# Multi–Layer Perceptron

Alexandre Devert
Software Engineering School of the USTC

March 28, 2012

## 1 Introduction

A *Multi–Layer Perceptron*, or MLP is a very simple and common example of *feedforward neuron network*.

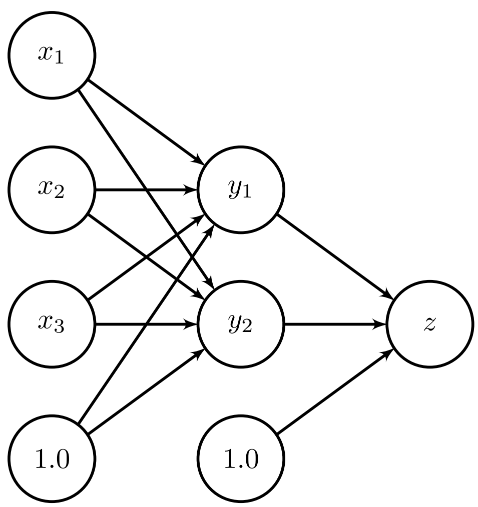


Figure 1: A MLP with 3 inputs and 1 hidden layer of 2 neurons, and 1 output neuron

Figure 1 shows a MLP. The left-most layer is called the *input layer*. The right-most part is the *output layer*. The inner layer is the *hidden layer*. One can see a MLP as a function $F(X, V, W) = z$.

- $n$ is the number of inputs

- $h$ is the number of hidden neurons
- $X$ is the input vector $\{x_1, x_2, \ldots, x_n\}$ of the MLP, dimension $n$
- $z$ is the output of the MLP
- $V$ and $W$ are the *weigth vectors* of the MLP.
- $V_i = \{v_{1,i}, v_{2,i}, \ldots, v_{n,i}, v_{n+1,i}\}$ where $V_i$ is the weight vector for hidden neuron $i$. $v_{n+1,i}$ is the *bias weight* of the hidden neuron $i$.
- $v_{j,i}$ is the weight between input $i$ and hidden neuron $j$.
- $W = \{w_1, w_2, \ldots, w_h, w_{h+1}\}$ where $w_i$ is the weight of the connection between hidden neuron $i$ and the output. $w_{h+1}$ is the *bias weight* of the output neuron.

By setting the $V$ and $W$ weights vectors with the proper values, a MLP can approximate any function. By using more neuron in the hidden layer, we can build more accurate approximations. A MLP is commonly used for:

- Regression tasks $\Rightarrow$ approximation of a function from which we only have noisy samples.
- Classification tasks $\Rightarrow$ learning a decision boundary from noisy examples.

# 2 Computing the output of a Multi–Layer Perceptron

A MLP is made of neurons. A neuron itself can be seen as a function $g$

$$G(X,U) = f(X.U + u_{n+1}) = f\left(u_{n+1} + \sum_{i=1}^{n} x_i u_i\right)$$

With

- $X$ is the input vector $\{x_1, x_2, \ldots, x_n\}$ of the neuron
- $U$ is the weight vector $\{u_1, u_2, \ldots, u_n, u_{n+1}\}$ of the neuron, of dimension $n+1$

- $f$ is the transfer function, tanh is a popular choice. See Figure 2 to have an idea of what that function looks like.

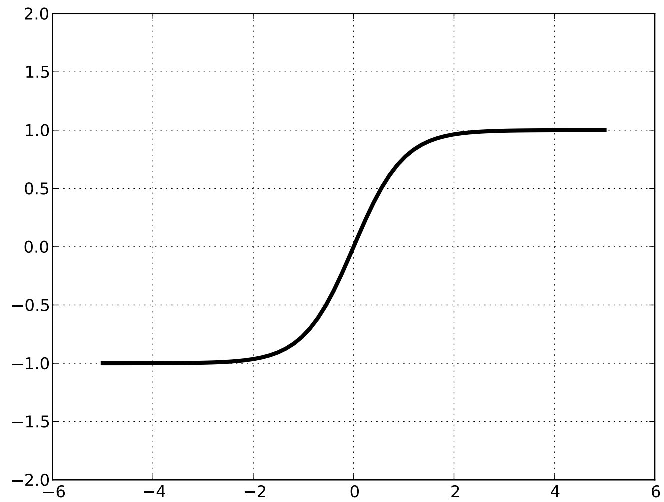


Figure 2: The tanh function

We can think of a MLP as a function $F(X, V, W) = z$. Usually, it is convinient to break the computation in 2 steps :

