Introduction to Artificial Neural Networks

Machine Learning (CIC1205)

Eduardo Bezerra

CEFET/RJ

Outline

Introduction

What is an Artificial Neural Network?

Definition:

- An artificial neural network (ANN) is a computational model inspired by the brain.
- It consists of layers of interconnected nodes (called *neurons*) that transform input data to predict or classify outputs.

What is an Artificial Neural Network?

Structure:

- **Input layer:** receives raw data (features).
- **Hidden layers:** perform intermediate transformations using learned weights and nonlinear activations.
- **Output layer:** produces the final prediction or decision.

What is an Artificial Neural Network?

Computation:

$$z = \mathbf{w}^T \mathbf{x} + b \quad , \quad a = \varphi(z)$$

where φ is a nonlinear activation function (e.g., sigmoid, ReLU).

Learning: weights $\mathbf{w}$ and biases b are updated via backpropagation using a loss function.

Analogy: Neural Networks as Decision Pipelines

Imagine that you are a loan officer reviewing applications:

- For each person, you consider features like income, credit score, and debt.

Analogy: Neural Networks as Decision Pipelines

Imagine that you are a loan officer reviewing applications:

- For each person, you consider features like income, credit score, and debt.

How do you decide?

- You mentally assign importance to each feature.
- Combine them into a score, and use your experience to interpret it.
- If the score is high enough → approve the loan; otherwise → reject.

Analogy: Neural Networks as Decision Pipelines

Imagine that you are a loan officer reviewing applications:

- For each person, you consider features like income, credit score, and debt.

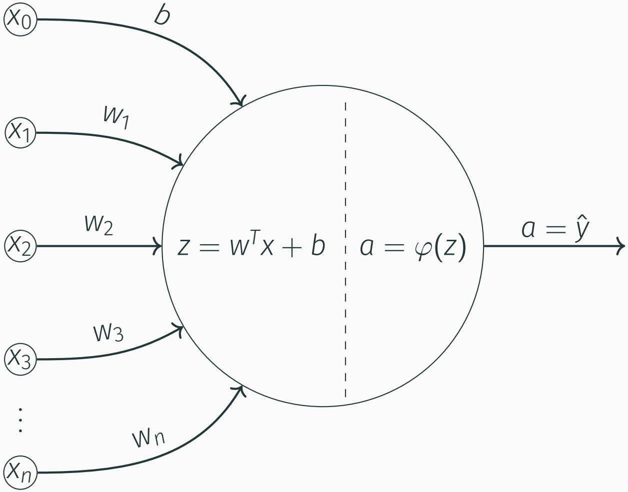
How do you decide?

- You mentally assign importance to each feature.
- Combine them into a score, and use your experience to interpret it.
- If the score is high enough → approve the loan; otherwise → reject.

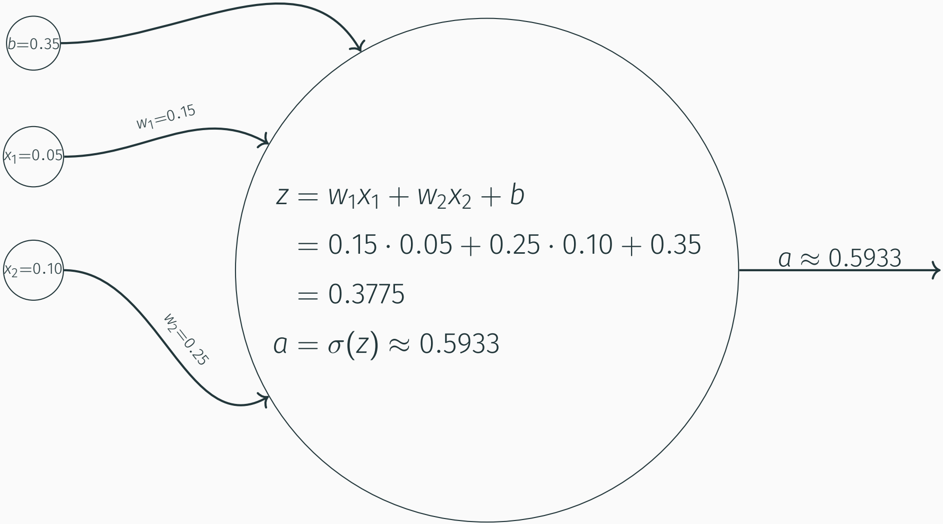
Analogy (to a neural network):

- The features are inputs x_i
- The importance is stored in weights w_i
- The decision is based on a score $z = \sum w_i x_i + b$, passed through an activation function

