

Higher Order and Recurrent Neural Architectures for Trading the EUR/USD Exchange Rate

by
Christian L. Dunis*
Jason Laws*
Georgios Sermpinis*
(* Liverpool Business School, CIBEF
Liverpool John Moores University)

February 2008

Abstract

The motivation for this paper is to investigate the use of alternative novel neural network architectures when applied to the task of forecasting and trading the Euro/Dollar (EUR/USD) exchange rate. This is done by benchmarking three different neural network designs representing a Higher Order Neural Network (HONN), a Psi Sigma Network and a Recurrent Network (RNN) with three successful architectures, the traditional Mutilayer Perceptron (MLP), the Softmax and the Gaussian Mixture (GM) models, as reported in Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004). More specifically, the trading performance of the six models is investigated in a forecast and trading simulation competition on the EUR/USD time series over a period of 8 years. These results are also benchmarked with more traditional models such as a moving average convergence divergence technical model (MACD), an autoregressive moving average model (ARMA) and a logistic regression model (LOGIT).

As it turns out, the MLP, the HONN, the Psi Sigma and the RNN models do all well and outperform the more traditional models in a simple trading simulation exercise. However, when more sophisticated trading strategies using confirmation filters and leverage are applied, the GM network produces remarkable results and outperforms all the other network architectures.

Keywords

Confirmation filters, Higher Order Neural Networks, Psi Sigma Networks, Recurrent Networks, Gaussian Mixture models, leverage, Multi-Layer Perceptron Networks, Probability Distributions, Quantitative Trading Strategies, Softmax Cross Entropy Networks.

Christian Dunis is Professor of Banking and Finance at Liverpool Business School and Director of the Centre for International Banking, Economics and Finance (CIBEF) at Liverpool John Moores University (E-mail: cdunis@tiscali.co.uk).

Jason Laws is Reader of Finance at Liverpool Business School and a member of CIBEF (E-mail: J.Laws@ljmu.ac.uk).

Georgios Sermpinis is an Associate Researcher with CIBEF (E-mail: G.Sermpinis@2005.ljmu.ac.uk) and currently working on his PhD thesis at Liverpool Business School.

CIBEF – Centre for International Banking, Economics and Finance, JMU, John Foster Building, 98 Mount Pleasant, Liverpool L3 5UZ.

1. INTRODUCTION

Neural networks are an emergent technology with an increasing number of real-world applications including Finance (Lisboa *et al.* (2000)). However their numerous limitations are often creating scepticism about their use among practitioners.

The motivation for this paper is to investigate the use of several new neural networks techniques that try to overcome these limitations. This is done by benchmarking three different neural network architectures representing a Higher Order Neural Network (HONN), a Psi Sgima network and a Recurrent Neural Network (RNN). Their trading performance on the Euro/Dollar (EUR/USD) time series is investigated and is compared with the three best models reported by Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004), the Multi-layer Perceptron (MLP), the Softmax and the Gaussian Mixture (GM) model. So in essence, this paper can be seen as a continuation of the research mentioned just above or as a forecasting competition among some of the most up-to-date forecasting techniques over a demanding series such as the EUR/USD exchange rate.

The results of our three networks can also be compared to the more traditional approaches also studied by Dunis and Williams (2002, 2003), namely a moving average convergence divergence technical model (MACD), an autoregressive moving average model (ARMA) and a logistic regression model (LOGIT).

As it turns out, the MLP, the HONN and the Psi Sigma demonstrate a similar good performance and outperform the more traditional models in a simple trading simulation exercise, while the GM model outperforms all models when more sophisticated trading strategies using confirmation filters and leverage are applied. This might be due to the ability of the GM model to use probability distributions to identify successfully trades with a high Sharpe ratio.

The rest of the paper is organised as follows. In section 2, we present the literature relevant to the Recurent Networks, the Higher Order Neural Networks and their variant Psi Sigma. Section 3 describes the dataset used for this research, actually the same as in Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004). An overview of the different neural network models is given in section 4. Section 5 gives the empirical results of all the models considered and investigates the possibility of improving their performance with the application of more sophisticated trading strategies. Section 6 provides some concluding remarks.

2. LITERATURE REVIEW

The motivation for this paper is to apply some of the most promising new neural networks architectures which have been developed recently with the purpose to overcome the numerous limitations of the more classic neural architectures and to assess whether they can achieve a higher performance in a trading simulation.

RNNs have an activation feedback which embodies short-term memory allowing them to learn extremely complex temporal patterns. Their superiority against feedforward networks when performing nonlinear time series prediction is well documented in Connor *et al.* (1993) and Adam *et al.* (1994). In financial applications, Kamijo *et al.* (1990) applied them successfully to the recognition of stock patterns of the Tokyo stock exchange while Tenti (1996) achieved remarkable results using RNNs to forecast the exchange rate of the Deutsche Mark. Tino *et al.* (2001) use them to trade successfully the volatility of the DAX and the FTSE 100 using straddles while Dunis and Huang

(2002), using continuous implied volatility data from the currency options market, obtain remarkable results for their GBP/USD and USD/JPY exchange rate volatility trading simulation.

HONNs were first introduced by introduced by Giles and Maxwell (1987) as a fast learning network with increased learning capabilities. Although their function approximation superiority over the more traditional architectures is well documented in the literature (see among others Redding *et al.* (1993), Kosmatopoulos *et al.* (1995) and Psaltis *et al.* (1998)), their use in finance so far has been limited. This has changed when scientists started to investigate not only the benefits of Neural Networks (NNs) against the more traditional statistical techniques but also the differences between the different NNs model architectures. Practical applications have now verified the theoretical advantages of HONNs by demonstrating their superior forecasting ability and put them in the front line of research in financial forecasting. For example Dunis *et al.* (2006b) use them to forecast successfully the gasoline crack spread while Fultcher *et al.* (2006) apply HONNs to forecast the AUD/USD exchange rate, achieving a 90% accuracy. However, Dunis *et al.* (2006a) show that, in the case of the futures spreads and for the period under review, the MLPs performed better compared with HONNs and recurrent neural networks.

Psi Sigma networks were first introduced as an architecture capable of capturing higher order correlations within the data while avoiding some of the HONNs limitations such as the combinatorial increase in weight numbers. Shin *et al.* (1991) and Ghosh *et al.* (1992) demonstrate these benefits and present empirical evidence on their forecasting ability. For financial applications, Ghazali *et al.* (2006) compare them with HONNs on the IBM common stock closing price and the US 10-year government bond series and prove their forecasting superiority while, in a similar paper, Hussain *et al.* (2006) present satisfactory results of the Psi Sigma forecasting power on the EUR/USD, the EUR/GBP and the EUR/JPY exchange rates.

3. THE EUR/USD EXCHANGE RATE AND RELATED FINANCIAL DATA

Our benchmark test is to trade the EUR/USD exchange rate based on daily forecasts of its London closing prices¹. All time series are daily closing data obtained from a historical database provided by Datastream and used in Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004).

Name of period	Trading days	Beginning	End
Total dataset	1749	17 October 1994	03 July 2001
Training dataset	1459	17 October 1994	18 May 2000
Out-of-sample dataset [Validation set]	290	19 May 2000	03 July 2001

Table 1: The EUR/USD dataset

¹ EUR/USD is quoted as the number of USD per Euro: for example, a value of 1.2657 is USD1.2657 per Euro. The EUR/USD exchange rate only exists from 4 January 1999: it was retroplated from 17 October 1994 to 31 December 1998 and a synthetic EUR/USD series was created for that period using the fixed EUR/DEM conversion rate agreed in 1998, combined with the USD/DEM daily market rate.

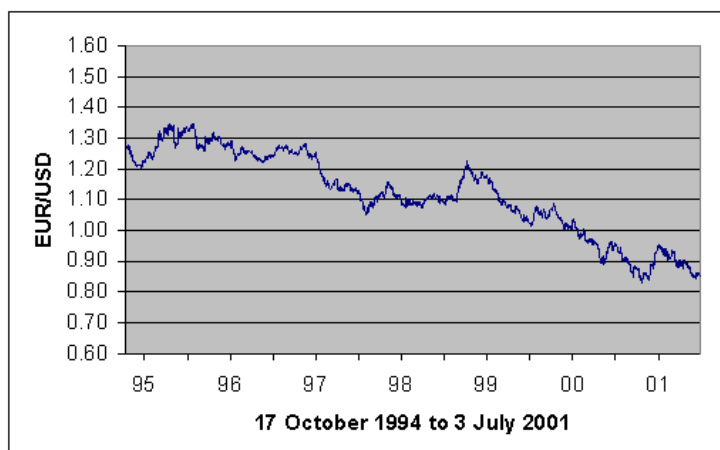


Fig. 1: EUR/USD London daily closing prices (total dataset)

Dunis and Williams (2002, 2003) carried out a variable selection and identified the explanatory variables listed in table 2.

