Lecture 6: An Introduction to Deep Learning

Pr. Ismail Merabet

Univ. of K-M-Ouargla

December 17, 2024

Contents

1. Introduction

2. What is an artificial neural networks?

3. What is Learning?

4. The General Set-up

5. Stochastic Gradient Method

Introduction

The approximation of continuous functions is a cornerstone of mathematical analysis and computational science, with applications spanning from numerical modeling of many real problems.
Neural networks (NNs) have emerged as a powerful tool for approximating continuous functions, thanks to their flexibility, generalization capabilities, and universal approximation properties.
A foundational result is: the Universal Approximation Theorem[1], which states that a feedforward neural network with:

- a single hidden layer,
- equipped with a non-linear activation function [2],

can approximate any continuous function on a compact domain to arbitrary precision, provided the network has a sufficient number of neurons.

[1] Cybenko 89

[2] such as the sigmoid or ReLU

What is an artificial neural networks?

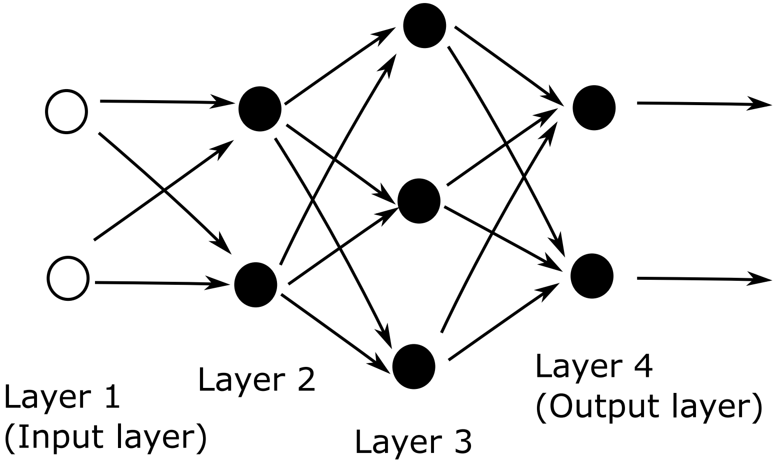

Figure: Artificial Neural Networks

What is an artificial neural networks?

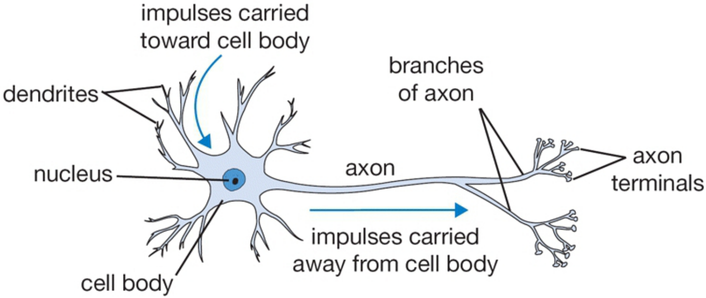

Figure: Natural Neuron

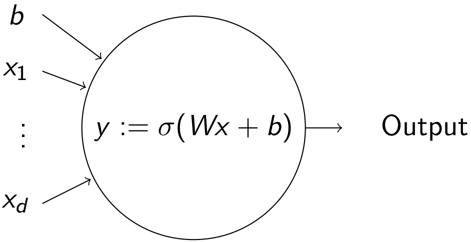

Figure: An artificial neuron

Definitions

- σ is called the activation function.
- W the weigth and b is bias.