A Single Neuron



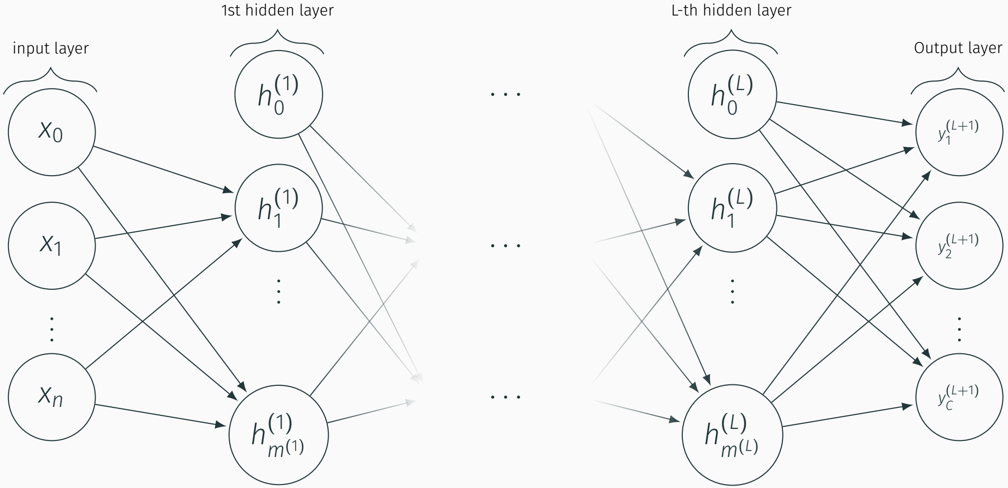
A Single Neuron - numerical example



$$
\begin{aligned}
z &= w_1 x_1 + w_2 x_2 + b \\
&= 0.15 \cdot 0.05 + 0.25 \cdot 0.10 + 0.35 \\
&= 0.3775 \\
a &= \sigma(z) \approx 0.5933
\end{aligned}
$$

Artificial neurons can be put together to build arbitrarily complex artificial neural networks...

Full Neural Network



Model:

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) \quad \text{where} \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

Used for: Binary classification.

Key idea: No hidden layers. Just one output neuron.

The Simplest Neural Network is... Logistic Regression!

One neuron, no hidden layer:

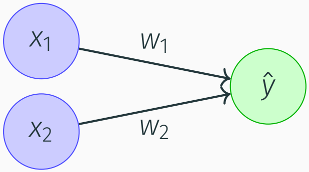
$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

Same structure as logistic regression!

- Inputs $\rightarrow$ Weights $\rightarrow$ Sigmoid activation
- Output neuron produces probability

A logistic regression model is a neural network with one layer and one activation.

Visual: Logistic Regression as a Neural Network



sigmoid activation

Takeaway

- Logistic regression **is** a neural network.
- It has **no hidden layers**.
- It uses a **sigmoid activation** at the output.

Before building deep networks, understand this single-neuron foundation.

But let's use a less simple example...

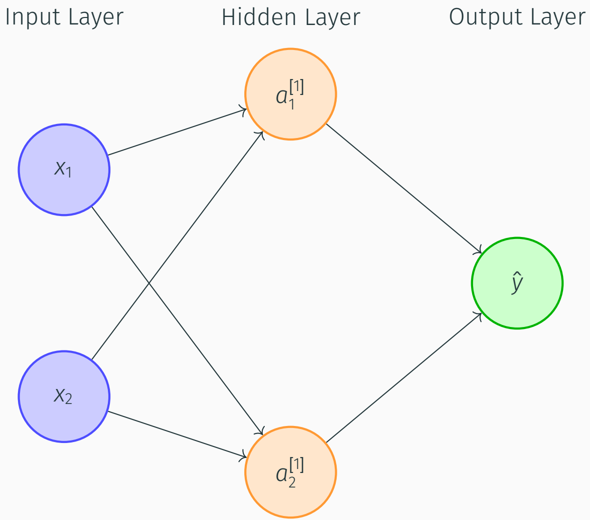
Neural Network Architecture

Let's use the following neural network as example:

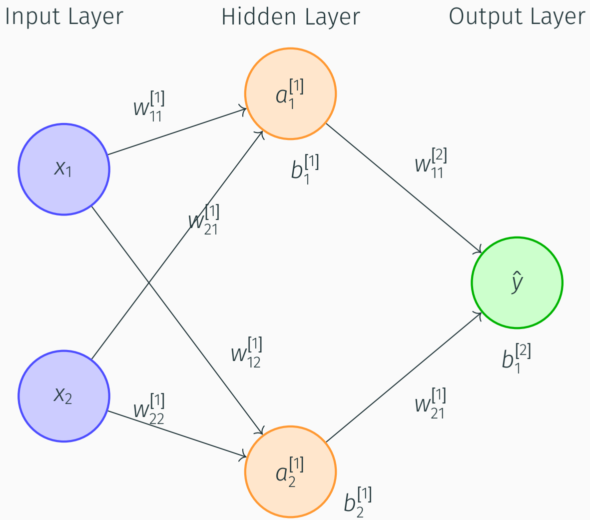
- Input layer: x_1, x_2
- Hidden layer: 2 neurons (h_1, h_2)
- Output layer: 1 neuron ($\hat{y}$)
- Activation: Sigmoid
- Loss: Squared Error

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \qquad \sigma(z) = \frac{1}{1 + e^{-z}}$$

Neural Network Architecture



Neural Network Architecture



Training a Neural Network

How an ANN is Trained: Overview

- Iteratively adjusting internal parameters (weights and biases).
- Goal: Minimize the difference between predictions and actual values.
- Uses a **training set** of labeled data.

