# Derivation of a Forward and Backpropagation Pass on a Feedforward Neural Network

Raul Morales Delgado

## Contents

# 1 Objective

The objective of this tutorial is to demonstrate, through thorough calculations, how to derive a forward pass and a back-propagation pass through a feedforward neural network (FFNN) of $n$ inputs, one hidden layer with $m$ nodes, and $p$ outputs. In this sense, this tutorial aims to be a general case application for any FFNN with a single hidden layer.

# 2 Scope

In this tutorial, the architecture of a FFNN with a single hidden layer will be presented. This architecture considers $n$ inputs, $m$ nodes in the hidden layer, and $p$ outputs. Also, it includes the use of biases for the hidden and output layers, as well as generic activation functions for each node of these layers.

For the back-propagation process, a variation of the Mean Squared Error (MSE) function will be used as the loss function in order to set the iterative process to aim to minimize the error of the network in each iteration.

Since there is plenty documentation, articles and tutorials about how to code and use a NN, and how to tweak their hyperparameters for better learning and resulting accuracy, these topics will not be covered here. This tutorial will only focus on the mathematical processes involved in a simple FFNN.

# 3 Introduction

Feedforward neural networks (FFNN) are among the simplest neural networks and are the core upon which much more complex neural networks are built. A few days ago (from when I started writing this tutorial), I was taking a course on Natural Language Processing (NLP) and, as a way to explain RNN and LSTM networks, FFNN were revisited in a detailed manner, such that the student would follow along with the derivations that allow one to obtain the results of a forward pass and of a backpropagation pass.

However, throughout the process, there were some inconsistencies in the material which did not allow me to correctly follow the derivation. Because of this, I decided to do my very own, which is the core of this tutorial. Although there are plenty examples out there of the results of this process, they tend to be extremely summarized — they present the network and they leap through the equations — such that the underlying math is not clear if you want to follow along. Furthermore, these examples are usually limited to a single node in the output layer (i.e. for binary classification, e.g. `dog` / `no_dog`) and very few neurons in the hidden layer (three in the course I took).

For these reasons, I decided to derive the general case of a feedforward neural network with $n$ input nodes, a single hidden layer with $m$ nodes, and $p$ output nodes. The derivation will consist on the mathematical procedures involved in one forward pass through the network and one backpropagation pass. To execute the backpropagation process, a loss function is considered, as previously explained.

Finally, please bear in mind that I am not a Computer Scientist, let alone a Mathematician. While there might be some lack of rigorosity in the definitions or derivations, I trust that the process has been performed adequately and that the results are correct. If you find any mistake, please do not hesitate to email me or submit a GitHub issue.

