$.
- The neuron computes a weighted sum:

$$z = \sum_{i=1}^{n} w_i x_i + b,$$

The Mathematical Model of a Neuron

- A neuron receives inputs $x_1, x_2, \ldots, x_n$.
- Each input is associated with a weight $w_1, w_2, \ldots, w_n$.
- The neuron computes a weighted sum:

$$z = \sum_{i=1}^{n} w_i x_i + b,$$

- The **weights** determine the importance of each input.
- The **bias** allows the model to shift the activation function, enabling it to fit the data better.
- z represents the net input to the activation function.

The Mathematical Model of a Neuron

- A neuron receives inputs $x_1, x_2, \ldots, x_n$.
- Each input is associated with a weight $w_1, w_2, \ldots, w_n$.
- The neuron computes a weighted sum:

$$z = \sum_{i=1}^{n} w_i x_i + b,$$

- The **weights** determine the importance of each input.
- The **bias** allows the model to shift the activation function, enabling it to fit the data better.
- z represents the net input to the activation function.

Output of the Neuron

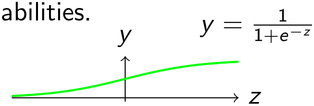
The output y is computed by applying an **activation function** f to z:

$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i + b\right).$$

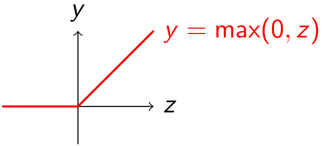
Activation functions introduce non-linearity, allowing the network to model complex patterns.

Activation Functions

- Common activation functions:
  - **Sigmoid**: $f(z) = \frac{1}{1+e^{-z}}$
    Smooth and bounded; outputs values in $(0, 1)$. Suitable for probabilities.

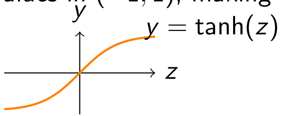


  - **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$
    Computationally efficient and helps mitigate vanishing gradient problems.

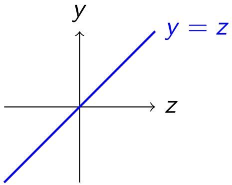


Activation Functions

- Common activation functions:
  - **Tanh**: $f(z) = \tanh(z)$
    Outputs values in $(-1, 1)$, making it zero-centered.



  - **Linear Unit**: $f(z) = z$



What is a Feed-forward Neural Network?

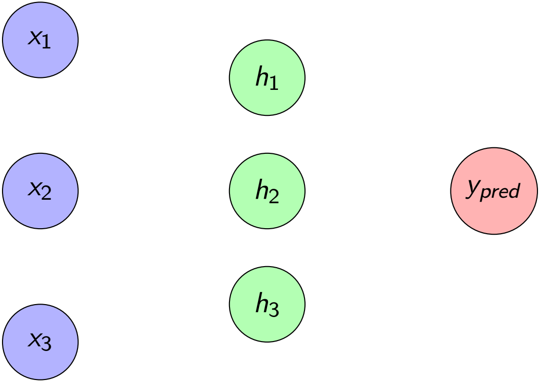
- A type of artificial neural network where connections between nodes do not form a cycle.
- Information moves in one direction — **forward** — from input to output.
- Composed of:
  - **Input Layer**: Accepts raw data.
  - **Hidden Layers**: Perform intermediate computations.
  - **Output Layer**: Produces final result.

Structure of a Feed-forward Neural Network

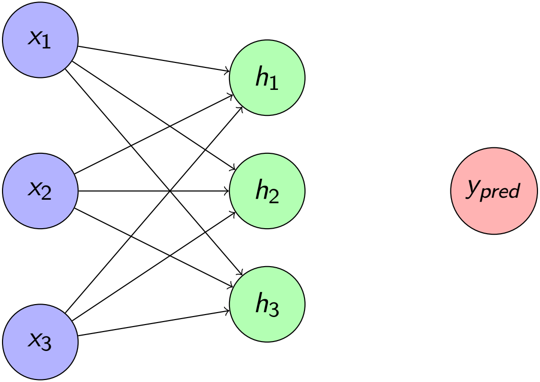
Key Characteristics

- No feedback connections.
- Each layer is fully connected to the next layer.
- Uses **back-propagation** for training.

