Neural Networks

Jayin Khanna
MAT399: UG Seminar Presentation 2 Report
Shiv Nadar University

March 30, 2025

Abstract

Neural networks have revolutionized machine learning and were the beginning of deep learning. It helps to analyze complex relations in data and lies at the heart of many AI fields such as Computer Vision, Natural Language Processing, GenAI etc. This report provides a comprehensive overview of neural networks, delving deep into their history, applications, architecture, mathematical formulation, forward propagation, backpropagation, gradient descent and activation functions.

1 Introduction

This report provides a comprehensive overview of neural networks, covering both the theoretical foundations and practical implementations. Specifically, the following topics will be discussed:

- History of Neural Networks (Section 1.1)
- Applications (Section 1.3)
- Theory (Section 2)
  - Perceptron (Section 2.1)
  - Multilayer Perceptron (Section 2.2)
  - Forward propagation (Section 2.3)
  - Generalization of Forward Propagation (Section 2.3.1)
  - Activation Functions (Section 2.4)
  - Backpropagation (Section 2.5)
- Demand Prediction Example (Section 3)
- Refrences (Section 4)

For detailed explanations, please refer to the respective sections linked above.

1.1 History and Origins of Neural Networks

The foundation of neural networks was laid in the 1940s when neurophysiologists and mathematicians began exploring artificial intelligence inspired by the human brain.

Early Concepts: The McCulloch-Pitts Model (1943)
Warren McCulloch, a neurophysiologist, and Walter Pitts, a mathematician, introduced the first mathematical model of an artificial neuron in 1943. Their model, known as the McCulloch-Pitts neuron, was a binary threshold unit that mimicked how neurons process information in the brain. This work established the idea that networks of simple processing units could perform logical operations.

Hebbian Learning (1949)

Donald Hebb proposed a theory of synaptic plasticity in 1949, stating that *"neurons that fire together, wire together."* This biological insight helped lay the foundation for learning rules in artificial neural networks.

Perceptron: Frank Rosenblatt (1958)
Frank Rosenblatt, a psychologist, developed the perceptron, a simple neural network that could classify patterns using adjustable weights. The perceptron was one of the first implementations of machine learning, although it was later criticized for its limitations in solving nonlinearly separable problems, as highlighted in Marvin Minsky and Seymour Papert's book *Perceptrons* (1969).

1.2 Power of the Mighty Neural Network in one Theorem:

Universal Approximation Theorem (UAT): It states that a neural network with a single hidden layer containing a sufficient number of neurons can approximate any continuous function on a compact subset of R n to any desired degree of accuracy, provided it uses a nonconstant, bounded, and monotonically increasing activation function.

1.3 Applications and Real-World Impact

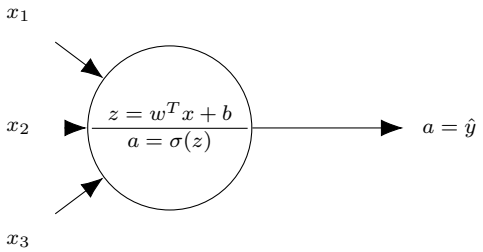
Neural networks have widespread applications across industries:

- **Computer Vision**: Facial recognition, medical image diagnosis, autonomous vehicles.
- **Natural Language Processing (NLP)**: Chatbots, machine translation, sentiment analysis.
- **Finance**: Algorithmic trading, fraud detection, credit scoring.
- **Healthcare**: Drug discovery, personalized medicine, disease diagnosis.
- **Robotics**: Autonomous systems, industrial automation.

2 Theory

2.1 Perceptron: Understanding the Network at the Atomic Level

We will consider a simple case to understand the smallest computation unit of a neural network i.e. a perceptron. Suppose that we have three features, x_1, x_2, x_3. A perceptron computes a linear combination of the input features with a bias term.e. $z = w_1x_1 + w_2x_2 + w_3x_3 + b$, and then applies a non-linear function, called an *activation*, to this linear combination i.e. $a = \sigma(z)$, to give an output.

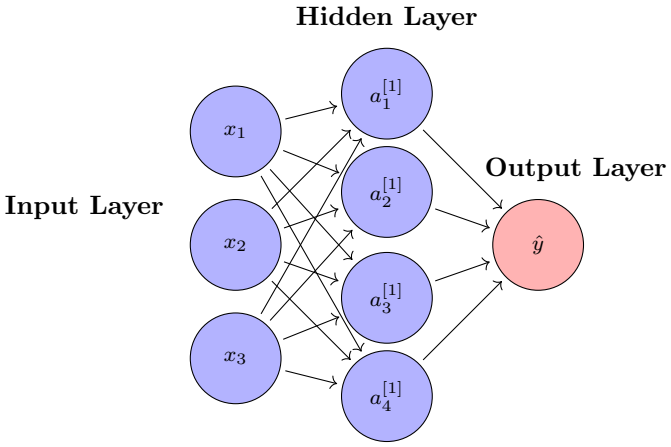


$$z = w^T x + b$$
$$a = \sigma(z)$$

2.2 Multilayer Perceptron:

Annotations:

- $z_1^{[1]} = w_1^{[1]} x + b_1^{[1]}$
- $a_1^{[1]} = \sigma(z_1^{[1]})$
- $[l]$ represents the layer index.
- A_i represents the node in the layer.



