
Quantitative Finance
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/rquf20

# Higher order and recurrent neural architectures for trading the EUR/USD exchange rate

Christian L. Dunis [a] , Jason Laws [a] & Georgios Sermpinis [a]
[a] Liverpool Business School, Liverpool John Moores University , Liverpool L3 5UZ, UK
Published online: 06 Apr 2010.

*Quantitative Finance*, Vol. 11, No. 4, April 2011, 615–629



# Higher order and recurrent neural architectures for trading the EUR/USD exchange rate

CHRISTIAN L. DUNIS, JASON LAWS* and GEORGIOS SERMPINIS

Liverpool Business School, Liverpool John Moores University, Liverpool L3 5UZ, UK

(*Received 2 April 2008; in final form 25 September 2009*)

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 Multilayer Perceptron (MLP), the Softmax and the Gaussian Mixture (GM) models. 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 all do 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*: Quantitative trading strategies; Volatility modelling; Risk management; Options volatility



## 1. Introduction

Neural networks are an emergent technology with an increasing number of real-world applications, including Finance (Lisboa and Vellido 2000). However, their numerous limitations often create 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 Sigma 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 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 with 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 perfomance 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 organized as follows. In section 2, we present the literature relevant to the Recurrent 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.

*Corresponding author. Email: j.laws@ljmu.ac.uk

*Quantitative Finance*
ISSN 1469–7688 print/ISSN 1469–7696 online © 2011 Taylor & Francis
http://www.informaworld.com
DOI: 10.1080/14697680903386348

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.

A good literature review on MLP, SCE and GM networks can be found in Dunis and Williams (2002, 2003) and Lindemann *et al*. (2004, 2005). Furthermore, Ozun (2006) forecasts with great statistical accuracy the EUR/USD exchange rate from 2003 to 2006 with feedfoward neural networks.

Other adaptive methods have been applied to the EUR/USD exchange rate with some success. Austin *et al*. (2004) use genetic algorithms (GAs) and evolutionary reinforced learning (EVL) to model and trade profitably among other exchange rates the EUR/USD intraday from January 1999 to January 2002. Hryshko and Downs (2006) also use GAs and EVL to make profitable trades on the EUR/USD in 2002. Jeng *et al*. (2006) introduce DNA forecasting which proves superior to an ARIMA model in forecasting the EUR/USD series. Ullrich *et al*. (2007) forecast and trade the EUR/USD with seven different support vector machine models using as a benchmark a series of traditional forecasting techniques: they find that the hyperbolic support vector machine (SVM) outperform all other models in terms of statistical performance and trading efficiency.

In any case neural network regression models have demonstrated their ability to help develop efficient decision trading tools, as reported by Dunis and Williams (2002, 2003) and Lindemann *et al*. (2004, 2005). They also have the great advantage of not making any assumption about the data generating process for the financial time series under review.

In this short literature review we focus primarily upon the three novel architectures applied in this research, namely RNNs, HONNs and Psi Sigma networks.

RNNs have an activation feedback which embodies short-term memory allowing them to learn extremely complex temporal patterns. Their superiority against feedfoward networks when performing nonlinear time series prediction is well documented by Connor and Atlas (1993) and Adam *et al*. (1994). In financial applications, Kamijo and Tanigawa (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 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*. 1988), its use in finance so far has been limited. This 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. More recently, Dunis *et al*. (2009) verified the robustness and stability of HONNs and MLPs in forecasting the EUR/USD ECB fixing over the period January 1999 to August 2008.

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 and Ghosh (1991) and Ghosh and Shin (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 prices.† All time series are daily closing data obtained

†EUR/USD is quoted as the number of USD per Euro: for example, a value of 1.2657 is USD 1.2657 per Euro. The EUR/USD exchange rate only exists from 4 January 1999: it was extrapolated 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.

Table 1. The EUR/USD dataset.

| 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 |

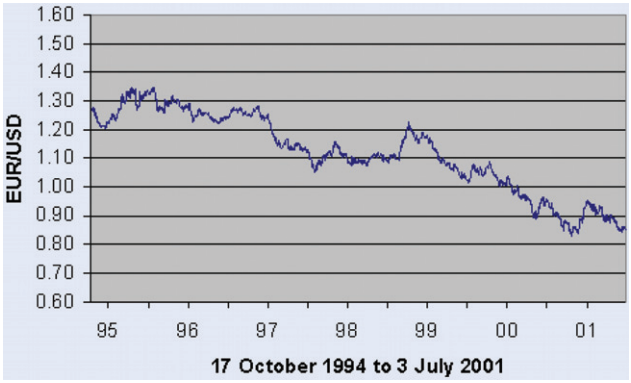


Figure 1. EUR/USD London daily closing prices (total dataset).

from a historical database provided by Datastream and used by Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004) (table 1, figure 1).

