Multilayer Perceptron

Guillaume Frèche

Version 1.0

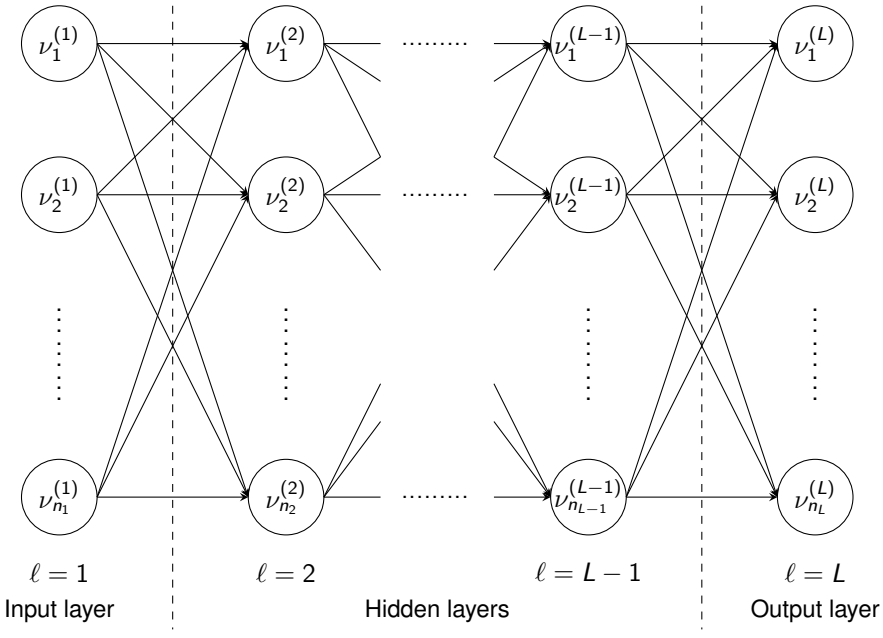
We have seen in the document about non-linear regression that with the basic structure of a single neuron, we can already perform classification tasks such as affine or quadratic separation. In this document, we use this simple element to build more elaborate structures: the **neural networks**.

1 Network structure

A **Multilayer Perceptron** (MLP) or **Artificial Neural Network** (ANN) is an oriented graph divided into $L \in \mathbb{N}$, $L \geq 2$, distinct layers.

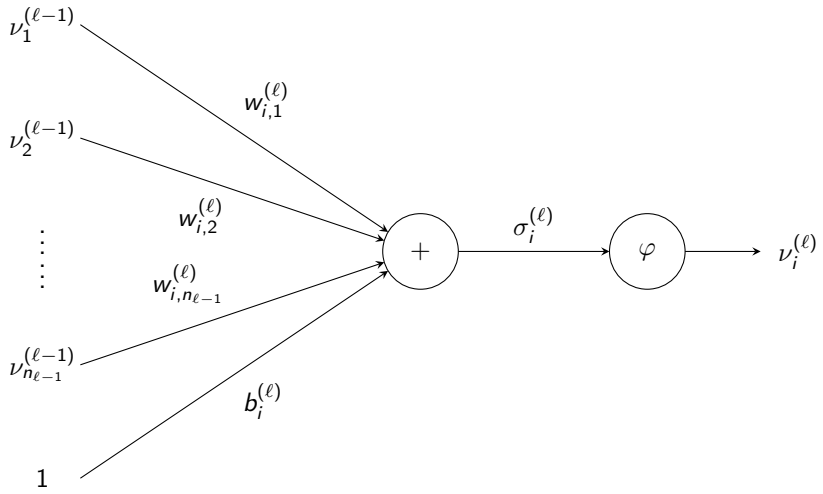
- Layer $\ell = 1$ is called the **input layer**;
- layer $\ell = L$ is called the **output layer**;
- any layer $\ell \in [\![2, L-1]\!]$ is called a **hidden layer**.

For any $\ell \in [\![1, L]\!]$, layer ℓ consists of $n_\ell \in \mathbb{N}^*$ **neurons** $\nu_1^{(\ell)}, \dots, \nu_{n_\ell}^{(\ell)}$.



