VBA for finance – ESILV 2018-2019

Neural network

You will be evaluated on this project. You will work in groups of two students of the same TD group. Before the end of Thursday 27th December, you will send your Excel file as well as any necessary information to your TD teacher.

In this tutorial, we will build a neural network for forecasting financial time series. You will compare the predicting abilities of this model with an AR-GARCH. This tutorial is based on this research article, which I invite you to consult:
https://pdfs.semanticscholar.org/87d3/eb3174f93abb1822343798084a50c3d18bff.pdf

I. AR-GARCH

In this first part, we will focus on AR-GARCH. 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:

$$AR(n) - GARCH(p,q): \begin{cases} y_t = \sum_{i=1}^{n} \eta_i y_{t-i} + \sigma_t \varepsilon_t \\ \sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-i}^2}, \end{cases}$$

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:

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

To estimate this model 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 download prices 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 minmized.
9) 4+n cells with the parameters of the model: intial 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			

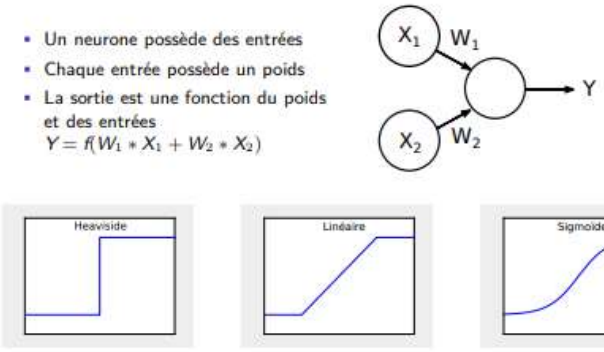
Once this is made, use the macro recorder and define a function returning the optimal parametrization of this model for a time interval given as an argument of the function.

II. Neural Network

In this second part, we will focus on ANN (Artificial Neural Network). A brief overview of these kind of methods is provided in https://www.labri.fr/perso/nrougier/downloads/Perceptron.pdf

II-1. Simple perceptron

You will begin with a simple perceptron (where X1 and X2 are, in our framework, price returns in t-1 and t-2 and Y the forecast by the ANN of the price return in t):



For the activation function (or transfer function), you will use the most widespread one, that is the sigmoid function:

$$f(x) = \frac{1}{1 + \exp(-x)}$$

The intial choice for the weights, w1 and w2, is provided randomly.

The update of the weights after each example learned is made accordingly to the following gradient-descent algorithm, in which you use a fixed (or decreasing) learning rate (a good intial value : 0.5).

In detail, for a given time, the output of the network is compared to the true price return. Then, the weight is corrected in the following way (which incorporates the gradient in the case of a sigmoid transfer function): $w_i \leftarrow w_i - \sigma x_i e$, where $e = (y - s)y(1 - y)$ is the derivative of the error regarding any weighted input $x_i w_i$ (in other words, it is the sensitivity of the error, at the level of the output node, to the contribution of a single neuron), wi is the weight, of the neuron i, xi is its input, y the output, s the correct value, and σ the learning rate. The global error is roughly (it is a linear approximation) $\sum_i x_i w_i e$.

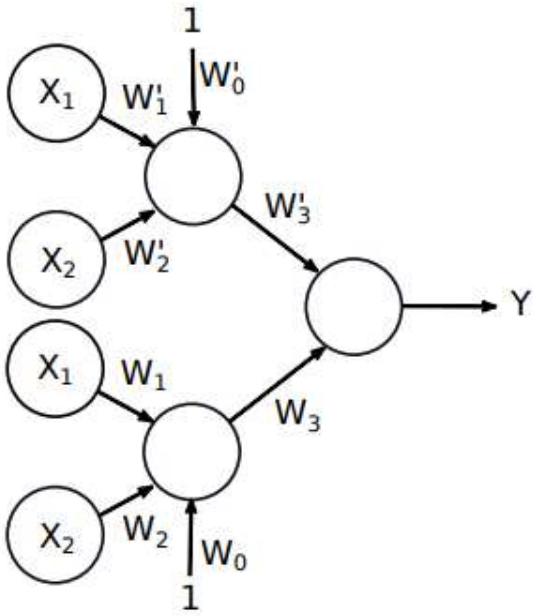
What we expect from you in VBA:

- Create a class SimplePerceptron, with:
  - an attribute containing all the weights, stored in an array of dimension 1,
  - a method output, with a given input (one-dimension array) as argument,
  - a method updateWeights, with a given input and expected output as argument.
- From a standard module (not the class module), make your network learn on one part of the sample and then forecast on the other part (the dates defining the learning sample and the testing sample are arguments of your sub).

Important remark about the scale of the parameters: The output is between 0 and 1, whereas a return should be between -100% and infinity. Therefore, the output must be translated in terms of returns.
1/ It can be done by applying a transform taking into account the minimal and maximal return reached in the sample (y(M-m)+m). The corresponding inverse transform could also be applied to the input so that we only work in the ANN with variables between 0 and 1.
2/ Alternatively, we can replace the activation function by the identity only in the last layer for a multilayer perceptron (next section).

II-2. Multi-layer perceptron

The multi-layer perceptron is made of several simple perceptrons:



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, n² weights have to be estimated. You can store these weights in an array (1 to n, 1 to n, 1 to p), in which $\boldsymbol{w_{i,j,k}}$ **provides you with the weight linking the output i of layer (k-1), noted $\boldsymbol{z_{i,k}}$, to the node j of layer k**. In other words, $z_{i,k+1} = f\left(\sum_{l=1}^{n} w_{l,i,k} z_{l,k}\right)$. The layer 0 is of course the input itself and $z_{i,1} = x_i$. It addition to this array, you have to add a one-dimensionnal array containing the weights between the last hidden layer and the output node, noted $w_{i,1,p+1}$.

The update of the weights of the last layer is made accordingly to the method explained in the previous section. 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 contribution of a single neuron.
- 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, and $f'(x) = f(x)(1 - f(x))$.
- After having calculated the sensitivity for each node of each layer, update all the weights, using $w_{i,j,k} \leftarrow w_{i,j,k} - \sigma z_{i,k} e_{i,k}$.

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.

II.3 Alternative learning rules

The backpropagation explained above can be seen as a standard learning rule. It can be ameliorated with the following methods:

- Momentum: the standard learning rule is mitigated with a stationary rule. A new parameter $\lambda \in [0,1]$ determines to which extent the weights should be updated and to which extent the previous change of weight should be used again: $\Delta w_{i,j,k}(t) = -\lambda\sigma z_{i,k-1}(t)e_{i,k}(t) + (1-\lambda)\Delta w_{i,j,k}(t-1)$.
- Non-constant learning rates.

Implement these two ameliorations.

III. Comparison

Compute the several accuracy functions defined in the article (R^2, MPE,MSRE). Define also the P&L of a strategy which consists in buying when the forecast return is positive and selling else.

Create a result worksheet for each model (GARCH, simple perceptron, multi-layer perceptron) and analyse your results.

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.