Machine Learning Project : Forecasting Electricity Consumption

ALAOUI, Mohammed Amine BELKHAYAT ZOUGARI, Younes
FELTIN, Clément LETOURNEL, Benoit

1er février 2017

1 Introduction

Neural networks have been used in the past decades in a variety of problems due to the increasing computing capabilities on the one hand and to the extensive work that have been done to explore the reach of such techniques on the other. Notable examples include natural language processing and image/video processing, where neural network based solution stand undefeated, especially when combined with clever pre-processing of data.

Nonetheless, neural networks, aside from their great feats, come with a few drawbacks of their own, amongst them lies the 'black box' aspect. As a matter of fact it is difficult to interpret what a neural network is trying to do with the data, the more so as its architecture is complex. Hence, many attempts at combining neural optimization with other more transparent techniques in order to get the best of the two worlds : easy to devise and understand systems. In this report we explore a combination of neural network and fuzzy rules. These latter have been widely used in control systems, and have the advantage of being directly understandable by a human operator as they consist of IF/THEN rules that deal with "fuzzy" facts. A fuzzy fact can be represented with a set $\mathcal{X}$ of which the membership function is a function $f : x \in \mathcal{X} \rightarrow [0, 1]$ such taht $f(x)$ expresses the possibility of $\mathcal{X}$ containing x. The architecture of the neural network is learned from the data which give this method the name of Self organizing fuzzy neural networks.

SOFNN is compared to standard MLPs which parameters are selected using a clever strategy : Genetic optimization over a population of algorithms . The comparison is performed over forecasting problems ranging from toy problems to a real one : that of energy consumption. ARIMA/SARIMA modeling is used as a baseline.

RTE (French Electricity Network) has released historical electricity consumption data, power generation data by energy source and the balance of electricity flows between regions. We will work on a time series from jan-2013 to dec-2015. The information of consumption is given every 30min, representing a total of 56924 timestamps.

2 Using ARIMA

Introduction

ARIMA (stands for Auto Regressive Integrated Moving Average) is a model widely used for forecasting time series. It has the advantage of simplicity and efficiency.

As we can see in its name, ARIMA is constituted of three parts (AR + I + MA).

— the Auto Regressive part exploits the correlation between an observation and a set of previous observations. The lag between the first and last observation is the order of the AR part and is denoted p.

— ARIMA is integrated because it helps stationnarize the time series by differentiating the time series data (as we will see later), we note d the degree of differentiating.

— the Moving Average aims for expressing the correlation between an observation and the error residual from a moving average of previous observations. The size of the window of the moving average is noted q.

We take benefit of this moment to point out to a good website that tackles the forecasting issues : http ://people.duke.edu/ rnau/411home.htm.

Our principal challenge is to find the adequate parameters (p, d, q) of the model that sticks the best to our data.

Our work will be divided into five parts

— Data visualization

— Time series stationnarization

— Autocorrelation Function & Partial Autocorrelation Function

— ARIMA Model Building

— Making predictions

2.1 Data Visualization

The first step to do at the beginning of every machine learning problem is to visualize the data. In our case (forecasting time series), the step aims to get to know the data, to see if there is any seasonality, trend etc. We obtain the following curve :

There are direct hypothesis that we can make on the time series once we've seen their allure :

— The data are periodic : with a period nearly (if not exactly) equal to 1 year.

— The data are stationary : there doesn't seem to be any special trend followed by data, and they appear to be in a permanent regime, the statistical specifications don't change during the time.

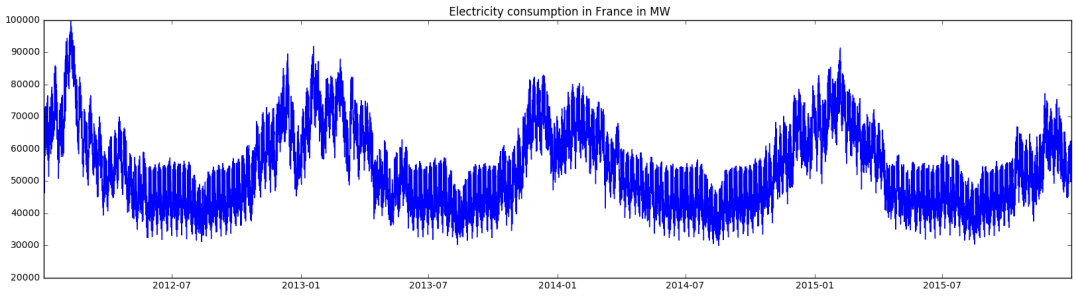

FIGURE 1 – Time series to forecast

2.2 Periodicity and Stationarity of the Time Series

2.2.1 Periodicity

Smoothing
Since our data know a lot of variations between indexes, it is a bit hard to detect the frequency of the time series. Therefore, we plotted a rolling mean with a 1-day window, we get a smooth curve, and manually, we would say that the frequency is equal to 1800 samples. We need a more practical way to find it more precisely, that's why we thought of the Fourier transformation (or even better : the Fast Fourier Transformation), that has the ability to give us the spectrum of the data. We will use a similar way for doing so through the PSD (Power Spectral Density).