Number	Variable	Mnemonics	Lag
1	US \$ TO UK £ (WMR) – EXCHANGE RATE	USDOLLR	12
2	JAPANESE YEN TO US \$ (WMR) – EXCHANGE RATE	JAPAYE\$	1
3	JAPANESE YEN TO US \$ (WMR) – EXCHANGE RATE	JAPAYE\$	10
4	BRENT CRUDE – Current Month, fob U\$/BBL	OILBREN	1
5	GOLD BULLION \$/TROY OUNCE	GOLDBLN	19
6	FRANCE BENCHMARK BOND 10 YR (DS) – RED. YIELD	FRBRYLD	2
7	ITALY BENCHMARK BOND 10 YR (DS) – RED. YIELD	ITBRYLD	6
8	JAPAN BENCHMARK BOND – RYLD. 10 YR (DS) – RED.	JPBRYLD	9
9	NIKKEI 225 STOCK AVERAGE – PRICE INDEX	JAPDOWA	1
10	NIKKEI 225 STOCK AVERAGE – PRICE INDEX	JAPDOWA	15

Table 2: Explanatory variables and Datastream mnemonics

The observed EUR/USD time series is non-normal (Jarque-Bera statistics confirmed this at the 99% confidence interval) containing slight skewness and low kurtosis. It is also nonstationary and Dunis and Williams (2002, 2003) decided to transform the EUR/USD as well as all the explanatory series into stationary series of rates of return².

Given the price level P_1, P_2, \dots, P_t , the rate of return at time t is formed by:

$$R_t = \left(\frac{P_t}{P_{t-1}} \right) - 1 \quad [1]$$

The summary statistics of the EUR/USD returns series reveal a slight skewness and high kurtosis. The Jarque-Bera statistic confirms again that the EUR/USD series is non-normal at the 99% confidence interval.

² Confirmation of its stationary property is obtained at the 1% significance level by both the Augmented Dickey Fuller (ADF) and Phillips-Perron (PP) test statistics.

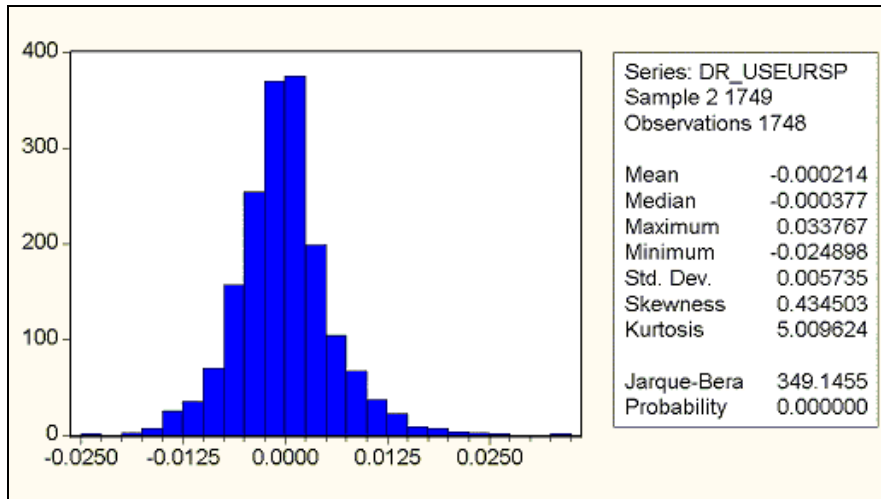


Fig. 2: EUR/USD returns summary statistics (total dataset)

A further transformation includes the creation of interest rates yield curve series, generated by:

$$yc = 10 \text{ year benchmark bond yields} - 3 \text{ month interest rates} \quad [2]$$

Following Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004), we divide our dataset as follows:

Name of period	Trading days	Beginning	End
Total data set	1749	17 October 1994	03 July 2001
Training data set	1169	17 October 1994	08 April 1999
Test data set	290	09 April 1999	18 May 2000
Out-of-sample data set [Validation set]	290	19 May 2000	03 July 2001

Table 3: The neural networks datasets

4. THE NEURAL NETWORKS FORECASTING MODELS

Neural networks exist in several forms in the literature. The most popular architecture is the Multi-layer Perceptron (MLP).

A standard neural network has at least three layers. The first layer is called the input layer (the number of its nodes corresponds to the number of explanatory variables). The last layer is called the output layer (the number of its nodes corresponds to the number of response variables). An intermediary layer of nodes, the hidden layer, separates the input from the output layer. Its number of nodes defines the amount of complexity the model is capable of fitting. In addition, the input and hidden layer contain an extra node, called the bias node. This node has a fixed value of one and has the same function as the intercept in traditional regression models. Normally, each node of one layer has connections to all the other nodes of the next layer.

The network processes information as follows: the input nodes contain the value of the explanatory variables. Since each node connection represents a weight factor, the information reaches a single hidden layer node as the weighted sum of its inputs. Each node of the hidden layer passes the information through a nonlinear activation function and passes it on to the output layer if the calculated value is above a threshold.

The training of the network (which is the adjustment of its weights in the way that the network maps the input value of the training data to the corresponding output value) starts with randomly chosen weights and proceeds by applying a learning algorithm called backpropagation of errors³ (Shapiro (2000)). The learning algorithm simply tries to find those weights which optimise an error function (normally the sum of all squared differences between target and actual values). Since networks with sufficient hidden nodes are able to learn the training data (as well as their outliers and their noise) by heart, it is crucial to stop the training procedure at the right time to prevent overfitting (this is called ‘early stopping’). This can be achieved by dividing the dataset into 3 subsets respectively called the training and test sets used for simulating the data currently available to fit and tune the model and the validation set used for simulating future values. The network parameters are then estimated by fitting the training data using the above mentioned iterative procedure (backpropagation of errors). The iteration length is optimised by maximising the forecasting accuracy for the test dataset. Finally, the predictive value of the model is evaluated applying it to the validation dataset (out-of-sample dataset).

4.1 THE MULTI-LAYER PERCEPTRON MODEL

4.1.1 The MLP network architecture

The network architecture of a ‘standard’ MLP looks as presented in figure 3⁴:

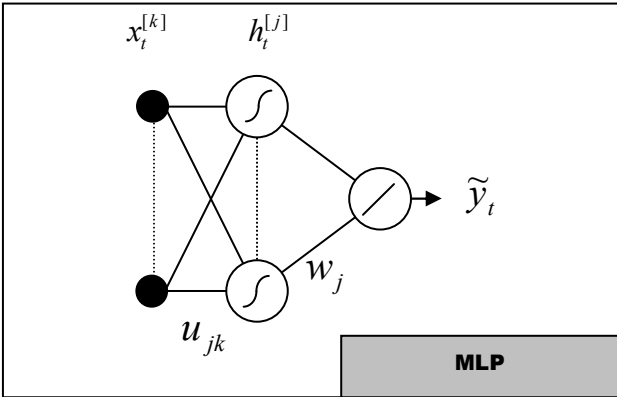


Fig. 3: A single output, fully connected MLP model

where:

$x_t^{[n]}$ ($n = 1, 2, \dots, k + 1$) are the model inputs (including the input bias node) at time t

$h_t^{[m]}$ ($m = 1, 2, \dots, j + 1$) are the hidden nodes outputs (including the hidden bias node)

\tilde{y}_t is the MLP model output

u_{jk} and w_j are the network weights

\textcircled{S} is the transfer sigmoid function: $S(x) = \frac{1}{1 + e^{-x}}$, [3]

\textcircled{L} is a linear function: $F(x) = \sum_i x_i$ [4]

³ Backpropagation networks are the most common multi-layer networks and are the most commonly used type in financial time series forecasting (Kaastra and Boyd (1996)).

⁴ The bias nodes are not shown here for the sake of simplicity.

The error function to be minimised is:

$$E(u_{jk}, w_j) = \frac{1}{T} \sum_{t=1}^T (y_t - \tilde{y}_t(u_{jk}, w_j))^2, \quad \text{with } y_t \text{ being the target value} \quad [5]$$

4.1.2 Empirical results of the MLP model

The results for the MLP achieved by Dunis and Williams (2002, 2003) are summarized in table 4. The benchmark model ‘naïve strategy’ follows the rule that the forecast return for tomorrow is today’s value. The trading strategy applied is simple: go or stay long when the forecast return is above zero and go or stay short when the forecast return is below zero. Appendix A.1 documents the performance measures used while Appendix A.2 gives the results of the more traditional techniques, namely a moving average convergence divergence technical model (MACD), an autoregressive moving average model (ARMA) and a logistic regression model (LOGIT). The MLP outperforms all benchmarks.

	NAIVE	MLP
<i>Sharpe Ratio (excluding costs)</i>	1.83	2.57
<i>Annualised Volatility (excluding costs)</i>	11.6%	11.6%
<i>Annualised Return (excluding costs)</i>	21.3%	29.7%
<i>Maximum Drawdown (excluding costs)</i>	-9.1%	-9.1%
<i>Taken Positions (annualised⁵)</i>	109	118

Table 4: Trading performance of the benchmark models

4.2 THE SOFTMAX CROSS ENTROPY MODEL

The Softmax cross entropy network (henceforth SCE) is a neural network with a cross entropy cost function and a Softmax activation function at the output nodes. The main idea of this model is to approximate the probability density function for the target value through a histogram representing the probability of the target value being within a range of predefined size. The output value of a SCE model is therefore a vector with as many elements as there are output nodes, 6 in our case (each node representing one bar of the histogram). The vector elements sum up to unity and represent the density function for the target value while each vector element stands for the probability that the target value lies in the value range the vector element represents.

