Artificial Neural Networks And Deep Learning

Theory

Christian Rossi

Academic Year 2024-2025

Abstract

Neural networks have matured into flexible and powerful non-linear data-driven models, effectively tackling complex tasks in both science and engineering. The emergence of deep learning, which utilizes neural networks to learn optimal data representations alongside their corresponding models, has significantly advanced this paradigm.

In the course, we will explore various topics in depth. We will begin with the evolution from the Perceptron to modern neural networks, focusing on the feedforward architecture. The training of neural networks through backpropagation and algorithms like Adagrad and Adam will be covered, along with best practices to prevent overfitting, including cross-validation, stopping criteria, weight decay, dropout, and data resampling techniques.

The course will also delve into specific applications such as image classification using neural networks, and we will examine recurrent neural networks and related architectures like sparse neural autoencoders. Key theoretical concepts will be discussed, including the role of neural networks as universal approximation tools, and challenges like vanishing and exploding gradients.

We will introduce the deep learning paradigm, highlighting its distinctions from traditional machine learning methods. The architecture and breakthroughs of convolutional neural networks (CNNs) will be a focal point, including their training processes and data augmentation strategies.

Furthermore, we will cover structural learning and long-short term memory (LSTM) networks, exploring their applications in text and speech processing. Topics such as autoencoders, data embedding techniques like word2vec, and variational autoencoders will also be addressed.

Finally, we will discuss transfer learning with pre-trained deep models, examine extended models such as fully convolutional CNNs for image segmentation (e.g., U-Net) and object detection methods (e.g., R-CNN, YOLO), and explore generative models like generative adversarial networks (GANs).

Contents

CHAPTER 1

Deep learning

1.1 Introduction

Definition (*Machine Learning*)**.** A computer program is considered to learn from experience E with respect to a specific class of tasks T and a performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Given a dataset $\mathcal{D} = x_1, x_2, \ldots, x_N$, Machine Learning can be broadly categorized into three types:

- *Supervised learning*: in this type of learning, the model is provided with desired outputs $t_1, t_2, \ldots, t_N$ and learns to produce the correct output for new input data. The primary tasks in supervised learning are:
  - *Classification*: the model is trained on a labeled dataset and returns a label for new data.
  - *Regression*: the model is trained on a dataset with numerical values and returns a number as the output.
- *Unsupervised learning*: here, the model identifies patterns and regularities within the dataset $\mathcal{D}$ without being provided with explicit labels. The main task in unsupervised learning is:
  - *Clustering*: the model groups similar data elements based on inherent similarities within the dataset.
- *Reinforcement learning*: in this approach, the model interacts with the environment by performing actions $a_1, a_2, \ldots, a_N$ and receives rewards $r_1, r_2, \ldots, r_N$ in return. The model learns to maximize cumulative rewards over time by adjusting its actions.

1.2 Deep Learning

Deep Learning, a subset of Machine Learning, focuses on utilizing large datasets and substantial computational power to automatically learn data representations. In certain cases, traditional classification may fail due to the presence of irrelevant or redundant features in the dataset.

Deep Learning addresses this issue by learning optimal features directly from the data, which are then used by Machine Learning algorithms to perform more accurate classifications. Essentially, Deep Learning involves learning how to represent data in a way that improves the performance of Machine Learning models.

1.3 Perceptron

In the 1940s, computers were already proficient at executing tasks exactly as programmed and performing arithmetic operations with impressive speed. However, researchers envisioned machines that could do much more. They wanted computers that could handle noisy data, interact directly with their environment, function in a massively parallel and fault-tolerant way, and adapt to changing circumstances. Their quest was for a new computational model, one that could surpass the constraints of the Von Neumann Machine.

1.3.1 Human neurons

The human brain contains an enormous number of computing units, with approximately 100 billion neurons, each connected to around 7,000 other neurons through synapses. In adults, this results in a total of 100 to 500 trillion synaptic connections, while in a three-year-old child, this number can reach up to 1 quadrillion synapses.

The brain's computational model is characterized by its distributed nature among simple, non-linear units, its redundancy which ensures fault tolerance, and its intrinsic parallelism. The perceptron, a computational model inspired by the brain, reflects these principles.

