VBA projet: The perceptron in finance

You will be evaluated on this project. You will work in groups of two students of the same TD group. Before the end of **Sunday 27th December**, you will send by mail to **your TD teacher** your Excel file as well as a short report (no more than 3 pages) containing interpretation answers. Every project with badly indented code, without the *option explicit*, without the name of the variable to be incremented after a *Next*, without the two bounds specified for each dimension in the declaration of an array, or without comments will be penalized.

Introduction

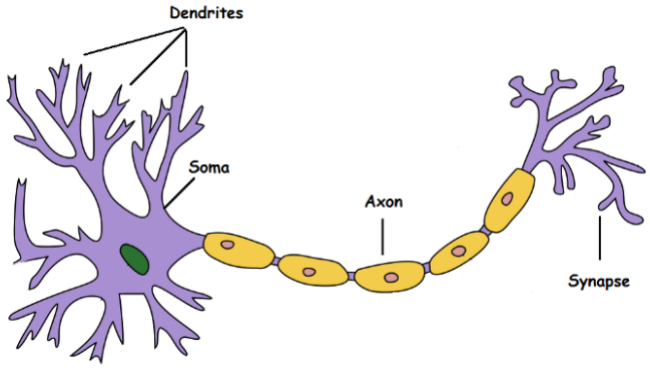
The purpose of this directed work is to implement the perceptron model in VBA. The historical background section will give you the big picture of the perceptron development theory (this part is optional to read). Section 2 presents the formal notations of a perceptron and how its parameters are optimized through the learning process. You should understand this part for a correct implementation of the perceptron. In particular, you will work on strategies for weights initialization and a plan to implement this model. In Section 3, you will apply this model to a financial dataset and compare this approach to a more traditional econometric model. Section 4 indicates the evaluation rules.

Note that this model can be implemented for either classification or regression purposes.

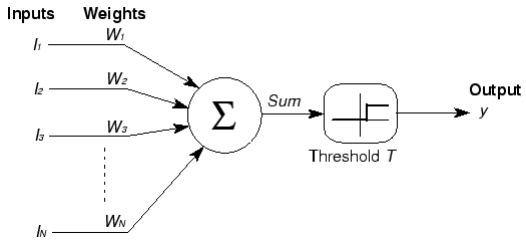
1 Historical background (for your culture)

1.1 The perceptron history

The first reference to the formal neuron is attributed to McCulloch and Pitts [5] who tried to mimic the functionality of a biological neuron.



(a) Representation of a biological neuron



(b) McCulloch and Pitt formal neuron

As you can see on figure (a), the simplified biological neuron is composed of:

- Dendrite: Receives signals from other neurons

- Soma: Processes the information
- Axon: Transmits the output of this neuron
- Synapse: Point of connection to other neurons

In their paper, McCulloch and Pitts consider that inputs (signals that comes from the dendrites) are either inhibitory (prevents the generation of an electrical signal) or excitatory (promotes the generation of an electrical signal) and can take the values 1 or 0. When an input is excitatory a weight $w = 1$ is associated to this input while if it is inhibitatory a weight $w = 0$ is applied. Thus, a weighted sum is then computed for all inputs (it corresponds to the soma). If this final sum is less than some value (which you decide, say τ), then the output is zero. Otherwise, the output is a one.

However, this model has serious limitations:

- it cannot process non-boolean inputs
- weights values are not optimized
- the threshold τ must be chosen by hand
- it implies that data are linearly separable

In 1957, Rosembaltt introduced a network composed of the units that were enhanced version of McCulloch-Pitts model [7]. The perceptron was born. This new model was the result of merger between two concepts from the 1940s, McCulloch-Pitts model of an artificial neuron and Hebbian learning rule of adjusting weights [3]. Changes from McCulloch and Pitts model included: processing of non-boolean inputs, automatic weights computation and introduction of a bias term.

Following this new idea, Minsky and Papert (1969) dedicated a book to the perceptron model [6]. This book has been published a decade after Rosenbaltt's model and the perceptron research had not yet produced a successful application in the real world. In fact, the most obvious application of the perceptron - computer vision - demands computing capabilities that far exceed what could be achieved with the technology of the 1960s. The book was widely interpreted as showing that neural networks are basically limited and fatally flawed. Following the publication of this book, research on perceptron-style learning machines remained unfashionable until the mid-1980s. Funding was no longer forthcoming. Compelled to change fields, perceptron researchers applied their unfashionable ideas wherever they went. This exodus helped spur the development of adaptive signal processing, which, ironically, was one of the earliest and most spectacular applications of learning techniques.