In order to apply the cross entropy cost function, the target values of the training data set have to be preprocessed so that one gets a target vector (rather than a single target value as with the MLP), where the target vector has as many elements as the SCE model has output nodes. The target vector consists of zeros and a single one. The value ‘one’ indicates which output node of the network covers the value range where the original target value lies in. Since the network forecasts should be used as a density function, one has to take care that the output vector sums up to unity. This is done by superimposing the Softmax function to the actual network outputs. The Softmax function keeps the internal relationship between the output values but transforms them in a way that their values add up to unity (see equation [8] below).

During the training phase (that is when the network weights are adjusted), the SCE model learns to map the input vector of the training data set to the target vector of the same data set. Since each target vector consists of a single ‘one’ representing a non-

⁵ The number of taken positions can differ from the number of trading days due to the possibility to hold a position for longer than 1 day.

overlapping range of possible output values (while the rest are zeros), the SCE model tries in fact to solve a classification task.

The network might face a situation where the same input vector is related to two different output values (at different times) so that the network has no other chance than to map the input vector to more than one output node. In doing so, the network generates a density function for the target value, while the integrated Softmax function ensures that the probabilities add up to unity.

4.2.1 The SCE network architecture

The difference in architecture with a MLP lies in the multiple output nodes. While the MLP has typically only one output node delivering a level estimation, the SCE network uses several output nodes to represent an approximation of the density function (while being trained on a classification task).

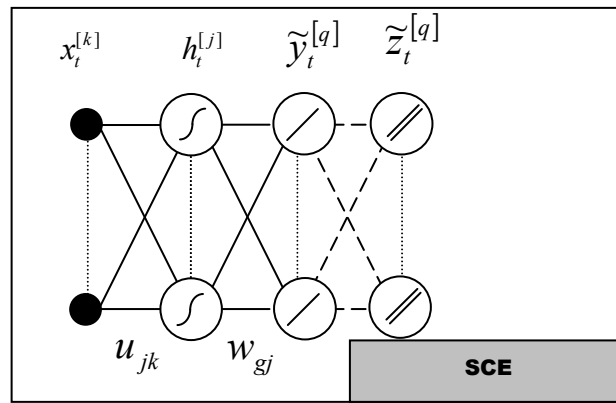


Fig. 4: A single output, fully connected SCE model

where:

- $x_t^{[n]}$ ($n = 1, 2, \dots, k+1$) are the model inputs (including the input bias node) at time t
- $h_t^{[m]}$ ($m = 1, 2, \dots, j+1$) are the hidden nodes outputs (including the hidden bias node)
- $\tilde{y}_t^{[g]}$ ($g = 1, 2, \dots, q$) is the SCE model output before applying the Softmax function
- $\tilde{z}_t^{[g]}$ ($g = 1, 2, \dots, q$) is the network value at the output node g
- u_{jk} and w_{gj} are the network weights

Ⓢ is the transfer sigmoid function: $S(x) = \frac{1}{1 + e^{-x}}$, [6]

Ⓛ is a linear function: $F(x) = \sum_i x_i$ [7]

Ⓜ is the Softmax function $A(g) = \tilde{z}_g = \frac{\exp(\tilde{y}_g)}{\sum_{g_1} \exp(\tilde{y}_{g_1})}$ [8]
with \tilde{y}_g being the output of the linear function

The error function to be minimised is:

$$E(u_{jk}, w_{gj}) = \sum_{t=1}^T \sum_{g=1}^q y_{tg} \cdot \log \left(\frac{y_{tg}}{\tilde{z}_{tg}(u_{jk}, w_{gj})} \right), \quad \text{with } y_{tg} \text{ being the target value} \quad [9]$$

4.2.2 Empirical results of the SCE model

Since neural networks start with random initialisation of their weights, each network (even with the same architecture) is unique and produces slightly different results. In order to get stable and reliable results from the SCE architecture, Lindemann *et al.* (2004) split the initial investment capital equally amongst 30 identical (except the initial weights) models. The result is therefore the average result of a committee of 30 SCE networks. The trading strategy consists of using the density function of each of the 30 SCE models to calculate the probability for an upmove. This is simply done by adding up the last 3 of 6 values of the output vector (since those 3 values cover the whole range of possible positive values for an upmove) and taking a long position if the probability for an upmove exceeds 50% (and a short position vice versa). A summary of the results achieved by the SCE committee is given table 5 below.

	NAIVE	MLP	SCE
<i>Sharpe Ratio (excluding costs)</i>	1.83	2.57	2.26
<i>Annualised Volatility (excluding costs)</i>	11.6%	11.6%	11.6%
<i>Annualised Return (excluding costs)</i>	21.3%	29.7%	26.3%
<i>Maximum Drawdown (excluding costs)</i>	-9.1%	-9.1%	-7.8%
<i>Positions Taken (annualised)</i>	109	118	143

Table 5: Trading performance results

4.3 THE GAUSSIAN MIXTURE MODEL

The GM network was first introduced by Husmeier (1999) and is applied to our EUR/USD time series in Lindemann *et al.* (2004).

The GM model represents the probability density of the data by a linear combination of a fixed number of normal distributions (where the distribution width is adapted to the whole set of training data while the locations of the distribution centres depend on the actual input data x_t and the dependent variable y_t). This is done in a hidden layer where each node represents a normal distribution. The actual network output is not the density function itself but the prediction of a single value⁶ which is the likelihood of the actual GM model parameters generating the observed value of the dependent variable y conditioned on the input data x .

To optimise the cost function (that is, to maximise the sum of likelihood values), the weights u_{jk} and w_{ij} , determining the location of the normal distribution centres (μ_t), have to be adapted so that the distance between y_t and μ_t is minimal. Doing so, the centres of the distribution are close to y_t and therefore the likelihood and with it the value of the cost function are high. See figure 5 below to illustrate that working principle.

4.3.1 The GM network architecture

The GM architecture differs in three main ways from the benchmark feedforward MLP network. First, as shown by Husmeier (1999), in order to be a universal approximator at least a second hidden layer is necessary. Second, both the independent and

⁶ Nevertheless, the whole density distribution can be constructed by varying the value of y over the interesting range of the searched density function.

dependent variable (\mathbf{x}, \mathbf{y}) are used as input data, since the aim is not to predict \mathbf{y} but its density distribution $P(\mathbf{y} | \mathbf{x})$ respectively the corresponding likelihood value. Third, the network uses Gaussian distributions in the second hidden layer.

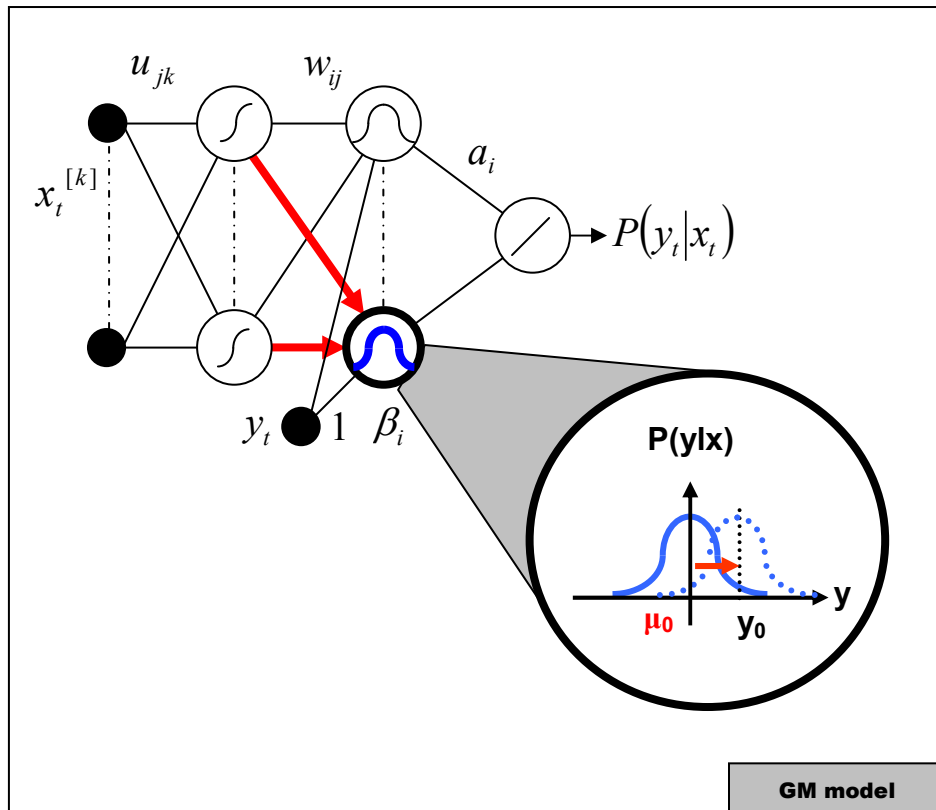


Fig. 5: GM network architecture

The following functions are applied within the GM model:

- $x_t^{[n]}$ ($n = 1, 2, \dots, k+1$) are the model inputs (including the input bias node) at time t
- y_t is the argument of the density function conditional on the values of the inputs (note that the weights of y_t are fixed to 1^7)
- u_{jk} and w_{ji} are the network weights
- β_i define the inverse widths of the Gaussian distributions
- a_i are the mixing coefficients, with $\sum_i a_i = 1$
- i is the number of applied Gaussian mixture distributions
- j is the number of applied network weights w_j
- k is the number of applied network weights u_{jk}

⁷ If we would not fix the weight to 1 the network could decrease the cost function not only by adjusting the centres of the Gaussian mixture functions but also by changing the original target value y_t .