Periodogram
In order to determine the periodicity of the signal, we plotted its periodogram (which expresses the Power Spectral Density). Thanks to the periodogram we were able to find the exact value of the frequency (which was equal to $1.02 year^{-1}$) so we got the period (17172 samples of 30 minutes)

2.2.2 Stationarity

Visually, we could notice that our data are already stationary (statistical properties don't change through time). There is a statistical way to be sure of our presumption : the Dickey-Fuller test. The hypothesis test takes a time series in argument, and returns a statistic test and critical values.

By applying this test, we obtain a statistic test equal to -7.95 whereas the critical value 1% is equal to -2.57, so we are 99% sure that the data are stationary.

We tried to differentiate the data, which is believed to increase the stationarity of time series. Indeed, the statistic test for the first differential is equal to -32.92, and the second differential is equal to -59.58.

We will chose the first degree as differentiating order.

2.3 Autocorrelation and Partial Autocorrelation

These two functions provides us with necessary information that helps us determining the orders for the ARIMA model.

To do so, we have to plot the auto-correlation function and partial auto-correlation function, and then, plot the confidence interval and choose p as the indice of the first time that the partial auto-correlation function crosses this interval, and q as the indice of the first time the auto-correlation function crosses the confidence interval.

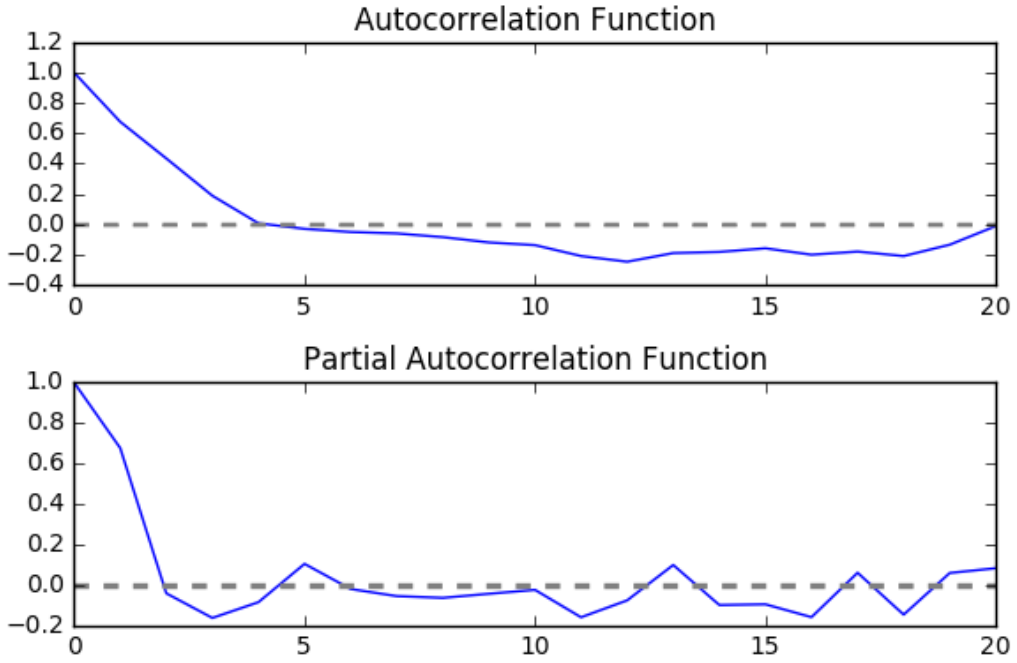

FIGURE 2 – Auto-correlation and Partial Auto-correlation Functions

Through the result shown on the figure 2, we choose $p = 2$ and $q = 4$ as parameters of the ARIMA model.

2.4 Building the model

Once the parameters are fixed, we had been able to build the ARIMA model thanks to `ARIMA` from `statsmodels.tsa.arima_model`. First, we tried with a simple `AR(2)` model, then we compare the fitted values with the differential of the original signal (since we chose the degree 1 for differentiation). Then, we did the same with a simple `MA(24)` model. At last, we defined the combined ARIMA model, and the comparison of the fitted values with the time series differential gives the results shown in the figure 3

In order to get to the original time series, we should integrate the fitted values, as we have done in figure 4. We notice that the signal and its fitted values have the same form (modulo same little modifications). So, after normalizing and putting the fitted values to the same mean as the original data, we obtain the results shown in figure 5.

We obtained a training error of 13% by using this method.