Decision Making: Hiring based on experience

σ is the Heaviside function

$$H(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{1}$$

Here $x =$ Experience (years): The condition: if the candidate has more than 3 years of experience then say yes.

$$H(x-3) = \begin{cases} 0 & \text{if } x \leq 3 \quad \text{(No)} \\ 1 & \text{if } x > 3 \quad \text{(Yes)} \end{cases} \tag{2}$$

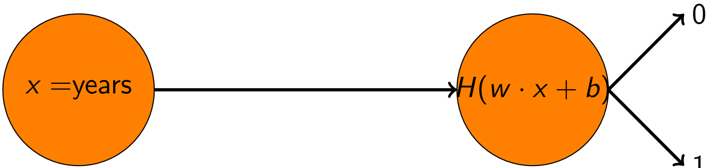

Figure: artificial neural network

Hiring based on experience and education level

Now x_1= experience and x_2= education level.

$$\sigma(w \cdot x + b), \quad x = (x_1, x_2), \quad w = (w_1, w_2)$$

Table:

Candidate	years	degree	score	decision
1	3	Phd (4)	6	Yes
2	6	Master (2)	5	Yes
3	4	Bachelor (1)	1	Yes
4	1	Master (2)	0	No

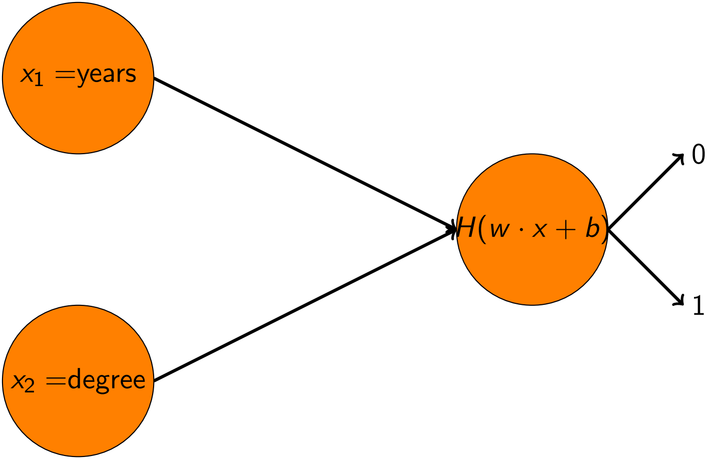

Figure: artificial neural network. $(w_1, w_2, b) = (1, 2, -5)$

Hiring with salary based on experience and education level

$$ReLU(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3}$$

Table:

Candidate	years	degree	salary
1	3	Phd (4)	12000
2	6	Master (2)	10000
3	4	Bachelor (1)	2000
4	1	Master (2)	0

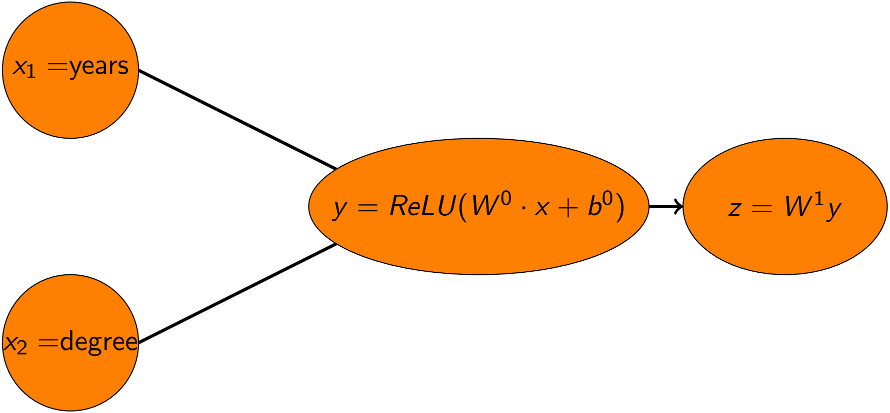

Figure: $\sigma = ReLU, \quad W^0 = (1, 2), \quad b^0 = -5, \quad W^1 = 2000$

What is Learning?

- Employees $p^i : i = 1, 2, ..., N$.
- Input $x^i = \left(\begin{array}{c} x_1^i \\ x_2^i \end{array} \right)$.
- Evaluation of p^i: $y^i \in \{0, 1, 2, 3, 4\}$.

Model:

$$f(x, \theta) = w^2 \sigma(w^1 \sigma(w^0 \cdot x + b^0) + b^1),$$

where, $\theta = (w^0, b^0, w^1, b^1, w^2)$ is unknown.