# 4 The Feedforward Neural Network (FFNN)

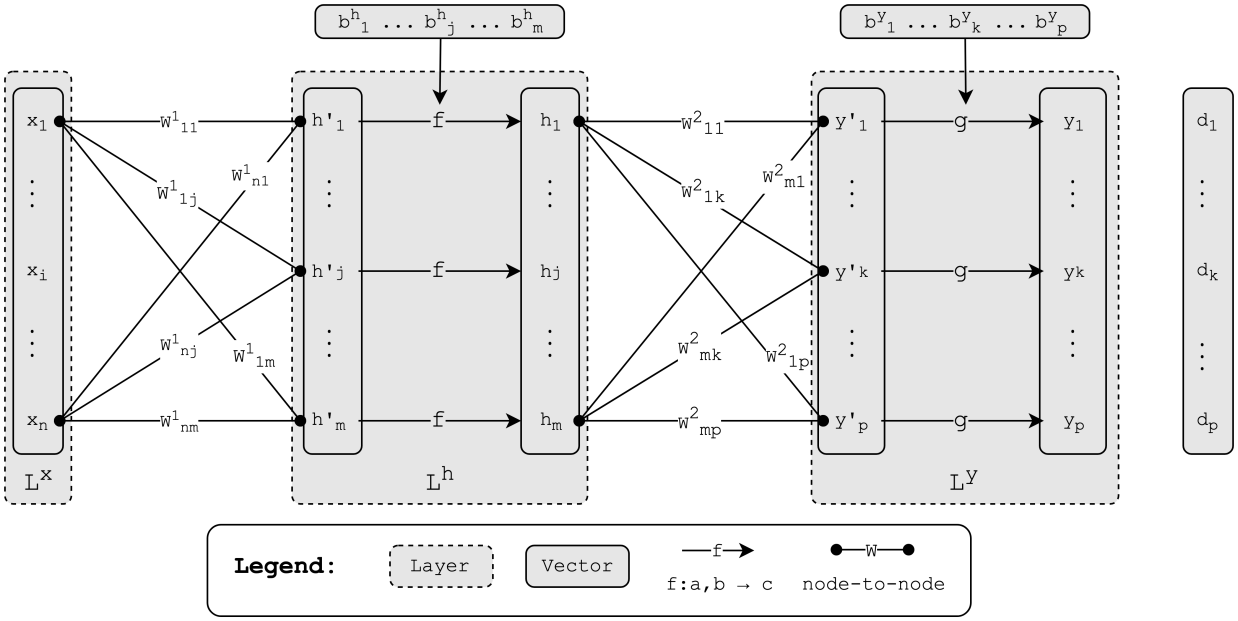


Figure 1: Diagram of a FFNN with a single hidden layer.

For the purpose of this tutorial, let's consider the FFNN of Figure 1. In this neural network, from left to right, the following can be defined:

- $\mathrm{L^x}$, $\mathrm{L^h}$, $\mathrm{L^y}$ Input, hidden and output layer, respectively.
- $\mathbf{x}$: Input vector of $n$ components, where $x_i$ represents the $i^{th}$ arbitrary component of the vector, such that:
$$\mathbf{x} = [x_1, \cdots, x_i, \cdots, x_n]$$
- $\mathbf{W^1}$: Matrix of weights 1, with size $n \times m$, which connects $\mathrm{L^x}$ to $\mathrm{L^h}$, where:
$$\mathbf{W^1} = \begin{bmatrix} W^1_{1,1} & \cdots & W^1_{1,j} & \cdots & W^1_{1,m} \\ \vdots & \ddots & \vdots & & \vdots \\ W^1_{i,1} & \cdots & W^1_{i,j} & \cdots & W^1_{i,m} \\ \vdots & & \vdots & \ddots & \vdots \\ W^1_{n,1} & \cdots & W^1_{n,j} & \cdots & W^1_{n,m} \end{bmatrix}$$
- $\mathbf{h'}$: Input vector to $\mathrm{L^h}$ of $m$ components, where $h'_j$ represents the $j^{th}$ arbitrary component of the vector, such that:
$$\mathbf{h'} = [h'_1, \cdots, h'_j, \cdots, h'_m]$$
- $\mathbf{b^h}$: Bias vector of $\mathrm{L^h}$ of $m$ components, where $b^h_j$ represents the $j^{th}$ arbitrary component of the vector, such that:
$$\mathbf{b^h} = [b^h_1, \cdots, b^h_j, \cdots, b^h_m]$$

- $f$: Activation function of the hidden layer, $\mathrm{L^h}$, such that $f : a, b \rightarrow c$, where $a$ and $b$ are inputs of the function, and $c$ its output.

- $\mathbf{h}$: Output vector to $\mathrm{L^h}$ of $m$ components, where $h_j$ represents the $j^{th}$ arbitrary component of the vector, such that:

$$\mathbf{h} = [h_1, \cdots, h_j, \cdots, h_m]$$

- $\mathbf{W^2}$: Matrix of weights 2, with size $m \times p$, which connects $\mathrm{L^h}$ to $\mathrm{L^y}$, where:

$$\mathbf{W^2} = \begin{bmatrix} W^2_{1,1} & \cdots & W^2_{1,k} & \cdots & W^2_{1,p} \\ \vdots & \ddots & \vdots & & \vdots \\ W^2_{j,1} & \cdots & W^2_{j,k} & \cdots & W^2_{j,p} \\ \vdots & & \vdots & \ddots & \vdots \\ W^2_{m,1} & \cdots & W^2_{m,k} & \cdots & W^2_{m,p} \end{bmatrix}$$