1.2 The aftermath of perceptron

Nowadays, the perceptron model is outdated since neural network architectures outperform traditional models in terms of accuracy. Neural network architectures (also known as "deep learning") are complex models that are theoretically supposed to approximate any function. This stated in the universal approximation theorem [4]:

> "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"

Four decades after Minsky and Papert's book, machine learning is a thriving research field. The effectiveness of deep neural networks for practical applications such as speech recognition and computer vision is undeniable. Cell phones have gained the capability to recognize speech and sometimes understand what we mean. Automobiles are increasingly capable of driving themselves. Although our theoretical understanding of learning machines has clearly progressed, these applications often rely on deep neural networks for which theory still offers very little guidance.

The following timeline give you an overview of the development of neural networks trough important discoveries for the field.

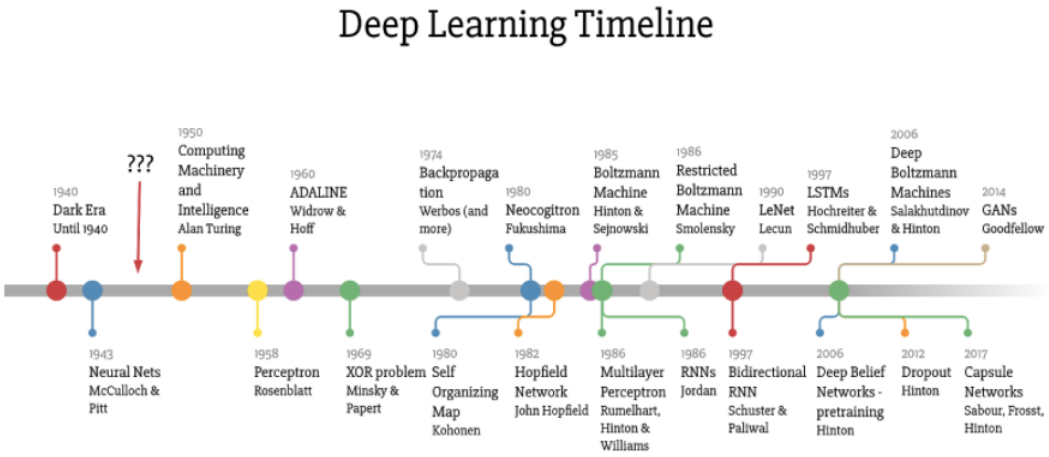
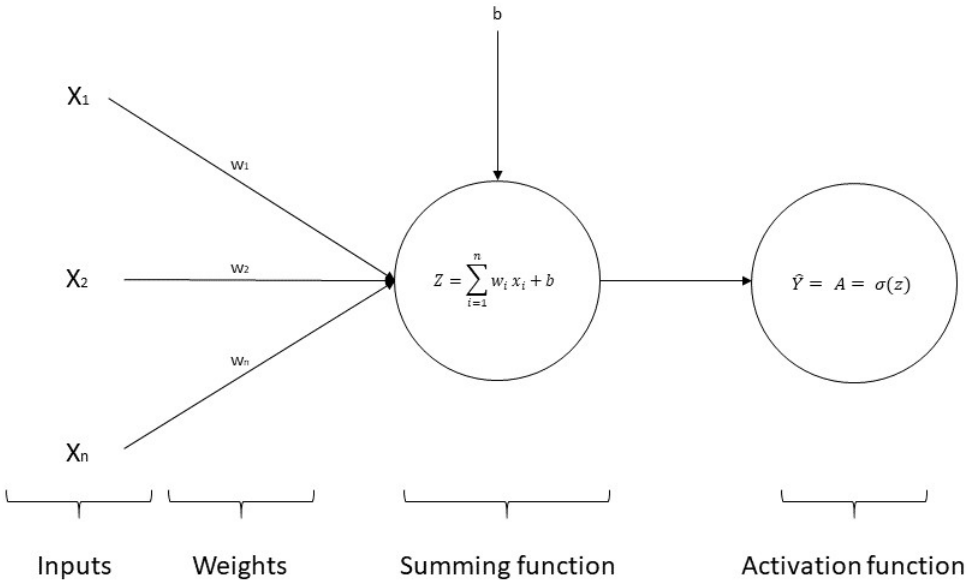


Figure 2: The deep learning timeline (source: Favio Vazquez)

2 Mathematical representation of the simple perceptron

2.1 The graphical model

The building blocks of today's perceptron formulation are summarized on the figure below:



- x_i: Input values -also called features - (training samples to learn).
- w_i: Weights (updated during training).
- Z: Weighted sum of inputs.
- A: Activation function (introduces non-linearities).