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

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 non-stationary 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 return.†

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

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

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.

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

$$yc = 10 \text{ year benchmark bond yields} - 3 \text{ month interest rates.} \tag{2}$$

Following Dunis and Williams (2002, 2003) and Lindemann *et al.* (2004), we divide our dataset as shown in table 3.

## 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 nodes corresponds to the number of explanatory variables). The last layer is called the output layer (the number of nodes corresponds to the number of response variables). An intermediary layer of nodes, the hidden layer, separates the input from the output layer. The 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 the weights in such a 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 errors‡ (Shapiro 2000). The learning algorithm simply tries to find those weights which optimize 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 three 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 optimized by maximizing the forecasting accuracy for the test dataset. Finally, the predictive value

†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.

‡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).

Table 2. Explanatory variables and Datastream mnemonics.

| 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 year (DS)—red. yield | FRBRYLD | 2 |
| 7 | Italy benchmark bond 10 year (DS)—red. yield | ITBRYLD | 6 |
| 8 | Japan benchmark bond—RYLD.10 year (DS)—red. yield | JPBRYLD | 9 |
| 9 | Nikkei 225 stock average—price index | JAPDOWA | 1 |
| 10 | Nikkei 225 stock average—price index | JAPDOWA | 15 |

Table 3. The neural networks datasets.

| 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 |

of the model is evaluated by applying it to the validation dataset (out-of-sample dataset).

As mentioned before, variable selection was taken from Dunis and Williams (2002, 2003). As for the codes used, they were developed in MatLab at CIBEF.

### 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 2.† In this figure: $x_t^{[n]}$ $(n = 1, 2, \ldots, k+1)$ are the model inputs (including the input bias node) at time $t$, $h_t^{[m]}$ $(m = 1, 2, \ldots, j+1)$ 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, the circle with a 'squiggle' inside is the transfer sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

and the circle with a slash inside is a linear function:

$$F(x) = \sum_i x_i \tag{4}$$

The error function to be minimized is

$$E(u_{jk}, w_j) = \frac{1}{T}\sum_{t=1}^{T} (y_t - \tilde{y}_t(u_{jk}, w_j))^2 \tag{5}$$

with $y_t$ being the target value.

**4.1.2. Empirical results of the MLP model.** The results for the MLP achieved by Dunis and Williams (2002, 2003)

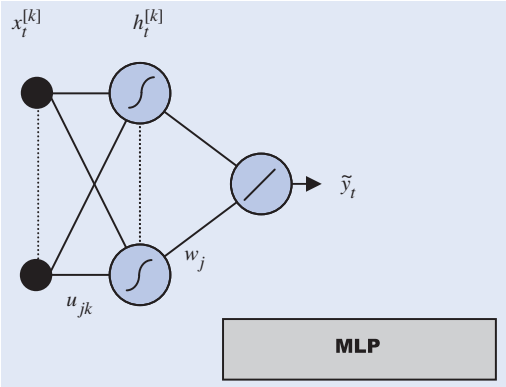


Figure 2. A single output, fully connected MLP model.

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.

### 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, six 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.

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

Table 4. Trading performance of the benchmark models.

| | NAIVE | MLP |
|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% |
| Annualized return (excluding costs) | 21.3% | 29.7% |
| Maximum drawdown (excluding costs) | −9.1% | −9.1% |
| Taken positions (annualized[a]) | 109 | 118 |

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

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 that 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 on 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)).