The Training Steps

Steps (repeat many times):

1. Forward Pass
2. Loss (Error) Computation
3. Backward Pass
4. Parameter Update

The Training Steps

Steps (repeat many times):

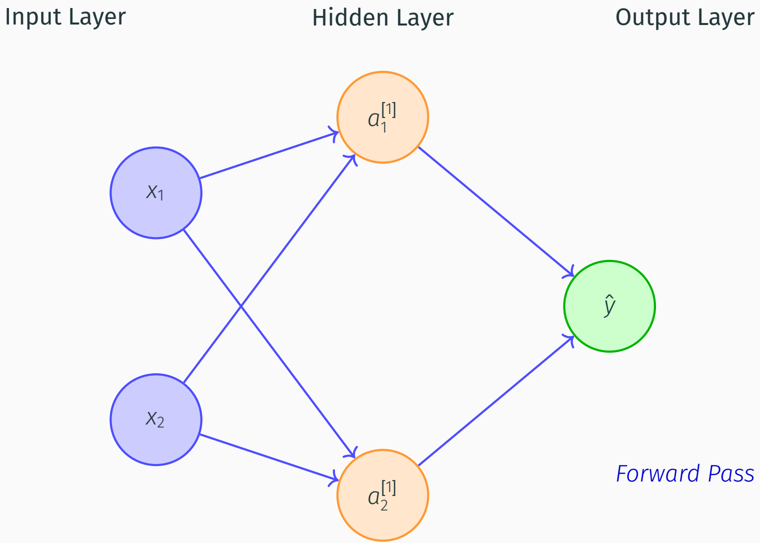
1. Forward Pass
2. Loss (Error) Computation
3. Backward Pass
4. Parameter Update

This iterative process (forward pass, error, backward pass, update) continues for many epochs until the **error** is minimized.

Step 1: Forward Pass

- For each input in the **training set**:
- Data propagates from the input layer, through hidden layers, to the output layer.
- Each neuron performs a weighted sum of inputs and applies an activation function.
- Generates the network's **prediction**.

Step 1: Forward Pass



Step 2: Loss (Error) Computation

- The network's prediction is compared to the true label from the **training set**.
- A **loss function** quantifies this discrepancy.
- Results in an **error** value.
- The **error** indicates how "wrong" the prediction was.

Step 2: Loss (Error) Computation

Standard notation:

$$\mathcal{L}(\hat{y}, y)$$

But actually:

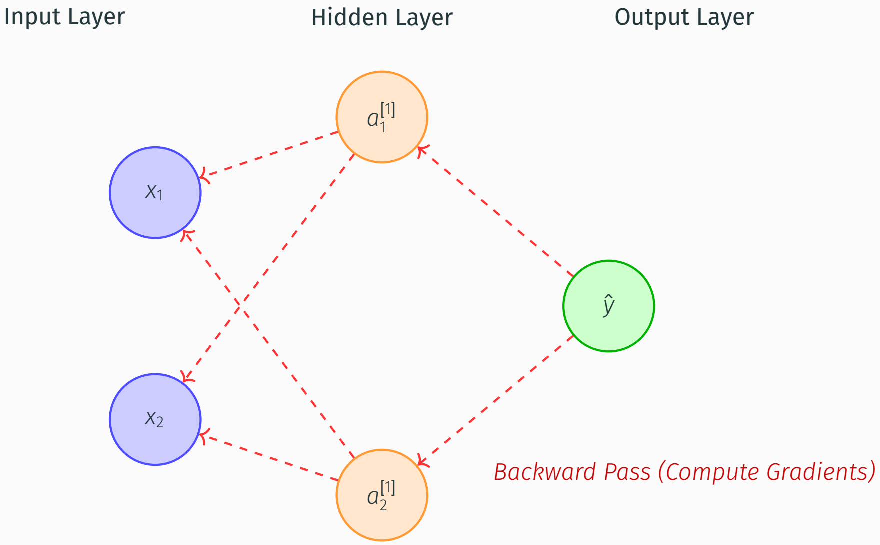
$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(h(x;\theta), y) \quad \text{where } \theta = \{W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\}$$

Training a neural network means minimizing the loss by adjusting all parameters in θ.

Step 3: Backward Pass (Backpropagation)

- The calculated **error** is propagated **backward** through the network.
- This process is called backpropagation.
- Using the chain rule of calculus, **backpropagation** efficiently computes the **gradient** of the error with respect to each weight and bias.
- That is, the gradient of the loss with respect to **each** parameter is computed.
- The **gradient** indicates the direction and magnitude for parameter adjustment to reduce the **error**.

