Introduction to Multilayer Perceptron

Multilayer Fully Connected Feedforward Neural Networks

Vitor Greati[1]

[1]Federal University of Rio Grande do Norte

Table of Contents

Concept

Defining Neural Networks

Neural Networks or Artificial Neural Networks are computational models inspired in the neural system, containing a **labelled directed graph** $G = (V, E)$ whose nodes are capable of performing some **simple computation** and where each edge $(u, v) \in E$ **carries the output** of such computation from u to v increased or diminished by the **edge weight** $w(u, v)$.

The weights indicate the importance degree of the signal of each connection.

In a Neural Network, these **weights are modified** during the learning process by **learning algorithms**.

Applications

Neural Networks for what?

Neural networks can be applied to supervised, unsupervised and semi-supervised learning tasks, given the right architecture.

Common applications are:

- classification;
- regression;
- clustering;
- vector quantization;
- pattern association;
- function approximation.

Neurons

Basic elements

The neuron's basic task is to take an input, perform some computation and output a value.

Main elements

- An input vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in \mathbb{R}^n$.
- A vector of weights $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle \in \mathbb{R}^n$.
- A value $b \in \mathbb{R}$, called *bias*.
- A computation between inputs and weights, like

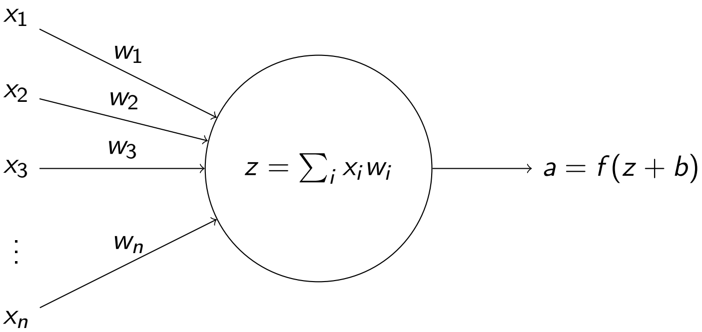
$$z = \mathbf{x} \cdot \mathbf{w} = \sum_i x_i w_i.$$

- An activation function f to produce an output $f(z + b)$.

Neurons

Graphical representation

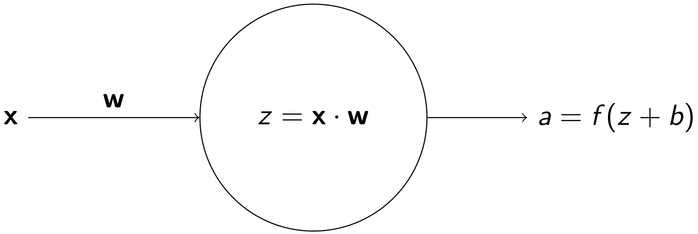
A common graphical representation highlights those elements:



Neurons

Graphical representation

The same representation, but using only vectors:



Activation functions

Sigmoid function

The sigmoid is a classical function in the neuron activation context. It is defined by:

$$f_{sig}(net) = \frac{1}{1 + e^{-net}}$$

When compared to the step function:

- f_{sig} is continuous and differentiable everywhere;
- f_{sig} is symmetric around the y-axis;
- f_{sig} asymptotically approaches its saturation values.

However:

- f_{sig} outputs are not zero centered;
- Saturated neurons essentially kill the gradient, since the delta will be extremely small.

Activation functions
Step function

The step function is defined as

$$f_s(net) = \begin{cases} 1, net > 0 \\ 0, net < 0 \end{cases}$$

Notice that:

- f_s only lets the signal pass if the computation results in a positive net;
- f_s is not differentiable, which produce problems for some learning algorithms;

This activation is used in the classic **Perceptron**.

Activation functions

Hyperbolic tangent function

The hyperbolic tangent function is defined as

$$f_{tanh}(net) = \tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}.$$

Notice that:

- f_{tanh} is zero centered;
- The problem for saturated neurons remains.

Activation functions

ReLU - Rectified Linear Unit

The ReLU function is defined as:

$$f_{relu}(net) = \max(0, net).$$

Notice:

- f_{relu} is not saturable and it is extremely efficient;
- f_{relu} is not differentiable at 0.

Activation functions

Leaky ReLU - Leaky Rectified Linear Unit

The Leaky ReLU is defined as:

$$f_{lrelu}(net) = \begin{cases} net, net \geq 0 \\ \alpha \times net, net < 0 \end{cases}$$

- f_{lrelu} allows a small, non-zero gradient at 0;
- f_{lrelu} allows negative values;
- PReLUs, Parametric ReLUs, allows α to be learned differently for each node.