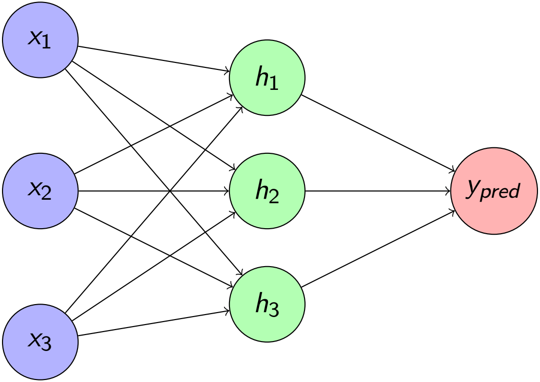
Feed-forward Neural Network: Overview



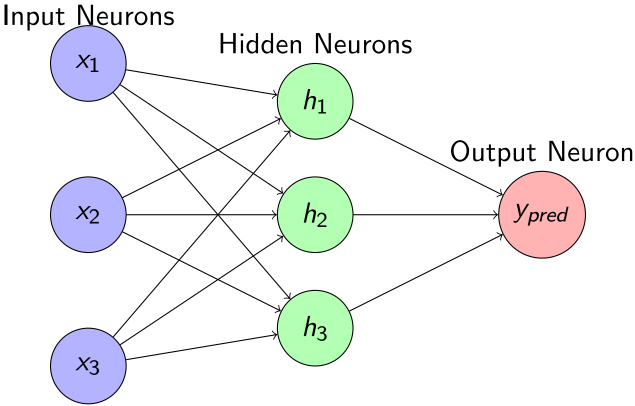
Feed-forward Neural Network: Overview



Feed-forward Neural Network: Overview



Feed-forward Neural Network: Overview



Forward Pass: Step-by-Step Computation

1. **Input to Hidden Layer:**

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j$$

Forward Pass: Step-by-Step Computation

1. **Input to Hidden Layer:**

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j$$

2. **Hidden Layer Activation:**

$$h_j = \sigma(z_j) = \frac{1}{1 + e^{-z_j}}$$

Forward Pass: Step-by-Step Computation

1. **Input to Hidden Layer:**

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j$$

2. **Hidden Layer Activation:**

$$h_j = \sigma(z_j) = \frac{1}{1 + e^{-z_j}}$$

3. **Output Layer Computation:**

$$y = \sum_{j=1}^{m} v_j h_j + c$$

Forward Pass: Step-by-Step Computation

1. **Input to Hidden Layer:**

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j$$

2. **Hidden Layer Activation:**

$$h_j = \sigma(z_j) = \frac{1}{1 + e^{-z_j}}$$

3. **Output Layer Computation:**

$$y = \sum_{j=1}^{m} v_j h_j + c$$

4. **Output Activation:**

$$y_{pred} = \sigma(y)$$

Back-propagation: Overview

Goal: Minimize the loss function $L(y_{pred}, y_{target})$.

Back-propagation: Overview

Goal: Minimize the loss function $L(y_{pred}, y_{target})$.

Steps in Back-propagation:

1. Compute the error at the output layer.
2. Back-propagate the error to the hidden layer.
3. Update weights and biases using the gradient descent algorithm.

Back-propagation: Step-wise Computation

1. **Compute Output Layer Error:**

$$\delta_{\text{output}} = Error \cdot y_{pred}(1 - y_{pred})$$

Back-propagation: Step-wise Computation

1. **Compute Output Layer Error:**

$$\delta_{\text{output}} = Error \cdot y_{pred}(1 - y_{pred})$$

2. **Back-propagate Error to Hidden Layer:**

$$\delta_j = \delta_{\text{output}} \cdot v_j \cdot h_j(1 - h_j)$$

Back-propagation: Step-wise Computation

1. **Compute Output Layer Error:**

$$\delta_{\text{output}} = Error \cdot y_{pred}(1 - y_{pred})$$

2. **Back-propagate Error to Hidden Layer:**

$$\delta_j = \delta_{\text{output}} \cdot v_j \cdot h_j(1 - h_j)$$

3. **Weight Updates:**

$$w_{ij}^{\text{new}} = w_{ij} - \eta \cdot \frac{\partial L}{\partial w_{ij}}$$

Back-propagation: Step-wise Computation

1. **Compute Output Layer Error:**
$$\delta_{\text{output}} = Error \cdot y_{pred}(1 - y_{pred})$$