Gaussian distribution: $G_{\beta_i}(y_t - \mu_i) = \sqrt{\frac{\beta_i}{2\pi}} \exp\left(-\frac{\beta_i \cdot (y_t - \mu_i)^2}{2}\right)$, [10]

with $\mu_i(x) := \sum_j w_{ij} S\left(\sum_k u_{jk} x_k\right)$, $\sigma_k = \sqrt{\frac{1}{\beta_k}}$, $\beta_i > 0$, $a_i \geq 0$, $\sum_i a_i = 1$



Sigmoid function: $S(x) = \frac{1}{1 + e^{-x}}$, [11]



Linear function: $P(y|x) = \sum_i a_i G_{\beta_i}[y - \mu_i(x)]$ [12]

The error function to be minimised is:

$$E(u_{jk}, w_{ij}, \beta_i, a_i) = -\frac{1}{T} \sum_{t=1}^T \ln(P(y_t | x_t, u_{jk}, w_{ij}, \beta_i, a_i)), \quad \text{with } y_t \text{ being the target value} \quad [13]$$

It is possible to update the parameters of the GM model by gradient descent, as was done with the MLP network. However this algorithm, due to the architectural complexity of the GM network, is very time consuming.

4.3.2 Empirical results of the GM model

In order to apply the GM model to the EUR/USD return time series, Lindemann *et al.* (2004) optimise its parameters as well as the stopping point⁸ on the EUR/USD test data set. The trading strategy consists of using the density functions to calculate the probability for a positive exchange rate change as well as for a negative change and taking a trading position where the probability is biggest (namely >50% since both add up to 100%).

In order to minimize the variance of the network forecasts, they also split the initial investment capital equally amongst 30 identical (except the initial weights) GM models. Their result is therefore the average result of a committee of 30 GM networks in order to minimise the chance to pick an outlier model. This is particularly important when trading on the tails of the density function. Table 6 includes a summary of the GMs forecasting performance.

	NAIVE	MLP	SCE	GM
<i>Sharpe Ratio (excluding costs)</i>	1.83	2.57	2.26	2.09
<i>Annualised Volatility (excluding costs)</i>	11.6%	11.6%	11.6%	11.6%
<i>Annualised Return (excluding costs)</i>	21.3%	29.7%	26.3%	24.2%
<i>Maximum Drawdown (excluding costs)</i>	-9.1%	-9.1%	-7.8%	-12.4%
<i>Positions Taken (annualised)</i>	109	118	143	162

Table 6: Trading performance results

As can be seen, the performance of the GM committee does not improve the result of the single benchmark MLP network.

However the GM model does provide more information than is actually used with this simple trading strategy as we have access to the complete distribution of the predicted

⁸ Even with regularisation, the additional implementation of early stopping improved results (see Lindemann *et al.* (2004)). The weights were fixed at the best result on the test data set during training.

move in the exchange rate. This should be helpful when applying more sophisticated strategies which are investigated in detail in section 5.

4.4 THE RECURRENT NETWORK

Our next model is the recurrent neural network. While a complete explanation of RNN models is beyond the scope of this paper, we present below a brief explanation of the significant differences between RNN and MLP architectures. For an exact specification of the recurrent network, see Elman (1990).

A simple recurrent network has activation feedback, which embodies short-term memory. The advantages of using recurrent networks over feedforward networks, for modelling non-linear time series, has been well documented in the past. However as described in Tenti (1996) “the main disadvantage of RNNs is that they require substantially more connections, and more memory in simulation, than standard backpropagation networks”, thus resulting in a substantial increase in computational time. However having said this RNNs can yield better results in comparison to simple MLPs due to the additional memory inputs.

4.4.1 The RNN architecture

A simple illustration of the architecture of an Elman RNN is presented below.

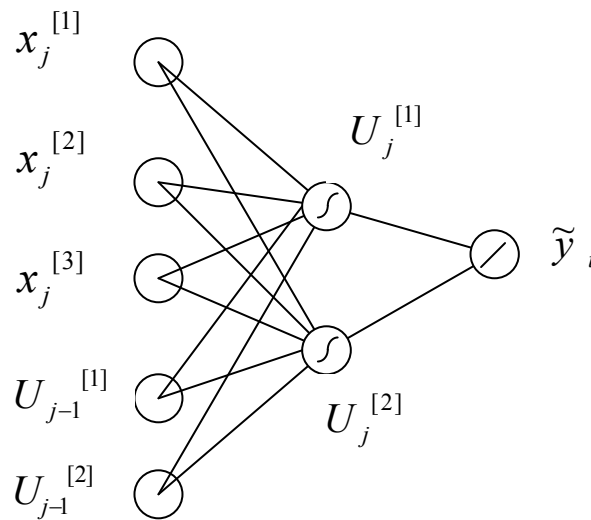


Fig. 6: Elman Recurrent neural network architecture with two nodes on the hidden layer.


where:


$x_t^{[n]}$ ($n = 1, 2, \dots, k + 1$), $u_t^{[1]}$, $u_t^{[2]}$ are the model inputs (including the input bias node) at time t

\tilde{y}_t is the recurrent model output

$d_t^{[f]}$ ($f = 1, 2$) and $w_t^{[n]}$ ($n = 1, 2, \dots, k + 1$) are the network weights

$U_t^{[f]}$ ($f = 1, 2$) is the output of the hidden nodes at time t

 is the transfer sigmoid function: $S(x) = \frac{1}{1 + e^{-x}}$, [14]

 is the linear output function: $F(x) = \sum_i x_i$ [15]

The error function to be minimised is:

$$E(d_t, w_t) = \frac{1}{T} \sum_{t=1}^T (y_t - \tilde{y}_t(d_t, w_t))^2 \quad [16]$$

In short, the RNN architecture can provide more accurate outputs because the inputs are (potentially) taken from all previous values (see inputs $U_{j-1}^{[1]}$ and $U_{j-1}^{[2]}$ in the figure above).

4.4.2 Empirical results of the RNN model

The RNNs are trained with gradient descent as for the MLPs. However, the increase in the number of weights, as mentioned before, makes the training process extremely slow: to derive our results, we needed about five times the time needed with the MLPs.

Mostly for this reason, we decided to use a single network and not the average of a committee, selecting in the end the network that demonstrated the best statistical performance criteria in the test and training period⁹. Moreover, with this methodology our results are directly compatible with those of Dunis and Williams (2002, 2003) and the forecasting competition seems fairer. The characteristics of the network that we used are on Appendix A3.

The trading strategy is that followed for the MLP. As shown in table 7 below, the RNN has an overall performance similar to that of the MLP model.

	NAIVE	MLP	SCE	GM	RNN
Sharpe Ratio (excluding costs)	1.83	2.57	2.26	2.09	2.57
Annualised Volatility (excluding costs)	11.6%	11.6%	11.6%	11.6%	11.6%
Annualised Return (excluding costs)	21.3%	29.7%	26.3%	24.2%	29.8%
Maximum Drawdown (excluding costs)	-9.1%	-9.1%	-7.8%	-12.4%	-13.8%
Positions Taken (annualised)	109	118	143	162	124

Table 7: Trading performance results

4.5 THE HIGHER ORDER NEURAL NETWORK

Higher Order Neural Networks (HONNs) were first introduced by Giles and Maxwell (1987) and were called “Tensor Networks”. Although the extent of their use in finance has so far been limited, Knowles *et al.* (2005) show that, with shorter computational times and limited input variables, “the best HONN models show a profit increase over the MLP of around 8%” on the EUR/USD time series (p. 7). For Zhang *et al.* (2002), a significant advantage of HONNs is that “HONN models are able to provide some rationale for the simulations they produce and thus can be regarded as “open box”

⁹ We choose the network with firstly the lowest Mean Absolute Error and secondly the lowest Root Mean Squared Error in the training and test period. With these criteria we got the best performance in the test and the training period.

rather than “black box”. Moreover, HONNs are able to simulate higher frequency, higher order non-linear data, and consequently provide superior simulations compared to those produced by ANNs (Artificial Neural Networks)” (p. 188).