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-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) (figure 3). In figure 3, $x_t^{[n]}$ $(n = 1, 2, \ldots, k+1)$ are the model inputs (including the input bias node) at time $t$, $h_t^{[m]}$ $(m = 1, 2, \ldots, j+1)$ are the hidden nodes' outputs (including the hidden bias node), $\tilde{y}_t^{[g]}$ $(g = 1, 2, \ldots, q)$ is the SCE model output before applying the Softmax function, $\tilde{z}_t^{[g]}$ $(g = 1, 2, \ldots, q)$ is the network value at the output node $g$, $u_{jk}$ and $w_{gj}$ are the network weights, the circle with a 'squiggle' inside is the transfer sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{6}$$

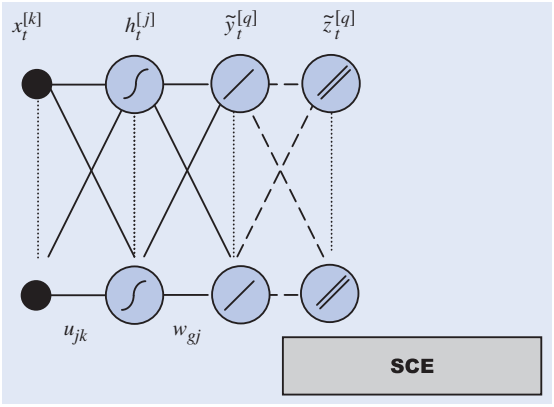


Figure 3. A single output, fully connected SCE model.

the circle with a slash inside is a linear function:

$$F(x) = \sum_i x_i, \tag{7}$$

and the circle with a double slash inside is the Softmax function:

$$A(g) = \tilde{z}_g = \frac{\exp(\tilde{y}_g)}{\sum_{g_1} \exp(\tilde{y}_{g_1})} \tag{8}$$

with $\tilde{y}_g$ being the output of the linear function.

The error function to be minimized is

$$E(u_{jk}, w_{gj}) = \sum_{t=1}^{T} \sum_{g=1}^{q} y_{tg} \cdot \log\left(\frac{y_{tg}}{\tilde{z}_{tg}(u_{jk}, w_j)}\right) \tag{9}$$

with $y_{tg}$ being the target value.

**4.2.2. Empirical results of the SCE model.** Since neural networks start with a random initialization 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 the last three of six values of the output vector (since those three 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.

### *4.3. The Gaussian mixture model*

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

Table 5. Trading performance results.

| | NAIVE | MLP | SCE |
|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 | 2.26 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% | 11.6% |
| Annualized return (excluding costs) | 21.3% | 29.7% | 26.3% |
| Maximum drawdown (excluding costs) | −9.1% | −9.1% | −7.8% |
| Positions taken (annualized) | 109 | 118 | 143 |

about the GM network forecasting ability in finance was given by Lindeman *et al.* (2005).

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 training data set 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 value† 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 optimize the cost function (that is, to maximize the sum of likelihood values), the weights $u_{jk}$ and $w_{ij}$, determining the location of the normal distribution centres ($\mu_t$), have to be adapted so that the distance between $y_t$ and $\mu_t$ 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 4 which illustrates 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 dependent variable $(x, y)$ are used as input data, since the aim is not to predict $y$ but its density distribution $P(y \mid x)$ and the corresponding likelihood value. Third, the network uses Gaussian distributions in the second hidden layer.

The following functions are applied within the GM model:

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

$y_t$ the argument of the density function conditional on the values of the inputs (note that the weights of $y_t$ are fixed to 1‡)

$u_{jk}$, $w_{ji}$ the network weights

$\beta_i$ define the inverse widths of the Gaussian distributions

$a_i$ the mixing coefficients, with $\sum_i a_i = 1$

$i$ the number of applied Gaussian mixture distributions

$j$ the number of applied network weights $w_j$

$k$ the number of applied network weights $u_{jk}$

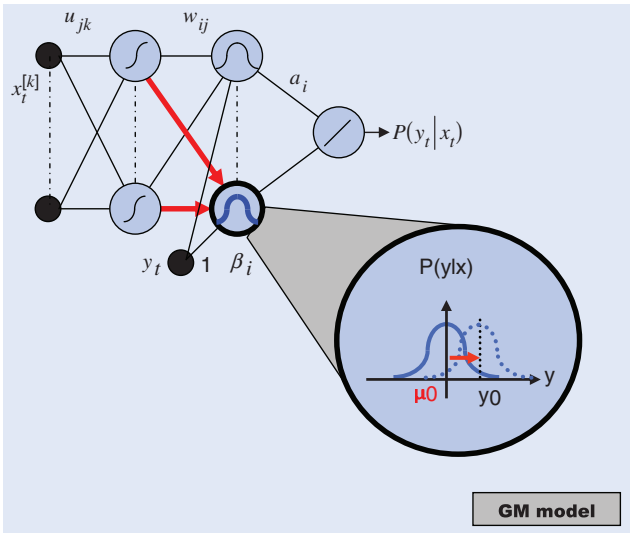


Figure 4. GM network architecture.