Step 3: Backward Pass



Step 3: Backward Pass

For our 2-2-1 network, the following partial derivatives are computed:

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{12}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{21}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{22}^{[1]}} \qquad \text{(input to hidden)}$$

$$\frac{\partial \mathcal{L}}{\partial b_{1}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial b_{2}^{[1]}} \qquad \text{(hidden biases)}$$

$$\frac{\partial \mathcal{L}}{\partial w_{1}^{[2]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{2}^{[2]}} \qquad \text{(hidden to output)}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} \qquad \text{(output bias)}$$

Step 3: Backward Pass

For our 2-2-1 network, the following partial derivatives are computed:

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{12}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{21}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{22}^{[1]}} \qquad \text{(input to hidden)}$$

$$\frac{\partial \mathcal{L}}{\partial b_{1}^{[1]}}, \quad \frac{\partial \mathcal{L}}{\partial b_{2}^{[1]}} \qquad \text{(hidden biases)}$$

$$\frac{\partial \mathcal{L}}{\partial w_{1}^{[2]}}, \quad \frac{\partial \mathcal{L}}{\partial w_{2}^{[2]}} \qquad \text{(hidden to output)}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} \qquad \text{(output bias)}$$

Together, these form the full gradient vector used in the parameter update.

Step 3: The Gradient Vector in Backpropagation (2-2-1 Network)

In the backward pass, we compute the gradient vector:

$$\nabla_\theta \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{12}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{21}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{22}^{[1]}}, \frac{\partial \mathcal{L}}{\partial b_1^{[1]}}, \frac{\partial \mathcal{L}}{\partial b_2^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_1^{[2]}}, \frac{\partial \mathcal{L}}{\partial w_2^{[2]}}, \frac{\partial \mathcal{L}}{\partial b^{[2]}} \right]^T$$

This vector tells us how to change each parameter in order to reduce the loss.

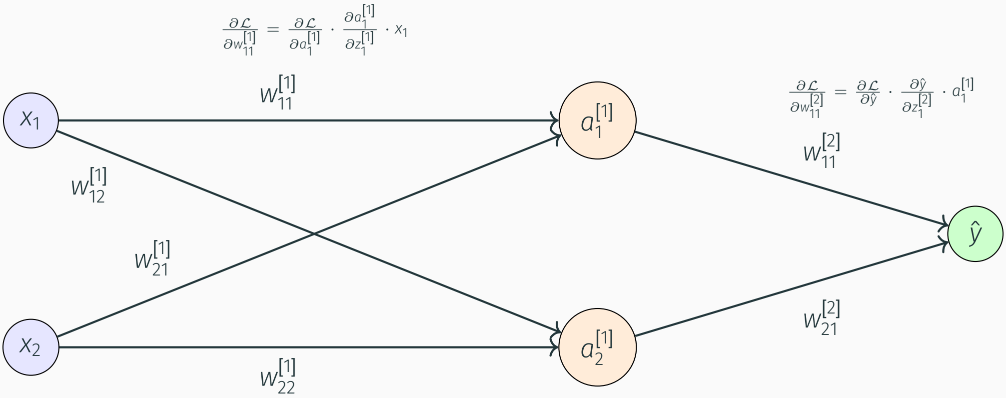
In the backward pass, we compute the gradient vector:

$$\nabla_\theta \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{12}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{21}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{22}^{[1]}}, \frac{\partial \mathcal{L}}{\partial b_{1}^{[1]}}, \frac{\partial \mathcal{L}}{\partial b_{2}^{[1]}}, \frac{\partial \mathcal{L}}{\partial w_{1}^{[2]}}, \frac{\partial \mathcal{L}}{\partial w_{2}^{[2]}}, \frac{\partial \mathcal{L}}{\partial b^{[2]}} \right]^T$$

This vector tells us how to change each parameter in order to reduce the loss.

In PyTorch, this is automatically computed when we call `loss.backward()`.

Step 3: Visual Map of Gradient Terms (2-2-1 Network)



Each edge corresponds to a weight, and its gradient is built from the local forward activations and backward error signals.

Step 3: Backpropagation Equations (2-2-1 Network)

Loss Function:

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

Output Neuron Gradient:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y \quad , \quad \frac{\partial \hat{y}}{\partial z_1^{[2]}} = \hat{y}(1 - \hat{y}) \quad , \quad \frac{\partial z_1^{[2]}}{\partial w_{i1}^{[2]}} = a_i^{[1]}$$

Hidden Neuron Gradient:

$$\frac{\partial \mathcal{L}}{\partial a_i^{[1]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1^{[2]}} \cdot w_{i1}^{[2]} \quad , \quad \frac{\partial a_i^{[1]}}{\partial z_i^{[1]}} = a_i^{[1]}(1 - a_i^{[1]}) \quad , \quad \frac{\partial z_i^{[1]}}{\partial w_{ji}^{[1]}} = x_j$$

Final Update Rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[1]}} = \frac{\partial \mathcal{L}}{\partial a_i^{[1]}} \cdot \frac{\partial a_i^{[1]}}{\partial z_i^{[1]}} \cdot x_j$$

Step 4: Parameter Update with Gradient Descent

Gradient descent (or its variants) uses the computed **gradients** to update the network's weights and biases.