2. **Back-propagate Error to Hidden Layer:**
$$\delta_j = \delta_{\text{output}} \cdot v_j \cdot h_j(1 - h_j)$$

3. **Weight Updates:**
$$w_{ij}^{\text{new}} = w_{ij} - \eta \cdot \frac{\partial L}{\partial w_{ij}}$$

4. **Gradient Descent Rule:**
$$\Delta w_{ij} = -\eta \cdot \delta_j \cdot x_i$$

Gradient Descent: Overview

- Gradient Descent is an optimization algorithm used to minimize a loss function $L(w)$.
- Update rule:
$$w_{t+1} = w_t - \eta \frac{\partial L(w_t)}{\partial w}$$
- η: Learning rate determines the step size in each iteration.
- Goal: Find the global minimum of $L(w)$.

Gradient Descent: Mathematical Analysis

Update Rule:

$$w_{t+1} = w_t - \eta \frac{\partial L(w_t)}{\partial w}$$

Error Function:

$$L(w) = w^2 + 1$$

Gradient:

$$\frac{\partial L(w)}{\partial w} = 2w$$

Gradient Descent: Mathematical Analysis

Update Rule:

$$w_{t+1} = w_t - \eta \frac{\partial L(w_t)}{\partial w}$$

Error Function:

$$L(w) = w^2 + 1$$

Gradient:

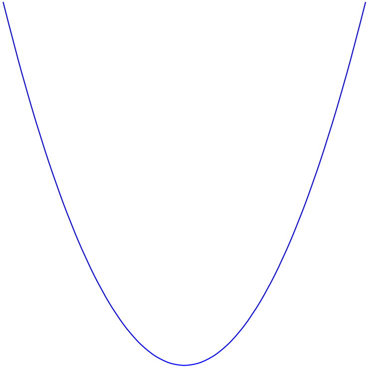
$$\frac{\partial L(w)}{\partial w} = 2w$$

Step Size:

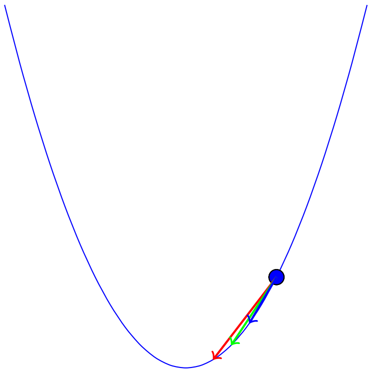
$$w_{t+1} = w_t - \eta \cdot 2w_t$$

Where the step size is proportional to η and w_t.

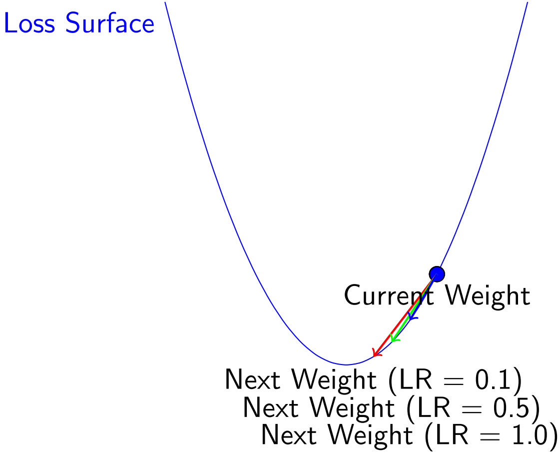
Gradient Descent: Effect of Learning Rates