In figure 4 the circle with the 'hump' inside is the Gaussian distribution:

$$G_{\beta_i}(y_t - \mu_i) = \sqrt{\frac{\beta_i}{2\pi}} \exp\left(-\frac{\beta_i \cdot (y_t - \mu_i)^2}{2}\right) \qquad (10)$$

with

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

the circle with the 'squiggle' inside is the Sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} \qquad (11)$$

and the circle with the slash inside is the linear function:

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

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

‡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 $y_t$.

Table 6. Trading performance results.

| | NAIVE | MLP | SCE | GM |
|---|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 | 2.26 | 2.09 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% | 11.6% | 11.6% |
| Annualized return (excluding costs) | 21.3% | 29.7% | 26.3% | 24.2% |
| Maximum drawdown (excluding costs) | −9.1% | −9.1% | −7.8% | −12.4% |
| Positions taken (annualized) | 109 | 118 | 143 | 162 |

The error function to be minimized is

$$E(u_{jk}, w_{ij}, \beta_i, a_i) = -\frac{1}{T}\sum_{t=1}^{T} \ln(P(y_t|x_t, u_{jk}, w_{ij}, \beta_i, a_i)), \tag{13}$$

with $y_t$ being the target value.

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) optimize its parameters as well as the stopping point† 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 minimize the chance of picking 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. 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 move in the exchange rate. This should be helpful when applying more sophisticated strategies, which are investigated in detail in section 5.

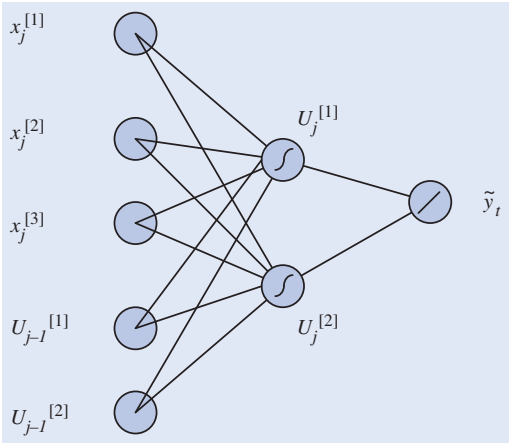


Figure 5. Elman recurrent neural network architecture with two nodes in the hidden layer.

### 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 nonlinear time series, has been well documented in the past. However, as described by 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 with 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 in figure 5. In this figure $x_t^{[n]}(n = 1, 2, \ldots, k+1)$, $u_t^{[1]}$ and $u_t^{[2]}$ 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 $w_t^{[n]}$ $(n = 1, 2, \ldots, k+1)$ are the network weights, $U_t^{[f]}$ $(f = 1, 2)$ are the output of the hidden nodes at time $t$, the circle with a 'squiggle' inside is the transfer sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{14}$$

and the circle with a slash inside is the linear output function:

$$F(x) = \sum_i x_i. \tag{15}$$

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

Table 7. Trading performance results.

| | NAIVE | MLP | SCE | GM | RNN |
|---|---|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 | 2.26 | 2.09 | 2.57 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% | 11.6% | 11.6% | 11.6% |
| Annualized 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 (annualized) | 109 | 118 | 143 | 162 | 124 |

The error function to be minimized is

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

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 figure 5).

**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 period.† 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 shown in appendix A.3.

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

### *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' 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).

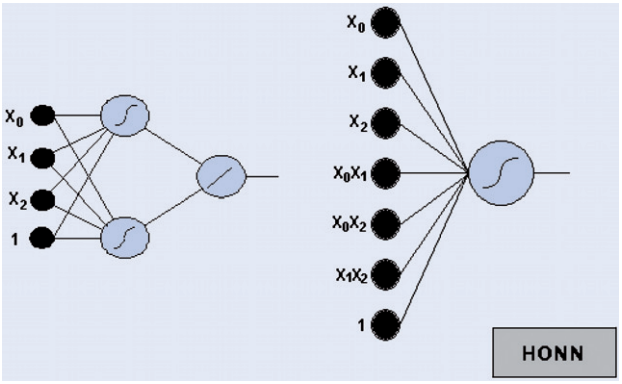


Figure 6. Left: MLP with three inputs and two hidden nodes. Right: second-order HONN with three inputs.