- Adjusts parameters in the direction that minimizes the **error**.
- Takes small "steps" down the error surface.

Step 4: Parameter Update with Gradient Descent

Goal: Move parameters in the direction that reduces the loss.

Step 4: Parameter Update with Gradient Descent

Goal: Move parameters in the direction that reduces the loss. We update all parameters using gradient descent:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$$

Where:

- θ: vector of all weights and biases
- α: learning rate (step size)
- $\nabla_\theta \mathcal{L}$: gradient vector computed in the backward pass

Step 4: Parameter Update with Gradient Descent

Goal: Move parameters in the direction that reduces the loss. We update all parameters using gradient descent:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$$

Where:

- θ: vector of all weights and biases
- α: learning rate (step size)
- $\nabla_\theta \mathcal{L}$: gradient vector computed in the backward pass

In PyTorch:

- *`loss.backward()`* computes $\nabla_\theta \mathcal{L}$
- *`optimizer.step()`* applies the update rule above

Training Cycle of a Neural Network (Step-by-Step)

1. Forward Pass: $x \rightarrow \hat{y}$

Training Cycle of a Neural Network (Step-by-Step)

1. Forward Pass: $x \to \hat{y}$

2. Loss Computation: $\mathcal{L}(\hat{y}, y)$

Training Cycle of a Neural Network (Step-by-Step)

1. Forward Pass: $x \to \hat{y}$

$\downarrow$

2. Loss Computation: $\mathcal{L}(\hat{y}, y)$

$\downarrow$

3. Backward Pass (Gradients Computation: $\nabla\mathcal{L}$)

Training Cycle of a Neural Network (Step-by-Step)

1. Forward Pass: $x \to \hat{y}$

↓

2. Loss Computation: $\mathcal{L}(\hat{y}, y)$

↓

3. Backward Pass (Gradients Computation: $\nabla\mathcal{L}$)

↓

4. Parameter Update: $\theta \leftarrow \theta - \alpha\nabla\mathcal{L}$

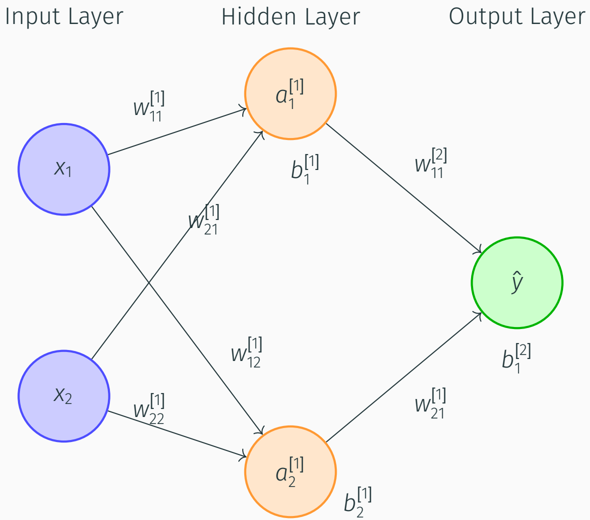
1. Forward Pass: $x \to \hat{y}$
2. Loss Computation: $\mathcal{L}(\hat{y}, y)$
3. Backward Pass (Gradients Computation: $\nabla\mathcal{L}$)
4. Parameter Update: $\theta \leftarrow \theta - \alpha\nabla\mathcal{L}$

Repeat these steps for multiple **epochs**.

Numerical Example

Follow the content of this section along with the companion notebook: ann_intro_companion.ipynb.

Example Neural Network



Input to Hidden Layer: Weights and Biases

Weight matrix $W^{[1]} \in \mathbb{R}^{2\times 2}$

$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} = \begin{bmatrix} 0.12 & 0.18 \\ 0.22 & 0.28 \end{bmatrix}$$

Bias vector $b^{[1]} \in \mathbb{R}^2$

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix} = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix}$$

Hidden to Output Layer: Weights and Bias

Weight vector $W^{[2]} \in \mathbb{R}^{1\times 2}$

$$W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} \end{bmatrix} = \begin{bmatrix} 0.40 & 0.45 \end{bmatrix}$$

Bias scalar $b^{[2]} \in \mathbb{R}$

$$b^{[2]} = 0.60$$

Matrix Computation in the Forward Pass

Step 1: Input to Hidden Layer

$$z^{[1]} = x \cdot W^{[1]} + b^{[1]} \quad \Rightarrow \quad a^{[1]} = \sigma(z^{[1]})$$

Matrix Computation in the Forward Pass

Step 1: Input to Hidden Layer

$$z^{[1]} = x \cdot W^{[1]} + b^{[1]} \quad \Rightarrow \quad a^{[1]} = \sigma(z^{[1]})$$

Step 2: Hidden to Output Layer

$$z^{[2]} = a^{[1]} \cdot W^{[2]} + b^{[2]} \quad \Rightarrow \quad a^{[2]} = \hat{y} = \sigma(z^{[2]})$$

Note: $\sigma(\cdot)$ is the sigmoid activation function.