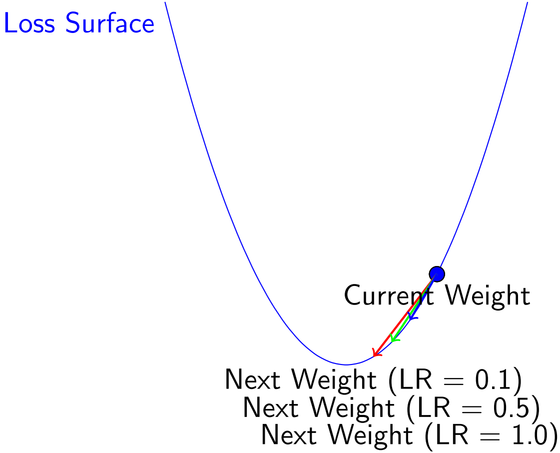
Gradient Descent: Effect of Learning Rates



Gradient Descent: Effect of Learning Rates



Gradient Descent: Effect of Learning Rates



Weight Update Formula:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial E}{\partial w}$$

Summary

- Gradient descent behavior heavily depends on the learning rate.
- Choosing the right learning rate is crucial for optimization:
  - Small η: Can result in slow convergence.
  - Large η: Can cause divergence or overshooting
  - Optimal η: An optimal learning rate balances convergence speed and stability.
- Visualizations provide an intuitive understanding of optimization dynamics.

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$
- Weights between second hidden layer and output layer $= 4 \times 1 = 4$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$
- Weights between second hidden layer and output layer $= 4 \times 1 = 4$
- Bias in hidden layer 1$=4$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$
- Weights between second hidden layer and output layer $= 4 \times 1 = 4$
- Bias in hidden layer 1$= 4$
- Bias in hidden layer 2$= 4$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$
- Weights between second hidden layer and output layer $= 4 \times 1 = 4$
- Bias in hidden layer 1$= 4$
- Bias in hidden layer 2$= 4$
- Bias in output layer 2$= 1$

Question 1

Question: If a feed-forward neural network has 3 input neurons, 2 hidden layers with 4 neurons each, and 1 output neuron, how many total weights are there in the network including bias?

Solution:

- Weights between input and first hidden layer $= 3 \times 4 = 12$
- Weights between first and second hidden layer $= 4 \times 4 = 16$
- Weights between second hidden layer and output layer $= 4 \times 1 = 4$
- Bias in hidden layer 1$=$ 4
- Bias in hidden layer 2$=$ 4
- Bias in output layer 2$=$ 1
- Weights with bias $= 32+9 = 41$

Question 2

Question: Given a feedforward neural network with an input layer of 5 neurons, one hidden layer of 10 neurons, and an output layer of 2 neurons, how many biases are there in the network?

Question 2

Question: Given a feedforward neural network with an input layer of 5 neurons, one hidden layer of 10 neurons, and an output layer of 2 neurons, how many biases are there in the network?

Solution:

- Biases in the hidden layer $= 10$

Question 2

Question: Given a feedforward neural network with an input layer of 5 neurons, one hidden layer of 10 neurons, and an output layer of 2 neurons, how many biases are there in the network?

Solution:

- Biases in the hidden layer $= 10$
- Biases in the output layer $= 2$

Question 2

Question: Given a feedforward neural network with an input layer of 5 neurons, one hidden layer of 10 neurons, and an output layer of 2 neurons, how many biases are there in the network?

Solution:

- Biases in the hidden layer $= 10$
- Biases in the output layer $= 2$
- Total biases $= 10 + 2 = 12$

Question 3

Question: If the activation function for a neuron is sigmoid, what is the output of the neuron when the input x is 0.5?

Question 3

Question: If the activation function for a neuron is sigmoid, what is the output of the neuron when the input x is 0.5?
Solution: $f(x) = \frac{1}{1+e^{-x}}$

Question 3

Question: If the activation function for a neuron is sigmoid, what is the output of the neuron when the input x is 0.5?
Solution: $f(x) = \frac{1}{1+e^{-x}}$
$f(0.5) = \frac{1}{1+e^{-0.5}}$

Question 3

Question: If the activation function for a neuron is sigmoid, what is the output of the neuron when the input x is 0.5?
Solution: $f(x) = \frac{1}{1+e^{-x}}$
$f(0.5) = \frac{1}{1+e^{-0.5}}$
≈ 0.622

Question 4

Question: Given a feed-forward neural network with 2 hidden layers, each with 6 neurons, and an output layer with 1 neuron, how many total neurons are there in the network (exclude input neurons)?