Structure of a multilayer perceptron

For any $\ell \in [\![2, L]\!]$ and any $i \in [\![1, n_\ell]\!]$, neuron $\nu_i^{(\ell)}$ is connected to neurons $\nu_1^{(\ell-1)}, \dots, \nu_{n_{\ell-1}}^{(\ell-1)}$ of the previous layer $\ell - 1$ through vertices with assigned **weights** $w_{i,1}^{(\ell)}, \dots, w_{i,n_{\ell-1}}^{(\ell)}$ and **bias** $b_i^{(\ell)}$. The following diagram shows the structure of a single neuron, that we are going to detail in the next section.



Structure of single neuron

Remarks:

- The particular single-neuron classifier is an MLP such that $\ell = 2$, $n_1 = n$ and $n_2 = 1$.
- This type of neural network is also referred to as a **dense neural network** (DNN) because a neuron is connected to all the neurons of the previous layer. In future documents, we will see other types of neural networks that are not dense, such as the convolutional neural networks.

2 Feedforward algorithm

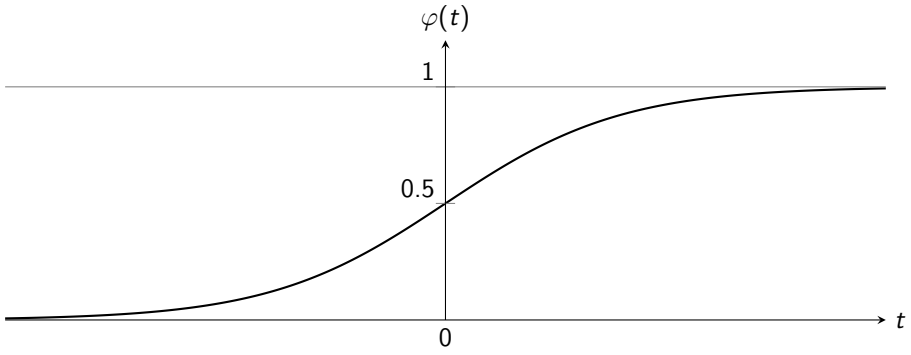
For simplicity, we denote $\nu_i^{(\ell)}$ the response of the corresponding neuron. A neural network is fed with an input vector $\mathbf{x} = (x_1, \dots, x_{n_1})$ and outputs the vector $\hat{\mathbf{y}} = \left(\nu_1^{(L)}, \dots, \nu_{n_L}^{(L)}\right)$ made of the responses of the output layer neurons.

- Every neuron in the input layer simply returns its associated component in the input vector, i.e. for any $i \in [\![1, n_1]\!]$, $\nu_i^{(1)} = x_i$.
- For any $\ell \in [\![2, L]\!]$ and any $i \in [\![1, n_\ell]\!]$, neuron response $\nu_i^{(\ell)}$ is defined from the neuron responses of the previous layer $\left(\nu_j^{(\ell-1)}\right)_{1 \leq j \leq n_{\ell-1}}$, weights $(w_{i,j})$, and bias $b_i^{(\ell)}$ by:

$$\sigma_i^{(\ell)} = \sum_{j=1}^{n_{\ell-1}} w_{i,j}^{(\ell)} \nu_j^{(\ell-1)} + b_i^{(\ell)} \qquad \nu_i^{(\ell)} = \varphi\left(\sigma_i^{(\ell)}\right) = \varphi\left(\sum_{j=1}^{n_{\ell-1}} w_{i,j}^{(\ell)} \nu_j^{(\ell-1)} + b_i^{(\ell)}\right) \tag{1}$$

  Bias can be seen as a weight connecting neuron $\nu_i^{(\ell)}$ to a neuron in the previous layer with no input and with output constantly equal to 1. In this expression, φ is the non-linear **activation function**. A commonly used activation function is the **sigmoid** defined by:

$$\varphi : \mathbb{R} \to [0, 1] \qquad t \mapsto \frac{1}{1 + e^{-t}}$$



This function is differentiable and its derivative can be easily expressed:

$$\forall t \in \mathbb{R} \qquad \varphi'(t) = (1 - \varphi(t))\,\varphi(t)$$

We can sum up this description in the following algorithm.