Activation functions

ELU - Exponential Linear Units

The ELU function is defined as:

$$f_{elu}(net) = \begin{cases} net, net \geq 0 \\ \alpha \times (e^{net} - 1), neq < 0 \end{cases}$$

Here, α is a constant, set when the network is instantiated. A common value for it is $\alpha = 1.0$.

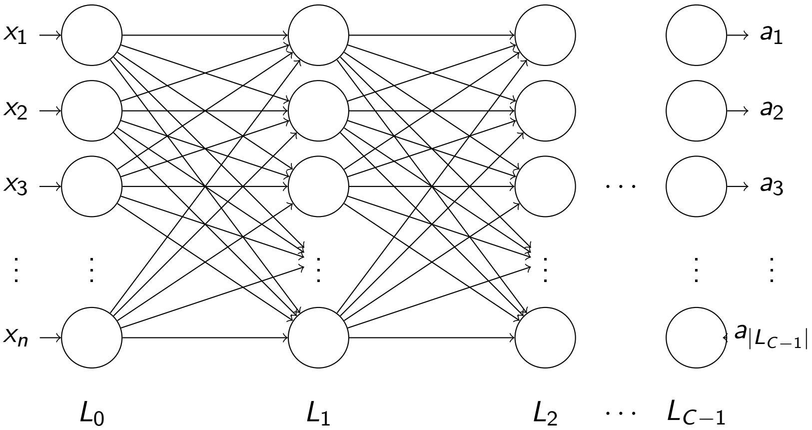
ELUs have presented better results than ReLUs.

Table of Contents

Multilayer Architecture

Fully connected feedforward architecture

An architecture with C layers, $L_0, L_1, \ldots, L_{C-1}$, is graphically represented as:



Multilayer Architecture

Fully connected feedforward architecture

Observations

- A layer L_i has its own size (number of neurons) $|L_i|$.
- L_0 is the **input layer**.
  - Neurons in this layer are **input neurons**.
  - An input neuron j takes the j-th component of the input vector.
  - Input neurons do not perform any computation or activation: just output the input value.
  - The output of L_0 is $\mathbf{x}$, the input vector.
- L_{C-1} is the **output layer** and its output is the network output.
- Any other layer is a **hidden layer**.
- This architecture is **fully connected** because each neuron in layer L_i is connected to every neuron in layer L_{i+1}.
- This architecture is **feedforward** because there is no back arrows forming cycles; otherwise, it would be *recurrent*.

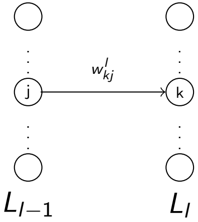
Multilayer Architecture

Fully connected feedforward architecture

How can we perform calculations and learning in this architecture? Mathematics, of course.

Representing connections between layers

Consider layer $L_l, l = 1, \ldots, C-1$, and neuron k of L_l. Since the architecture is fully connected, k is connected to all neurons j in L_{l-1}. Denote by w^l_{kj} the weight of the connection between neuron j of L_{l-1} and neuron k of L_l:



Multilayer Architecture

Fully connected feedforward architecture

In this way, we can represent all weights between two layers using only one matrix $\mathbf{W}^l = (w^l_{kj})$:

$$\mathbf{W}^l = \begin{bmatrix} w^l_{11} & w^l_{12} & \cdots & w^l_{1|L_{l-1}|} \\ w^l_{21} & w^l_{22} & \cdots & w^l_{2|L_{l-1}|} \\ \vdots & \vdots & \ddots & \vdots \\ w^l_{|L_l|1} & w^l_{|L_l|2} & \cdots & w^l_{|L_l||L_{l-1}|} \end{bmatrix}.$$

Notice that:

- $\mathbf{W}^l$ has dimensions $|L_l| \times |L_{l-1}|$.
- The weight vector of connections to neuron k in L_l is the k-th row of $\mathbf{W}^l$, denoted by $\mathbf{w^l_k}$.
- All connections in the network are represented! They are all in matrices $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^{C-1}$.