Initial Values

Inputs and Target:

$$x_1 = 0.05, \quad x_2 = 0.10, \quad y = 0.01$$

Weights and Biases:

$$W^{[2]} = \begin{bmatrix} 0.15 & 0.20 \\ 0.25 & 0.30 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 0.40 & 0.45 \end{bmatrix}, \quad b^{[2]} = 0.60$$

Step 1: Forward Pass – Hidden Layer

$$a_1^{[1]} = \sigma(0.05 \cdot 0.15 + 0.10 \cdot 0.25 + 0.35) = \sigma(0.3775) \approx 0.59327$$

$$a_2^{[1]} = \sigma(0.05 \cdot 0.20 + 0.10 \cdot 0.30 + 0.35) = \sigma(0.3925) \approx 0.59688$$

Step 1: Forward Pass – Hidden Layer

$$a_1^{[1]} = \sigma(0.05 \cdot 0.15 + 0.10 \cdot 0.25 + 0.35) = \sigma(0.3775) \approx 0.59327$$

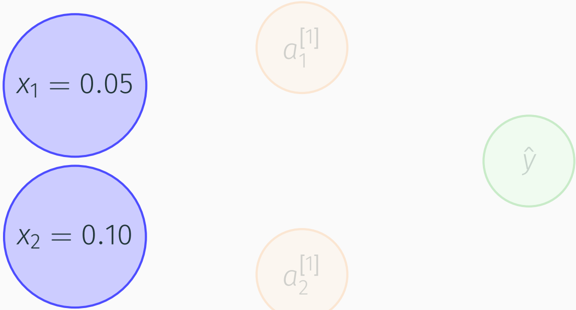
$$a_2^{[1]} = \sigma(0.05 \cdot 0.20 + 0.10 \cdot 0.30 + 0.35) = \sigma(0.3925) \approx 0.59688$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix}$$

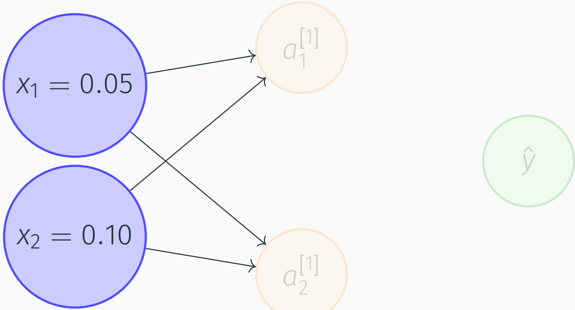
Step 1: Forward Pass – Output Layer

$$
\begin{aligned}
\hat{y} &= \sigma(0.59327 \cdot 0.40 + 0.59688 \cdot 0.45 + 0.60) \\
&= \sigma(1.1059) \approx 0.75136
\end{aligned}
$$

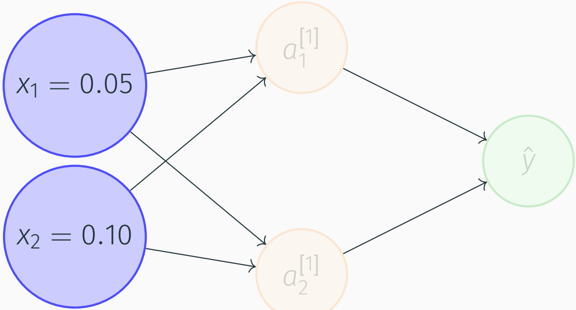
Step 1: Forward Pass - Input Layer



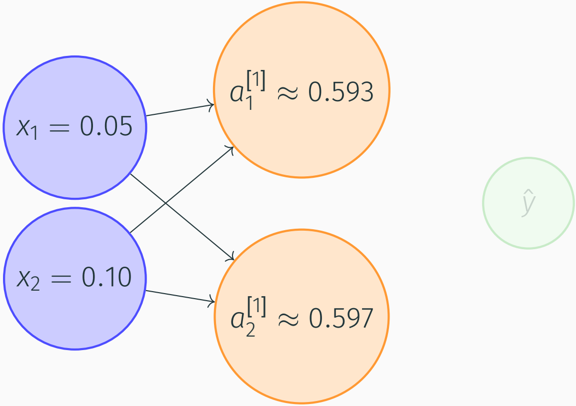
Step 1: Forward Pass - Input Layer



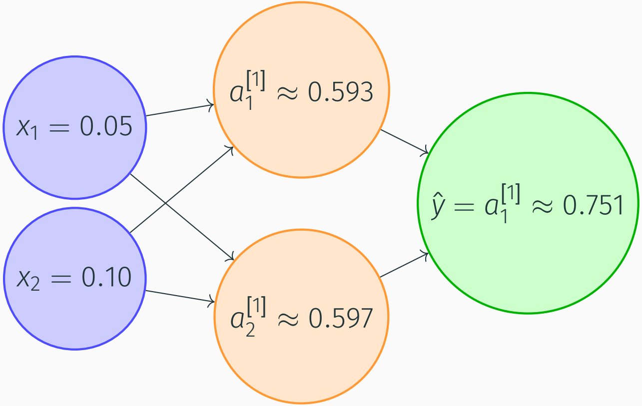
Step 1: Forward Pass - Input Layer



Step 1: Forward Pass - Hidden Layer Activation



Step 1: Forward Pass - Output Computation



Step 2: Loss Computation

Given the input x, the expected output was $y = 0.01$.

