Understanding Artificial Neural Networks

Theory, Logic, and Python Implementation

AI Education Series

December 24, 2025

Lecture Overview

What is an Artificial Neural Network (ANN)?

- **Definition:** A computational model inspired by the biological neural networks of the human brain.
- **Core Purpose:** To simulate the brain's ability to learn and identify patterns from data.
- **Mechanism:** Consists of interconnected nodes (neurons) that process information using mathematical functions.

Biological vs. Artificial Neurons

Biological Neuron:

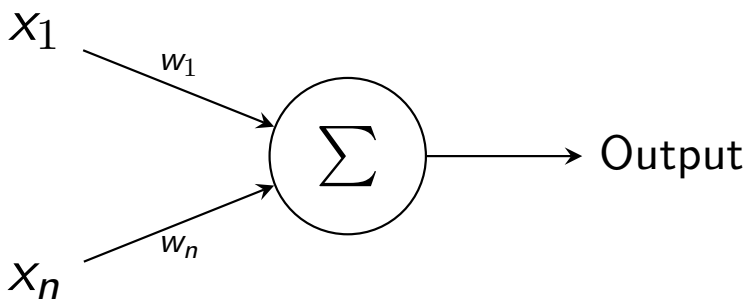
- *Dendrites:* Receive signals.
- *Soma:* Processes signals.
- *Axon:* Transmits output.

Human Neuron Diagram
Showing Dendrites, Cell Body (Soma), and Axon.

Figure: Biological Structure

Artificial Neuron:

- *Inputs:* Data points (x_n).
- *Weights/Bias:* Signal strength.
- *Activation:* Decision making.

