Memo on the Neural Network formulation

Summary

1 Introduction

This short note is dealing with the leaning process of a Neural Network (NN).

Neural networks are very simple devices with number of numbers: an organisation with multiple layers help to formalise the equations between layers. By adjusting the numbers adequately (i.e. the learning process), the outputs of the NN can be used to forecast values.

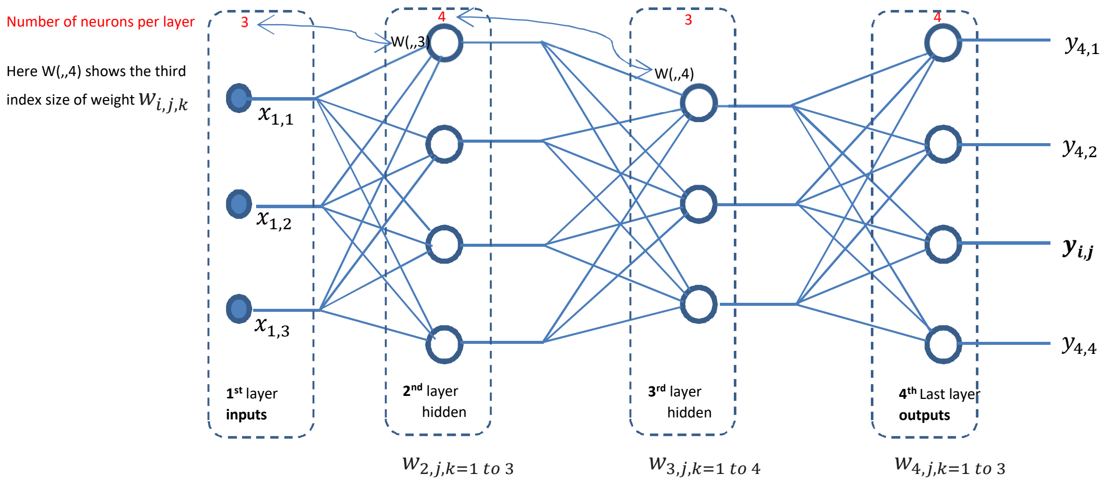
So, one important process of a NN is the learning process, this is obtained by adjusting the values of the weights taken into account for output the resulting values of the NN.

This learning process is based on the Gradient Descent principle within a backward propagation.

This short note is essentially dealing with the evaluation of the gradient, i.e. the partial derivatives of the error with respect to the weights in each layer. A recurrence form is output for the implementation of the learning process into a computer code.

2 Basic sketch of a Neural Network

See first § 5 and §6 below. The following figure is a 4 layers network, adapted from [R 1]:



$$\boldsymbol{y_{i,j}} = \varphi\left(\sum_{k=1}^{L_{i-1}.N} y_{i-1,k} w_{i,j,k}\right) \quad \textit{Eq. 1}$$

Hence, each neurons of layer i depends on number of neurons of previous layer $L_{i-1}.N$ for the size of the corresponding weights $w_{,,k=1\ to\ L_{i-1}.N}$. Those weights cannot be saved in the layer $i-1$ because those weights shall be set independently for each neuron j in layer i.

2.1 Notations for indexes in Neural Network (NN)

- In order to avoid confusions, it is important to distinguish the meanings of the successive indexes used, particularly for the neuron's output value $\boldsymbol{y}$ and for the weights $\boldsymbol{w}$:

first index for the layer: i the layer index

second index for the neuron: j the neuron index in layer i

$y_{i,j}$

second index for the neuron: j the neuron index in layer i

third index for the neuron in previous layer: k of the layer $i-1$

$w_{i,j,k}$

- With a TYPE definition, one set the dimensions of the NN with NL layers to :

$L_{i=1\ to\ NL}.\mathrm{N}$	the number of neurons in layer i
$L_{i=1\ to\ NL}.NN_{j=1\ to\ L_i.N}.y$	abbreviated here as $y_{i,j}$
$L_{i=1\ to\ NL}.NN_{j=1\ to\ L_i.N}.w_{k=1\ to\ L_{i-1}.N}$	abbreviated here as $w_{i,j,k}$

Hence the short "$y_{i-1,k}w_{i,j,k}$" is the product of the output value of the neuron k in layer $i-1$ by the weight in the neuron j of layer i dedicated to the neuron k of layer $i-1$.