- $\mathbf{y'}$: Input vector to $\mathrm{L^y}$ of $p$ components, where $y'_k$ represents the $k^{th}$ arbitrary component of the vector, such that:

$$\mathbf{y'} = [y'_1, \cdots, y'_k, \cdots, y'_p]$$

- $\mathbf{b^y}$: Bias vector of $\mathrm{L^y}$ of $p$ components, where $b^y_k$ represents the $k^{th}$ arbitrary component of the vector, such that:

$$\mathbf{b^y} = [b^y_1, \cdots, b^y_k, \cdots, b^y_p]$$

- $g$: Activation function of the output layer, $\mathrm{L^y}$, such that $g : a, b \rightarrow c$, where $a$ and $b$ are inputs of the function, and $c$ its output.

- $\mathbf{y}$: Output vector to $\mathrm{L^y}$ of $p$ components, where $y_k$ represents the $k^{th}$ arbitrary component of the vector, such that:

$$\mathbf{y} = [y_1, \cdots, y_k, \cdots, y_p]$$

- $\mathbf{d}$: Target vector (vector of *true* values) of $p$ components, where $d_k$ represents the $k^{th}$ arbitrary component of the vector, such that:

$$\mathbf{d} = [d_1, \cdots, d_k, \cdots, d_p]$$

A forward pass in this FFNN can be defined as the set of signals that start at the input layer, $\mathrm{L^x}$, and finish in the output layer, $\mathrm{L^y}$; they go from left to right. Inversely, a backpropagation pass starts at $\mathrm{L^y}$ and finish at $\mathrm{L^x}$. In this FFNN, all nodes are interconnected (some of them, though, have not been connected in the diagram to not clutter it).

Also, for the purpose of this tutorial, vectors will be considered as 1-dimensional matrices. For instance, $\mathbf{x}$ is a matrix of 1 row and $n$ columns.

## 5 Analysis of the Forward Pass

Given the input vector $\mathbf{x}$ and the weight matrix $\mathbf{W^1}$, the input vector to the hidden layer, $\mathbf{h'}$, can be described by:

$$\mathbf{h'} = \mathbf{x}\mathbf{W^1} \tag{1}$$

In the hidden layer, $\mathrm{L^h}$, an activation function $f$, whose input parameters are vectors $\mathbf{h'}$ and $\mathbf{b^h}$, is used to yield the output vector $\mathbf{h}$, such that $f : h'_j, b^h_j \rightarrow h_j$. Therefore:

$$\mathbf{h} = f\left(\mathbf{h'} + \mathbf{b^h}\right) \tag{2}$$

A good summary about activation functions can be found in this here. Now, expanding the matrix multiplication of Eq.(1):

$$\mathbf{h'} = [x_1, \cdots, x_i, \cdots, x_n] \begin{bmatrix} W^1_{1,1} & \cdots & W^1_{1,j} & \cdots & W^1_{1,m} \\ \vdots & \ddots & \vdots & & \vdots \\ W^1_{i,1} & \cdots & W^1_{i,j} & \cdots & W^1_{i,m} \\ \vdots & & \vdots & \ddots & \vdots \\ W^1_{n,1} & \cdots & W^1_{n,j} & \cdots & W^1_{n,m} \end{bmatrix};$$

And evaluating the result for the $j^{th}$ component of $\mathbf{h'}$:

$$h'_j = x_1 \cdot W^1_{1j} + \ldots + x_i \cdot W^1_{ij} + \ldots + x_n \cdot W^1_{nj};$$

$$\therefore h'_j = \sum_i^n x_i \cdot W^1_{ij}. \tag{3}$$

Consequently, the $j^{th}$ component in $\mathbf{h}$ can be described by:

$$h_j = f\left(\sum_i^n x_i \cdot W^1_{ij} + b^h_j\right). \tag{4}$$

If we define $\psi_j$ as:

$$\psi_j = \sum_i^n x_i \cdot W^1_{ij} + b^h_j \tag{5}$$

Then, Eq.(4) can be rewritten as:

$$h_j = f\left(\psi_j\right). \tag{6}$$

Moving forward in the FFNN, the previous process can be repeated between the hidden layer, $\mathrm{L^h}$, and the output layer, $\mathrm{L^y}$. Therefore,

$$\mathbf{y'} = \mathbf{h}\mathbf{W^2} \tag{7}$$

And, in the output layer $\mathrm{L^y}$, the activation function $g$, whose input parameters are $\mathbf{y'}$ and $\mathbf{b^y}$, is used to yield the output vector $\mathbf{y}$, such that $g : y'_k, b^y_k \rightarrow y_k$, and:

$$\mathbf{y} = g\left(\mathbf{y'} + \mathbf{b^y}\right). \tag{8}$$

Consequently, the evaluated $k^{th}$ component of $\mathbf{y}$ can be described by:

$$y_k = g\left(\sum_j^m h_j \cdot W_{jk}^2 + b_k^y\right). \tag{9}$$

If we define $\omega_k$ as:

$$\omega_k = \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \tag{10}$$

Then, Eq.(9) can be rewritten as:

$$y_k = g\left(\omega_k\right). \tag{11}$$

Finally, Eq.(4) can be plugged into Eq.(9) and the expression rearranged for improved readability:

$$y_k = g\left(b_k^y + \sum_j^m f\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right) \cdot W_{jk}^2\right). \tag{12}$$

Therefore, from the previous equation, it can be stated that:

$$y_k = y_k\left(x_i, W_{ij}^1, W_{jk}^2, b_j^h, b_k^y\right) \tag{13}$$

And when the learning process (training) is completed, $\mathbf{W^1}$, $\mathbf{W^2}$, $\mathbf{b^h}$ and $\mathbf{b^y}$ are constant. Consequently, for all components of $\mathbf{y}$:

$$\mathbf{y} = \mathbf{y}\left(\mathbf{x}\right). \tag{14}$$

In summary, the previous expression indicates that when the training is completed a set of optimized weights and bias matrices — namely, $\mathbf{W^1}$, $\mathbf{W^2}$, $\mathbf{b^h}$ and $\mathbf{b^y}$ for this FFNN — have been found such that the model will output a vector, $\mathbf{y}$, as a function of a given input vector, $\mathbf{x}$.

## 6 Defining the Loss Function and its Gradient

Deep Learning neural networks, like many other AI models, require an iterative process during the training phase to adjust their parameters, like weights and bias matrices. Particularly, in supervised learning, where a model is trained with a labeled dataset, this process allows to improve the model's ability to predict a set of features from a given input. To achieve this, the iterative process uses a loss function (or, more generally, a cost function) to compare the output vector of the neural network to the target vector — whose difference is called the *error*. In general, the minimization of the output of this function — the minimization of the error — is performed using gradient descent and it allows to: a) define the path that will allow to reduce the error by adequately updating the neural network's parameters, and, b) set the limit as to where the model has reached an adequate learning (e.g. an early stop callback for when the error reaches a certain value).

To accomplish its goal, the loss function operates in the backpropagation pass of the network, which happens once after every forward pass during the training phase. The backpropagation pass will be analyzed in detail in the following section, but for now it is important to highlight that this pass goes from right to left — it starts in the output layer $\mathrm{L^y}$ and finishes in the input layer $\mathrm{L^x}$ (more specifically, it finishes in the weight matrix *before* the input layer).

The loss function considered for this tutorial will be a variation of the Mean Squared Error (MSE). More information about this function can be found in this link. If the MSE is defined as the loss function to calculate the error, $E$, then:

$$E = \sum_{k}^{p} \frac{(d_k - y_k)^2}{2}. \tag{15}$$

From the previous equation and from Eq.(13), it can be stated that:

$$E = E\left(d_k, x_i, W_{ij}^1, W_{jk}^2, b_j^h, b_k^y\right). \tag{16}$$

During the backpropagation pass, which is when the loss function is applied to the neural network to calculate the error, the input vector $\mathbf{x}$ and the output vector $\mathbf{y}$ are constant — the error is analyzed for this state of the network. In consequence, the independent variables of $E$ can be reduced to:

$$E = E\left(W_{ij}^1, W_{jk}^2, b_j^h, b_k^y\right). \tag{17}$$

To minimize the error of the network using gradient descent, the loss function must be differentiated with respect to its independent variables to find the gradient of the function, $\nabla E$, which can be stated as:

$$\nabla E = \left(\frac{\partial E}{\partial W_{ij}^1}, \frac{\partial E}{\partial W_{jk}^2}, \frac{\partial E}{\partial b_j^h}, \frac{\partial E}{\partial b_k^y}\right). \tag{18}$$