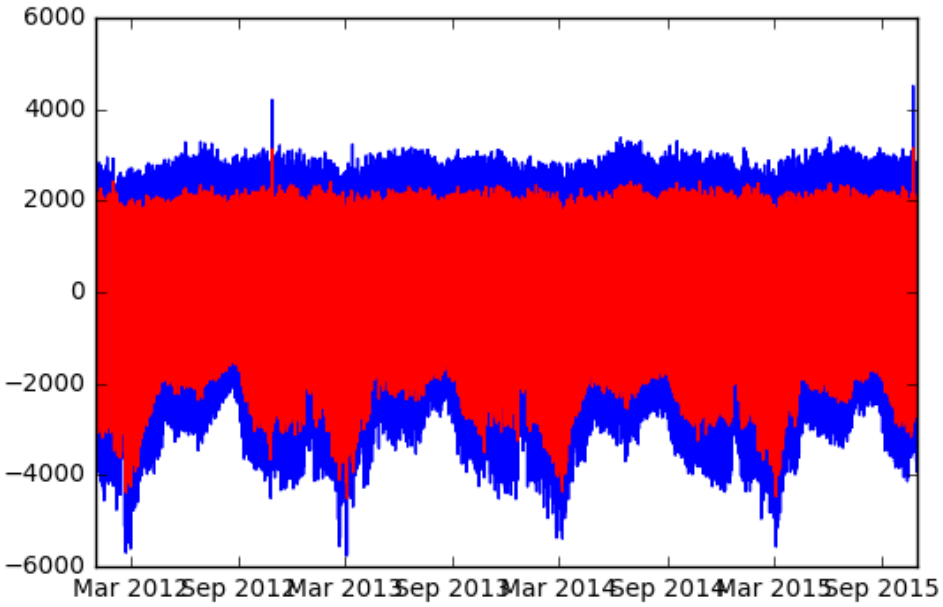

FIGURE 3 – Fitted values (red) and Observations (blue)

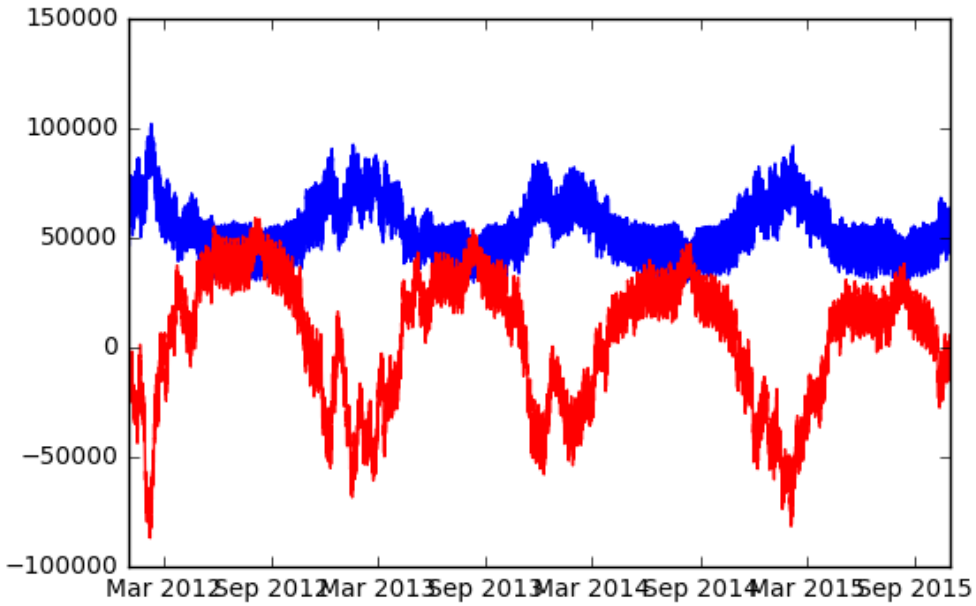

FIGURE 4 – Original Signal (blue) and Integration of fitted value (red)

2.5 Forecasting the Time Series

First, we tried to predict just the one next value. So, for $nb\_train = 15$, we trained the model, forecast a value, then add the real value and train again for the next value in the list. By doing