- $\mathcal{L}$: Loss function - not represented here - (minimized during training).

Several activation functions can be used to introduce non-linearity in the model. Below are some examples of popular activation functions:

$$\text{Sigmoid: } \sigma(x) = \frac{1}{1+e^{-x}}$$

$$\text{Hyperbolic tangent: } Tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}-e^{-x}}$$

$$\text{Rectified Linear Unit: } ReLU(x) = max(0,x)$$

Important note: in the special case of the sigmoid activation function, the perceptron becomes the logistic regression.

2.2 Optimization and training

In order to learn from the data, the model's parameters w_i associated to each features should be updated. Several optimisation strategies can be implemented but we will focus on gradient descent to simplify.

We denote $\hat{y}$ the predicted probabilities of the model which takes values in [0,1] and y the true labels whose values are {0,1}. We can now define the loss function of the logistic regression (the cross-entropy):

$$\mathcal{L}(\hat{y}, y) = -[y\log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

$$\begin{cases} \mathcal{L} = -\log(\hat{y}) & \text{if } y = 1 \\ \mathcal{L} = -\log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

Then, we define the cost function J which is a measure of how well the algorithm is doing on the entire dataset (all the samples are included).

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y^i}, y^i) = -\frac{1}{m}\sum_{i=1}^{m}[(y^i\log(\hat{y^i}) + (1-y^i)\log(1-\hat{y^i}))]$$

Now that we have defined our model, the question that arises is how to learn the parameters w_i? This question is equivalent to solve the following optimization program: $\underset{w,b}{\text{Min}}\ J(w,b)$.

The cost function for m examples can be rewritten as

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^i, y^i)$$

where $a^{(i)} = \hat{y^i} = \sigma(Z^i) = \sigma(w^T x^{(i)} + b)$
Then for updating the weights and bias we have to compute:

$$\frac{\partial J(w,b)}{\partial w_i} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial\mathcal{L}(a^{(i)}, y^{(i)})}{\partial w_i}$$

In fact, the learning process can be seen as a succession of forward and backward loop. In the forward phase, we compute the output based on the learned weights. In the backward phase, the error (difference between the predicted value and the actual output) is backpropagated inside the network in order to update the weights. This is done until convergence of weights. In our framework, the error to be backpropagated is mesured by the cross-entropy and we want the update the weigths according to this error. Thus we simply want to compute:

$$\frac{\partial \mathcal{L}}{w_i} \text{ and } \frac{\partial \mathcal{L}}{b}$$

Applying the chain rule, we have:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

$$\frac{\partial \mathcal{L}}{\partial a} = -[y\frac{\partial \log(a)}{\partial a} + (1-y)\frac{\partial \log(1-a)}{\partial a}] = -[y\frac{1}{a} + (1-y)\frac{1}{1-a}(-1)] \tag{1}$$

$$\frac{\partial a}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z)) = a(1-a) \tag{2}$$

$$\frac{\partial z}{\partial w} = \frac{\partial (wX+b)}{\partial w} = X \tag{3}$$

$$\frac{\partial z}{\partial b} = \frac{\partial (wX+b)}{\partial b} = 1 \tag{4}$$

Thus we have the following expressions:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w} &= [\frac{-y}{a}a(1-a) + (1-y)\frac{-1}{1-a}a(1-a)]X \\ &= -[y(1-a) - (1-y)a]X \\ &= (a-y)X\end{aligned}$$

Then we have the expression for the weights update:

$$w = w - \alpha(a-y)X \tag{5}$$
$$b = b - \alpha(a-y)1 \tag{6}$$

In pseudo-code here is what the computations should look like:

Algorithm 1 Gradient descent algorithm for loss minimization

Initialization: j=0, $dw_i = 0$, db = 0, b=0, w_i = random
for i =1 to m **do**
  $z^{(i)} = w^T x^{(i)} + b$
  $a^{(i)} = \sigma(z^i)$
  $J = -[y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})$
  $dz^{(i)} = a^{(i)} - y^{(i)}$
  $dw_i = dw_i + x^{(i)} dz^{(i)}$
  $db = db + dz^{(i)}$
end for
$J = J/m$
$dw_i = \frac{dw_i}{m}$
$db = \frac{db}{m}$
$w_i = w_i - \alpha dw_i$
$b = b - \alpha db$

The parameter α is called the learning rate and should be adjusted to control the speed of learning. α should be chosen carefully since a high value will cause the weights to diverge and a too small value will slow the learning process.

