Forecasting using Neural Networks : A new approach to dynamic architecture network

Georgia Papadaki*, Fotios Amaxopoulos [1,*]

*Tanaka Business School, Imperial College London, Exhibition Road, London, SW7 2AZ

1. Introduction

During the decades researchers across the globe have used various techniques in order to achieve forecasting capability of time series. From a simple ARIMA model to Artificial Intelligence approaches dispute still divides the science community regarding which method can yield the best forecast. Amid these controversies Neural Networks which where first proposed for forecasting in the late 1980's - early 1990's, still dominate and continue to gain more territory in our collective consciousness as an efficient technique to predict the future.

Despite the various Neural Network models that already exist in the contemporary bibliography, the answer to the question which one of those performs better still elude us. The most commonly used however is Artificial Neural Network with feed forward back propagation algorithm (FFBP) which has been a subject of research in order to test its efficiency more than any other model. By some recent estimates it accounts for the 90 percent of all Neural Network applications and in particular occasions it performed adequately well. Its architecture initially involves empirical trial and error testing regarding the number of neurons in each layer as well as the number of layers that will be used. The inputs as well as a set of weights that are allocated to each separate neuron, are propagated forward to the neurons of the consecutive hidden layer. There, under the application of an activation function they are forwarded to the following `the optimal weights that minimize an error function (usually Square Error Function) is gradient decent or other alternatives like conjugate gradient. Of course a variety of other training approaches are implemented until a certain tolerance error value is reached. Although FFBB Neural Networks are a universal functional approach, there are some shortcomings as the model's efficiency to avoid multiple local minima.

The development of other alternatives including Hopfield networks (Recurrent), Probabilistic, Fuzzy Logic, Self - Organized Networks etc, also share a part in the endeavor to make more accurate predictions under a Neural Network scope. In this paper however, in an attempt to approach one of the most contemporary forecasting model techniques, we implemented the Dynamic Architecture for Artificial Neural Networks (also known as DAN2). It was first developed by M.Ghiassi, H.Saiane (2005) and then in 2006 M.Ghiassi, H.Saiane, D.K.Zibra also implements it to number of commonly used in experiments time series. This model completely diverges from the common and it was a source of inspiration in our search for superior outcomes.

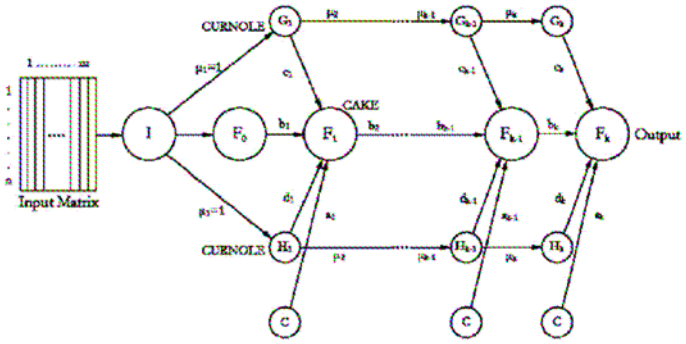
2. Model description

The intuition behind **DAN2** is based upon collecting and propagating gathered Knowledge as a whole rather than propagating the outputs of the activation functions of each individual neuron to the next layer. In this concept DAN2 can be seen as a simple feed forward neural network comprised of

[1] Corresponding author, Tel +447946947570, e-mail: Fotios.amaxopoulos05@imperial.ac.uk

an input layer, hidden layers and an output layer. The difference however between them is that in the latter we usually have to define not only the proper input variables but also the number of hidden layers as well as the number of neurons in each layer. In DAN2 however, the number of neurons is fixed and the optimal number of hidden layers is found dynamically according to some minimum optimal criteria that have to be reached. Thus what is only required from the researcher is to identify the input variables. Throughout the years of experimentation a vast number of papers have been published in an attempt to set a certain framework, based on statistics, in order to identify proper inputs and architecture of a neural network. Some conspicuous examples are Medeiros et al. (2006), Swanson and White (1997), Refenes and Zapranis (1999) and so many others that attempted to clear the multidisciplinary nature of the subject. Despite all of these attempts there is a different point of view, as lots of other researchers regard input variable selection procedure actually as "an art" and thus it is a matter of pure experiment (trial and error) rather than an implementation of a theory in order to meet a number of certain criteria.

Encapsulating the architecture of DAN2, we can describe it as follows: first and foremost the inputs are disseminated to the network as a matrix, i.e. simultaneously, rather than one at a time providing in this way "a training environment that ensures monotonically increasing learning"[2]. After that this collective matrix is propagated to the hidden layers. Under the general concept of DAN2, at each layer we define the linear and the non-linear relationships of the data separately. We combine and forward them to the next layer as a whole accumulated knowledge, and then to the next one, until we reach the output when our criteria for optimal error tolerance are met. More precisely, each hidden layer of DAN2 has a fixed number of neurons, which are four. The first is the so called "threshold unit" or C, which equals 1 and plays the role of the constant in regular regression by OLS. The main neuron is the "Current accumulated Knowledge element" or CAKE and gathers knowledge of all previous layers. The third and the fourth neurons are the "current residual non-linear elements" or CURNOLE and have as a purpose to gather the non linear relationships of the data using as inputs all outputs of the previous layers.