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})
\end{aligned}
$$

2.3 Forward Propagation

Let $\mathbf{W}^{[1]} \in \mathbb{R}^{4\times 3}$ be the weight matrix and $\mathbf{b}^{[1]} \in \mathbb{R}^4$ be the bias vector for the hidden layer. The hidden layer pre-activation values $\mathbf{z}^{[1]}$ and activations $\mathbf{a}^{[1]}$ are given by:

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \tag{1}$$

$$\mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \tag{2}$$

where $\sigma(\cdot)$ is an activation function (e.g., sigmoid, ReLU).
For the output layer, let $\mathbf{W}^{[2]} \in \mathbb{R}^{1\times 4}$ and $b^{[2]} \in \mathbb{R}$. The output is computed as:

$$z^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + b^{[2]} \tag{3}$$

$$\hat{y} = \sigma(z^{[2]}) \tag{4}$$

2.3.1 Generalization of Forward Propagation

For the sake of simplicity, let's assume that the input example is $\mathbf{x} \in \mathbb{R}^d$ and that our hidden layer does not include a bias term. Here the intermediate variable is:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}, \tag{5}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ is the weight parameter of the hidden layer. After running the intermediate variable $\mathbf{z} \in \mathbb{R}^h$ through the activation function ϕ we obtain our hidden activation vector of length h:

$$\mathbf{h} = \phi(\mathbf{z}). \tag{6}$$

The hidden layer output $\mathbf{h}$ is also an intermediate variable. Assuming that the parameters of the output layer possess only a weight of $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$, we can obtain an output layer variable with a vector of length q:

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}. \tag{7}$$

Assuming that the loss function is l and the example label is y, we can then calculate the loss term for a single data example,

$$L = l(\mathbf{o}, y). \tag{8}$$

2.4 Common Activation Functions

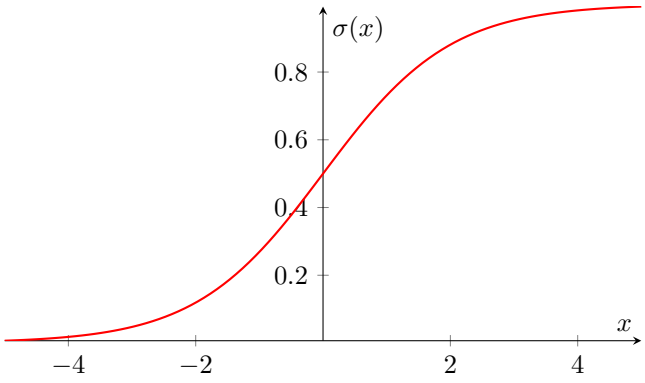
In a neural network, activation functions introduce **non-linearity**, which allows the network to learn complex patterns. Without activation functions, deep networks would behave like a single-layer perceptron because a composition of linear functions is still linear. The activation functions break this linearity and help the network model non-linear relationships.

2.4.1 1. Sigmoid Function

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

Properties:

- Range: $(0, 1)$
- Derivative: $\sigma(x)(1 - \sigma(x))$
- Problem: Vanishing gradient for large $|x|$



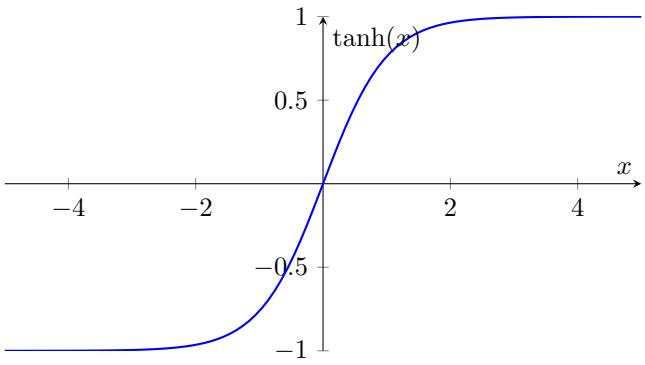
2.4.2 2. Hyperbolic Tangent (Tanh) Function

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

Properties:

- Range: $(-1, 1)$

- Derivative: $1 - \tanh^2(x)$
- Better than sigmoid since output is zero-centered.

