Neural networks

Houcine Senoussi

February 25, 2020

Introduction

- We introduce a new supervised learning method : neural networks.
- We first give definitions then we explain how this model **"learns"**.

What is a neural network ?

- Seen as a black box (figure5), a neural network is a system with **numerical** inputs and outputs that can be used to solve classification and regression problems.
- Classification problems :
  1. Given an instance X described by a numerical n-tuple $(x_1, ..., x_n)$, we have to find its class C in a set $\{C_1, ..., C_p\}$.
  2. In this case, the NN has n inputs and p outputs.
- Regression problems :
  1. Given an instance X described by a numerical n-tuple, we have to find the value $f(X) \in \mathbb{R}$.
  2. In this case, the NN has n inputs and 1 output.

What is a neural network ?

Figure: Neural network as a black box.

What is inside the black box ?

- The neural network consists of several **layers** (figure2): the input layer, the hidden layers and the output layer.
- The input of each layer comes from the previous layer.

What is inside the black box ?

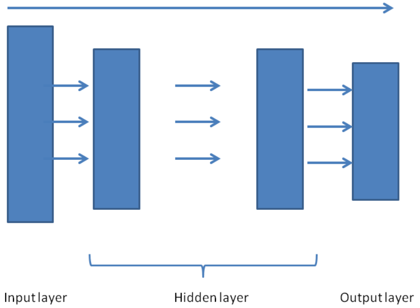

Figure: Layers of NN.

Let us zoom in on any layer

- Layers are sets of neurons.
- Each neuron in a layer $l+1$ is connected to each neuron of the previous layer l.
    - Its output is a function of a linear combination of those of the previous layer neurons.
- Structure and functionning of artificial neurons are inspired from those of natural neurons.

Neurons

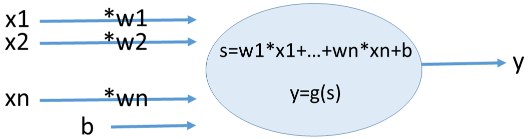

Figure: An artifical neuron.

Neurons

1. w_i's are the weights (of different inputs of the neuron). b is the bias.
   - The weight and the bias depend on the neuron.
2. g is the activation function. The two main functions are the step function (also called Heaviside) and the sigmoid function (figure 4).

Activation function

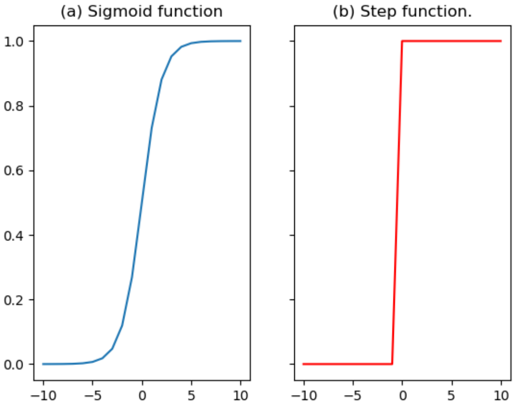

Figure: The two main activation functions

Notations

1. Number of layers : L.
   - Layer 1 : Input layer.
   - Layer 2-$(L-1)$: Hidden layers.
   - Layer L : Output layer.
2. Each neuron position in the network is defined by a couple (l, j), where l is the layer and j is the neuron position in the layer.
3. We call $x_j^{(l)}$ the output of the neuron (l, j).
   - For the output layer we also use the notation y_j as an equivalent to $x_j^{(L)}$.

Notations-2

1. We call w^l_{jk} the weight of the connexion between the neuron $(l-1,k)$ and the neuron (l,j).
2. We deduce that $x^{(l)}_j = g(\sum_{k\in(l-1)} w^l_{jk} x^{(l-1)}_k)$.
3. The sum $\sum_{k\in(l-1)} w_{jk} x^{(l-1)}_k$, called **weighted input** of the neuron (l,j), will be noted $z^{(l)}_j$.
4. It follows that $x^{(l)}_j = g(z^{(l)}_j)$.