## 7 Gradient Descent and the Learning Rate

During the iteration process of a neural network, the current iteration step can be defined as $t$, and the immediate next step as $t + 1$. In a single step $t$, a forward pass and a backpropagation pass are completed, and as mentioned earlier, the result of the backpropagation pass will be an updated set of the neural network's parameters that will be used in the following step. Furthermore, these parameters are updated such that the error, calculated with the loss function, is reduced by using a gradient descent algorithm — a type of optimization algorithm — to perform a small displacement in each of the independent variables values in the direction of the negative gradient — i.e. to slightly modify each of the parameters of the neural network. More information about gradient descent can be found here.

To execute the small displacement on these variables, the gradient descent algorithm introduces a learning rate $\alpha$ and uses it to set the magnitude for the displacement at each step. For instance, for the FFNN of this tutorial, the displacements of the loss function's independent variables — $\Delta W_{ij}^1$, $\Delta W_{jk}^2$, $\Delta b_j^h$ and $\Delta b_k^y$ — can be defined as follows:

$$W_{ij@t+1}^1 - W_{ij@t}^1 = \Delta W_{ij}^1, \tag{19}$$

$$W_{jk@t+1}^2 - W_{jk@t}^2 = \Delta W_{jk}^2, \tag{20}$$

$$b_{j@t+1}^h - b_{j@t}^h = \Delta b_j^h, \tag{21}$$

$$b^y_{k@t+1} - b^y_{k@t} = \Delta b^y_k; \tag{22}$$

Where,

$$\Delta W^1_{ij} = -\alpha \cdot \frac{\partial E}{\partial W^1_{ij}}, \tag{23}$$

$$\Delta W^2_{jk} = -\alpha \cdot \frac{\partial E}{\partial W^2_{jk}}, \tag{24}$$

$$\Delta b^h_j = -\alpha \cdot \frac{\partial E}{\partial b^h_j}, \tag{25}$$

$$\Delta b^y_k = -\alpha \cdot \frac{\partial E}{\partial b^y_k}. \tag{26}$$

From the last four equations, it is important to highlight that $\alpha \in \mathbb{R} > 0$, and the negative sign in the expressions indicate that the displacement will be performed in the direction of the negative gradient. A simple explanation about how a gradient descent algorithm takes advantage of the negative gradient to minimize the error can be found in Appendix A.

Finally, when the small displacements are calculated and the network's parameters updated, the resulting error forms a monotonic sequence throughout the iteration process, such that:

$$E_{t-1} \geq E_t \geq E_{t+1} \geq \cdots . \tag{27}$$

# 8 Analysis of the Backpropagation Pass

In simple terms, the backpropagation pass in iteration step $t$ is used to minimize the error, such that an *optimized* set of parameters is calculated and used in the next step, $t+1$. The optimization of these values is defined by Eq.(19–22), and to solve those equations, Eq.(23–26) must be derived first. To keep a sequential order, this process will be carried out from the output layer towards the input layer, starting with the weights- and then the bias-related variables.