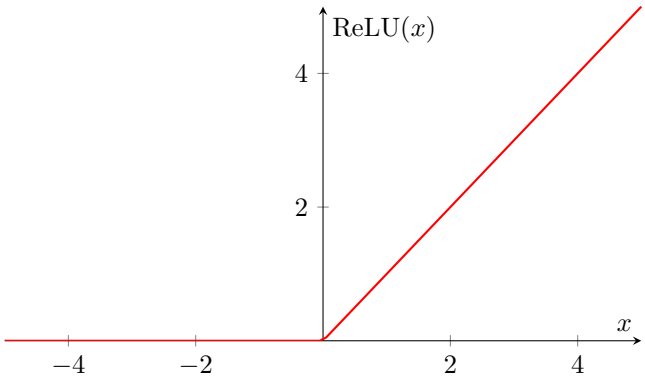


2.4.3 3. Rectified Linear Unit (ReLU)

$$\text{ReLU}(x) = \max(0, x)$$

Properties:

- Range: $[0, \infty)$
- Derivative: 1 for $x > 0$, 0 for $x \leq 0$
- Helps solve vanishing gradient but has the "dying ReLU" problem where neurons can get stuck with zero output.



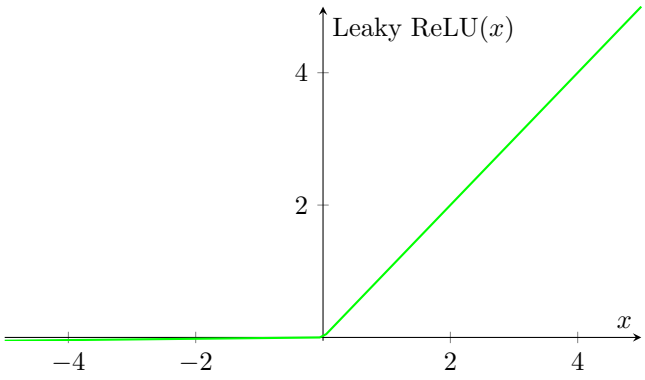
2.4.4 4. Leaky ReLU

$$\text{Leaky ReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

where α is a small slope (e.g., 0.01).

Properties:

- Solves the dying ReLU problem by allowing a small negative slope.
- Often used in deep learning architectures.



2.5 Backpropagation Algorithm

To update weights using gradient descent, we first compute derivatives of the loss J with respect to parameters, and then update the parameters using Gradient Descent.

2.5.1 Gradient Computation

Let the loss function be $J(\hat{y}, y)$. Define the error term for the output layer:

$$\delta^{[2]} = \frac{\partial J}{\partial z^{[2]}} = (\hat{y} - y)\sigma'(z^{[2]}) \tag{9}$$

The gradient for the output layer parameters:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \delta^{[2]}\mathbf{a}^{[1]T} \tag{10}$$

$$\frac{\partial J}{\partial b^{[2]}} = \delta^{[2]} \tag{11}$$

For the hidden layer, define the back propagated error:

$$\boldsymbol{\delta}^{[1]} = (\mathbf{W}^{[2]T}\delta^{[2]}) \odot \sigma'(\mathbf{z}^{[1]}) \tag{12}$$

where $\odot$ represents element-wise multiplication. The gradients for the hidden layer parameters:

$$\frac{\partial J}{\partial \mathbf{W}^{[1]}} = \boldsymbol{\delta}^{[1]}\mathbf{x}^T \tag{13}$$

$$\frac{\partial J}{\partial \mathbf{b}^{[1]}} = \boldsymbol{\delta}^{[1]} \tag{14}$$

2.5.2 Parameter Update

Using gradient descent with learning rate α, where $\alpha \in (0, 1]$:

$$\mathbf{W}^{[2]} \leftarrow \mathbf{W}^{[2]} - \alpha\frac{\partial J}{\partial \mathbf{W}^{[2]}} \tag{15}$$

$$b^{[2]} \leftarrow b^{[2]} - \alpha\frac{\partial J}{\partial b^{[2]}} \tag{16}$$

$$\mathbf{W}^{[1]} \leftarrow \mathbf{W}^{[1]} - \alpha\frac{\partial J}{\partial \mathbf{W}^{[1]}} \tag{17}$$