4.5.1 The HONN Architecture

While they have already experienced some success in the field of pattern recognition and associative recall¹⁰, the use of HONNs in finance is not yet widespread. The architecture of a three input second order HONN is shown below:

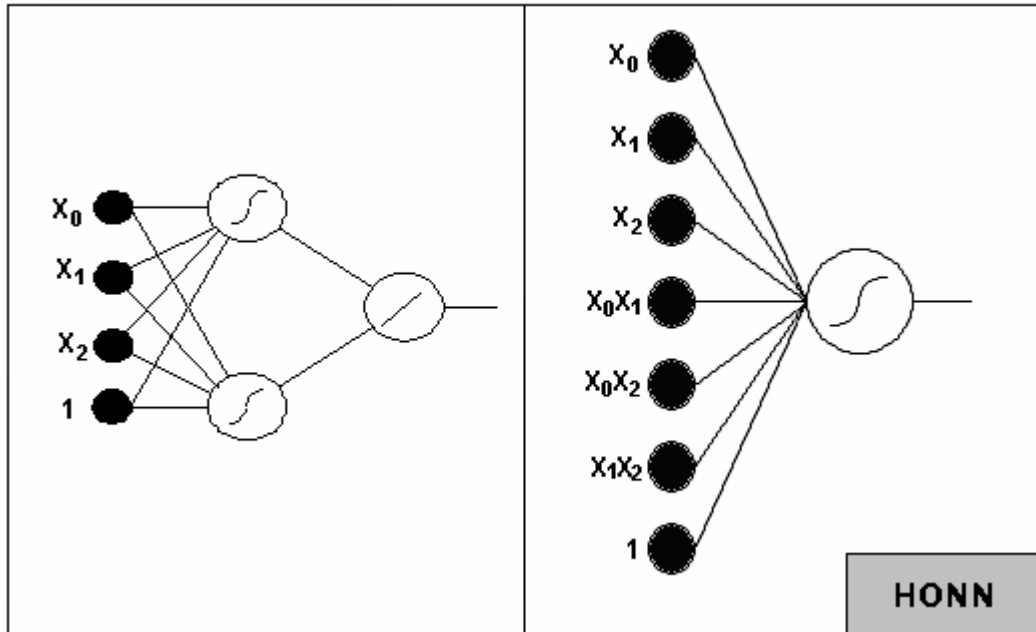


Fig. 7: Left, MLP with three inputs and two hidden nodes; right, second order HONN with three inputs

where:

$x_t^{[n]}$ ($n = 1, 2, \dots, k + 1$) are the model inputs (including the input bias node) at time t

\tilde{y}_t is the HONNs model output

u_{jk} are the network weights



are the model inputs.



is the transfer sigmoid function: $S(x) = \frac{1}{1 + e^{-x}}$, [17]



is a linear function: $F(x) = \sum_i x_i$ [18]

The error function to be minimised is:

$$E(u_{jk}, w_j) = \frac{1}{T} \sum_{t=1}^T (y_t - \tilde{y}_t(u_{jk}))^2, \quad \text{with } y_t \text{ being the target value} \quad [19]$$

¹⁰ Associative recall is the act of associating two seemingly unrelated entities, such as smell and colour. For more information see Karayiannis *et al.* (1994).

HONNs use joint activation functions; this technique reduces the need to establish the relationships between inputs when training. Furthermore this reduces the number of free weights and means that HONNS are faster to train than even MLPs. However because the number of inputs can be very large for higher order architectures, orders of 4 and over are rarely used.

Another advantage of the reduction of free weights means that the problems of overfitting and local optima affecting the results of neural networks can be largely avoided. For a complete description of HONNs see Knowles *et al.* (2005).

4.5.2 Empirical results of the HONN model

We follow the same methodology as we did with RNNs for the selection of our optimal HONN and again we use a single network and not the average of a committee. The trading strategy is that followed for the MLP. A summary of our findings is presented in table 8 below while the characteristics of the network that we used are on Appendix A3.

	NAIVE	MLP	SCE	GM	RNN	HONN
<i>Sharpe Ratio (excluding costs)</i>	1.83	2.57	2.26	2.09	2.57	2.58
<i>Annualised Volatility (excluding costs)</i>	11.6%	11.6%	11.6%	11.6%	11.6%	11.6%
<i>Annualised Return (excluding costs)</i>	21.3%	29.7%	26.3%	24.2%	29.8%	29.8%
<i>Maximum Drawdown (excluding costs)</i>	-9.1%	-9.1%	-7.8%	-12.4%	-13.8%	-9.2%
<i>Positions Taken (annualised)</i>	109	118	143	162	124	129

Table 8: Trading performance results

We can see that our results slightly improve with the introduction of HONNs and that in general terms HONNs, the Recurrent and the MLP seems to have the same forecasting strength on this specific dataset.

4.6 THE PSI SIGMA NETWORK

Psi Sigma networks can be considered as a class of feedforward fully connected HONNs. First introduced by Ghosh and Shin (1991), the Psi Sigma network utilizes product cells as the output units to indirectly incorporate the capabilities of higher-order networks while using a fewer number of weights and processing units. Their creation was motivated by the need to create a network combining the fast learning property of single layer networks with the powerful mapping capability of HONNs while avoiding the combinatorial increase in the required number of weights. While the order of the more traditional HONN architectures is expressed by the complexity of the inputs, in the context of Psi Sigma, it is represented by the number of hidden nodes.

4.6.1 The Psi Sigma architecture

In a Psi Sigma network the weights from the hidden to the output layer are fixed to 1 and only the weights from the input to the hidden layer are adjusted, something that greatly reduces the training time. Moreover, the activation function of the nodes in the hidden layer is the summing function while the activation function of the output layer is a sigmoid. The figure below shows a Psi Sigma with one output layer.

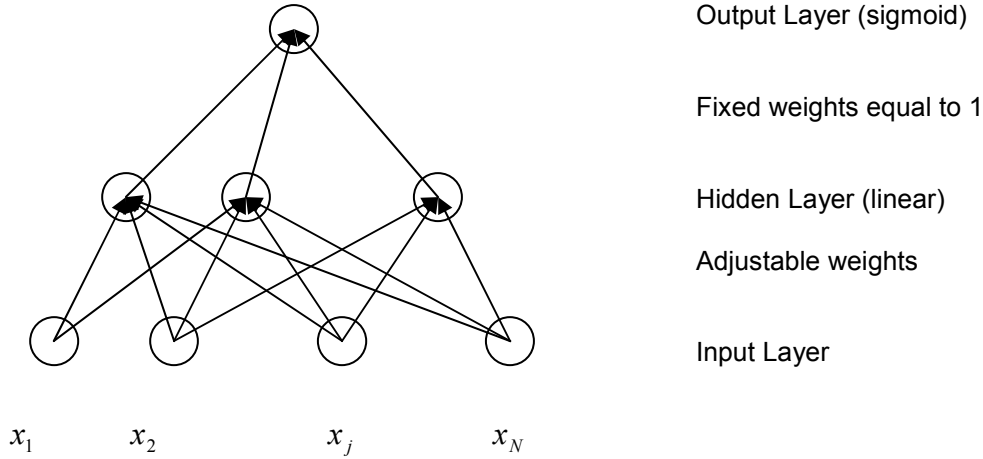


Fig. 8: A Psi Sigma network with one output layer

where:

x_i ($n = 1, 2, \dots, k + 1$) are the model inputs (including the input bias node)

\tilde{y}_t is the Psi Sigma output

w_j is the adjustable weights

$h(x) = \sum_i x_i$ is the hidden layer activation function [20]

$\sigma(x) = \frac{1}{1 + e^{-xc}}$ is the output unit adaptive sigmoid activation function [21]
with c the adjustable term

The error function to be minimised is:

$$E(c, w_j) = \frac{1}{T} \sum_{t=1}^T (y_t - \tilde{y}_t(w_k, c))^2 \quad \text{with } y_t \text{ being the target value} \quad [22]$$

For example let us consider a Psi Sigma network which is fed with a $N+1$ dimensional input vector $x = (1, x_1, \dots, x_N)^T$. These inputs are weighted by K weight factors $w_j = (w_{0j}, w_{1j}, \dots, w_{Nj})^T$, $j = 1, 2, \dots, K$ and summed by a layer of K summing units, where K is the desired order of the network. So the output of the j -th summing unit, h_j in

the hidden layer, is given by: $h_j = w_j^T x = \sum_{k=1}^N w_{kj} x_k + w_{0j}$, $j = 1, 2, \dots, K$ while the output \tilde{y}

of the network is given by $\tilde{y} = \sigma\left(\prod_{j=1}^K h_j\right)$ (in our case we selected for σ the sigmoid

function $\sigma(x) = \frac{1}{1 + e^{-xc}}$ [21]). Note that by using products in the output layer we directly

incorporate the capabilities of higher order networks with a smaller number of weights and processing units. For example, a k -th degree HONN with d inputs needs

$\sum_{i=0}^k \frac{(d+i-1)!}{i!(d+1)!}$ weights if all products of up to k components are to be incorporated while

a similar Psi Sigma network needs only $(d+1)*k$ weights. Also note that the sigmoid function is neuron adaptive. As the network is trained not only the weights but also c in [21] is adjusted. This strategy seems to provide better fitting properties and increases

the approximation capability of a neural network by introducing an extra variable in the estimation, compared to classical architectures with sigmoidal neurons (Vecci *et al.* (1998)).

4.6.2 Empirical results of the Psi Sigma model

The price for the flexibility and speed of Psi Sigma networks is that they are not universal approximators. We need to choose a suitable order of approximation (or else the number of hidden units) by considering the estimated function complexity, amount of data and amount of noise present. To overcome this, our code runs simulations for orders two to six and we then select the best network based on statistical criteria on the test and training sample as for the RNN and HONN models. The characteristics of the network that we used are presented on Appendix A3. The trading strategy is that followed for the MLP. A summary of our findings is presented in table 9 below.