Question 4

Question: Given a feed-forward neural network with 2 hidden layers, each with 6 neurons, and an output layer with 1 neuron, how many total neurons are there in the network (exclude input neurons)?
Solution:

$$\text{Total neurons} = 6 + 6 + 1 = 13 \quad \text{(excluding input neurons)}$$

Training FFNN for AND Gate

Truth Table for AND Gate:

x_1	x_2	y (Target)
0	0	0
0	1	0
1	0	0
1	1	1

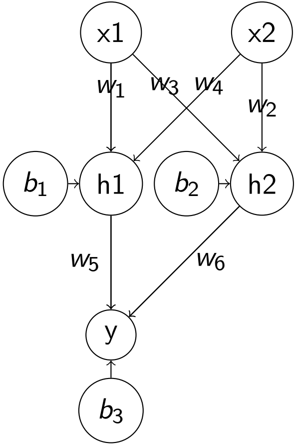
Neural Network Architecture:

- Inputs: x_1, x_2
- Hidden Layer: 2 neurons (with sigmoid activation)
- Output Layer: 1 neuron (with sigmoid activation)

Initial Weights and Biases:

$$w_1 = 0.5, \quad w_2 = 0.5$$
$$w_3 = 0.5, \quad w_4 = 0.5$$
$$w_5 = 0.5, \quad w_6 = 0.5$$
$$b_1 = 0.1, \quad b_2 = 0.1, \quad b_3 = 0.1$$

Neural Network Architecture for AND Gate with Weights and Biases



Forward Pass Calculation for AND Gate

For input $x_1 = 1, x_2 = 1$, let's compute the forward pass:

Hidden Layer Calculations:

$$h_1 = \sigma(w_1 x_1 + w_4 x_2 + b_1) = \sigma(0.5 \cdot 1 + 0.5 \cdot 1 + 0.1) = \sigma(1.1)$$

$$h_2 = \sigma(w_3 x_1 + w_2 x_2 + b_2) = \sigma(0.5 \cdot 1 + 0.5 \cdot 1 + 0.1) = \sigma(1.1)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{so} \quad \sigma(1.1) \approx 0.7503$$

Hence, both a_1 and a_2 are approximately 0.7503.

Output Layer Calculation:

$$y = \sigma(w_5 h_1 + w_6 h_2 + b_3) = \sigma(0.5 \cdot 0.7503 + 0.5 \cdot 0.7503 + 0.1)$$

$$y = \sigma(0.37515 + 0.37515 + 0.1) = \sigma(0.8503)$$

$$\sigma(0.8503) \approx 0.7006$$

So, the predicted output for input $(x_1 = 1, x_2 = 1)$ is $y_{pred} = 0.7006$.

Error Calculation and Back-propagation for AND Gate

Error (Mean Squared Error):

$$E = \frac{1}{2}(y_{\text{target}} - y_{pred})^2 = \frac{1}{2}(1-0.7006)^2 = \frac{1}{2}(0.2994)^2 \approx 0.0448$$

Back-propagation:

Output Layer Gradients:

$$\delta_3 = (y_{\text{target}} - y_{pred}) \cdot \sigma'(y)$$

$$\sigma'(y) = y(1-y) = 0.7006 \cdot (1-0.7006) = 0.7006 \cdot 0.2994 \approx 0.2098$$

$$\delta_3 = (1 - 0.7006) \cdot 0.2098 \approx 0.2994 \cdot 0.2098 \approx 0.0628$$