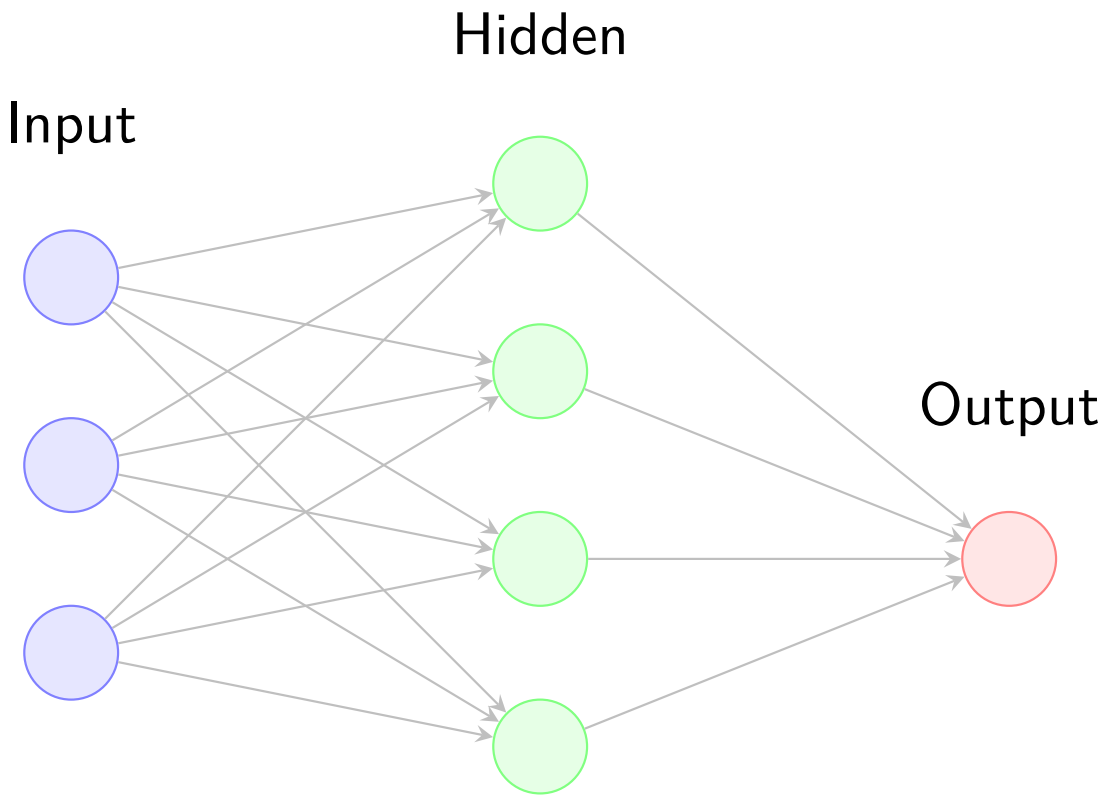


The Architecture: Layers

An ANN is typically organized into three types of layers:

1. **Input Layer:** Receives raw data from the external world.
2. **Hidden Layers:** Where the "learning" happens. These layers extract features.
3. **Output Layer:** Provides the final prediction or classification.

Visualizing the Structure

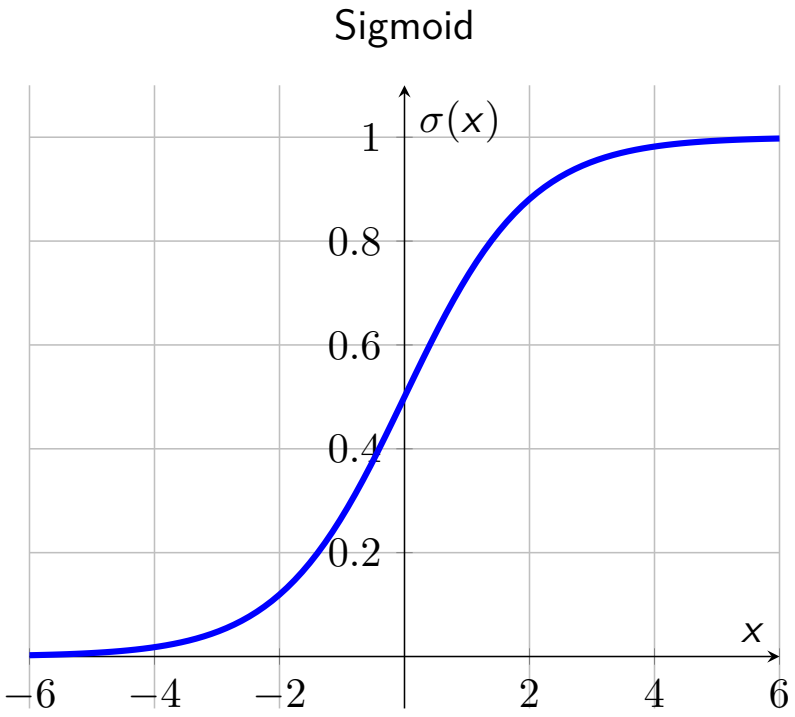


Activation: Sigmoid Function

Equation:

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

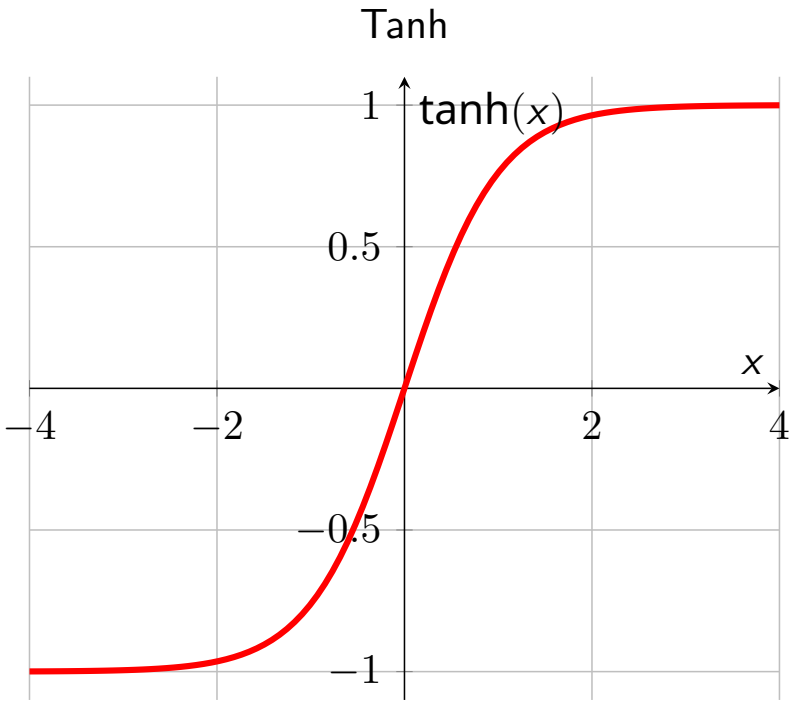
- **Range:** (0, 1)
- **Use Case:** Output layer for binary classification.
- **Issue:** Vanishing Gradient problem for large/small values.