	NAIVE	MLP	SCE	GM	RNN	HONN	Psi Sigma
<i>Sharpe Ratio (excluding costs)</i>	1.83	2.57	2.26	2.09	2.57	2.58	2.55
<i>Annualised Volatility (excluding costs)</i>	11.6%	11.6%	11.6%	11.6%	11.6%	11.6%	11.6
<i>Annualised Return (excluding costs)</i>	21.3%	29.7%	26.3%	24.2%	29.8%	29.8%	29.5%
<i>Maximum Drawdown (excluding costs)</i>	-9.1%	-9.1%	-7.8%	-12.4%	-13.8%	-9.2%	-5.9%
<i>Positions Taken (annualised)</i>	109	118	143	162	124	129	133

Table 9: Trading performance results

Once again our results are similar to those obtained by Dunis and Williams (2002, 2003) with a MLP. The theoretical advantage of Psi Sigma and HONN models to capture higher order correlations in the data could make us believe that our results would be significantly better than the ones achieved with the MLP and RNN models and this was not the case. However, the other major theoretical advantage of Psi Sigma networks, namely their speed, was clearly confirmed as we achieved about the same results as the HONNs and the RNNs with respectively half and one tenth of their training time.

5. TRADING COSTS, FILTERS AND LEVERAGE

Up to now, we have presented the trading results of all our models without considering transaction costs. Since some of our models trade quite often, taking transaction costs into account might change the whole picture.

We therefore introduce transaction costs as well as a filtered trading strategy for each model. The aim is to devise a trading strategy filtering only those trades which have a high probability of being successful. This should help to reduce the negative effect of transaction costs as trades with an expected gain lower than the transaction costs should be omitted.

5.1 TRANSACTION COSTS

The transaction costs for a tradable amount, say USD 5-10 million, are about 3 pips (0.0003 EUR/USD) per trade (one way) between market makers. But, as noted by Dunis and Williams (2002, 2003), since the EUR/USD time series is a series of bid rates, we have to pay the costs only one and not two times per taken position.

With an average exchange rate of EUR/USD of 0.8971 for the out-of-sample period, a cost of 3 pips is equivalent to an average cost of 0.033% per position.

5.2 CONFIRMATION FILTER STRATEGIES

5.2.1 Confirmation Filters

We now introduce trading strategies devised to filter out those trades with expected returns below the 0.033% transaction cost. Due to the architecture of our models, the trading strategy for the MLP, the RNN, the Psi Sigma and the HONN networks consists of one single parameter while the strategy applied to the SCE and GM model uses two parameters. This is because of the additional available information which the SCE and GM models offer in terms of probability distributions.

Up to now, the trading strategies applied to the models use a zero threshold: they suggest to go long when the forecast is above zero and to go short when the forecast is below zero. In the following, we examine how the models behave if we introduce a threshold d around zero (see figure 9) and what happens if we vary that threshold.

The filter rule for the MLP, RNN, HONN and Psi Sigma models is presented in figure 9 below.

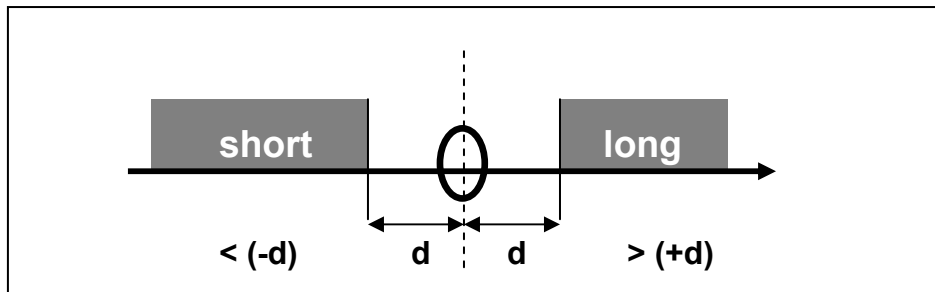


Fig. 9: Filtered trading strategy with one single parameter

Since the forecast of the SCE and GM models provide more information than the other models, we are able to introduce a second parameter for the trading strategy, which is the probability level.

As a result, and following Lindemann *et al.* (2004), all those trading signals are filtered out which are (a) not indicating a price move (in either direction) bigger than the threshold d (which has to be a multiple of the bin size in the SCE case) and in addition (b) not indicating a probability higher than $x\%$ for the forecast price move (which is the sum of the histogram bars for the SCE model and the space under the density function curve for the GM model). If both conditions are fulfilled at the same time for an up- as well as for a downmove, the strategy picks the trading signal with the higher probability.

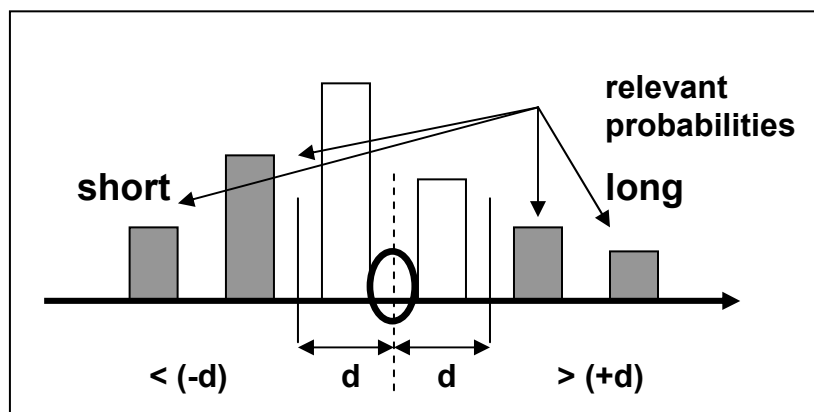


Fig. 10: Filtered trading strategy for the SCE model

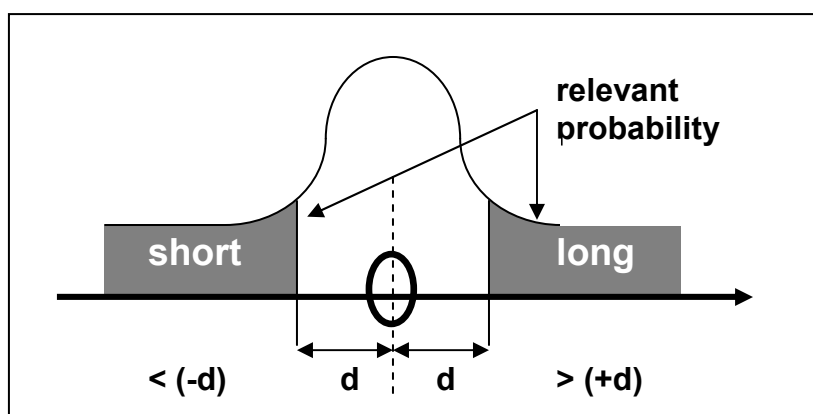


Fig. 11: Filtered trading strategy for the GM model

The thresholds chosen by Lindemann *et al.* (2004) for the GM, the Softmax and the MLP networks are given in table 10 below.

Model	Threshold (d)
MLP	= 0.00
SCE	= 0.25 (move size > 0.3%)
GM	= 0.00 (probability > 0.0%)

Table 10: Chosen parameters in Lindemann *et al.* (2004)

5.2.2 Empirical Results of the RNN, HONN and Psi Sigma models

Following the methodology of Lindemann *et al.* (2004), we proceed with the selection of the optimal thresholds. Taking the test period results, we choose the threshold that gives the higher return and Sharpe ratio. Our chosen parameters are presented in the table below while the detailed results leading to their choice are documented in Appendix A.4.

Model	Threshold (d)
RNN	= 0.00
HONN	= 0.00
Psi Sigma	= 0.00

Table 11: Chosen parameters for each trading strategy

For all networks, we leave the threshold at zero ($d=0.0$) since the profit on the test dataset is largest at this value. The value of $d=0.1$ looks promising in the case of Psi Sigma from a Sharpe ratio point of view but the lower level of profit deterred us from choosing it as a threshold. We stick therefore to $d=0.0$ in all cases.

A summary of the out-of-sample trading performance of our three models benchmarked against the Naïve, the MLP, the SCE and the GM networks using the selected thresholds as reported by Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004) is presented in table 12 below.

We can see that the MLP, the RNN, the HONN and the Psi Sigma networks show about the same performance based on the annualised return and the Sharpe ratio. However, it is worth mentioning that the time used to derive these results with the Psi Sigma network is half that needed with HONNs and one tenth that needed with RNNs.

	NAIVE	MLP	SCE	GM	RNN	HONN	Psi Sigma
Sharpe Ratio (excluding costs)	1.83	2.57	2.67	2.09	2.57	2.58	2.55
Annualised Volatility (excluding costs)	11.6%	11.6%	8.5%	11.6%	11.6%	11.6%	11.6%
Annualised Return (excluding costs)	21.3%	29.7%	22.7%	24.2%	29.8%	29.8%	29.5%
Maximum Drawdown (excluding costs)	-9.1%	-9.1%	-5.7%	-12.4%	-13.8%	-9.2%	-5.9%
Positions Taken (annualised)	109	118	120	162	124	129	133
Transaction costs	3.6%	3.9%	3.9%	5.3%	4.0%	4.3%	4.4%
Annualised Return (including costs)	17.7%	25.8%	18.8%	18.9%	25.7%	25.6%	25.1%

Table 12: Out-of-sample results for the chosen parameters

5.3 Leverage to exploit high Sharpe ratios

As we have seen, the application of a filtered trading strategy does not improve the results in this case, since all 3 models stick to a threshold of zero. The question then is whether we can gain higher risk-adjusted profits by using leverage.