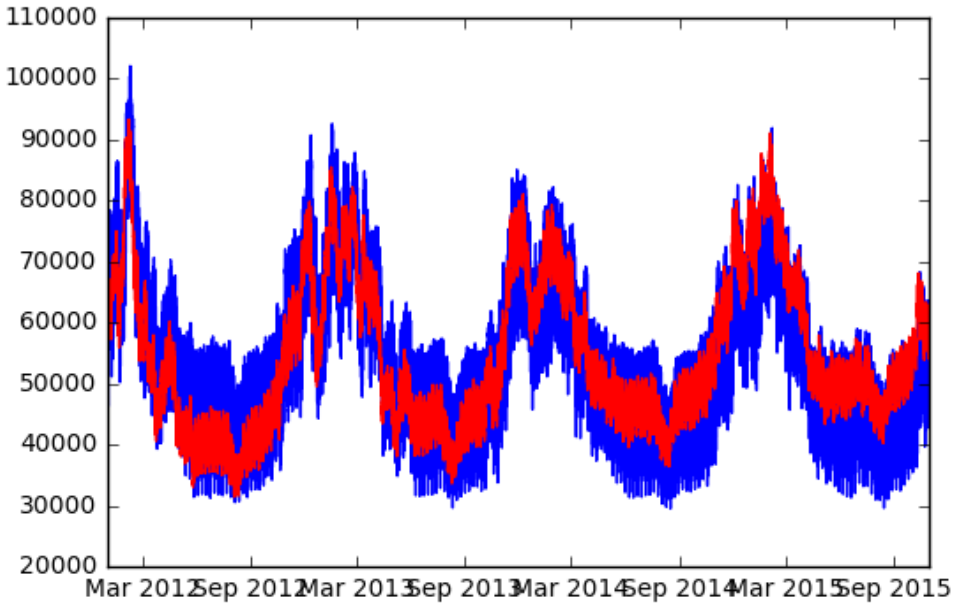

FIGURE 5 – Original Signal (blue) and Modified Integration of fitted value (red)

so, we got a prediction error of 5%.

But once we want to predict more values, we directly find ourselves far from the observations, as we can see in the figure 6

Secondly, we did the same as before, but we train on a dynamic data set that we aliment at each iteration with the last predicted value, and we predict the next one, we get a prediction error of 1.31% on 15 consecutive values.

The ARIMA gives good prediction for values which are coming soon, but very bad prediction (12% mean error) for the rest. Let's try another approach.

2.6 A second approach

Since we had troubles predicting the following values with a simple ARIMA model, another lead was to use SARIMA (or Seasonal ARIMA), that exploits the seasonality (periodicity) of the data to give more satisfying results.

The issue with SARIMAX was that it is a sublibrary of statsmodel that appeared in the 0.8 version, which is not a release version (the last release version available on pip and conda is statsmodels 0.6). Therefore, we installed it from the source code by downloading it for its GitHub repository, and install it through `setup.py`.

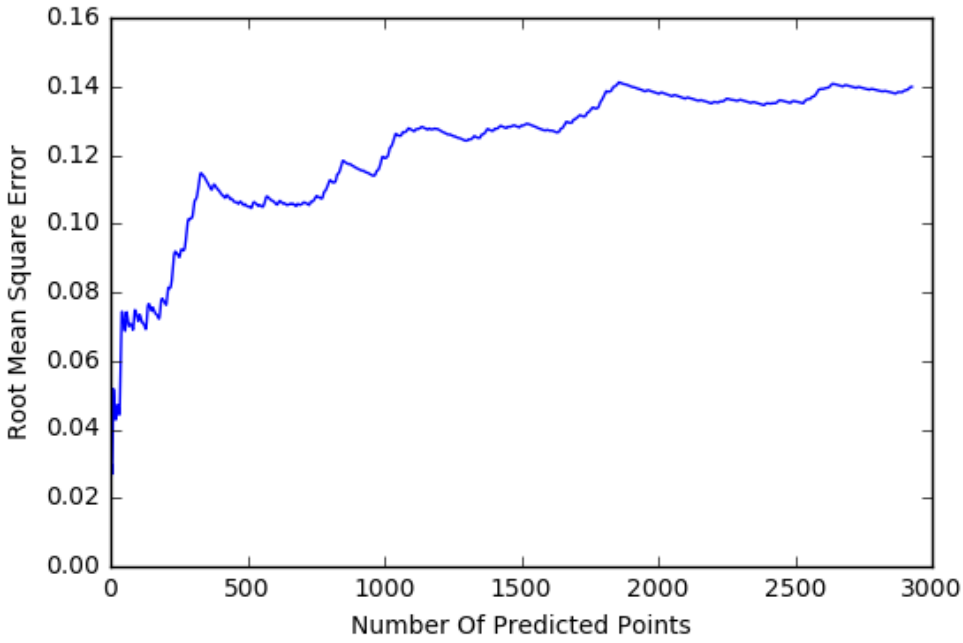

FIGURE 6 – The RMSE function of the number of predicted points

In addition to the information given to ARIMA (concerning the differentiation order, the parameters of AR and MA), SARIMAX takes in arguments seasonal information (seasonal parameter of AR and MA and seasonal parameter of differenciating, the number of periods in 1 year), and also the trend (which in our case is constant).

We got some problems with SARIMAX since there wasn't a convergence for the parameters chosen, so we did some tuning and still, the AIC (Akaike Information Criterion) is really large (10^6), and even if there is convergence, the prediction is not satisfying at all.