Activation: Hyperbolic Tangent (Tanh)

Equation:

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



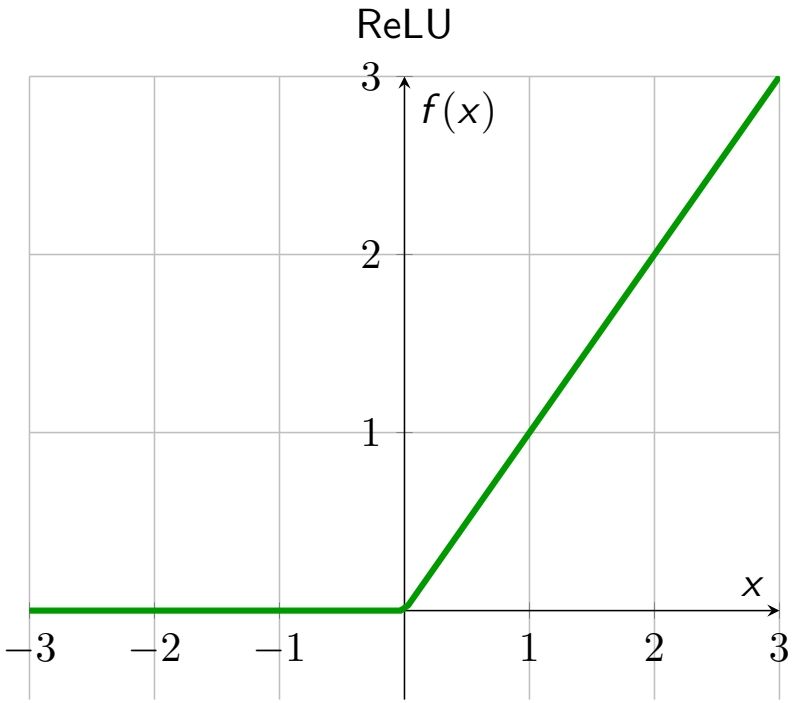
- **Range:** (-1, 1)
- **Key Feature:** Zero-centered (outputs have mean near 0).
- **Use Case:** Often preferred over Sigmoid in hidden layers.

Activation: Rectified Linear Unit (ReLU)

Equation:

$$f(x) = \max(0, x)$$

- **Range:** $[0, \infty)$
- **Benefits:** Computationally efficient; reduces likelihood of vanishing gradient.
- **Standard:** Most common for Hidden Layers.