### Derivation of $\Delta W^2_{jk}$

To derive $\Delta W^2_{jk}$, Eq.(15) is plugged into Eq.(24), such that:

$$\Delta W^2_{jk} = -\alpha \cdot \frac{\partial \left( \sum_k^p \frac{(d_k - y_k)^2}{2} \right)}{\partial W^2_{jk}}. \tag{28}$$

If Eq.(9) is plugged into the previous expression and the sum expanded:

$$\Delta W^2_{jk} = -\alpha \cdot \left( \frac{\partial \left( \frac{\left( d_1 - g\left( \sum_j^m h_j \cdot W^2_{j1} + b^y_1 \right) \right)^2}{2} \right)}{\partial W^2_{jk}} + \cdots + \frac{\partial \left( \frac{\left( d_k - g\left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right) \right)^2}{2} \right)}{\partial W^2_{jk}} + \cdots + \frac{\partial \left( \frac{\left( d_p - g\left( \sum_j^m h_j \cdot W^2_{jp} + b^y_p \right) \right)^2}{2} \right)}{\partial W^2_{jk}} \right), \tag{29}$$

If each of the terms of the previous expression is differentiated with respect to $W^2_{jk}$, only the middle term — the one who has the $W^2_{jk}$ component in the numerator — would remain, such that:

$$\Delta W^2_{jk} = -\alpha \cdot \left( \frac{\partial \left( \frac{\left( d_k - g\left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right) \right)^2}{2} \right)}{\partial W^2_{jk}} \right),$$

And after differentiating by $\partial W^2_{jk}$, the expression can be simplified back to its previous form:

$$\Delta W^2_{jk} = \alpha \cdot (d_k - y_k) \cdot \frac{\partial y_k}{\partial W^2_{jk}}. \tag{30}$$

If $\delta_{\mathrm{W}^2}$ is defined as:

$$\delta_{\mathrm{W}^2} = \frac{\partial y_k}{\partial W^2_{jk}}, \tag{31}$$

Then, by substituting Eq.(9) into the previous expression:

$$\delta_{\mathrm{W}^2} = \frac{\partial g\left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)}{\partial W^2_{jk}}, \tag{32}$$

And by applying the chain rule and replacing with Eq.(10), the expression can be simplified to:

$$\delta_{\mathrm{W}^2} = \frac{\partial g\left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)}{\partial \left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)} \cdot \frac{\partial \left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)}{\partial W^2_{jk}};$$

$$\therefore \delta_{\mathrm{W}^2} = g'(\omega_k) \cdot h_j; \tag{33}$$

Where:

$$g'(\omega_k) = \frac{\partial g(\omega_k)}{\partial \omega_k} = \frac{\partial g\left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)}{\partial \left( \sum_j^m h_j \cdot W^2_{jk} + b^y_k \right)}. \tag{34}$$

Note how the summation disappears in the chain derivation when differentiating with respect to $W^2_{jk}$. This process is explained in Appendix B.

Finally, replacing Eq.(33) into Eq.(31), and the latter into Eq.(30):

$$\Delta W^2_{jk} = \alpha \cdot (d_k - y_k) \cdot g'(\omega_k) \cdot h_j. \tag{35}$$

### Derivation of $\Delta b_k^y$

The derivation of $\Delta b_k^y$ is very much like that of $\Delta W_{jk}^2$:

$$\Delta b_k^y = -\alpha \cdot \frac{\partial \left( \sum_k^p \frac{(d_k - y_k)^2}{2} \right)}{\partial b_k^y}, \tag{36}$$

After expanding the summation, because the differentiation is with respect to $b_k^y$, only the following remains:

$$\Delta b_k^y = -\alpha \cdot \left( \frac{\partial \left( \frac{\left( d_k - g\left( \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \right) \right)^2}{2} \right)}{\partial b_k^y} \right),$$

And after differentiating:

$$\Delta b_k^y = \alpha \cdot (d_k - y_k) \cdot \frac{\partial y_k}{\partial b_k^y}. \tag{37}$$

If $\delta_{\mathrm{b}^y}$ is defined as:

$$\delta_{\mathrm{b}^y} = \frac{\partial y_k}{\partial b_k^y}, \tag{38}$$

Then, by substituting Eq.(9) into the previous expression:

$$\delta_{\mathrm{b}^y} = \frac{\partial g\left( \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \right)}{\partial b_k^y}, \tag{39}$$