To find θ, we need to solve a minimization problem:

$$\theta^* = \arg\min L(\theta)$$

where,

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \| f(x^i, \theta) - y^i \|^2,$$

For the example 2 the exact function is

$$f(x, \theta) = \mathrm{ReLu}\left(2000 \times \mathrm{ReLu}(x_1 - 2x_2 - 5)\right)$$

Loss function

Definition

The functio L defined by:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \| f(x^i, \theta) - y^i \|^2$$

is called the loss function.

We emphasize that Loss is a function of the weights and biases; the data points are fixed.

Choosing the weights and biases in a way that minimizes the loss function is refered to as *training* the network.

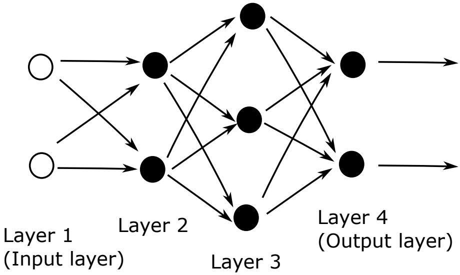

Figure: A four layers example

The four layer example Fig. 7 introduces the idea of neurons, represented by the sigmoid function, acting in layers.

The General Set-up

At a general layer, each neuron receives the same input one real value from every neuron at the previous layer and produces one real value, which is passed to every neuron at the next layer.
There are two exceptional layers.:

- At the input layer, there is no "previous layer" and each neuron receives the input vector.
- At the output layer, there is no "next layer"' and these neurons provide the overall output.

Definition

The layers in between these two are called *hidden layers*.

We now spell out the general form. We suppose that the network:

- has L layers,
- with layers 1 and L being the input and output layers, respectively.
- layer l, for $l = 1, 2, 3, \ldots, L$ contains n_l neurons.

So n_1 is the dimension of the input data.

Overall, the network maps from $\mathbb{R}^{n_1} \to \mathbb{R}^{n_L}$.

We use

- $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ to denote the matrix of weights at layer l.
- $w_{jk}^{[l]}$ is the weight that neuron j at layer l applies to the output from neuron k at layer $l-1$.
- Similarly, $b^{[l]} \in \mathbb{R}^{n_l}$ is the vector of biases for layer l, so neuron j at layer l uses the bias $b_j^{[l]}$.

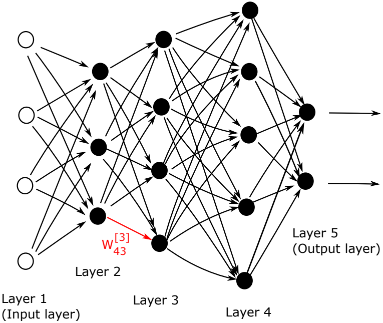

Figure: A five layers example

For this example,

- $L = 5$ layers,
- $n_1 = 4,\ n_2 = 3,\ n_3 = 4,\ n_4 = 5$ and $n_5 = 2$,
- $W^{[2]} \in \mathbb{R}^{3\times4},\ W^{[3]} \in \mathbb{R}^{4\times3},\ W^{[4]} \in \mathbb{R}^{5\times4},\ W^{[5]} \in \mathbb{R}^{2\times5}$,
- $b^{[2]} \in \mathbb{R}^{3},\ b^{[3]} \in \mathbb{R}^{4},\ b^{[4]} \in \mathbb{R}^{5}$ and $b^{[5]} \in \mathbb{R}^{2}$

Given an input $x \in \mathbb{R}^{n_1}$, we may then neatly summarize the action of the network by letting $a_j^{[l]}$ denote the output, or *activation*, from neuron j at layer l. So, we have

$$
\begin{aligned}
a^{[1]} &= x \in \mathbb{R}^{n_1}, && (4)\\
a^{[l]} &= \sigma\left(W^{[l]} a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad \text{for } l = 2, 3, \ldots, L. && (5)
\end{aligned}
$$

It should be clear that (4) and (5) amount to an algorithm for feeding the input forward through the network in order to produce an output $a^{[L]} \in \mathbb{R}^{n_L}$.