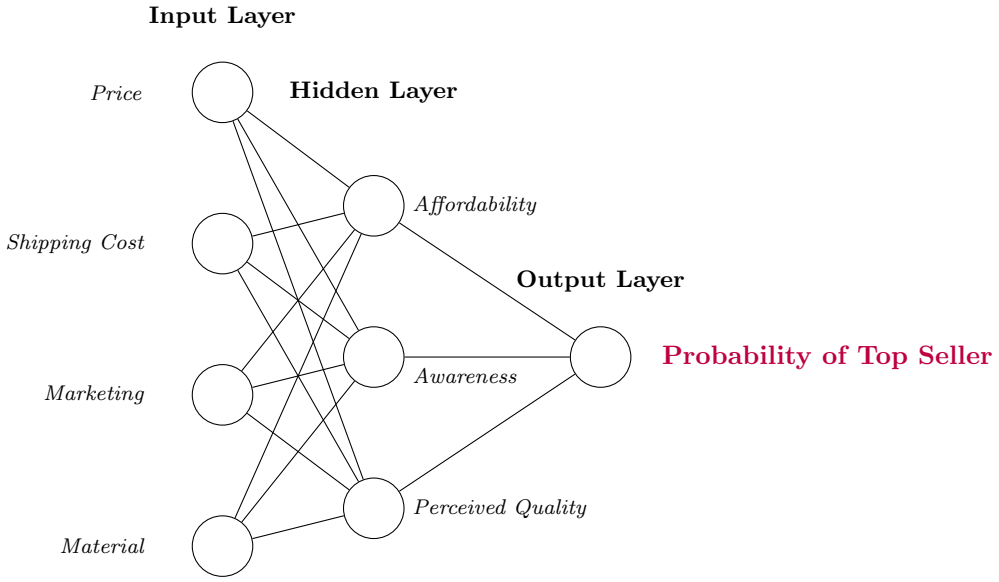
$$\mathbf{b}^{[1]} \leftarrow \mathbf{b}^{[1]} - \alpha\frac{\partial J}{\partial \mathbf{b}^{[1]}} \tag{18}$$

The following parameters are updated until convergence. In practice, it is updated for 1,000 - 10,000 iterations

3 Understanding Feature Mapping through Demand Prediction Example

This section illustrates how a neural network can be used for demand prediction by mapping input features to an output probability. The diagram represents a simple feedforward neural network, where the **input layer** consists of four key factors influencing demand: *Price, Shipping Cost, Marketing, and Material.* These features are transformed through a **hidden layer**, which learns abstract representations such as *Affordability, Awareness, and Perceived Quality.* Finally, the **output layer** provides a prediction, labeled as the **Probability of Top Seller**, indicating the likelihood of a product becoming a bestseller.

By utilizing weighted connections between layers, the neural network captures complex relationships between pricing strategies, marketing efforts, and consumer perception. This visualization emphasizes the power of neural networks in extracting meaningful insights from structured data, making them essential for predictive analytics in business and e-commerce.



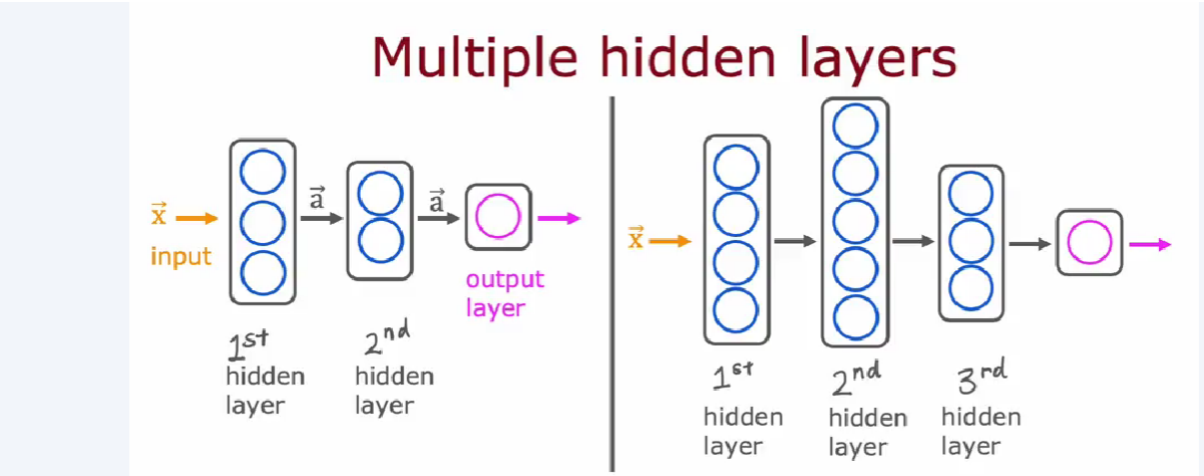


Figure 1: Examples of Neural Networks

References

[1] Deep Learning.AI and Stanford University. *Advanced Learning Algorithms - Module 1: Neural Networks*. Available on Coursera, 2024. https://www.coursera.org/learn/advanced-learning-algorithms?specialization=machine-learning-introduction.

[2] A. Zhang, M. Li, Z. Lipton, and A. J. Smola. *Dive into Deep Learning*. Publisher : Cambridge University Press; 1st edition (December 7, 2023), Chapter 5. https://d2l.ai/.