3 Main relation of a NN

The main relation of a NN is that the output value $y_{i,j}$ of layer i and neuron j is coming from the weighted sum of outputs of previous layer $i-1$, where φ is the activation function:

$$y_{i,j} = \varphi(v_{i,j}) \qquad v_{i,j} = \sum_{k=1}^{L_{i-1}.N} y_{i-1,k} w_{i,j,k} \qquad \boxed{y_{i,j} = \varphi\left(\sum_{k=1}^{L_{i-1}.N} y_{i-1,k} w_{i,j,k}\right)} \quad \textit{Eq. 2}$$

In the same way, for lower layers:

$$y_{i-1,k} = \varphi(v_{i-1,k}), \qquad v_{i-1,k} = \sum_{\lambda=1}^{L_{i-2}.N} y_{i-2,\lambda} w_{i-1,k,\lambda} \qquad so \qquad \boxed{v_{i,j} = \sum_{k=1}^{L_{i-1}.N} \varphi(v_{i-1,k}) w_{i,j,k}} \quad \textit{Eq. 3}$$

and so on down to the second layer and first layer with the inputs of the NN.

$$y_{i-2,\lambda} = \varphi(v_{i-2,\lambda}) \qquad v_{i-2,\lambda} = \sum_{m=1}^{L_{i-3}.N} y_{i-3,m} w_{i-2,\lambda,m} \qquad v_{i-1,k} = \sum_{\lambda=1}^{L_{i-2}.N} \varphi(v_{i-2,\lambda}) w_{i-1,k,\lambda}$$

$$\boxed{v_{i,j} = \sum_{k=1}^{L_{i-1}.N} \varphi\left(\sum_{\lambda=1}^{L_{i-2}.N} \varphi(v_{i-2,\lambda}) w_{i-1,k,\lambda}\right) w_{i,j,k}} \ \ldots \quad \textit{Eq. 4}$$

4 Criterium based on the observed error

4.1 Error to be minimized

For the neurons in the **last layer** i, the error criterium of the overall result is:

$$E = \frac{1}{2}\sum_{j=1}^{L_i.N}\left(goal_j - y_{i,j}\right)^2 \qquad E = \frac{1}{2}\sum_{j=1}^{L_i.N} \varepsilon_j^2 \qquad \textit{Eq. 5}$$

with $goal_j$ the expected result for neuron j and $y_{i,j}$ the signal output: $goal_j - y_{i,j} = \varepsilon_j$ *Eq. 6*

To adjust the neurons weights to minimize the overall error criteria, each weight shall be trimmed to follow the opposite direction of the derivative $\frac{\partial E}{\partial w_{i,j,k}}$ (Gradient Descent).

4.2 Gradient assessment: partial derivative with respect to the weights

Let z being the weight with respect to which one want a partial derivative:

$E = \frac{1}{2}\sum_{j=1}^{L_i.N} \varepsilon_j^2 \quad \frac{\partial E}{\partial z} = \frac{1}{2}\frac{\partial\left[\sum_{j=1}^{L_i.N} \varepsilon_j^2\right]}{\partial z}$ the partial derivatives are applied to each term of the sum, so:

$\frac{\partial E}{\partial z} = \frac{1}{2}\sum_{j=1}^{L_i.N}\frac{\partial \varepsilon_j^2}{\partial z}$ which gives $\frac{\partial E}{\partial z} = \frac{1}{2}\sum_{j=1}^{L_i.N} 2\varepsilon_j \frac{\partial \varepsilon_j}{\partial z}$ i.e. $\frac{\partial E}{\partial z} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial z}$ *Eq. 7*

Because one has for every index i, j : $y_{i,j} = \varphi\left(v_{i,j}\right)$, $v_{i,j} = \sum_{k=1}^{L_{i-1}.N} y_{i-1,k}w_{i,j,k}$ and $\frac{\partial v_{i,j}}{\partial v_{i-1,k}} = \sum_{k=1}^{L_{i-1}.N}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}$

the general form of the error can be developed as seen in § 3 (here with 4 sums):

$$\frac{\partial E}{\partial z} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}\left[\sum_{k=1}^{L_{i-1}.N}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}\left\{\sum_{\lambda=1}^{L_{i-2}.N}\frac{\partial y_{i-2,\lambda}w_{i-1,k,\lambda}}{\partial v_{i-2,\lambda}}\left[\sum_{m=1}^{L_{i-3}.N}\frac{\partial y_{i-3,m}w_{i-2,\lambda,m}}{\partial v_{i-3,m}}\frac{\partial v_{i-3,m}}{\partial z}\right]\right\}\right]\text{...etc.} \qquad \textit{Eq. 8}$$