Hidden Layer Gradients:

$$\delta_1 = \delta_3 \cdot w_5 \cdot \sigma'(h_1) = 0.0628 \cdot 0.5 \cdot \sigma'(0.7503)$$

$$\sigma'(a_1) = 0.7503 \cdot (1 - 0.7503) = 0.7503 \cdot 0.2497 \approx 0.1873$$

$$\delta_1 \approx 0.0628 \cdot 0.5 \cdot 0.1873 \approx 0.0059$$

$$\delta_2 = \delta_3 \cdot w_6 \cdot \sigma'(h_2) = 0.0628 \cdot 0.5 \cdot \sigma'(0.7503) \approx 0.0059$$

Weight updates

Weight Updates:

$$w_{5_{new}} \leftarrow w_5 - \eta \cdot \delta_3 \cdot h_1 = 0.5 - 0.3 \cdot 0.0628 \cdot 0.7503 \approx 0.5 - 0.01413 = 0.4857$$

$$w_{6_{new}} \leftarrow w_6 - \eta \cdot \delta_3 \cdot h_2 = 0.5 - 0.3 \cdot 0.0628 \cdot 0.7503 \approx 0.4857$$

$$w_{1_{new}} \leftarrow w_1 - \eta \cdot \delta_1 \cdot x_1 = 0.5 - 0.3 \cdot 0.0059 \cdot 1 = 0.5 - 0.00177 = 0.4982$$

$$w_{2_{new}} \leftarrow w_2 - \eta \cdot \delta_2 \cdot x_2 = 0.5 - 0.3 \cdot 0.0059 \cdot 1 = 0.4982$$

$$w_{3_{new}} \leftarrow w_3 - \eta \cdot \delta_2 \cdot x_1 = 0.5 - 0.3 \cdot 0.0059 \cdot 1 = 0.4982$$

$$w_{4_{new}} \leftarrow w_4 - \eta \cdot \delta_1 \cdot x_2 = 0.5 - 0.3 \cdot 0.0059 \cdot 1 = 0.4982$$

Bias Updates:

$$b_{3_{new}} \leftarrow b_3 - \eta \cdot \delta_3 = 0.1 - 0.3 \cdot 0.0628 = 0.08116$$

$$b_{1_{new}} \leftarrow b_1 - \eta \cdot \delta_1 = 0.1 - 0.3 \cdot 0.0059 = 0.09823$$

$$b_{2_{new}} \leftarrow b_2 - \eta \cdot \delta_2 = 0.1 - 0.3 \cdot 0.0059 = 0.09823$$

Exercise

Repeat the above calculations for $x_1 = 0, x_2 = 0$ and $y = 0$

Neural Network Architecture:

- Inputs: x_1, x_2
- Hidden Layer: 2 neurons (with sigmoid activation)
- Output Layer: 1 neuron (with sigmoid activation)

Initial Weights and Biases:

$$w_1 = 0.5, \quad w_2 = 0.5$$

$$w_3 = 0.5, \quad w_4 = 0.5$$

$$w_5 = 0.5, \quad w_6 = 0.5$$

$$b_1 = 0.1, \quad b_2 = 0.1, \quad b_3 = 0.1$$

Learning Rate: $\eta = 0.3$

Forward Propagation - Hidden Layer

Hidden Layer Calculations:

$$
\begin{aligned}
z_1 &= w_1 x_1 + w_2 x_2 + b_1 \\
&= (0.5 \times 0) + (0.5 \times 0) + 0.1 \\
&= 0.1 \\
h_1 &= \sigma(z_1) = \frac{1}{1 + e^{-0.1}} \\
&= \frac{1}{1 + 0.9048} \approx 0.525
\end{aligned}
$$

$$
\begin{aligned}
z_2 &= w_3 x_1 + w_4 x_2 + b_2 \\
&= (0.5 \times 0) + (0.5 \times 0) + 0.1 \\
&= 0.1 \\
h_2 &= \sigma(z_2) = \frac{1}{1 + e^{-0.1}} \\
&= \frac{1}{1 + 0.9048} \approx 0.525
\end{aligned}
$$