Given the input x, the expected output was $y = 0.01$.

But the model's output was $\hat{y} = 0.75136$.

Step 2: Loss Computation

Given the input x, the expected output was $y = 0.01$.

But the model's output was $\hat{y} = 0.75136$.

Let's compute the error:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0.01 - 0.75136)^2 \approx 0.2748$$

Step 2: Loss Computation

Given the input x, the expected output was $y = 0.01$.

But the model's output was $\hat{y} = 0.75136$.

Let's compute the error:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0.01 - 0.75136)^2 \approx 0.2748$$

This is the error we want to reduce using gradient descent!

From Forward Pass:

$$\hat{y} = 0.75136, \quad y = 0.01$$

$$a_1^{[1]} = 0.59327, \quad a_2^{[1]} = 0.59688$$

Step 3.1: Output Layer Gradients

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y = 0.74136$$

$$\frac{\partial \hat{y}}{\partial z_1^{[2]}} = \hat{y}(1 - \hat{y}) = 0.75136 \cdot (1 - 0.75136) \approx 0.18681$$

$$\delta_{\text{out}} = \frac{\partial \mathcal{L}}{\partial z_1^{[2]}} = 0.74136 \cdot 0.18681 \approx 0.1385$$

Step 3.2: Hidden to Output Weights

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = \delta_{\text{out}} \cdot a_1^{[1]} = 0.1385 \cdot 0.59327 \approx 0.0822$$

$$\frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}} = \delta_{\text{out}} \cdot a_2^{[1]} = 0.1385 \cdot 0.59688 \approx 0.0826$$

Step 3.3: Hidden Layer Errors (Backpropagated)

$$\delta_1 = \delta_{\text{out}} \cdot w_{11}^{[2]} \cdot a_1^{[1]}(1 - a_1^{[1]}) = 0.1385 \cdot 0.40 \cdot 0.59327 \cdot (1 - 0.59327) \approx 0.0134$$

$$\delta_2 = \delta_{\text{out}} \cdot w_{21}^{[2]} \cdot a_2^{[1]}(1 - a_1^{[1]}) = 0.1385 \cdot 0.45 \cdot 0.59688 \cdot (1 - 0.59688) \approx 0.0150$$

Step 3.4: Input-to-Hidden Gradients

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = \delta_1 \cdot x_1 = 0.0134 \cdot 0.05 = 0.00067 \quad , \quad \frac{\partial \mathcal{L}}{\partial w_{21}^{[1]}} = \delta_1 \cdot x_2 = 0.0134 \cdot 0.10 = 0.00134$$

$$\frac{\partial \mathcal{L}}{\partial w_{12}^{[1]}} = \delta_2 \cdot x_1 = 0.0150 \cdot 0.05 = 0.00075 \quad , \quad \frac{\partial \mathcal{L}}{\partial w_{22}^{[1]}} = \delta_2 \cdot x_2 = 0.0150 \cdot 0.10 = 0.00150$$

Step 4: Parameter Updates

Gradient Descent Update Rule:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$$

Assume: $\alpha = 0.5$

Updated Hidden → Output Weights:

$$w_{11}^{[2]} = 0.40 - 0.5 \cdot 0.0822 = 0.3589 \quad , \quad w_{21}^{[2]} = 0.45 - 0.5 \cdot 0.0826 = 0.4087$$

Updated Input → Hidden Weights:

$$w_{11}^{[1]} = 0.15 - 0.5 \cdot 0.00067 = 0.1497 \quad , \quad w_{21}^{[1]} = 0.25 - 0.5 \cdot 0.00134 = 0.2493$$

$$w_{12}^{[1]} = 0.20 - 0.5 \cdot 0.00075 = 0.1996 \quad , \quad w_{22}^{[1]} = 0.30 - 0.5 \cdot 0.00150 = 0.2993$$

Exercises

Exercise 1: Forward Pass

Given:

- $x_1 = 0.10, x_2 = 0.20$
- $W^{[1]} = \begin{bmatrix} 0.12 & 0.18 \\ 0.22 & 0.28 \end{bmatrix}, b^{[1]} = [0.35, 0.35]$
- $W^{[2]} = [0.40, 0.45], b^{[2]} = 0.60$

Exercise 1: Forward Pass

Given:

- $x_1 = 0.10$, $x_2 = 0.20$
- $W^{[1]} = \begin{bmatrix} 0.12 & 0.18 \\ 0.22 & 0.28 \end{bmatrix}$, $b^{[1]} = [0.35, 0.35]$
- $W^{[2]} = [0.40, 0.45]$, $b^{[2]} = 0.60$

Task:

1. Compute $a_1^{[1]}$ and $a_2^{[1]}$ using sigmoid activation
2. Compute the output $\hat{y}$

Hint: Use $\sigma(z) = \frac{1}{1+e^{-z}}$

Exercise 1: Forward Pass

Discussion Point: Which hidden neuron—$a_1^{[1]}$ or $a_2^{[1]}$—do you expect to contribute more to the final output? Why?