4.2.1 Case of last layer *i*

For the last layer i, the weights are $z = w_{i,j,k}$, so the development includes the two first sums:

$$\frac{\partial E}{\partial w_{i,j,k}} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}\left[\sum_{k=1}^{L_{i-1}.N}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial w_{i,j,k}}\right] \quad \text{(from eq. 8)}$$

- Note 1: For the last layer, $\varepsilon_j = goal_j - y_{i,j}$ so $\varepsilon_j = goal_j - \varphi\left(v_{i,j}\right)$, hence $\frac{\partial \varepsilon_j}{\partial v_{i,j}} = (-1)\frac{\partial y_{i,j}}{\partial v_{i,j}}$. *Eq. 9*
- Note 2: $y = \varphi(v)$ being a sigmoid, i.e. $\varphi(v) = \frac{1}{1+e^{-v}}$, its derivative is: $\frac{\partial y}{\partial v} = y(1-y)$.
- Note 3: because in the sums over j and k only one term is dependent on $w_{i,j,k}$, thus those sums disappear.

Hence eventually one gets: $\frac{\partial E}{\partial w_{i,j,k}} = \varepsilon_j(-1)y_{i,j}\left(1 - y_{i,j}\right)y_{i-1,k}$

For a recurrence one sets $\frac{\partial E}{\partial w_{i,j,k}} = \delta_{i,j}y_{i-1,k}$ with $\delta_{i,j} = \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}$ i.e. $\delta_{i,j} = \varepsilon_j(-1)y_{i,j}\left(1 - y_{i,j}\right)$ *Eq. 10*

4.2.2 Case of lower layers

- For other layers, starting with $i-1$, the weights are of the form $z = w_{i-1,k,\lambda}$ so, from eq. 8, the development includes 3 sums:

$$\frac{\partial E}{\partial w_{i-1,k,\lambda}} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}\left[\sum_{k=1}^{L_{i-1}.N}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}\left\{\sum_{\lambda=1}^{L_{i-2}.N}\frac{\partial y_{i-2,\lambda}w_{i-1,k,\lambda}}{\partial w_{i-1,k,\lambda}}\right\}\right]$$

And because in the last sums over k and λ only one term is dependent on $w_{i-1,k,\lambda}$, those sums disappear, but it remains for sure the sum over j :

$$\frac{\partial E}{\partial w_{i-1,k,\lambda}} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}\frac{\partial y_{i-2,\lambda}w_{i-1,k,\lambda}}{\partial w_{i-1,k,\lambda}} = \sum_{j=1}^{L_i.N} \varepsilon_j(-1)y_{i,j}\left(1 - y_{i,j}\right)w_{i,j,k}y_{i-1,k}\left(1 - y_{i-1,k}\right)y_{i-2,\lambda}$$

The recurrence remains $\frac{\partial E}{\partial w_{i-1,k,\lambda}} = \delta_{i-1,k}y_{i-2,\lambda}$ with $\delta_{i-1,k} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}$ *Eq. 11*

i.e. $\delta_{i-1,k} = \sum_{j=1}^{L_i.N} \delta_{i,j}\frac{\partial y_{i-1,k}w_{i,j,k}}{\partial v_{i-1,k}}$ i.e. $\delta_{i-1,k} = \sum_{j=1}^{L_i.N} \delta_{i,j}w_{i,j,k}y_{i-1,k}\left(1 - y_{i-1,k}\right)$

For other layers $i-2$, the weights are of the form $z = w_{i-2,\lambda,m}$ so, from eq. 8:

$$\frac{\partial E}{\partial w_{i-2,\lambda,m}} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}} \left[\sum_{k=1}^{L_{i-1}.N} \frac{\partial y_{i-1,k} w_{i,j,k}}{\partial v_{i-1,k}} \left\{\sum_{\lambda=1}^{L_{i-2}.N} \frac{\partial y_{i-2,\lambda} w_{i-1,k,\lambda}}{\partial v_{i-2,\lambda}} \left[\sum_{m=1}^{L_{i-3}.N} \frac{\partial y_{i-3,m} w_{i-2,\lambda,m}}{\partial w_{i-2,\lambda,m}}\right]\right\}\right] \quad \textit{Eq. 12}$$