Forward Propagation - Output Layer

Output Layer Calculation:

$$
\begin{aligned}
z_3 &= w_5 h_1 + w_6 h_2 + b_3 \\
&= (0.5 \times 0.525) + (0.5 \times 0.525) + 0.1 \\
&= 0.2625 + 0.2625 + 0.1 \\
&= 0.625 \\
y_{\text{pred}} &= \sigma(z_3) = \frac{1}{1 + e^{-0.625}} \\
&= \frac{1}{1 + 0.535} \\
&\approx 0.652
\end{aligned}
$$

Error Calculation

$$
\begin{aligned}
E &= \frac{1}{2}(y_{\text{target}} - y_{\text{pred}})^2 \\
&= \frac{1}{2}(0 - 0.652)^2 \\
&= \frac{1}{2}(0.425) \\
&= 0.213
\end{aligned}
$$

Back-propagation - Output Layer

Error Gradient:

$$\begin{aligned}\delta_3 &= (y_{\text{target}} - y_{\text{pred}})y_{\text{pred}}(1 - y_{\text{pred}}) \\ &= (0 - 0.652) \times (0.652) \times (1 - 0.652) \\ &= 0.652 \times 0.652 \times 0.348 \\ &\approx -0.148\end{aligned}$$

Weight and Bias Updates:

$$\begin{aligned}w_{5_{new}} &= w_5 - \eta \cdot \delta_3 \cdot h_1 \\ &= 0.5 - 0.3 \times -0.148 \times 0.525 \approx 5.2331 \\ w_{6_{new}} &= w_6 - \eta \cdot \delta_3 \cdot h_2 \\ &= 0.5 - 0.3 \times -0.148 \times 0.525 \approx 5.2331 \\ b_{3_{new}} &= b_3 - \eta \cdot \delta_3 \\ &= 0.1 - 0.3 \times -0.148 \approx 0.1444\end{aligned}$$

Back-propagation - Hidden Layer

Hidden Layer Gradients:

$$\begin{aligned}\delta_1 &= \delta_3 \cdot w_5 \cdot h_1(1 - h_1)\\ &= -0.148 \times 0.5 \times 0.525 \times (1 - 0.525)\\ &= -0.148 \times 0.5 \times 0.525 \times 0.475 \approx -0.0184\\ \delta_2 &= \delta_3 \cdot w_6 \cdot h_2(1 - h_2)\\ &= -0.148 \times 0.5 \times 0.525 \times 0.475 \approx -0.0184\end{aligned}$$

Weight and Bias Updates:

$$\begin{aligned}b_{1_{new}} &= b_1 - \eta \cdot \delta_1\\ &= 0.1 - 0.3 \times -0.0184\\ &\approx 0.1055\\ b_{2_{new}} &= b_2 - \eta \cdot \delta_2\\ &= 0.1 - 0.3 \times -0.0184\\ &\approx 0.1055\end{aligned}$$

Back-propagation - Hidden Layer

Weight Updates for w_1, w_2, w_3, w_4:

$$
\begin{aligned}
w_{1_{new}} &= w_1 - \eta \cdot \delta_1 \cdot x_1 \\
&= 0.5 - 0.3 \times -0.0184 \times 0 \\
&= 0.5 \\
w_{2_{new}} &= w_2 - \eta \cdot \delta_1 \cdot x_2 \\
&= 0.5 - 0.3 \times -0.0184 \times 0 \\
&= 0.5 \\
w_{3_{new}} &= w_3 - \eta \cdot \delta_2 \cdot x_1 \\
&= 0.5 - 0.3 \times -0.0184 \times 0 \\
&= 0.5 \\
w_{4_{new}} &= w_4 - \eta \cdot \delta_2 \cdot x_2 \\
&= 0.5 - 0.3 \times -0.0184 \times 0 \\
&= 0.5
\end{aligned}
$$

Example of AND Gate

```python
import numpy as np
import matplotlib.pyplot as plt

class NeuralNetwork:
        def __init__(self, input_size, hidden_size, output_size):
                self.input_size = input_size
                self.hidden_size = hidden_size
                self.output_size = output_size

# Initialize weights and biases
                self.weights_input_hidden = np.random.randn(self.input_size,    self.hidden_size)
                self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)
                self.bias_hidden = np.zeros((1, self.hidden_size))
                self.bias_output = np.zeros((1, self.output_size))

        def sigmoid(self, x):
                return 1 / (1 + np.exp(-x))

        def sigmoid_derivative(self, x):
                return x * (1 - x)

        def feedforward(self, X):
                self.hidden_activation = np.dot(X, self.weights_input_hidden) +                 self.bias_hidden
                self.hidden_output = self.sigmoid(self.hidden_activation)
                self.output_activation = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
                self.predicted_output = self.sigmoid(self.output_activation)
                return self.predicted_output
```