2.3 Choosing the appropriate weights initialization

Weights initialization is particularly a matter of concerns for deep learning architectures (when the model has multiple hidden layers) because it will determine how the gradient flows into the network. Depending on the activation function, you can use different weights initialization schemes which will speed up learning.

In the case of Tanh activation function we prefer Xavier initialization [1].

In the case of ReLU activation function, Kaiming initialisation is prefered [2].

2.4 Exercise: implementation of the perceptron

In this exercise we want to implement the perceptron model for stock prediction. We want to implement the special case of perceptron where the activation function is sigmoid.

2.4.1 First step: Data preparation

1. Create a "functions" module where you will place all the functions below. All sub must fill an array whose rows correspond to dates.
2. Returns: write a sub which computes the daily stock returns R_i.
3. Variable to predict
   (a) Write a sub which computes the forward return (i.e. R_{i+1}).
   (b) Apply the following rule to get your target $\begin{cases} Y = 1 & \text{if } R_{i+1} > 0 \\ Y = 0 & \text{if } R_{i+1} < 0 \end{cases}$
4. Simple moving average (SMA): write a sub which computes the rolling mean over 20 days.
5. Bollinger Bands: write a sub which computes the Bollinger bands
   (a) Compute the standard deviation of the SMA over 20 days.
   (b) Upper band: $SMA_{20} + 2 \times SMAStandardDeviation_{20}$.
   (c) Lower band: $SMA_{20} - 2 \times SMAStandardDeviation_{20}$.
6. In a "main" module retrieve the Google stock data and store them into an array. Run the above functions. You should end up with 3 features and your target.

2.4.2 Second step: Model Implementation

1. Implement a class SimplePerceptron with the following methods:
   (a) Initialization: initialize the weights array to random numbers.
   (b) Forward computation: take an input array of dimension 1 as argument.
   (c) Backward computation: take an input array of dimension 1 and true output as argument.
2. Set the default parameters to the following values: learning rate = 0.1, number of iterations = 20.
3. In a standard module, make the perceptron learn on one part of the data (say 80%) and forecast on the other part (20%).
4. Store the weights, bias and loss at each iteration.

2.4.3 Third step: Model testing and interpretation

1. Plot the graph of the loss at the end of the learning process. Did the perceptron converge?
2. Interpret the weights values.
3. Compute the confusion matrix. Is it a good model?
4. Add an hyperbolic tangent activation method to your perceptron class and run the code. Are the results better than using the sigmoid activation?
5. Implement the Glorot initialization method for hyperbolic tangent activation (see [1] for details). Are the results improved? Why?

3 Financial application

Along this section, you will have to put yourself in the shoes of a hedge fund quantitative analyst. That hedge fund is particularly famous for its ability to anticipate short-term stock fluctuations thanks to fine statistical modelling. More precisely, that hedge fund's performance is heavily driven by time-series analysis via econometric models. One day, your manager presents you a specific request: a new modelling framework called neural networks just got published by Yann Le Cun. You must study the profitability of such new methods and compare them to the benchmark AR-GARCH.

Here are some specifications provided by your manager:

- You will have to provide an Excel+VBA tool allowing the user to compare several models
- Your tool should showcase an AR-GARCH and an MLP.
- To put your models into perspective, proceed as follows for each one of them: 1) Prove that you coded it right (train the model on deterministically generated data then forecast on a new dataset generated the same way, precision should be around 100%). It is important for the user to be able to do this test from the click of a button on a separate spreadsheet. 2) Test it on real stock market data: train the model on the training set, test it on some other set, then get performance metrics. 3) Compare results with those from other models.
- Your tool should be user-friendly: a dashboard holding input interface (model metaparameters, buttons, ...) visual elements (graphs, ...) and key figures (performance metrics, backtest performance, ...). However, do not hide any spreadsheet nor other content.
- Your tool should run on any computer without setting issues. No third-party references/tools/libraries. If a Microsoft reference is to be added before use, notify it on your side report.
- **All relevant supplementary work will be considered.** In other words: creative students will be rewarded.