And because in the last sums over λ and m only one term is dependent on $w_{i-2,\lambda,m}$, those sums disappear, but it remains the sums over j and k:

$$\frac{\partial E}{\partial w_{i-2,\lambda,m}} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}} \left[\sum_{k=1}^{L_{i-1}.N} \frac{\partial y_{i-1,k} w_{i,j,k}}{\partial v_{i-1,k}} w_{i-1,k,\lambda} y_{i-2,\lambda}\left(1 - y_{i-2,\lambda}\right) y_{i-3,m}\right]$$

Thanks to the distributive property of multiplication over addition, $\sum_j a_j [\sum_k b_k] = \sum_j \sum_k a_j\, b_k = \sum_k \sum_j a_j b_k$ one can reorganize the sums for the previous recurrence $\delta_{i-1,k} = \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}} \frac{\partial y_{i-1,k} w_{i,j,k}}{\partial v_{i-1,k}}$ to appears clearly:

$$\frac{\partial E}{\partial w_{i-2,\lambda,m}} = \sum_{k=1}^{L_{i-1}.N} \sum_{j=1}^{L_i.N} \varepsilon_j \frac{\partial \varepsilon_j}{\partial v_{i,j}} \frac{\partial y_{i-1,k} w_{i,j,k}}{\partial v_{i-1,k}} w_{i-1,k,\lambda} y_{i-2,\lambda}\left(1 - y_{i-2,\lambda}\right) y_{i-3,m} \quad \textit{Eq. 13}$$

$$\frac{\partial E}{\partial w_{i-2,\lambda,m}} = \sum_{k=1}^{L_{i-1}.N} \delta_{i-1,k} w_{i-1,k,\lambda} y_{i-2,\lambda}\left(1 - y_{i-2,\lambda}\right) y_{i-3,m}$$

The recurrence remains $\frac{\partial E}{\partial w_{i-1,\lambda,m}} = \delta_{i-2,\lambda} y_{i-3,m}$ with $\delta_{i-2,\lambda} = \sum_{k=1}^{L_{i-1}.N} \delta_{i-1,k} w_{i-1,k,\lambda} y_{i-2,\lambda}\left(1 - y_{i-2,\lambda}\right)$

For other layers $i-3$ and below, the weights are of the form $z = w_{i-3,\mathrm{m,n}}$ the recurrence equation remains: $\frac{\partial E}{\partial w_{i-2,m,n}} = \delta_{i-3,m} y_{i-4,n}$ with $\delta_{i-3,m} = \sum_{\lambda=1}^{L_{i-2}.N} \delta_{i-2,\lambda} w_{i-2,\lambda,m} y_{i-3,m}\left(1 - y_{i-3,m}\right)$

4.3 Gradient descent recurrence form from last layer i down to the second layer

One summarises the following equations suited for a recurrence form:

When ***i*** is the **last layer,** $\frac{\partial E}{\partial w_{i,j,k}} = \delta_{i,j}\, y_{i-1,k}$ with $\delta_{i,j} = -\varepsilon_j\, y_{i,j}\left(1 - y_{i,j}\right)$ *Eq. 14*

and for $\boldsymbol{i-1}$ (so not the **last layer**), $\frac{\partial E}{\partial w_{i-1,k,\lambda}} = \delta_{i-1,k}\, y_{i-2,\lambda}$ $\delta_{i-1,k} = \sum_{j=1}^{L_i.N} \delta_{i,j}\, w_{i,j,k} y_{i-1,k}\left(1 - y_{i-1,k}\right)$ but *Eq. 15*

while renaming i to $i+1$, k to m, j and λ to k, m to j one gets an obvious form of recurrence:

When ***i*** is **not the last layer,** $\frac{\partial E}{\partial w_{i,j,k}} = \delta_{i,j}\, y_{i-1,k}$ with $\delta_{i,j} = y_{i,j}\left(1 - y_{i,j}\right) \sum_{k=1}^{L_{i+1}.N} \delta_{i+1,k}\, w_{i+1,k,j}$ *Eq. 16*

4.4 Weights trim recurrence form from first layer up to last layer

$$w_{i,j,k} = \left(w_{i,j,k}\right)_{previous\ step} + \eta \frac{-\partial E}{\partial w_{i,j,k}} \quad \textit{Eq. 17}$$

with $1 \geq \eta \geq 0$ the learning coefficient. This is the “Delta rule”. The negative sign is needed for the iterative process to converge to the local minimum error, i.e. in opposite direction of the gradient direction.