The training procedure of DAN2 is similar to the backpropagation idea. The network is trained until an error function (usually the MSE or the SSE) falls under an acceptable value. Firstly, in a special layer, CAKE neuron captures the linear relationship between the inputs and the desired outputs, under the standard OLS concept. If the error criteria are met then the training stops and we have a linear relationship between the inputs and the output. In general the CAKE neuron combines linearly previous layers' CAKE, CURNOLE and C reassuring in this way that already gained Knowledge is not lost but readjusted and carried until the output neuron is reached. After the input data have been linearly transformed in the fist CAKE node, they are transformed in the subsequent nodes through an algorithm in order to capture the nonlinearities of the process. The algorithm that is used by the creators of DAN2 is a trigonometric function which is represented as follows:

[2] Ghiassi and Shaidane 2005, 'A dynamic architecture for artificial neural networks, Neurocomputing 63 p.397-413

Cosine ($\mu\alpha_i + \theta$)

where α_i is an angle between the record i and the reference vector R, used to train the network and is updated at each subsequent node. It represents the transformed and normalized inputs while μ represents a coefficient and θ represents constant. In order to facilitate our calculations we replace θ with the following trigonometric equation:

$$A \text{ Cosine } (\mu\alpha_i) + B \text{ Sine } (\mu\alpha_i) .$$

The above equation is used by the CURNOLE nodes facilitating the reduction of the nonlinear parameters from one to two. If the equation does not sufficiently represents the nonlinearities that exist, an new layer with an extra node is added to the process as well as one more Cosine ($\mu\alpha_i + \theta$) equation for this extra node. Taking into account what has already been mention above DAN2 differs from each contemporary opponents as its architecture is depended upon the complexity of the problem and is defines dynamically. The general equation that represents the combination of the elements of each layer k in order to produce the output is :

$$F_k = \alpha_k + b_k F_{k-1}(X_i) + c_k G_k(X_i) + d_k H_k(X_i),$$

where X_i is the n independent input record

F_k is the output of layer k

$G_k(X_i)$ = Cosine ($\mu_k\alpha_i$)

$H_k(X_i)$ = Sine ($\mu_k\alpha_i$) and

α_k, b_k, c_k, d_k , μ_k are parameters for optimization at k iteration.

Our contribution to the above presented model will be the implementation of a more efficient algorithm in order to facilitate the estimation of the nonlinear parameter μ_k. While the first four parameters can be estimated by standard OLS, the nonlinear parameter will be estimated using the Conjugate Gradient algorithm. Additionally the Simulated Annealing algorithm will be used as a medium to avoid the classical problem of local minima. In brief, the obstacle of multiple minima rather than one global minimum is widely encountered in the neural network optimization procedures and that is particularly due to our decision when the gradient is zero. In this concept we should never prefer convergence based on small gradients but those based on upward trends of the gradient. Simulated Annealing assist an optimization by 'randomly perturbing the independent variables (weights in the case of a neural network) and keeping track of the best (lowest error) function value for each randomized set of variables.'[3] Firstly we use a random number generator to produce as many as possible iterations in order to find those weights that give the minimum error, we keep track of those weights and we use them as a center from which perturbation for the next random number generator will begin with lower standard deviation this time.

Concerning the final model architecture DAN2 in contrast with the usual feed forward neural networks adds dynamically hidden layers of four neurons each until a specified accuracy measure or a certain number of iterations are reached. Moreover due to the dynamic nature of the model we add two further training stopping criteria so as to avoid under-fitting (alternatively under-training) or over-fitting (over-training). In the first case in order to avoid such a problem we set stopping criterion ,$\varepsilon_1 = (SSE_k - SSE_{k+1}) / SSE_k \leq \varepsilon_1^*$. This is to reassure that adding our neural network has the optimal number of layers and none redundant that could increase the overall prediction error or would make the process slower without adding any further Knowledge. Regarding the other more commonly encountered problem in forecasting using neural networks, this of an over-fitted model, there are an additional criterion set to overcome it. We use $\varepsilon_2 = \left| MSE_T - MSE_v \right| / MSE_T \leq \varepsilon_2^*$,where 'T'

[3] Timothy Masters,1993, 'Practical Neural Network Recipes in C++', p.119

subscript stands for training and 'V' for validation. The training procedure stops when the above criteria are fully met. Something that worths to be mentioned is that we should be extremely careful regarding the desired level we set for ε_1 and ε_2. This is because if this level is too low we will definitely reach it too soon and the whole model should then be reconsidered setting this time a new level for them.

3. Model evaluation and results

For the evaluation of our model we will use a number of the most classical and widely used error functions like Mean Square Error, Mean Absolute Error, Mean Absolute Deviation and finally in the context of the competition's requirements we will add Symmetric Mean Absolute Percent Error. Our results indicate a promising out-performance of the statistical methods mentioned even in problems involving long horizons and inadequate number of observations to forecast.

References

Bishop C.M., 1995, Neural Networks for Pattern Recognition, Oxford University Press.

Ghiassi M., Saidane H., 2005 A dynamic architecture for artificial neural networks, Neurocomputing 63, 397–413.

Ghiassi, M., Saidane H., Zimbra D., 2005, A dynamic artificial neural network model for forecasting time series events, International Journal of Forecasting 21, 341–362.

Ghiassi, M., Zimbra D., Saidane H., 2006, Medium Term System Load Forecasting with a Dynamic Artificial Neural Network Model., Electric Power Systems Research 76, 302–316.

Masters, T., 1995, Advanced algorithms for neural networks : a C++ sourcebook**,** New York, NY, Wiley.

Masters, T., 1993, Practical neural network recipes in C++, San Diego, London , Academic Press.

Medeiros M.C., Teräsvirta T., Rech G., 2002, Building Neural Network Models for Time Series: A Statistical Approach, Journal of Forecasting, 25, 49–75.

Swanson NR., White H., 1997, A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks, Review of Economic and Statistics, 79, 540–550.

Welstead, S., 1994, Neural network and fuzzy logic applications in C/C++, New York, Chichester, Wiley.

Zapranis A., Refenes A-P., 1999, Principles of Neural Model Identification, Selection and Adequacy: With Applications to Financial Econometrics. Springer-Verlag, Berlin.