2.7 Conclusion on ARIMA

In conclusion, we were able to predict some values with ARIMA (even good results when it comes to the first elements of the test set), but once we got a little far in the test set, the result began to diverge from the observation, it is a good short term predictor, a bad long term one.

SARIMAX was a natural development of ARIMA for periodic trends, but the choice of the parameters wasn't easy nor accurate.

What should we do then ? We could have tried to do some signal processing on the data before the training such as the subsampling, since we have a very large data set, the ARMA model couldn't bear a large q value, maybe with subsampling, the results would have been better. We will try next with some more sophisticated algorithms.

For more information about the used methods, please refer to [1][2][4]

3 Genetic Algorithm for MLP

Introduction

As ARIMA models are quite popular for time series prediction, we wanted to benchmark the performance of ARIMA models compared to a Multi-Layer Perceptron (MLP). Similarly to ARIMA models, one main factor for success for a good forecasting is the correct choice of the time lags identified for the first layer of the perceptron. In this part, we will study the implications of a Genetic Algorithm, applied to time series forecasting. Inspired by the biological process of natural evolution, the Genetic Algorithm (GA) rely on cross-over and mutation, similar to biology evolution, to optimize the parameters of our problems. Cross-over and mutations are set to explore the space of parameters and will work as a cross-validation pipeline to select optimal parameters.

3.1 The Genetic Algorithm

The Genetic Algorithm is derived from evolution process in biology. In this algorithm, a population of individuals forms the potential solutions to our optimization problem. This population evolves toward a better solution after each iteration of the algorithm.

Genetic representation

Our phenotype for each individual will be composed of only two parameters of our MLP, the number of neurons in the input and hidden layers. The number of neurons in the input layer is equal to the time lag of our feature engineering.

— Each phenotype/individual is defined by the parameters or chromosomes : {'time_lags' : 63, 'hidden_units' : 70}

— Fitness function : the performance of each individual will be defined by the fitness function :

$$Fitness = \frac{1}{1 + MSE}$$

Parameters of the GA algorithm

— Population_size : size of the population at each generations. Compared to human population, the size is not growing in our algorithm

— Max_generations : the GA stops after the max_generations is reached

— Crossover_weights : parameter used for the cross-over operation

— Params_minmax = {'time_lags' : [1, 100], 'hidden_units' : [1, 100]} : minimum and maximum space in which each cross-over and mutation can explore.

Fixed parameters for each MLP

— Nb_epoch : number max of epochs for the forward-backward

— Batch_size : batch size of the MLP. In our case, the number of data for training does not fit in memory or is very slow for each update. We decide to use mini-batch descent to optimize the MLP For ease of implementation, we used Keras to implement our MLP with the code below :

```
model = Sequential()
model.add(Dense(hidden_units, input_dim= time_lags, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(data_train, labels_train, nb_epoch=100, batch_size=32)
```

3.2 Procedure for the GA

1. Initialize the population : the initial population is generated randomly (within params_minmax range), allowing the entire range of possible solutions to be explored
2. Selection : two chromosomes are selected randomly to perform a cross-over, that is to say a mix of parents phenotype so as to generate a new population.
3. Genetic operations are composed of two steps :
   - Cross-Over : two parents are producing four offspring with the equations below :
   - Mutation : from the generated Offspring, mutations can occur, generating three new chromosomes. While Cross-over keeps the offspring in the range of their parents, mutation can change radically the solution.
4. Natural selection : Each of the seven Offspring Chromosomes are then evaluated. The best cross-over offspring replaces the worst individual in the population. Then each of the three mutations have a chance to be accepted to the population regardless of its fitness. Otherwise, it is accepted, if and only if its fitness is greater than the worst individual in the population.

Details on the GA algorithm can be found on the paper : **A new intelligent system methodology for Time Series Forecasting with Artificial Neural Networks**, Ferreira, Vasconcelos, Adeodato [3].

3.3 Results

3.3.1 Statements after implementation

After having implemented the GA algorithm, We found out that the time required to run it was extremely high on our machines. For each phenotype of cross-over and mutation, a new neural network need to be trained. The number of networks to be trained is equal to $initiale\_popu\_size + (4cross\_over + 3mutations) * Max\_generations$. To make things possible, we decided to cut the training set size to one month instead of two years and each individual network has to be stopped after N_epoch, before it converges completely. Not converging completely creates a bias in our method as all weights are initialized randomly. The fitness reached after N_epoch epochs is highly dependent on initialization. If the frame of the network was fixed, we could fix the seed from which the weights are randomized but in our case, the size of the MLP varied from on individual to another.

3.3.2 Results

Overall, the GA algorithm tends to select the best phenotypes from cross-over and mutation to improve the global fitness of the population. After running the algorithm on a 20 individual population for 20 generations, we can see the global fitness is increasing reaching 0.998 fitness.