A further form “Generalized delta rule” of the trim equation takes into account a part of the error from the previous learning step, with $1 \geq \alpha \geq 0$ a so-called momentum coefficient (form used in [R 2]):

$$w_{i,j,k} = \left(w_{i,j,k}\right)_{previous\ step} + \eta \frac{-\partial E}{\partial w_{i,j,k}} + \alpha \left(\eta \frac{-\partial E}{\partial w_{i,j,k}}\right)_{previous\ step} \quad \textit{Eq. 18}$$

Hence, this chapter concludes the possible implementation of the backward propagation of errors and the trim of the weights (i.e. the learning process) in a computer code.

5 Artificial neuron From Wikipedia, the free encyclopedia

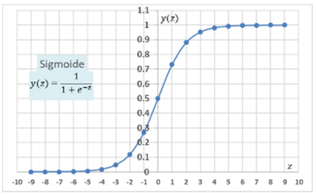
An **artificial neuron** is a mathematical function conceived as a model of biological neurons in their neural network with **200 types**.

Artificial neurons are elementary units in an artificial neural network.

The artificial neuron $\boldsymbol{v_k} \quad k = 1, n$

1. receives one or more inputs $x_i \quad i = 1, m$ (representing synapse excitatory postsynaptic potentials and inhibitory postsynaptic potentials at neural dendrites)
2. and sums them
3. to produce an output y_k (or activation, representing a neuron's action potential which is transmitted along its axon).
4. But usually each input is separately weighted, $\boldsymbol{w_{k,i}} \quad k = 1, n \quad i = 1, m$
5. and the sum is passed through an activation function φ or transfer function (non-linear function).

The transfer functions usually have a sigmoid shape



but they may also take the form of other non-linear functions, piecewise linear functions, or step functions. They are also often monotonically increasing, continuous, differentiable and bounded. *Non-monotonic, unbounded and oscillating activation functions with multiple zeros that outperform sigmoidal and ReLU (Rectified Linear Unit) like activation functions on many tasks have also been recently explored.*

The thresholding function has inspired building logic gates referred to as threshold logic; applicable to building logic circuits resembling brain processing.

For example, new devices such as memristors have been extensively used to develop such logic in recent times.[2]

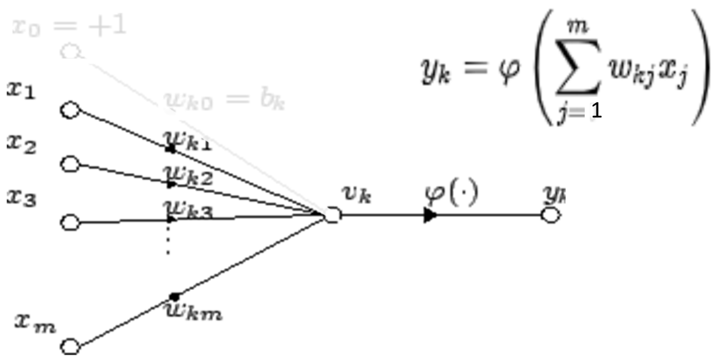
The artificial neuron transfer function should not be confused with a linear system's transfer function.

6 Basic structure

For a given artificial neuron k, let there be m inputs with signals x_1 through x_m and weights w_{k1} through w_{km}. *Usually, the x_0 input is assigned the value +1, which makes it a bias input with $w_{k0} = b_k$. This leaves only m actual inputs to the neuron: from x_1 to x_m.* The output of the k^{th} neuron is, with φ phi the transfer function (commonly a threshold function):

The output is analogous to the axon of a biological neuron, and its value propagates to the input of the next layer, through a synapse. It may also exit the system, possibly as part of an output vector.

It has no learning process as such. Its transfer function weights are calculated and threshold value are predetermined.



$$y_k = \varphi\left(\sum_{j=1}^{m} w_{kj} x_j\right)$$

7 References

[R 1] Marc Parizeau, « Le perceptron multicouche et son algorithme de rétropropagation des erreurs, » Université Laval (Canada)

[R 2] Patrice Dargenton, « Configuration d'un réseau de neurones avec un méta-réseau de neurones, » *patrice.dargenton@free.fr*

[R 3] Wikipedia, the free encyclopedia.