Example of AND Gate

```python
    def backward(self, X, y, learning_rate):
        output_error = y - self.predicted_output
        output_delta = output_error * self.sigmoid_derivative(self.predicted_output)
        hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
        hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_output)

        self.weights_hidden_output += np.dot(self.hidden_output.T, output_delta) *      learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) *       learning_rate
        self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) *       learning_rate

        return np.mean(np.square(output_error))
    def train(self, X, y, epochs, learning_rate):
        losses = []
        for epoch in range(epochs):
            self.feedforward(X)
            loss = self.backward(X, y, learning_rate)
            losses.append(loss)
            if epoch % 2000 == 0:
                print(f"Epoch {epoch}, Loss: {loss:.6f}")
        return losses
```

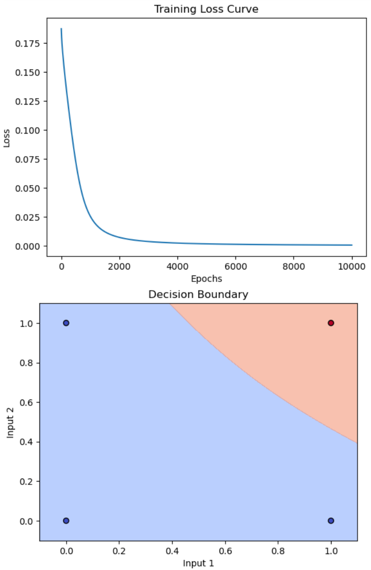
Example of AND Gate

```
# AND Gate input and output
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [0], [0], [1]])

nn = NeuralNetwork(input_size=2, hidden_size=4, output_size=1)
losses = nn.train(X, y, epochs=10000, learning_rate=0.1)

# Print final weights
print("Final Weights - Input to Hidden:")
print(nn.weights_input_hidden)
print("Final Weights - Hidden to Output:")
print(nn.weights_hidden_output)

# Print final predictions
final_predictions = nn.feedforward(X)
print("\nFinal Predictions:")
print(final_predictions)

# Plot training loss curve
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Curve")
plt.show()
```

Example of AND Gate

```python
# Decision Boundary
xx, yy = np.meshgrid(np.linspace(-0.1, 1.1, 100), np.linspace(-0.1, 1.1, 100))
X_grid = np.c_[xx.ravel(), yy.ravel()]
preds = nn.feedforward(X_grid).reshape(xx.shape)

plt.contourf(xx, yy, preds, levels=[0, 0.5, 1], alpha=0.6, cmap="coolwarm")
plt.scatter(X[:, 0], X[:, 1], c=y[:, 0], edgecolors='k', cmap="coolwarm")
plt.xlabel("Input 1")
plt.ylabel("Input 2")
plt.title("Decision Boundary")
plt.show()
```

Example of AND Gate



Summary: Feed-forward and Back-propagation

Feed-forward:

- Compute activations layer by layer from input to output.
- Final prediction $\hat{y}$ is produced.

Summary: Feed-forward and Back-propagation

Feed-forward:

- Compute activations layer by layer from input to output.
- Final prediction $\hat{y}$ is produced.

Back-propagation:

- Compute gradients of the loss function with respect to weights.
- Propagate errors backward from output to input.

Summary: Feed-forward and Back-propagation

Feed-forward:

- Compute activations layer by layer from input to output.
- Final prediction $\hat{y}$ is produced.

Back-propagation:

- Compute gradients of the loss function with respect to weights.
- Propagate errors backward from output to input.

Gradient Descent:

- Adjust weights and biases to minimize the loss function.
- Iteratively converge to optimal parameters.