Information in the brain is transmitted through chemical processes. Dendrites gather signals from synapses, which can be either inhibitory or excitatory. When the cumulative charge reaches a certain threshold, the neuron fires, releasing the charge.

1.3.2 Artificial neuron

The mathematical model of a neuron is represented as follows:

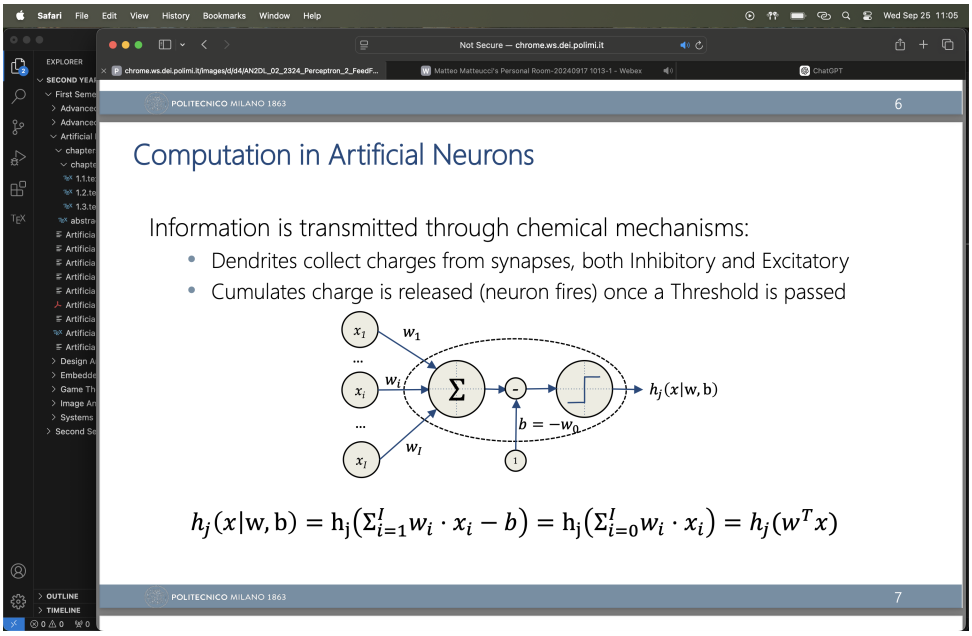

Figure 1.1: Artificial neuron

In this model, the output function $h_j(\mathbf{x}|\mathbf{w}, b)$ is defined as:

$$h_j(\mathbf{x}|\mathbf{w}, b) = h_j\left(\sum_{i=1}^{I} w_i x_i - b\right) = h_j\left(\sum_{i=0}^{I} w_i x_i\right) = h_j\left(\mathbf{w}^T \mathbf{x}\right)$$

The function used in an artificial neuron can either be a step function, with values ranging from 0 to 1, or a sine function, with values ranging from -1 to 1.

History Several researchers were actively investigating models for the brain during the mid-20th century. In 1943, Warren McCulloch and Walter Pitts proposed the Threshold Logic Unit, also referred to as the Linear Unit, where the activation function was a threshold unit, equivalent to the Heaviside step function. A few years later, in 1957, Frank Rosenblatt developed the first Perceptron, with weights encoded in potentiometers, and weight adjustments during learning were performed by electric motors. By 1960, Bernard Widrow introduced a significant advancement by representing the threshold value as a bias term in the ADALINE (Adaptive Linear Neuron or later, Adaptive Linear Element).

Example:
Consider a neuron designed to implement the OR operation:

x_0	x_1	x_2	**OR**
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

The corresponding neuron is illustrated below:

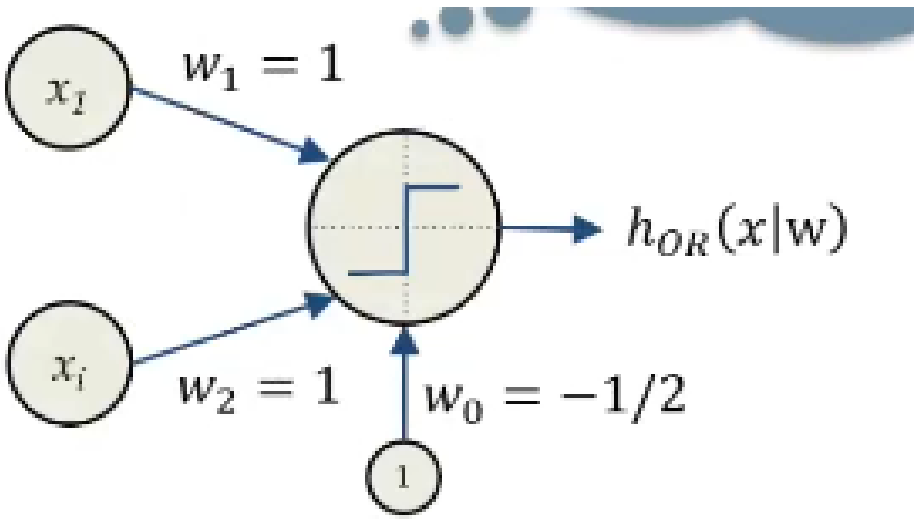

Figure 1.2: OR artificial neuron

The output function for this neuron is defined as:

$$h_{\text{OR}}(w_0 + w_1x_1 + w_2x_2) = h_{\text{OR}}\left(-\frac{1}{2} + x_1 + x_2\right) = \begin{cases} 1 & \text{if } \left(-\frac{1}{2} + x_1 + x_2\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Now, consider a neuron designed to implement the AND operation:

x_0	x_1	x_2	**AND**
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

The corresponding neuron is shown below:

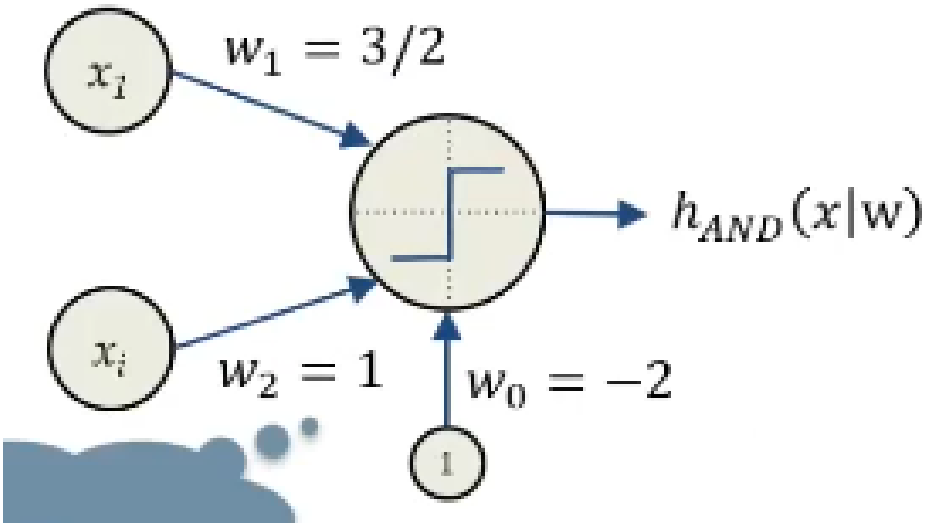

Figure 1.3: AND artificial neuron

The output function for this neuron is given by:

$$h_{\text{AND}}(w_0 + w_1x_1 + w_2x_2) = h_{\text{AND}}\left(-2 + \frac{3}{2}x_1 + x_2\right) = \begin{cases} 1 & \text{if } \left(-2 + \frac{3}{2}x_1 + x_2\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

1.3.3 Hebbian learning

The strength of a synapse increases based on the simultaneous activation of the corresponding input and the desired target. Hebbian learning can be summarized as follows:

1. Begin with a random initialization of the weights.
2. Adjust the weights for each sample individually (online learning), and only when the sample is not correctly predicted.

Mathematically, this is expressed as:

$$\begin{cases} w_i^{k+1} = w_i^k + \Delta w_i^k \\ \Delta w_i^k = \eta x_i^k t^k \end{cases} \implies w_i^{k+1} = w_i^k + \eta x_i^k t^k$$

Here, η represents the leraning rate, x_i^k is the i-th input to the perceptron at time k, and t^k is the desired output at time k.