Secondly, we can notice from figures 8 and 9 that, globally, the optimized phenotypes after 20 generations have less hidden units than from the initial population. The neural network is more robust on the test set with fewer hidden layers.

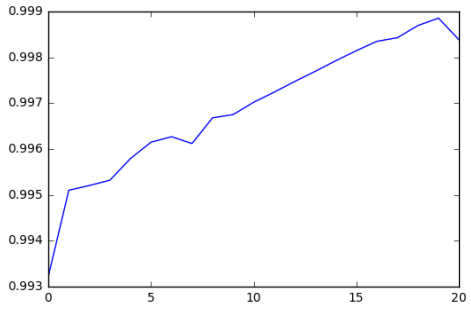

FIGURE 7 – Population fitness per generation

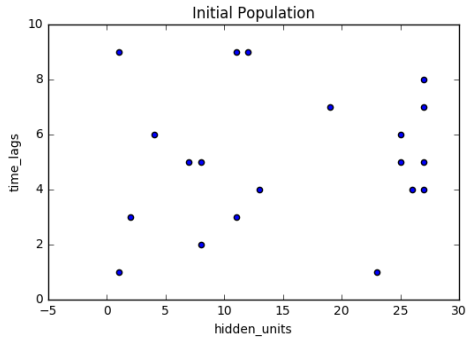

FIGURE 8 – Parameters of initial randomized population

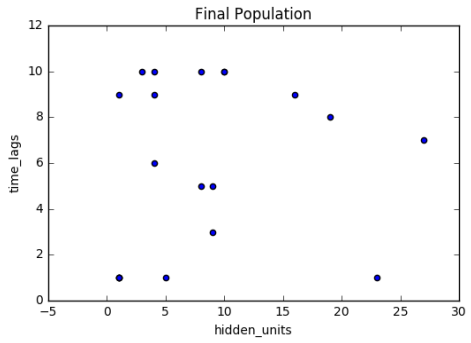

FIGURE 9 – Parameters of final optimized population

3.4 Conclusion on GA applied to MLP

The Genetic Algorithm show good results or at least improving results over time. Nevertheless, we can point out that, from the limitation of our machine, we could not make MLP converge

entirely for all individuals in all generations. As a consequence, final score of each MLP could be highly correlated to initialization. One final result pointed out that individuals with smaller hidden layer were chosen over higher layers. This statement can be false as the GA could select mainly the "simplest" individuals who converges quickly on first iterations.

4 Self Organised Fuzzy Neural Networks (SOFNN)

4.1 Setting

The idea behind SOFNN [5] is to use a neural network to learn the parameters of a fuzzy rule system, grossly speaking. As a matter of fact while fuzzy systems are straight forward to interpret they are not as easy to devise and often need deep expert understanding of the field of application in other to be performing. Using the capabilities of a neural network enables the automation of the design process so as to alleviate the involvement of the expert. The result is a system easy to design and to understand. Designing a FNN often goes through the choice of membership functions their parameters, and the way they're combined.

Before tackling the SOFNN algorithms let us discuss first in slightly more details fuzzy logic. As stated before it can be deemed as an extension of classical logic where we are allowed to be uncertain about a set containing a variable, in other terms going from a binary membership function to a continuous one. Fuzzy logic also utilizes an extension of logic operators (conjunction and disjunction) via functions on $\mathcal{X} \times \mathcal{X} \to [0, 1]$ called t-norms and t-conorms. There are many ways to define these latter, each resulting in different semantics. Such technical details are out of the scope of this project, so we restrict ourselves to the extensions used in the SOFNN of interest : The Gaussian membership functions and the product t-norm.

The SOFNN is as described by fig. 11. The data is fed to a series of membership functions that are combined into several rules. The rules are then weighed to compute the results. Formally, assuming that the aim is to predict a variable y from a set of variables $x_1, \cdots x_n$, each x_i is fed to a series of Gaussian membership functions $G_{i,j}(x_i) = \exp(-(x_i - c_{i,j})^2/\sigma_{i,j})$. these latter are combined into rules $R_1, \cdots, R_L$ using the product t-norm : $R_j(x) = \prod_i G_{i,j}(x_i)$.

The main step of the SOFNN algorithm is to adapt the structure of the neural network and its initial parameters to the training data. This is done by a competitive clustering technique which is detailed in the next section.

4.2 Adapting and training the FNN

The structure is learned using a clustering algorithm upon the data (including the labels). The resulting clusters should contain a considerable proportion of the data (more than a certain threshold parameter). It is based on rival penalized competitive learning RPCL, alg 1 shows the detailed steps.