Exercise 1: Forward Pass

Discussion Point: Which hidden neuron—$a_1^{[1]}$ or $a_2^{[1]}$—do you expect to contribute more to the final output? Why?

Result from the computation:

$$a_1^{[1]} \approx 0.6001, \quad a_2^{[1]} \approx 0.6044 \quad \text{with} \quad W^{[2]} = [0.40,\ 0.45]$$

Exercise 1: Forward Pass

Discussion Point: Which hidden neuron—$a_1^{[1]}$ or $a_2^{[1]}$—do you expect to contribute more to the final output? Why?

Result from the computation:

$$a_1^{[1]} \approx 0.6001, \quad a_2^{[1]} \approx 0.6044 \quad \text{with} \quad W^{[2]} = [0.40,\ 0.45]$$

Contribution Estimates:

$$a_1^{[1]} \cdot w_{11}^{[2]} \approx 0.6001 \cdot 0.40 = 0.2400 \quad , \quad a_2^{[1]} \cdot w_{21}^{[2]} \approx 0.6044 \cdot 0.45 = 0.2720$$

Exercise 1: Forward Pass

Discussion Point: Which hidden neuron—$a_1^{[1]}$ or $a_2^{[1]}$—do you expect to contribute more to the final output? Why?

Result from the computation:

$$a_1^{[1]} \approx 0.6001, \quad a_2^{[1]} \approx 0.6044 \quad \text{with} \quad W^{[2]} = [0.40,\ 0.45]$$

Contribution Estimates:

$$a_1^{[1]} \cdot w_{11}^{[2]} \approx 0.6001 \cdot 0.40 = 0.2400 \quad , \quad a_2^{[1]} \cdot w_{21}^{[2]} \approx 0.6044 \cdot 0.45 = 0.2720$$

Conclusion:

- h_2 contributes more to $\hat{y}$
- This is due to a slightly higher activation and a stronger outgoing weight

Exercise 2: Output Layer Gradients

Given:

$$\hat{y} = 0.80, \quad y = 0.10$$

$$a_1^{[1]} = 0.60, \quad a_2^{[1]} = 0.55$$

Exercise 2: Output Layer Gradients

Given:

$$\hat{y} = 0.80, \quad y = 0.10$$

$$a_1^{[1]} = 0.60, \quad a_2^{[1]} = 0.55$$

Task:

1. Compute $\frac{\partial \mathcal{L}}{\partial \hat{y}}$
2. Compute $\frac{\partial \hat{y}}{\partial z_1^{[2]}}$
3. Compute gradients $\frac{\partial \mathcal{L}}{\partial w_{11}^{[2]}}, \frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}}$

Hint: Use $\hat{y}(1 - \hat{y})$ for the sigmoid derivative.

Discussion Point: Should we update $w_{11}^{[2]}$ and $w_{21}^{[2]}$ in the same direction? Why or why not?

Exercise 3: Hidden Layer Gradients

Given:

$$\delta_{\text{out}} = 0.14, \quad w_{11}^{[2]} = 0.42, \quad a_1^{[1]} = 0.60$$

Exercise 3: Hidden Layer Gradients

Given:

$$\delta_{\text{out}} = 0.14, \quad w_{11}^{[2]} = 0.42, \quad a_1^{[1]} = 0.60$$

Task:

1. Compute $\delta_1 = \delta_{\text{out}} \cdot w_{11}^{[2]} \cdot a_1^{[1]}(1 - a_1^{[1]})$
2. Compute $\frac{\partial \mathcal{L}}{\partial w_{11}^{[2]}}$ and $\frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}}$

Exercise 3: Hidden Layer Gradients

Discussion Point: Which input—x_1 or x_2—has a larger influence on $a_1^{[1]}$'s gradient? Why?

Exercise 4: Weight Updates

Given:

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = 0.0008, \quad \alpha = 0.1, \quad w_{11}^{[1]} = 0.15$$

Exercise 4: Weight Updates

Given:

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[1]}} = 0.0008, \quad \alpha = 0.1, \quad w_{11}^{[1]} = 0.15$$

Task:

1. Compute the new value of $w_{11}^{[1]}$
2. Discuss: What happens if α is too large? Too small?

Exercise 4: Weight Updates

Discussion Point: How would your result change if the learning rate were doubled? What if it were ten times smaller?