Now suppose: we have N pieces of data, or *training points*, in $\mathbb{R}^{n_1}$, $\{x^{\{i\}}\}_{i=1}^{N}$, for which there are given target outputs $\{y(x^{\{i\}})\}_{i=1}^{N}$ in $\mathbb{R}^{n_L}$. The quadratic loss function that we wish to minimize has the form

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \tfrac{1}{2} \| y(x^{\{i\}}) - a^{[L]}(x^{\{i\}}) \|_2^2, \tag{6}$$

Here, the factor $\frac{1}{2}$ is included for convenience; it simplifies matters when we start differentiating.

We note that, in principle, rescaling an objective function does not change an optimization problem. We should arrive at the same minimizer if we change L to, for example, $100\,L$ or $L/30$. So the factors $1/10$ and $1/2$ should have no effect on the outcome.

GD and SGD

We now introduce a classical method in optimization that is often referred to as *steepest descent* or *gradient descent*. The method proceeds iteratively, computing a sequence of vectors in $\mathbb{R}^s$ with the aim of converging to a vector that minimizes the cost function. Suppose that our current vector is θ.

How should we choose a perturbation, $\delta\theta$, so that the next vector, $\theta + \delta\theta$, represents an improvement? If $\delta\theta$ is small, then ignoring terms of order $\| \delta\theta \|^2$, a Taylor series expansion gives

$$Loss(\theta + \delta\theta) \approx Loss(\theta) + \sum_{r=1}^{s} \frac{\partial\, Loss(\theta)}{\partial\, \theta_r} \delta\theta_r. \tag{7}$$

Here $\partial\, Loss(\theta)/\partial\, \theta_r$ denotes the partial derivative of the loss function with respect to the rth parameter. For convenience, we will let $\nabla Loss(\theta) \in \mathbb{R}^s$ denote the vector of partial derivatives, known as the *gradient*, so that

$$(\nabla Loss(\theta))_r = \frac{\partial\, Loss(\theta)}{\partial\, \theta_r}.$$

Then (7) becomes

$$Loss(\theta + \delta\theta) \approx L(\theta) + \nabla Loss(\theta)^T \delta\theta. \tag{8}$$

Our aim is to reduce the value of the cost function. The relation (8) motivates the idea of choosing $\delta\theta$ to make $\nabla Loss(\theta)^T \delta\theta$ as negative as possible. So the most negative that $f^T g$ can be is $-\| f \|_2 \| g \|_2$, which happens when

$$f = -g.$$

Hence, based on (8), we should choose $\delta\theta$ to lie in the direction $-\nabla Loss(\theta)$. Keeping in mind that (8) is an approximation that is relevant only for small $\delta\theta$, we will limit ourselves to a small step in that direction. This leads to the update

$$p \to p - \eta \nabla Loss(\theta). \tag{9}$$

Here η is small stepsize that, in this context, is known as the *learning rate*.

Our function (6) involves a sum of individual terms. It follows that the partial derivative $\nabla Loss(\theta)$ is a sum of individual partial derivatives, i.e,

$$\nabla Loss(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla C_{x^{\{i\}}}(\theta). \tag{10}$$

$$C_{x^{\{i\}}} = \tfrac{1}{2} \| y(x^{\{i\}}) - a^{[L]}(x^{\{i\}}) \|_2^2. \tag{11}$$

When we have a large number of parameters and a large number of training points, computing the gradient vectorat every iteration of the steepest descent method (9) can be prohibitively expensive. A much cheaper alternative is to replace the mean of the individual gradients over all training points by the gradient at a single, randomly chosen training point. This leads to the simplest form of what is called the *stochastic gradient* method.

A single step may be summarized as

1. Choose an integer i uniformly at random from $\{1, 2, 3, \ldots, N\}$.
2. Update

$$\theta \to \theta - \eta \nabla C_{x^{\{i\}}}(\theta). \tag{12}$$