The resulting centers are used to both determine the FNN architecture, and to initialize its parameters. The number of clusters gives the number fuzzy rules to be used in the FNN. Each rule computes the product of p membership functions, p being the number of input nodes. Let $w_k =$

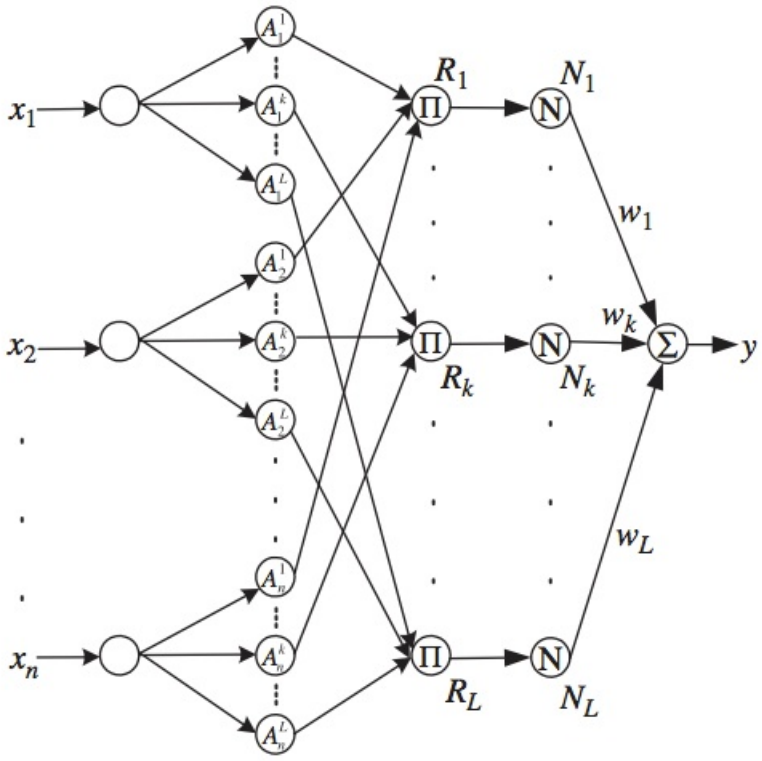

FIGURE 10 – Structure of the SOFNN
A_i^j are the membership functions, R_i the fuzzy rules (using the product t-norm here), N_i is normalization of $R_i : N_i = R_i / \sum_j R_j$

$(w_{k,1}, \cdots, w_{k,p}, w_{k,p+1})$ be the center of the ith cluster. The means of the membership functions connected to the rule R_i $c_{i,k}, k = 1, \cdots, p$ are set to $w_{k,i}$. The standard deviations are given by $\|w_{k,i} - r_{k,u}\|/r$ with r an overlap parameter. The weight of the connection linking the rule R_i to the output node are set to $w_{k,p+1}$.

The resulting neural network is then trained via a standard gradient descent. The parameters to optimize are of course the weights of the connections to the output node, but also the means and standard deviations of each membership function.

We believe that the competitive clustering algorithm can be replaced with a more sophisticated clustering algorithm that could lead to better performances. As a matter of fact, using RPCL relies heavily only on 2-NN without dimensionality reduction, which can be of important impact in the case of forecasting. Spectral clustering for example can be a better alternative but that remains to be tested.

4.3 Feature Engineering & Training

The SOFNN takes a time series $X = (x_{t=t_0}, ..., x_{t=t_0+n\tau})$ as input for training, and transforms it into a matrix defined by $[X_0, ..., X_{N_t}]$ where each X_i is a column vector defined by $X_i = (x_{t=t_0-i\tau}, ..., x_{t=t_0+(n-i)\tau})$ where τ is the sampling period and N_t is an hyperparameter,

Algorithm 1: RPCL

Function CompetitiveClustering;
Input : Initial number of clusters L_0, threshold ϵ, max iterations T, rival learning rate α_r, learning rate α_w, convergence parameter H, data X
Output: a set centers of clusters C

$h \leftarrow 0$;
$C \leftarrow L_0$ random samples from X **for** $c \in C$ **do**
  $W_c \leftarrow 1$;
end
while $h < H$ **do**
  while $t < T$ **do**
    $t \leftarrow 0$;
    $L \leftarrow$ number of centers in C ;
    $z \leftarrow$ random sample from X ;
    $r \leftarrow$ neirest neighbor of z ;
    $w \leftarrow$ second neirest neighbor of z ;
    $w \leftarrow w + \alpha_w(1 - 1/T)(z - w)$;
    $r \leftarrow r - \alpha_r(1 - 1/T)(z - r)$;
    $W_c \leftarrow W_c + 1$;
  end
  assign each sample to its nearest center ;
  remove centers for which $W_c / \sum_{c'} W'_c < \epsilon$;
  if *no center was removed* **then**
    $h \leftarrow h + 1$;
  end
end

representing the number of samples used for computing a prediction.