The leverage factors applied are calculated in such a way that each model has a common volatility of 10%¹¹ on the test data set.

Since we now have additional information (which is the leveraged trading results based on the test dataset), we can rethink our former choice of thresholds. The thresholds that we select in the end are presented in the table below while an insight about our selection process can be found in Appendix A.4.

Model	Threshold (d)
RNN	= 0.05
HONN	= 0.00
Psi Sigma	= 0.00

Table 13: Parameters for the leveraged trading strategies

For the HONN and the Psi Sigma network we leave the threshold at 0 as the profit is maximized for this value on the test dataset. For the RNN we choose $d = 0.05$ which gives the highest profit and Sharpe ratio on the test dataset and filters out only 3 trades per year.

The thresholds reported by Lindemann *et al.* (2004) who follow the same methodology are presented in table 14 below.

Model	Threshold (d)
MLP	= 0.05
SCE	= 0.25 (move size > 0.3%)
GM	= 0.35 (probability = 0.25)

Table 14: Parameters for the leveraged trading strategies

¹¹ Since most of the models (using a threshold of zero) have a volatility of about 10%, we have chosen this level as our basis. The leverage factors retained are given in table 12 below.

The transaction costs are calculated by taking 0.033% per position into account, while the costs of leverage (interest payments for the additional capital) are calculated with 4% p.a. (that is 0.016% per trading day¹²). Our final results are presented in table 15 below.

	NAIVE	MLP	SCE	GM	RNN	HONN	Psi Sigma
<i>Sharpe Ratio (excluding costs)</i> ¹³	1.83	2.30	2.67	3.80	2.57	2.58	2.55
<i>Annualised Volatility (excluding costs)</i>	11.9%	13.4%	12.5%	12.2%	11.9%	12.3%	11.9%
<i>Annualised Return (excluding costs)</i>	21.8%	30.8%	33.2%	46.4%	30.7%	31.7%	30.4%
<i>Maximum Drawdown (excluding costs)</i>	-9.3%	-10.3%	-8.5%	-11.3%	-14.3%	-9.8%	-6.1%
<i>Leverage Factor</i>	1.03	1.62	1.46	3.99	1.03	1.03	1.03
<i>Positions Taken (annualised)</i>	109	89	120	68 ¹⁴	121	129	133
<i>Transaction and leverage costs</i>	3.7%	6.1%	7.1%	12.5%	4.0%	4.3%	4.6%
<i>Annualised Return (including costs)</i>	18.1%	24.7%	26.1%	33.9%	26.7%	27.0%	25.8%

Table 15: Trading performance - final results¹⁵

As can be seen from table 15, GM networks are able to take advantage of the combination of a confirmation filter and leverage and to deliver higher Sharpe ratios and returns. HONNs achieve the highest annualised return net of transaction costs among the other five competing models while the Psi Sigma, the RNN and the SCE models achieve similar performances. It seems that the ability of HONNs and Psi Sigma to capture higher order correlations within our dataset and the ability of the RNN

¹² The interest costs are calculated by considering a 4% interest rate p.a. divided by 252 trading days. In reality, leverage costs also apply during non-trading days so that we should calculate the interest costs using 360 days per year. But for the sake of simplicity, we use the approximation of 252 trading days to spread the leverage costs of non-trading days equally over the trading days. This approximation prevents us from keeping track of how many non-trading days we hold a position.

¹³ The calculation is done without transaction and leverage costs due to a better comparability to other published numbers (which are generally calculated in this way).

¹⁴ The SCE and GM committees have actually taken more trades than reported in the table above (e.g. the GM model has actually taken 134 positions). The reason why Lindemann *et al.* (2004) report a smaller number of trades is that SCE and GM committees are able to invest less than 100% of their total capital per position (this is due to the fact that the position size is determined by the average number of committee members generating a trading signal). Since our transaction costs of 0.033% per position are based on the assumption of 100% of invested total capital, we have to recalculate the 134 positions of partially invested total capital into the equivalent number of positions with 100% of invested capital (which are the above shown 68 positions).

¹⁵ Not taken into account are the following effects:

- a) The interest that could be earned during times where the capital is not traded [non-trading days] and could therefore be invested;
- b) The SCE and GM committees are not forced to use 100% of their capital when trading (leaving out a leverage factor <1), since the amount is determined by the average forecast of the 30 models. If the committees invest therefore only a few per cent of the capital available but apply the leverage factor (>1), the additional capital has not to be borrowed (since there is still own money available) and therefore leverage costs would not be incurred. Those 'savings' are not taken into account here.

to embody short term memory does not help them to exploit the leverage and the confirmation filter and to achieve higher trading performance. Overall, our three models perform remarkably well (see table 12), however they do not manage to take advantage of more sophisticated trading strategies using confirmation filters and leverage contrary to density distribution networks (see table 15).

6. CONCLUDING REMARKS

In this paper, we apply Recurrent, Higher Order and Psi Sigma neural networks to a one-day-ahead forecasting and trading task of the EUR/USD time series. We develop these different prediction models over the period October 1994 - May 2000 and validate their out-of-sample trading efficiency over the following period from May 2000 through July 2001. Our results are benchmarked against those of the Gaussian Mixture, the Softmax Entropy and the Multi-layer Perceptron models presented by Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004) who study the same series over the same time period.

Trading strategies that should filter out potentially unsuccessful trades by using a confirmation threshold have not worked out and the Psi Sigma, HONN and RNN models fail to exploit leverage for the asset and time period under review.

Nevertheless, the trading results of the Psi Sigma, HONN and RNN models are similar to the best model of Dunis and Williams (2002, 2003), the MLP, when applied without confirmation filter and leverage. When more sophisticated trading strategies are applied, our results are not improved significantly although HONNs still perform remarkably.

It is also important to note that the Psi Sigma network which presents similar results to HONNs and RNNs needs far less training time than all other network architectures, a much desirable feature in a real-life quantitative investment and trading environment: in the circumstances, our results should go some way towards convincing a growing number of quantitative fund managers to experiment beyond the bounds of the traditional MLP model.

APPENDIX

A.1 Performance measures

The performance measures are calculated as follows¹⁶:

Performance Measure	Description	
Annualised Return	$R^A = 252 * \frac{1}{N} \sum_{t=1}^N R_t$ <p>with R_t being the daily return</p>	[14]
Cumulative Return	$R^C = \sum_{t=1}^N R_t$	[15]
Annualised Volatility	$\sigma^A = \sqrt{252} * \sqrt{\frac{1}{N-1} * \sum_{t=1}^N (R_t - \bar{R})^2}$	[16]
Sharpe Ratio	$SR = \frac{R^A}{\sigma^A}$ <p>Maximum negative value of $\sum (R_t)$ over the period</p>	[17]
Maximum Drawdown	$MD = \text{Min}_{i=1, \dots, t; t=1, \dots, N} \left(\sum_{j=i}^t R_j \right)$	[18]

Table 16: Trading simulation performance measures

A.2 Results of alternative benchmark models

	NAIVE	MACD	ARMA	LOGIT	MLP
Sharpe Ratio (excluding costs)	1.83	0.97	1.10	1.81	2.57
Annualised Volatility (excluding costs)	11.6%	11.7%	11.7%	11.6%	11.6%
Annualised Return (excluding costs)	21.3%	11.3%	12.9%	21.1%	29.7%
Maximum Drawdown (excluding costs)	-9.1%	-7.8%	-10.1%	-5.8%	-9.1%
Positions Taken (annualised)	109	22	112	123	118

Table 17: Out-of-sample trading performance results for traditional models as reported by Dunis and Williams (2003, table 1.20, p. 35)

¹⁶ For more details see Dunis and Williams (2002, 2003).

A.3 Networks characteristics

Below are presented the characteristics of the networks for the different architectures that presented the best statistical performance on the training and on the test sub-period and that we used on this paper.

Parameters	Reccurent	HONNs	Psi Sigma
<i>Learning algorithm</i>	<i>Gradient descent</i>	<i>Gradient descent</i>	<i>Gradient descent</i>
<i>Learning rate</i>	<i>0.001</i>	<i>0.001</i>	<i>0.5</i>
<i>Momentum</i>	<i>0.003</i>	<i>0.003</i>	<i>0.5</i>
<i>Iteration steps</i>	<i>500</i>	<i>500</i>	<i>500</i>
<i>Initialisation of weights</i>	<i>N(0,1)</i>	<i>N(0,1)</i>	<i>N(0,1)</i>
<i>Input nodes</i>	<i>10</i>	<i>10</i>	<i>10</i>
<i>Hidden nodes (1layer)</i>	<i>5</i>	<i>0</i>	<i>5</i>
<i>Output node</i>	<i>1</i>	<i>1</i>	<i>1</i>

Table 18: Network characteristics

A.4 Empirical results

The table below shows the results of the filtered trading strategy applied to the test dataset for different values of d . We choose the threshold that gives the highest return.