Representing the biases

Remember that each neuron has its own bias. So, let b_k^l denote the bias of neuron k in layer L_l. Then, the biases in layer l are represented as just a vector

$$\mathbf{b}^l = \langle b_1^l, b_2^l, \ldots, b_{|L_l|}^l \rangle.$$

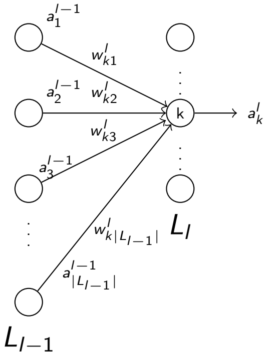
In this way, all biases are represented by vectors $\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^{C-1}$.

Multilayer Architecture

Fully connected feedforward architecture

Representing layer outputs

Denote by a^l_k the output of neuron k in layer L_l, and by $\mathbf{a^l}$ the output vector for layer L_l. How to compute such output? First, look:



Multilayer Architecture

Fully connected feedforward architecture

Denote by z_k^l the computation performed by the neuron, i.e:

$$z_k^l = \sum_j^{|L_{l-1}|} w_{kj}^l a_j^{l-1} = \mathbf{w_k^l a^{l-1}}.$$

Then, the neuron output is

$$a_k^l = f(z_k^l + b_k^l).$$

And the **layer output** is

$$\mathbf{a}^l = \mathbf{f}(\mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l).$$

Multilayer Architecture

Fully connected feedforward architecture

Didn't get it? Look:

$$
\begin{aligned}
\mathbf{f}\left(\mathbf{W}^{l}\cdot\mathbf{a}^{\mathbf{l}-\mathbf{1}}+\mathbf{b}^{\mathbf{l}}\right) &= \mathbf{f}\left(\begin{bmatrix} w_{11}^{l} & w_{12}^{l} & \dots & w_{1|L_{l-1}|}^{l} \\ w_{21}^{l} & w_{22}^{l} & \dots & w_{2|L_{l-1}|}^{l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{|L_l|1}^{l} & w_{|L_l|2}^{l} & \dots & w_{|L_l||L_{l-1}|}^{l} \end{bmatrix}\cdot\begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_{|L_{l-1}|}^{l-1} \end{bmatrix}+\begin{bmatrix} b_1^{l} \\ b_2^{l} \\ \vdots \\ b_{|L_l|}^{l} \end{bmatrix}\right) \\
&= \mathbf{f}\left(\begin{bmatrix} \sum_j^{|L_{l-1}|} w_{1j}a_j^{l-1}+b_1^l \\ \sum_j^{|L_{l-1}|} w_{2j}a_j^{l-1}+b_2^l \\ \vdots \\ \sum_j^{|L_{l-1}|} w_{|L_l|j}a_j^{l-1}+b_{|L_l|}^l \end{bmatrix}\right) \\
&= \mathbf{f}\left(\begin{bmatrix} \mathbf{w}_1^{\mathbf{l}}\mathbf{a}^{\mathbf{l}-\mathbf{1}}+b_1^l \\ \mathbf{w}_2^{\mathbf{l}}\mathbf{a}^{\mathbf{l}-\mathbf{1}}+b_2^l \\ \vdots \\ \mathbf{w}_{|\mathbf{L}_\mathbf{l}|}^{\mathbf{l}}\mathbf{a}^{\mathbf{l}-\mathbf{1}}+b_{|L_l|}^l \end{bmatrix}\right) \\
&= \begin{bmatrix} f(z_1^l) & f(z_2^l) & \dots & f(z_{|L_l|}^l) \end{bmatrix}^T \\
&= \begin{bmatrix} a_1^l & a_2^l & \dots & a_{|L_l|}^l \end{bmatrix}^T \\
&= \mathbf{a}^{\mathbf{l}}
\end{aligned}
$$

Multilayer Architecture

Computing the network output

Given an input vector $\mathbf{x}$, how to compute the output of the network?

We have an equation to compute any $\mathbf{a}^l$ and the algorithm just goes like this:

Input: Input vector $\mathbf{x}$
Output: Network output $\mathbf{a}^{C-1}$

```
begin
    a^0 ← x
    for l ← 1 to C − 1 do
        a^l ← W^l · a^(l−1) + b^l
    end
    return a^(C−1)
end
```

$$
\begin{array}{l}
\textbf{begin} \\
\quad \mathbf{a}^0 \leftarrow \mathbf{x} \\
\quad \textbf{for } l \leftarrow 1 \textbf{ to } C-1 \textbf{ do} \\
\quad\quad \mathbf{a}^l \leftarrow \mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l \\
\quad \textbf{end} \\
\quad \textbf{return } \mathbf{a}^{C-1} \\
\textbf{end}
\end{array}
$$