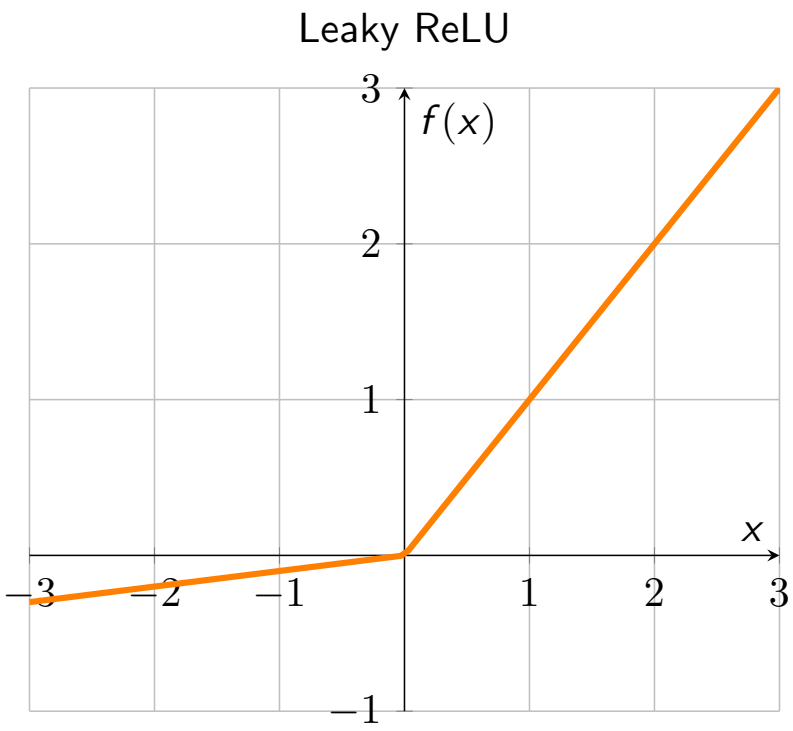


Activation: Leaky ReLU

Equation:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{if } x \leq 0 \end{cases}$$

- **Purpose:** Fixes the "Dying ReLU" problem by allowing a small gradient when $x < 0$.
- **Range:** $(-\infty, \infty)$



The Mathematics of a Single Neuron

The output Y of a neuron is calculated as:

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

Where:

- x_i: Input signals.
- w_i: Synaptic weights (Importance of input).
- b: Bias (Threshold adjustment).
- f: Activation function (Sigmoid, ReLU, etc.).

Weights and Bias: The Learning Parameters

- **Weights (w):** Represent the strength of the connection. Higher weight means more influence.
- **Bias (b):** Shifts the activation function to allow flexibility in the model.
- **Goal of Training:** Finding optimal w and b to minimize error.

The Forward Pass (Inference)

1. Data enters the input layer.
2. Multiplied by weights, bias added.
3. Passed through activation function.
4. Moves to the next layer until it reaches the output.
5. **Result:** A raw prediction.

Measuring Error: Loss Functions

- **MSE (Mean Squared Error):** For regression.
- **Cross-Entropy:** For classification.

Backpropagation: The Chain Rule

- Propagates the error backward through the network.
- Calculates the gradient of the loss function with respect to weights.
- This is the core engine of AI learning.

Optimization: Gradient Descent

- Algorithm to update weights: $w_{new} = w_{old} - \alpha \frac{\partial Loss}{\partial w}$.
- **Learning Rate (α):** Step size of the update.

Python Tools for Neural Networks

- **NumPy:** Fundamental package for scientific computing.
- **TensorFlow/Keras:** High-level API for building and training models.
- **PyTorch:** Popular research-focused deep learning library.

Step 1: Importing Libraries & Data

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Sample Data (XOR Problem)
X = np.array([[0,0], [0,1], [1,0], [1,1]], "float32")
y = np.array([[0], [1], [1], [0]], "float32")
```

Step 2: Defining the Model

```
model = Sequential()

# Input layer + Hidden Layer (4 neurons)
model.add(Dense(4, input_dim=2, activation='relu'))

# Output Layer (1 neuron for binary outcome)
model.add(Dense(1, activation='sigmoid'))
```

Step 3: Compiling

```
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Step 4: Training the Model (Fit)

```python
# Training for 500 iterations (epochs)
history = model.fit(X, y, epochs=500, verbose=0)

# Evaluate performance
scores = model.evaluate(X, y)
print(f"Accuracy: {scores[1]*100}%")
```

Key Takeaways

1. ANN is a bio-inspired mathematical model.
2. Forward + Backpropagation is the learning cycle.
3. Python (Keras/TensorFlow) makes building networks accessible.
4. Preprocessing and tuning are as important as the architecture.

Thank you! Time for Coding!