Algorithm 1 Feedforward algorithm

procedure FEEDFORWARD($\mathbf{x}$)
  Input $\mathbf{x} = (x_1, \dots, x_{n_1})$
  for $i \in [\![1, n_1]\!]$ **do**
    $\nu_i^{(1)} \leftarrow x_i$
  end for
  for $\ell \in [\![2, L]\!]$ **do**
    for $i \in [\![1, n_\ell]\!]$ **do**
      $\nu_i^{(\ell)} \leftarrow \varphi\left(\sigma_i^{(\ell)}\right) = \varphi\left(\sum_{j=1}^{n_{\ell-1}} w_{i,j}^{(\ell)} \nu_j^{(\ell-1)} + b_i^{(\ell)}\right)$
    end for
  end for
  Return $\hat{\mathbf{y}} = \left(\nu_1^{(L)}, \dots, \nu_{n_L}^{(L)}\right)$
end procedure

3 Backpropagation algorithm

We denote $\mathbf{w} = \left(w_{i,j}^{(\ell)}, b_i^{(\ell)}\right)$ the vector representing all the weights and biases in the network. The feedforward algorithm can be represented as a mapping $\mathcal{N}$ such that $\hat{\mathbf{y}} = \mathcal{N}(\mathbf{x}, \mathbf{w})$. During training, the MLP is fed with examples of input $\mathbf{x}_1, \dots, \mathbf{x}_N$ and output $\mathbf{y}_1, \dots, \mathbf{y}_N$ and aims at minimizing mean square error:

$$E = \frac{1}{N}\sum_{k=1}^{N} \|\hat{\mathbf{y}}_k - \mathbf{y}_k\|^2 = \frac{1}{N}\sum_{k=1}^{N} \|\mathcal{N}(\mathbf{x}_k, \mathbf{w}) - \mathbf{y}_k\|^2 = \frac{1}{N}\sum_{k=1}^{N} \xi\left(\mathbf{x}_k, \mathbf{y}_k, \mathbf{w}\right)$$

As discussed in the non-linear regression document, the network will rather minimize the individual error ξ as it gets new data on the fly. Weights and biases are updated according to the following **gradient descent**:

$$\widehat{\mathbf{w}}^{(k+1)} = \widehat{\mathbf{w}}^{(k)} - \mu_{k+1} \nabla \xi\left(\mathbf{x}_{k+1}, \mathbf{y}_{k+1}, \widehat{\mathbf{w}}^{(k)}\right)$$

where μ_{k+1} is the **learning factor**. To determine gradient $\nabla\xi$, we need to compute partial derivatives $\dfrac{\partial \xi}{\partial w_{i,j}^{(\ell)}}$ and $\dfrac{\partial \xi}{\partial b_i^{(\ell)}}$. As its name indicates, the backpropagation goes through the network backward, updating corresponding partial derivatives,

weights, and biases based on previous computations. Therefore, we start with weights and biases of the output layer, then we work on the hidden layers.

3.1 Output layer

By definition, $\xi = \sum_{i=1}^{n_L} \left(\nu_i^{(L)} - y_i\right)^2$. For any $i \in [\![1, n_L]\!]$,

$$\frac{\partial \xi}{\partial \nu_i^{(L)}} = 2\left(\nu_i^{(L)} - y_i\right) \tag{2}$$

If we consider bias $b_i^{(L)}$ and weights $w_{i,j}^{(L)}$ pointing to neuron $\nu_i^{(L)}$, then Equation (1) yields:

$$\frac{\partial \nu_i^{(L)}}{\partial b_i^{(L)}} = \varphi'\left(\sigma_i^{(L)}\right) \qquad \frac{\partial \nu_i^{(L)}}{\partial w_{i,j}^{(L)}} = \nu_j^{(L-1)}\varphi'\left(\sigma_i^{(L)}\right) = \nu_j^{(L-1)}\frac{\partial \nu_i^{(L)}}{\partial b_i^{(L)}} \tag{3}$$

Using the chain rule,

$$\frac{\partial \xi}{\partial b_i^{(L)}} = \frac{\partial \xi}{\partial \nu_i^{(L)}}\frac{\partial \nu_i^{(L)}}{\partial b_i^{(L)}} = 2\left(\nu_i^{(L)} - y_i\right)\varphi'\left(\sigma_i^{(L)}\right) \qquad \frac{\partial \xi}{\partial w_{i,j}^{(L)}} = \frac{\partial \xi}{\partial \nu_i^{(L)}}\frac{\partial \nu_i^{(L)}}{\partial w_{i,j}^{(L)}} = 2\left(\nu_i^{(L)} - y_i\right)\nu_j^{(L-1)}\varphi'\left(\sigma_i^{(L)}\right) \tag{4}$$