3.1 AR-GARCH model

We now focus on AR-GARCH as a benchmark model. As a reminder, GARCH means Generalized Autoregressive Conditional Heteroscedasticity. This AR-GARCH model is composed of two parts, one defining returns (AR) and the other the volatility (GARCH). It is defined as:

$$AR(n) - GARCH(p,q) : \left\{ \begin{array}{rcl} y_t & = & \sum_{i=1}^{n} \eta_i y_{t-i} + \sigma_t \varepsilon_t \\ \sigma_t^2 & = & \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2, \end{array} \right.$$

where y is the price return of an asset, σ its volatility, and ε a random variable supposed to be a standard Gaussian variable (ε is the innovation, $\sigma\varepsilon$ the residual). For this tutorial, we will use ARCH(n)-GARCH(1,1), so the model is finally defined as:

$$AR(n) - GARCH(1,1) : \left\{ \begin{array}{rcl} y_t & = & \sum_{i=1}^{n} \eta_i y_{t-i} + \sigma_t \varepsilon_t \\ \sigma_t^2 & = & \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2. \end{array} \right.$$

To estimate this model (that is to find the parameters relevant with the data) we need to calculate its likelihood. You will do this using directly Excel and its solver, then you will automatize it by recording a macro. Here is how you must build your sheet:

1. First column: price (you can use the dataset provided or download another price series of a stock or stock index, from abcbourse.com, for example).
2. Second column: price returns.
3. Third column: predicted returns (AR part of the model; **if the user changes the value of n, the formula must be changed accordingly automatically**).
4. Fourth column: residual of the AR part.
5. Fifth column: σ.
6. Sixth column: innovation.
7. Seventh column: log-likelihood.
8. A cell with the sum of all the log-likelihoods, to be minimized.
9. $4+n$ cells with the parameters of the model: initial σ^2, the three parameters of the GARCH regression and the n of the AR part. They are the variables of the optimization.
10. A cell with the empirical variance of the innovation, which should be close to 1 (a constraint of the optimization problem).

	A	B	C	D	E	F	G	H	I	J	K
1	Date	SMI	price return	predicted return	residual of the AR part	sigma	epsilon	Log-likelihood			
2	01/11/2016	7761,34									
3	02/11/2016	7700,41	-0,79%							sigma² 0	0,000100468
4	03/11/2016	7640,94	-0,77%								
5	04/11/2016	7593,2	-0,62%	-0,36%	-0,26%	0,010023	-0,26207	-0,95328		alpha 0	1,12415E-05
6	07/11/2016	7735,22	1,87%	-0,36%	2,23%	0,005997	3,712738	-7,81115		alpha 1	2,78E-04
7	08/11/2016	7744,03	0,11%	-0,29%	0,40%	0,06197	0,064877	-0,92104		beta 1	5,63E-02
8	09/11/2016	7897,84	1,99%	0,86%	1,12%	0,01512	0,743038	-1,19499			
9	10/11/2016	7928,77	0,39%	0,05%	0,34%	0,013319	0,254565	-0,95134		eta 1	1,85937E-05
10	11/11/2016	7880,29	-0,61%	0,92%	-1,53%	0,006263	-2,43917	-3,8937		eta 2	0,461242943
11	14/11/2016	7896,85	0,21%	0,18%	0,03%	0,040807	0,007234	-0,91896			
12	15/11/2016	7909,2	0,16%	-0,28%	0,44%	0,010247	0,427828	-1,01046		Log-likelihood	-709,039656
13	16/11/2016	7914,02	0,06%	0,10%	-0,04%	0,008244	-0,04365	-0,91989		variance of epsilon	0,949999771
14	17/11/2016	7964,68	0,64%	0,07%	0,57%	0,003949	1,438198	-1,95315			
15	18/11/2016	7904,55	-0,75%	0,03%	-0,78%	0,024215	-0,32338	-0,97123			
16	21/11/2016	7849,86	-0,69%	0,30%	-0,99%	0,008561	-1,15306	-1,58371			
17	22/11/2016	7741,82	-1,38%	-0,35%	-1,03%	0,019609	-0,52431	-1,05639			
18	23/11/2016	7752,24	0,13%	-0,32%	0,45%	0,01045	0,434188	-1,0132			
19	24/11/2016	7798,5	0,60%	-0,63%	1,23%	0,00835	1,474836	-2,00651			
20	25/11/2016	7881,53	1,06%	0,06%	1,00%	0,024881	0,402959	-1,00013			
21	28/11/2016	7823,23	-0,74%	0,28%	-1,01%	0,009549	-1,06294	-1,48386			
22	29/11/2016	7845,01	0,28%	0,49%	-0,21%	0,018168	-0,11706	-0,92579			

Figure 3: What your spreadsheet should look like.