And by applying the chain rule and replacing with Eq.(10), the expression can be simplified to:

$$\delta_{\mathrm{b}^y} = \frac{\partial g\left( \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \right)}{\partial \left( \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \right)} \cdot \frac{\partial \left( \sum_j^m h_j \cdot W_{jk}^2 + b_k^y \right)}{\partial b_k^y};$$

$$\therefore \delta_{\mathrm{b}^y} = g'(\omega_k) \cdot (1) = g'(\omega_k); \tag{40}$$

Finally, replacing Eq.(40) into Eq.(38), and the latter into Eq.(37):

$$\Delta b_k^y = \alpha \cdot (d_k - y_k) \cdot g'(\omega_k). \tag{41}$$

**Derivation of $\Delta W_{ij}^1$**

To derive $\Delta W_{ij}^1$, first, Eq.(15) is plugged into Eq.(23):

$$\Delta W_{ij}^1 = -\alpha \cdot \frac{\partial \left( \sum_k^p \frac{(d_k - y_k)^2}{2} \right)}{\partial W_{ij}^1}. \tag{42}$$

If Eq.(12) is plugged into the previous expression and the sum expanded:

$$\Delta W_{ij}^1 = -\alpha \cdot \left( \frac{\partial \left( \frac{\left( d_1 - g\left( b_1^y + \sum_j^m f\left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right) \cdot W_{j1}^2 \right) \right)^2}{2} \right)}{\partial W_{ij}^1} + \cdots + \frac{\partial \left( \frac{\left( d_k - g\left( b_k^y + \sum_j^m f\left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right) \cdot W_{jk}^2 \right) \right)^2}{2} \right)}{\partial W_{ij}^1} + \cdots + \frac{\partial \left( \frac{\left( d_p - g\left( b_p^y \right. \right.}{} \right. \tag{43}$$

Because each of the components of the sum $\sum_k^p$ has a $W_{ij}^1$ term nested inside, none of the components is zero after differentiation by $\partial W_{ij}^1$; thus, the summation remains. Consequently, proceeding with the differentiation:

$$\Delta W_{ij}^1 = \alpha \cdot \sum_k^p (d_k - y_k) \cdot \frac{\partial y_k}{\partial W_{ij}^1}. \tag{44}$$

If $\delta_{\mathrm{W}^1}$ is defined as:

$$\delta_{\mathrm{W}^1} = \frac{\partial y_k}{\partial W_{ij}^1}, \tag{45}$$

Then, by substituting Eq.(12) into the previous expression:

$$\delta_{\mathrm{W}^1} = \frac{\partial g \left( b_k^y + \sum_j^m f\left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right) \cdot W_{jk}^2 \right)}{\partial W_{ij}^1}, \tag{46}$$

To derive the previous expression, Eq.(4) is introduced to use the chain rule, such that:

$$\delta_{\mathrm{W}^1} = \frac{\partial g \left( b_k^y + \sum_j^m h_j \cdot W_{jk}^2 \right)}{\partial h_j} \cdot \frac{\partial h_j}{\partial W_{ij}^1}$$

$$\delta_{\mathrm{W}^1} = \frac{\partial g \left( b_k^y + \sum_j^m h_j \cdot W_{jk}^2 \right)}{\partial \left( b_k^y + \sum_j^m h_j \cdot W_{jk}^2 \right)} \cdot \frac{\partial \left( b_k^y + \sum_j^m h_j \cdot W_{jk}^2 \right)}{\partial h_j} \cdot \frac{\partial f \left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right)}{\partial \left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right)} \cdot \frac{\partial \left( b_j^h + \sum_i^n x_i \cdot W_{ij}^1 \right)}{\partial W_{ij}^1}$$

$$\therefore \delta_{\mathrm{W}^1} = g'(\omega_k) \cdot W_{jk}^2 \cdot f'(\psi_j) \cdot x_i, \tag{47}$$

Where:

$$f'(\psi_j) = \frac{\partial f(\psi_j)}{\partial \psi_j} = \frac{\partial f\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right)}{\partial\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right)}. \tag{48}$$

Finally, replacing Eq.(47) into Eq.(45), and the latter into Eq.(44) and rearranging the expression:

$$\Delta W_{ij}^1 = \alpha \cdot f'(\psi_j) \cdot x_i \cdot \sum_k^p (d_k - y_k) \cdot g'(\omega_k) \cdot W_{jk}^2. \tag{49}$$