3.2 Hidden layer

Now we consider a hidden layer $\ell \in [\![2, L-1]\!]$. Assume that we have already applied the backpropagation algorithm on layer $\ell + 1$ and we have access to all the partial derivatives $\dfrac{\partial \xi}{\partial \nu_i^{(\ell+1)}}$ and $\dfrac{\partial \xi}{\partial w_{i,j}^{(\ell+1)}}$. From chain rule and Equation (1), we have

$$\frac{\partial \xi}{\partial \nu_i^{(\ell)}} = \sum_{h=1}^{n_{\ell+1}} \frac{\partial \xi}{\partial \nu_h^{(\ell+1)}}\frac{\partial \nu_h^{(\ell+1)}}{\partial \nu_i^{(\ell)}} \qquad \text{with} \qquad \frac{\partial \nu_h^{(\ell+1)}}{\partial \nu_i^{(\ell)}} = w_{h,i}^{(\ell+1)}\varphi'\left(\sigma_h^{(\ell+1)}\right) \tag{5}$$

As for the output layer, we use these partial derivatives to get

$$\frac{\partial \xi}{\partial b_i^{(\ell)}} = \frac{\partial \xi}{\partial \nu_i^{(\ell)}}\frac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}} \qquad \text{with} \qquad \frac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}} = \varphi'\left(\sigma_i^{(\ell)}\right) \tag{6}$$

$$\frac{\partial \xi}{\partial w_{i,j}^{(\ell)}} = \frac{\partial \xi}{\partial \nu_i^{(\ell)}}\frac{\partial \nu_i^{(\ell)}}{\partial w_{i,j}^{(\ell)}} \qquad \text{with} \qquad \frac{\partial \nu_i^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = \nu_j^{(\ell-1)}\varphi'\left(\sigma_i^{(\ell)}\right) = \nu_j^{(\ell-1)}\frac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}} \tag{7}$$

3.3 Algorithm

We can wrap up these equations in the following algorithm.

Algorithm 2 Backpropagation algorithm

procedure BACKPROPAGATION($\mathbf{y}$, μ)
  Input expected output $\mathbf{y} = (y_1, \dots, y_{n_L})$
  Input learning factor μ
  for $\ell \in [\![2, L]\!]$ in decreasing order **do**
    for $i \in [\![1, n_\ell]\!]$ **do**
      if $\ell = L$ **then**
        $\dfrac{\partial \xi}{\partial \nu_i^{(L)}} \leftarrow 2\left(\nu_i^{(L)} - y_i\right)$
      else
        for $h \in [\![1, n_{\ell+1}]\!]$ **do**
          $\dfrac{\partial \nu_h^{(\ell+1)}}{\partial \nu_i^{(\ell)}} \leftarrow w_{h,i}^{(\ell+1)} \varphi'\left(\sigma_h^{(\ell+1)}\right)$
        end for
        $\dfrac{\partial \xi}{\partial \nu_i^{(\ell)}} \leftarrow \sum_{h=1}^{n_{\ell+1}} \dfrac{\partial \xi}{\partial \nu_h^{(\ell+1)}} \dfrac{\partial \nu_h^{(\ell+1)}}{\partial \nu_i^{(\ell)}}$
      end if
      $\dfrac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}} \leftarrow \varphi'\left(\sigma_i^{(\ell)}\right)$ ▷ Bias derivative
      $b_i^{(\ell)} \leftarrow b_i^{(\ell)} - \mu \dfrac{\partial \xi}{\partial \nu_i^{(\ell)}} \dfrac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}}$ ▷ Bias update
      for $j \in [\![1, n_{\ell-1}]\!]$ **do**
        $w_{i,j}^{(\ell)} \leftarrow w_{i,j}^{(\ell)} - \mu \nu_j^{(\ell-1)} \dfrac{\partial \xi}{\partial \nu_i^{(\ell)}} \dfrac{\partial \nu_i^{(\ell)}}{\partial b_i^{(\ell)}}$ ▷ Weight update
      end for
    end for
  end for
end procedure