Once this is made, use the macro recorder and define a function returning the optimal parametrization of this model for a time interval given as an argument of the function.

3.2 Artificial neural network

In Section 2, you already worked on the simple perceptron. You will now combine several perceptrons to create multi-layer perceptrons (MLP)

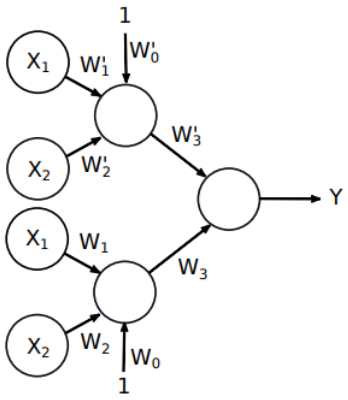


Figure 4: Example of MLP.

Several architectures are possible. To keep things simple, we suppose that, for n inputs, we have p hidden layers of n nodes each, and then an output layer with a single node. Between each pair of adjacent layers, n^2 weight parameters have to be estimated. You can store these weights in an array of dimension (1 to n, 1 to n, 1 to p), in which $w_{i,j,k}$ provides you with the weight linking the output i of layer $(k-1)$, noted $z_{i,k}$, to the node j of layer k. In other words, $z_{i,k+1} = f(\sum_{l=1}^{n} w_{l,i,k} z_{l,k})$, with f the activation function. The layer 0 is of course the input itself and $z_{i,1} = x_i$. In addition to this array, you have to add a one-dimensional array containing the weights between the last hidden layer and the output node, noted $w_{i,1,p+1}$.

The iterative update of the weights of the last layer can be made accordingly to the method explained in section 3 (to simplify, you can consider a null bias b). You will use backpropagation to update the weights of the other layers: begin with the layers the closer to the output, and go backward:

- For the output layer, define e, which is the sensitivity of the error, at the level of the output node, to the
- For the node i of the hidden layer k (make a loop for k decreasing from p to 1), the sensitivity of the output error to its input is now $e_{i,k} = f'(\sum_{l=1}^{n} w_{l,i,k} z_{l,k}) \sum_j w_{i,j,k+1} e_{j,k+1}$, where $e_{j,p+1)}$ is simply e.
- After having calculated the sensitivity for each node of each layer, update all the weights, using $w_{i,j,k)} \leftarrow w_{i,j,k} - \alpha z_{i,k} e_{i,k}$.

Create a new class for this multi-layer perceptron. Make your network learn on one part of the sample and then forecast on the other part.

3.3 Comparison

Determine the confusion matrix for the three models: AR-GARCH, simple perceptron, MLP and comment them.

Several metaparameters must be defined by the user:

- the first and last dates of the learning sample,
- the first and last dates of the testing sample,
- the number of lags taken into account in the AR part of the AR-GARCH model or in the ANN,

- parameters used for the learning rule of the ANN.

Create a userform in which the user will indicate the desired metaparmaters. Make the calculation accordingly to this input and display the results.

4 Evaluation

Here is how the project will be marked:

- Excel (/8):
  - Overall user experience /2
  - AR-GARCH implementation (including the automatic correction of the in-cell formulas when the user changes n) /3
  - Quality of the empirical test of each model, and of the comparison userform/interface /3
- VBA (/10):
  - Object-oriented simple perceptron implementation /4
  - Object-oriented MLP implementation /4
  - Code organisation, interpretability and cleanness /2
- Side report (/2)
- Extras

References

[1] Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks. 2010.

[2] He K., Zhang X., Ren S., Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

[3] Hebb, D. O. The organization of behavior. *Journal of clinical psychology*, page 335.

[4] Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[5] McCulloch W. S., Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[6] Minsky M., Papert S. Perceptrons : An introduction to computational geometry. 1969.

[7] Rosenbaltt F. The perceptron: a perceiving and recognizing automaton. 1957.