Exercices

1. Propose a simple NN to compute the logical functions *AND*, *OR*.
2. Can we do the same thing for *XOR* ? Propose another solution.
3. Use *R* to define a NN for the iris data.

Parameters of a neural network

- Number of layers : depends on the problem. The most frequent case is 3 layers : Input+1 Hidden layer + Output.
- Number of neurons in each layer :
  - Input and Output layers : see above.
  - Hidden layers : there is no rule. In general many values are tried to find the optimal (or a good) one.
  - Activation function : depends on the problem.
  - The parameters given above are called **hyper-parameters**.
- Weights and biases : They are **learned** using a training set $D = \{(X_i, Y_i), i = 1, ..., n\}$.

Softmax layer

- In the case when we have a classification problem with p classes we can use the **softmax** function as an activation function of the output layer.
- Let us call $x_1, ..., x_l$ the inputs of the output layer (coming from the previous layer).
- Using the weights and biases of the output layer neurons we compute the following variables :
  - $t_i = \sum_{j=1,..,l} w_{ij} * x_j + b_i$
- then the following variables considered as the outputs of the network :
  - $\hat{y}_i = \frac{exp(t_j)}{\sum_{j=1,..,l} exp(t_j)}$

Computing the error

- We randomly initialize w's and b's.
- For each element X_i of D the network computes a value $\hat{Y}_i = \begin{pmatrix} \hat{y_{i1}} \\ \dots \\ \hat{y_{ip}} \end{pmatrix}$ of the output.
- For each example, we have $Y_i{=}C_q \in \{C_1, \dots, C_p\}$, but to compute the error we represent it as a vector $\begin{pmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}$ containing a 1 in the q^{th} position and 0's elsewhere.

Computing the error

- The difference between Y's and $\hat{Y}$'s gives us an error.
- This error is a function $E(W, B)$ (W : weights, B : biases). It can be defined in several ways. For example :
  - $E(W, B) = \frac{1}{2n} \sum_i \| Y_i - \hat{Y}_i(W, B) \|^2$
- To **minimize** this function, we use a very famous algorithm : **Gradient Descent**.

Gradient Descent-Principle

- Objective : finding the minimum of a function f.
- f can be a function of one or many variables.
  - $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
- Method : create a sequence x_i that will converge to the minimum.
- The general definition of this sequence is the following :
  - x_0 is any vector (can be defined randomly).
  - $x_{i+1} = x_i$ - $\eta \overrightarrow{gradf}(x_i)$
    - η is a parameter called **learning rate**.

Gradient Descent-Algorithm

- Input :
  - A precision $\epsilon \in \mathbb{R}$.
  - A maximal number of iteration *nb_max_iter*.
- Result : x_0=(x_{01}, ..., x_{0n}), $f(x_0)$.

1. Initialization : x_0 randomly choosed, $y_0 = f(x_0)$, nb_iter=0.
2. Repeat
   1. $nb\_iter = nb\_iter + 1$
   2. $x_1 = x_0$ - $\eta \overrightarrow{grad}(x_0)$
   3. $y_1 = f(x_1)$
   4. diff = absolute-value(y_1-y_0)
   5. $(x\_0,\ y\_0) = (x\_1,\ y\_1)$
3. While (diff $> \epsilon$) AND ($nb\_iter < nb\_max\_iter$)
4. Return (x_0, y_0).

Backpropagation

- To apply the gradient descent algorithm to our error function we need the following functions (components of the gradient) for each layer l, for each neuron of this layer (index i) and for each neuron of the previous level (index j) :
  - $\frac{\partial E}{\partial w^l_{jk}}$
  - $\frac{\partial E}{\partial b^l_j}$
- An efficient algorithm computes these values for the output layer (l=L), then the level $l = L - 1$ then more generally for each level l in function of the values for level $l + 1$: It is the **backpropagation** algorithm.