Selection of the optimal threshold based on the test period	Threshold									
	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
RNN	28.4% (2.93)	27.8% (2.39)	12.9% (2.87)	4.36% (1.81)	1.43% (0.98)	-0.6% (-1.1)	-0.5% (-0.9)	-0.1% (-0.2)	0.2% (1.28)	0.3% (1.29)
HONN	22.5% (2.31)	17.0% (1.94)	13.9% (1.76)	5.99% (0.88)	-4.8% (-0.9)	-0.1% (-0.1)	-0.2% (-0.1)	0.5% (0.2)	-0.9% (-0.4)	0.6% (0.37)
Psi Sigma	24.9% (2.51)	21.8% (2.12)	14.1% (3.74)	4.3% (2.37)	1.44% (1.3)	0.52% (0.92)	0.51% (0.92)	0.52% (0.92)	0.00% (0.00)	0.00% (0.00)

Table 19: Results for alternative threshold values

Note: The entries represent the annualized return values while the values in parenthesis represent the Sharpe ratio.

The table below shows the results of the filtered trading strategy applied to the test dataset for different values of d after taking leverage into account. We choose the threshold that gives the highest return.

Selection of the optimal threshold based on the test period	Threshold									
	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
RNN	29.2% (2.93)	29.7% (4.3)	13.2% (2.87)	4.5% (1.81)	1.47% (0.98)	-0.6% (-1)	-0.5% (-0.9)	-0.1% (-0.2)	0.3% (1.3)	0.27% (1.3)
HONN	23.2% (2.31)	17.5% (1.94)	14.3% (1.76)	6.17% (0.88)	-4.9% (-0.3)	-0.1% (-0.2)	-0.2% (-0.1)	0.52% (0.2)	-0.9% (-0.4)	0.7% (0.4)
Psi Sigma	25.1% (2.5)	22.4% (2.1)	14.3% (3.74)	4.45% (2.5)	1.48% (1.3)	0.52% (0.93)	0.52% (0.93)	0.52% (0.93)	0.00% (0.00)	0.00% (0.00)

Table 20: Results for alternative threshold values

Note: The entries represent the annualized return values while the values in parenthesis represent the Sharpe ratio.

REFERENCES

Adam, O., Zarader, L. and Milgram, M. (1994), 'Identification and Prediction of Non-Linear Models with Recurrent Neural Networks', *Laboratoire de Robotique de Paris*.

Connor, J. and Atlas, L. (1993), 'Recurrent Neural Networks and Time Series Prediction', *Proceedings of the International Joint Conference on Neural Networks*, 301-306.

Dunis, C. and Huang, X. (2002), 'Forecasting and Trading Currency Volatility: An Application of Recurrent Neural Regression and Model Combination', *Journal of Forecasting*, 21, 5, 317-354.

(DOI: 10.1002/0470013265.ch4)

Dunis, C., Laws, J. and Evans B. (2006a), 'Trading Futures Spreads: An application of Correlation and Threshold Filters', *Applied Financial Economics*, 16, 1-12.

(DOI: 10.1080/09603100500426432)

Dunis, C., Laws, J. and Evans B. (2006b), 'Modelling and Trading the Gasoline Crack Spread: A Non-Linear Story', *Derivatives Use, Trading and Regulation*, 12, 126-145.

(DOI: 10.1057/palgrave.dutr.1840046)

Dunis, C. and Williams, M. (2002), 'Modelling and Trading the EUR/USD Exchange Rate: Do Neural Network Models Perform Better?', *Derivatives Use, Trading and Regulation*, 8, 3, 211-239.

(DOI: 10.1002/for.935)

Dunis, C. and Williams, M. (2003), 'Applications of Advanced Regression Analysis for Trading and Investment', in C. Dunis, J. Laws and P. Naïm [eds.], *Applied Quantitative Methods for Trading and Investment*, John Wiley, Chichester.

(DOI: 10.1002/0470013265.ch1)

Elman, J. L. (1990), 'Finding Structure in Time', *Cognitive Science*, 14, 179-211.

(DOI :10.1016/0364-0213(90)90002-E)

Hussain, A., Ghazali, R and Al-Jumeily D. (2006), 'Dynamic Ridge Polynomial Neural Network for Financial Time Series Prediction', IEEE International conference on Innovation in Information Technology, IIT06, Dubai.

Ghazali, R., Hussain, A. and Merabti, M. (2006), 'Higher Order Neural Networks for Financial Time Series Prediction', *The 10th IASTED International Conference on Artificial Intelligence and Soft Computing*, Palma de Mallorca, Spain, 119-124.

Ghosh, J. and Shin, Y. (1992) 'Efficient Higher-Order Neural Networks for Classification and Function Approximation', *International Journal of Neural Systems*, 3, 4, 323-350.

(DOI: 10.1142/S0129065792000255)

Giles, L. and Maxwell, T. (1987) 'Learning, Invariance and Generalization in Higher Order Neural Networks', *Applied Optics*, 26, 4972-4978.

Fulcher, J., Zhang, M. and Xu, S. (2006), *The Application of Higher-Order Neural Networks to Financial Time Series*, Artificial Neural Networks in Finance and Manufacturing, Hershey, PA: Idea Group, London.

Husmeier, D. (1999), *Neural Networks for Conditional Probability Estimation - Forecasting Beyond Point Predictions (Perspectives in Neural Computing)*, Springer, London.

Kaasra, I. and Boyd, M. (1996), 'Designing a Neural Network for Forecasting Financial and Economic Time Series', *Neurocomputing*, 10, 215-236.

(DOI: 10.1016/0925-2312(95)00039-9)

Kamijo, K. and Tanigawa, T. (1990), 'Stock Price Pattern Recognition: A Recurrent Neural Network Approach', *In Proceedings of the International Joint Conference on Neural Networks*, 1215-1221.

Karayiannis, N. and Venetsanopoulos, A. (1994), 'On The Training and Performance of High-Order Neural Networks', *Mathematical Biosciences*, 129, 143-168.

(DOI: 10.1016/0025-5564(94)00057-7)

Knowles, A., Hussein, A., Deredy, W., Lisboa, P. and Dunis, C. L. (2005), 'Higher-Order Neural Networks with Bayesian Confidence Measure for Prediction of EUR/USD Exchange Rate', *CIBEF Working Papers*. Available at www.cibef.com.

Kosmatopoulos, E., Polycarpou, M., Christodoulou, M. and Ioannou, P. (1995), 'High-Order Neural Network Structures for Identification of Dynamical Systems', *IEEE Transactions on Neural Networks*, 6, 422-431.

Lindemann, A., Dunis, C., and Lisboa P. (2004), 'Level Estimation, Classification and Probability Distribution Architectures for Trading the EUR/USD Exchange Rate'. *Neural Network Computing & Applications*, 14, 3, 256-271.

(DOI: 10.1007/s00521-004-0462-8)

Lisboa, P. J. G. and Vellido, A. (2000), 'Business Applications of Neural Networks', vii-xxii, in P. J. G. Lisboa, B. Edisbury and A. Vellido [eds.] *Business Applications of Neural Networks: The State-of-the-Art of Real-World Applications*, World Scientific, Singapore.

(DOI: 10.1016/S0377-2217(02)00302-8)

Psaltis, D., Park, C. and Hong, J. (1988), 'Higher Order Associative Memories and their Optical Implementations.', *Neural Networks*, 1, 149-163.

Redding, N., Kowalczyk, A. and Downs, T. (1993), 'Constructive Higher-Order Network Algorithm that is Polynomial Time', *Neural Networks*, 6, 997-1010.

(DOI: 10.1016/S0893-6080(03)00188-6)

Shapiro, A. F. (2000), 'A Hitchhiker's Guide to the Techniques of Adaptive Nonlinear Models', *Insurance, Mathematics and Economics*, 26, 119-132.

(DOI: 10.1016/S0167-6687(99)00058-X)

Shin, Y. and Ghosh, J. (1991) 'The Pi-Sigma Network: An Efficient Higher-Order Neural Network for Pattern Classification and Function Approximation', *Proceedings IJCNN*, Seattle, July, 13-18.

Tenti, P. (1996), 'Forecasting Foreign Exchange Rates Using Recurrent Neural Networks', *Applied Artificial Intelligence*, 10, 567-581.

(DOI: 10.1080/088395196118434)

Tino, P., Schittenkopf, C. and Doffner, G. (2001), 'Financial Volatility Trading Using Recurrent Networks', *IEEE Transactions in Neural Networks*, 12, 4, 865-874.

Vecci, L., Piazza, F. and Uncini, A. (1998), 'Learning and Approximation Capabilities of Adaptive Spline Activation Neural Networks', *Neural Networks*, 11, 259-270.

(DOI: 10.1016/S0893-6080(97)00118-4)

Zhang, M., Xu, S., X. and Fulcher, J. (2002), 'Neuron-Adaptive Higher Order Neural-Network Models for Automated Financial Data Modelling', *IEEE Transactions on Neural Networks*,13,1, 188-204.

(DOI:10.1109/72.977302)