1. From the input layer to the hidden layer
2. From the hidden layer to the output layer

It is can be done by storing the outputs of the hidden layer neurons in a vector $Y$

$$F(V, W, X) = G(Y, W) = f(w_{h+1} + W.Y) = f\left(w_{h+1} + \sum_{i=1}^{h} w_i y_i\right)$$

$$y_i = G(V_i, X) = f(v_{n+1,i} + V_i.X) = f\left(v_{n+1,i} + \sum_{j=1}^{n} v_{j,i} x_j\right)$$

- $Y$ is a vector $\{y_1, y_2, \ldots, y_n\}$ of dimension $h$
- $W$ is the weight vector of the output neuron, dimension $h + 1$
- $V_1, V_2, \ldots, V_h$ are the weight vectors of the $h$ hidden neurons

# 3 Stochastic Gradient Descent for Multi–Layer Perceptron

To find MLP weight's such as the MLP minimize the error on a set of samples, we will use *stochastic gradient descent*. The idea is that for a randomly chosen example point $X$ with desired output $z^*$, we define the *error* of the MLP as

$$e = \frac{1}{2}(z - z^*)^2$$

To do reduce the error on that point, all the weights of the MLP will be modified as following

$$v_{a,b}(t+1) = v_{a,b}(t) + \eta \frac{\partial e}{\partial v_{a,b}}$$

$$w_b(t+1) = w_b(t) + \eta \frac{\partial e}{\partial w_b}$$

Where $\eta$ is the *learning rate*, how strong is the pertubation we apply to the MLP weights. The central idea of *stochastic gradient descent* is that, by repeating this for many points, with proper value for $\eta$, the MLP's will have a low error for most points.

## 3.1 Value for $\frac{\partial e}{\partial w_b}$

$$\frac{\partial e}{\partial w_b} = (z - z^*) \frac{\partial z}{\partial w_b}$$

$$\frac{\partial z}{\partial w_b} = y_b f'(w_{h+1} + W.Y)$$

We can thus conclude

$$\frac{\partial e}{\partial w_b} = (z - z^*) y_b f'(w_{h+1} + W.Y)$$

## 3.2 Value for $\frac{\partial e}{\partial v_{a,b}}$

$$\frac{\partial e}{\partial v_{b,a}} = (z - z^*) \frac{\partial z}{\partial v_{b,a}}$$

$$\frac{\partial z}{\partial v_{b,a}} = w_b \frac{\partial y_b}{\partial v_{b,a}} f'(w_{h+1} + W.Y)$$
$$\frac{\partial y}{\partial v_{b,a}} = x_a f'(v_{b,h+1} + V_b.X)$$

We can thus conclude

$$\frac{\partial e}{\partial v_{b,a}} = (z - z^*) w_b x_a f'(w_{h+1} + W.Y) f'(v_{b,h+1} + V_b.X)$$

# 4 Back-propagation algorithm

The *back-propagation algorithm* is a *stochastic gradient descent* method specialized for the MLP, where $\frac{\partial e}{\partial v_{b,a}}$ and$\frac{\partial e}{\partial w_b}$ are computed efficiently, using a minimal amount of memory. It is an iterative algorithm, repeating the following steps

1. Pick a pair $(X, z^*)$ randomly from the training set
2. *forward* pass
3. *backward* pass

## 4.1 The *forward* pass

This pass computes and store, in this order

1. $y_i$ and $y'_i$, where $y_i = f(s_i)$ and $y'_i = f'(s_i)$ with $s_i = v_{n+1,i} + V_i.X$
2. $z$ and $z'$, where $z = f(s_{h+1})$ and $z = f'(s_{h+1})$with $s_{h+1} = w_{h+1} + W.Y$

## 4.2 The *backward* pass

This pass computes, in this order

1. $\rho^{\text{out}} = (z - z^*) z'$
2. $\rho_b^{\text{in}} = w_b \rho^{\text{out}} y'_b$
3. $\frac{\partial e}{\partial w_b} = y_b \rho^{\text{out}} \Rightarrow w_b(t+1) = w_b(t) + \eta y_b \rho^{\text{out}}$
4. $\frac{\partial e}{\partial v_{b,a}} = x_a \rho_b^{\text{in}} \Rightarrow v_{a,b}(t+1) = v_{a,b}(t) + \eta x_a \rho_b^{\text{in}}$