Backpropagation

- Input : the training set $D = \{(X_i, Y_i), i = 1, ..., n\}$
- Initialize randomly W and B.
- Repeat
  1. Foreach i=1, ..., n
     1. Put X_i in the input layer.
     2. For $l = 2, ..., L$ compute the output of the layer l (using the present valurs of the weights and the biases and the outputs of the level $l-1$). **(Feedforward)**
     3. The output of the level L is the vector $\hat{Y}_i$.
     4. Use the Y_i's and $\hat{Y}_i$'s to compute the error E. $E = \frac{1}{2} \parallel Y_i - \hat{Y}_i \parallel^2 = E = \frac{1}{2} \sum_j (y_{ij} - \hat{y}_{ij})^2$
     5. For $l = L, L-1, ..., 1$ **(Backpropagation)** Compute the values $\frac{\partial E}{\partial w_{jk}^l}$ and $\frac{\partial E}{\partial b_j^l}$ Update the values w_{jk}^l and b_j^l.
- Until stopping criterion is met.

Backpropagation

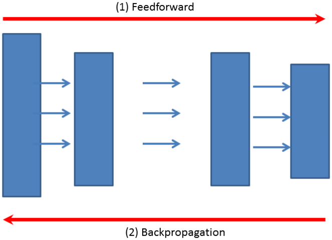

Figure: Steps of learning.

Backpropagation

- Updating the the values w^l_{jk} and b^l_j.
- For that, let us first introduce the variable δ^l_j that we call the error in the j^{th} neuron in the l^{th} layer, and which is defined as follows :
  - $\delta^l_j = \frac{\partial E}{\partial z^{(l)}_j}$ (remember the meaning of the variable $z^{(l)}_j$!).
  - δ^l : the vector of δ^l_j in the level l.
- Once we have these variables, we will use them to compute $\frac{\partial E}{\partial w^l_{jk}}$ and $\frac{\partial E}{\partial b^l_j}$.

Backpropagation

- Let us first consider δ^L, the computing error of the output layer.
- For each output neuron j we have :
  - $\hat{y}_j = x_j^L = g(z_j^L)$.
- It follows that :
  - $\delta_j^L = \frac{\partial E}{\partial z_j^{(L)}} = \frac{\partial E}{\partial x_j^{(L)}} \frac{dx_j^{(L)}}{dz_j^{(L)}} = \frac{\partial E}{\partial x_j^{(L)}} \, g'(z_j^{(L)})$.
- We see that δ_j^l is the product of two terms that can be easily computed :
  1. We have $E = \frac{1}{2} \sum_j (y_{ij} - x_j^{(L)})^2$. Therefore it is easy to compute $\frac{\partial E}{\partial x_j^{(L)}}$.
  2. $z_j^{(L)}$ is the linear combination of the inputs of the neuron and g' is the derivative of a well known function. Therefore, it is easy to compute

Backpropagation

- Now, let us compute δ^l in function of δ^{l+1}, for any layer l.
- We have (the proof is let as an exercice) :
  - $\delta^l = ((W^{l+1})^T \ \delta^{l+1} \bigodot g'(z^l)$.
- where $((W^{l+1})^T$ is the transpose of the weight matrix for thr layer $l+1$.

Backpropagation

- Now, we have to use the values δ^l to compute $\frac{\partial E}{\partial w^l_{jk}}$ and $\frac{\partial E}{\partial b^l_j}$.
- For that it is easy to see that for each neuron we have :
  1. (Derivative Error/bias) : $\frac{\partial E}{\partial b^l_j} = \frac{\partial E}{\partial z^l_j} = \delta^l_j$.
  2. (Derivative Error/weights) : $\frac{\partial E}{\partial w^l_{jk}} = x^{l-1}_k \ \delta^l_j$

Conclusion

- Neural networks : layers and neurons.
- Neurons : inputs, outputs, weights and biases.
- Parameters and hyperparameters and other parameters.
- Learning in a neural network : backpropagation.

Bibliographie

- M.A. Nielsen, "Neural networks and deep learning". Determination Press, 2015.