4.3.1 Choice of N_t

Instinctively, we wish we could have used high values of N_t to be able to make longer term forecasting, but it made training time explode, and it also increased the appearance of saturation non-linearities in the output. Therefore, we decided to limit the scope of our forecasts to $N_t = 3$.

4.3.2 Choice of the learning rate

The choice of the learning rate is a critical part of the hyperparameter selection. Indeed, having a too high value of the learning rate lead to the presence of NaN-valued weights after training, which makes prediction impossible. On the other hand, a low valued learning rate can significantly increase training time.

4.4 Performance of SOFNN forecasting

After tuning the hyperparameters of our SOFNN, we eventually managed to achieve satisfactory time series prediction tasks, with RMSE around 3%, which tends to prove the efficiency of this method.

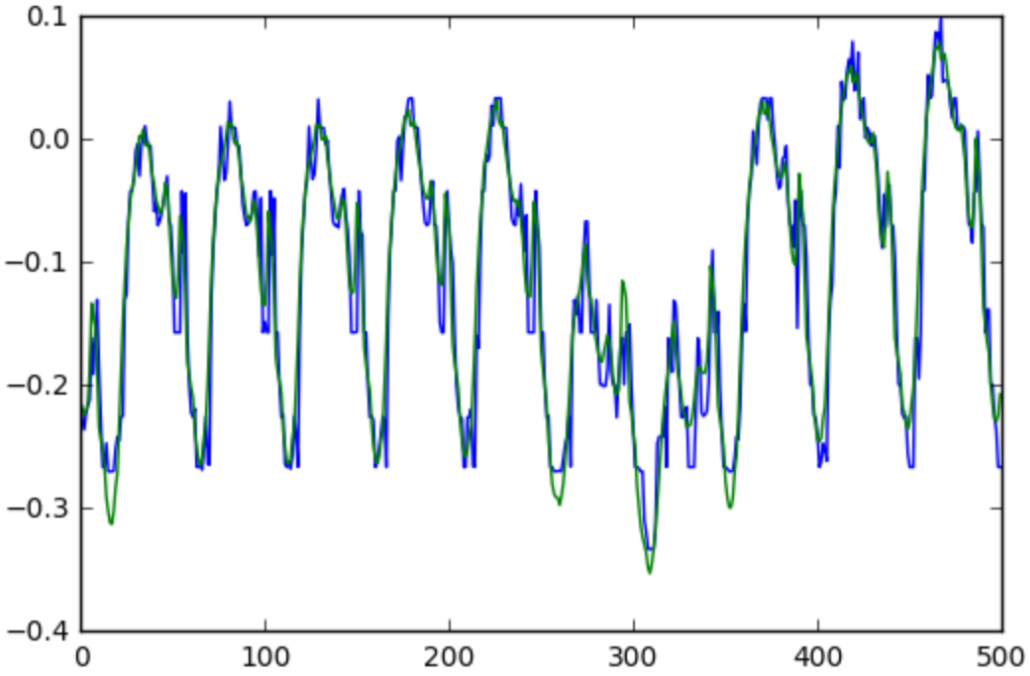

FIGURE 11 – Forecasted Time Series (in blue) compared to the label (in green)

5 Conclusion

This project has presented three different approaches for time series forecasting. The first model uses auto-regressive models (ARIMA and SARIMA). The second approach implements a greedy Genetic Algorithm, derived from natural evolution process, on Multi-Layer Perceptrons. The last approach tested self-organizing fuzzy neural networks (SOFNN).

Overall results were hardly comparable as some methods could not converge entirely due to time machine limitation.

Results of the GA were seemed pretty promising and ensure an overall increased performance over time. Nevertheless, this method is extremely costly for bigger data sets and should be kept if one would want to explore some extreme parameters values.

Also, the SOFNN forecasting tends to give encouraging results and is indeed less computation heavy than meta optimization approaches over MLPS, However, the parameters of the clustering part need to be well tuned in order to render something exploitable. With too much rules the learned function is not smooth and often takes the form of a saturated signal. With too little rules the target function remains out of reach. We have yet to investigate the impact of the clustering methods on the performance of the resulting SOFNN.

With more human and computing time, we would also investigate day ahead forecasting and longer term forecasting.

Références

[1] S. Abu. Seasonal arima with python. 2016.

[2] A.JAIN. A comprehensive beginner's guide to create a time series forecast. 2016.

[3] Tiago A. Ferreira, Germano C. Vasconcelos, and Paulo J. Adeodato. A new intelligent system methodology for time series forecasting with artificial neural networks. *Neural Process. Lett.*

[4] J.Brownlee. How to create an arima model for time series forecasting with python. 2017.

[5] Junfei Qiao and Huidong Wang. A self-organizing fuzzy neural network and its applications to function approximation and forecast modeling. *Neurocomputing*, 71, 2008.