## Derivation of $\Delta b_j^h$

The derivation of $\Delta b_j^h$ is somewhat similar to that of $\Delta W_{ij}^1$. Plugging Eq.(15) into Eq.(25):

$$\Delta b_j^h = -\alpha \cdot \frac{\partial\left(\sum_k^p \frac{(d_k - y_k)^2}{2}\right)}{\partial b_j^h}, \tag{50}$$

And differentiating by $\partial b_j^h$:

$$\Delta b_j^h = \alpha \cdot \sum_k^p (d_k - y_k) \cdot \frac{\partial y_k}{\partial b_j^h}. \tag{51}$$

If $\delta_{\mathrm{b}^h}$ is defined as:

$$\delta_{\mathrm{b}^h} = \frac{\partial y_k}{\partial b_j^h}, \tag{52}$$

Then, by substituting Eq.(12) into the previous expression:

$$\delta_{\mathrm{b}^h} = \frac{\partial g\left(b_k^y + \sum_j^m f\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right) \cdot W_{jk}^2\right)}{\partial b_j^h}, \tag{53}$$

To derive the previous expression, Eq.(4) is introduced to use the chain rule, such that:

$$\delta_{\mathrm{b}^h} = \frac{\partial g\left(b_k^y + \sum_j^m h_j \cdot W_{jk}^2\right)}{\partial h_j} \cdot \frac{\partial h_j}{\partial b_j^h}$$

$$\delta_{\mathrm{b}^h} = \frac{\partial g\left(b_k^y + \sum_j^m h_j \cdot W_{jk}^2\right)}{\partial\left(b_k^y + \sum_j^m h_j \cdot W_{jk}^2\right)} \cdot \frac{\partial\left(b_k^y + \sum_j^m h_j \cdot W_{jk}^2\right)}{\partial h_j} \cdot \frac{\partial f\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right)}{\partial\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right)} \cdot \frac{\partial\left(b_j^h + \sum_i^n x_i \cdot W_{ij}^1\right)}{\partial b_j^h}$$

$$\delta_{\mathrm{b}^h} = g'(\omega_k) \cdot W^2_{jk} \cdot f'(\psi_j) \cdot (1)$$

$$\therefore \delta_{\mathrm{b}^h} = g'(\omega_k) \cdot W^2_{jk} \cdot f'(\psi_j). \tag{54}$$

Finally, replacing Eq.(54) into Eq.(52), and the latter into Eq.(51) and rearranging the expression:

$$\Delta b^h_j = \alpha \cdot f'(\psi_j) \cdot \sum_k^p (d_k - y_k) \cdot g'(\omega_k) \cdot W^2_{jk}. \tag{55}$$

# 9 Conclusions

- In a forward pass throughout the FFNN, a output vector $\mathbf{y}$ is generated, and its $k^{th}$ component is described by Eq.(12):

$$y_k = g\left(b^y_k + \sum_j^m f\left(b^h_j + \sum_i^n x_i \cdot W^1_{ij}\right) \cdot W^2_{jk}\right)$$

  In this equation, the sums $\sum_j^m$ and $\sum_i^n$ indicate that calculation of $y_k$ takes into account all the weights and biases of the previous layers, which is what was proposed at the beginning of this FFNN — that previous layers must be fully interconnected.

- In the backpropagation pass, the effect of interconnecting layers is appreciated as the pass starts delving into the layers. For instance, the equations for $\Delta W^2_{jk}$ and $\Delta b^y_k$ (Eq.(35) and Eq.(41), respectively), depend only on the $k^{th}$ component of $\mathbf{y}$, while the equations for $\Delta W^1_{ij}$ and $\Delta b^h_j$ (Eq.(49) and Eq.(55), respectively), depend on all the components of $\mathbf{y}$ and only on the $j^{th}$ component of $\mathbf{h}$. If there were more hidden layers, these dependencies would keep adding up until the last weights and bias matrices previous to the input layer.

# Appendix A

This Appendix is under construction.

# Appendix B

This Appendix is under construction.