Example:
We aim to learn the weights necessary to implement the OR operator with a sinusoidal output. The modified OR truth table is as follows:

x_0	x_1	x_2	**OR**
1	-1	-1	-1
1	-1	1	1
1	1	-1	1
1	1	1	1

We begin with random weights:

$$\mathbf{w} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

The learning rate is set to $\eta = \frac{1}{2}$. The output function is defined as:

$$h(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} = 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

The training involves iterating through the data records and adjusting the weights for incorrectly classified samples until all records are correctly predicted.

Starting from the first row, we have:

$$y_{\text{first row}} = x_0w_0 + x_1w_1 + x_2w_2 = 1 \cdot 0 + (-1) \cdot 0 + (-1) \cdot 0 = 0$$

This does not match the expected output of -1. We adjust the weights:

$$w_0^{\text{new}} = w_0 + \eta x_0 y = 0 + \frac{1}{2} \cdot 1 \cdot (-1) = -\frac{1}{2}$$

$$w_1^{\text{new}} = w_1 + \eta x_1 y = 0 + \frac{1}{2} \cdot (-1) \cdot (-1) = \frac{1}{2}$$

$$w_2^{\text{new}} = w_2 + \eta x_2 y = 0 + \frac{1}{2} \cdot (-1) \cdot (-1) = \frac{1}{2}$$

Now, the weights vector is:

$$\mathbf{w} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

For the second row, we have:

$$y_{\text{second row}} x_0 w_0 + x_1 w_1 + x_2 w_2 = 1 \cdot \left(-\frac{1}{2}\right) + (-1) \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = -\frac{1}{2}$$

This does not match the expected output of 1. We adjust the weights:

$$w_0^{\text{new}} = w_0 + \eta x_0 y = \left(-\frac{1}{2}\right) + \frac{1}{2} \cdot 1 \cdot 1 = 0$$

$$w_1^{\text{new}} = w_1 + \eta x_1 y = \frac{1}{2} + \frac{1}{2} \cdot (-1) \cdot 1 = 0$$

$$w_2^{\text{new}} = w_2 + \eta x_2 y = \frac{1}{2} + \frac{1}{2} \cdot 1 \cdot 1 = 1$$

Now, the weights vector is:

$$\mathbf{w} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

For the third row, we have:

$$y_{\text{third row}} x_0 w_0 + x_1 w_1 + x_2 w_2 = 1 \cdot 0 + 1 \cdot 0 + (-1) \cdot 1 = -1$$

This does not match the expected output of 1. We adjust the weights:

$$w_0^{\text{new}} = w_0 + \eta x_0 y = 0 + \frac{1}{2} \cdot 1 \cdot 1 = \frac{1}{2}$$

$$w_1^{\text{new}} = w_1 + \eta x_1 y = 0 + \frac{1}{2} \cdot 1 \cdot 1 = \frac{1}{2}$$

$$w_2^{\text{new}} = w_2 + \eta x_2 y = 1 + \frac{1}{2} \cdot (-1) \cdot 1 = \frac{1}{2}$$

Now, the weights vector is:

$$\mathbf{w} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

For the third row, we have:

$$y_{\text{fourth row}} x_0 w_0 + x_1 w_1 + x_2 w_2 = 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{3}{2}$$

This matches the expected output of 1.

After verifying the outputs for all rows, we recognize that further iterations (epochs) are needed for full convergence. We repeat the training until all records produce the desired outputs.

After multiple epochs, the final weights vector is:

$$\mathbf{w} = \begin{bmatrix} -\frac{1}{2} & 1 & 1 \end{bmatrix}$$

The number of epochs required depends on both the initialization of the weights and the order in which the data is presented.

A perceptron computes a weighted sum and returns the sign (thresholding) of the result:

$$h_j(\mathbf{x}|\mathbf{w}) = h_j\left(\sum_{i=0}^{I} w_i x_i\right) = \text{Sign}(w_0 + w_1 x_1 + \cdots + w_I x_I)$$

This forms a linear classifier, where the decision boundary is represented by the hyperplane:

$$w_0 + w_1 x_1 + \cdots + w_I x_I = 0$$

The linear boundary explains how the perceptron implements Boolean operators. However, if the dataset does not have a linearly separable boundary, the perceptron fails to work. In such cases, alternative approaches are needed, including non-linear boundaries or different input representations. This concept forms the basis for Multi-Layer Perceptrons (MLPs).

CHAPTER 2

Feed forward neural networks