**4.5.1. The HONN architecture.** While they have already experienced some success in the field of pattern recognition and associative recall,‡ the use of HONNs in finance is not yet widespread. The architecture of a three input second order HONN is shown in figure 6. In this figure $x_t^{[n]}$ $(n = 1, 2, \ldots, k+1)$ 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, the filled circles are the model inputs, the circle with a 'squiggly' line inside is the transfer sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{17}$$

and the circle with a slash inside is a linear function:

$$F(x) = \sum_i x_i. \tag{18}$$

The error function to be minimized is

$$E(u_{jk}, w_j) = \frac{1}{T}\sum_{t=1}^{T}(y_t - \tilde{y}_t(u_{jk}, w_j))^2 \tag{19}$$

with $y_t$ being the target value.

HONNs use joint activation functions; this technique reduces the need to establish the relationships between

†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.
‡Associative recall is the act of associating two seemingly unrelated entities, such as smell and colour. For more information, see Karayiannis and Venetsanopoulos (1994).

Table 8. Trading performance results.

| | NAIVE | MLP | SCE | GM | RNN | HONN |
|---|---|---|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 | 2.26 | 2.09 | 2.57 | 2.58 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% | 11.6% | 11.6% | 11.6% | 11.6% |
| Annualized return (excluding costs) | 21.3% | 29.7% | 26.3% | 24.2% | 29.8% | 29.8% |
| Maximum drawdown (excluding costs) | −9.1% | −9.1% | −7.8% | −12.4% | −13.8% | −9.2% |
| Positions taken (annualized) | 109 | 118 | 143 | 162 | 124 | 129 |

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 four 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 while the characteristics of the network that we used are give in appendix A.3.

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 feedfoward fully connected HONNs. First introduced by Ghosh and Shin (1992), the Psi Sigma network utilizes product cells as the output units to indirectly incorporate the capabilities of higher order networks while using fewer 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 greatly reducing the training time. Moreover, the activation function of the nodes in the hidden layer is the summing function

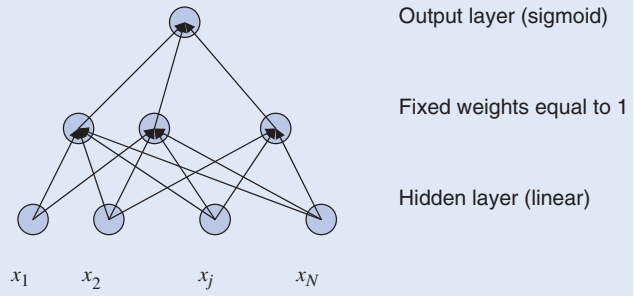


Figure 7. A Psi Sigma network with one output layer.

while the activation function of the output layer is a sigmoid. Figure 7 shows a Psi Sigma network with one output layer. In this figure $x_t$ $(n = 1, 2, \ldots, k+1)$ are the model inputs (including the input bias node), $\tilde{y}_t$ is the Psi Sigma output, $w_j$ are the adjustable weights,

$$h(x) = \sum_i x_i \tag{20}$$

is the hidden layer activation function, and

$$\sigma(x) = \frac{1}{1 + e^{-xc}} \tag{21}$$

is the output unit adaptive sigmoid activation function with $c$ the adjustable term.

The error function to be minimized is

$$E(c, w_j) = \frac{1}{T}\sum_{t=1}^{T}(y_t - \tilde{y}_t(w_k, c))^2 \tag{22}$$

with $y_t$ being the target value.

For example, let us consider a Psi Sigma network which is fed with an $N+1$-dimensional input vector $x = (1, x_1, \ldots, x_N)^T$. These inputs are weighted by $K$ weight factors $w_j = (w_{0j}, w_{1j}, \ldots, w_{Nj})^T$, $j = 1, 2, \ldots, K$, 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

$$h_j = w_j^T x = \sum_{k=1}^{N} w_{kj}x_k + w_{oj}, \quad j = 1, 2, \ldots, K,$$

while the output $\tilde{y}$ of the network is given by $\tilde{y} = \sigma(\prod_{j=1}^{K} h_j)$ (in our case we selected for $\sigma$ the sigmoid function shown in equation (21)). Note that by using products in the output layer we directly incorporate the capabilities of higher order networks with a smaller

Table 9. Trading performance results.

| | NAIVE | MLP | SCE | GM | RNN | HONN | Psi Sigma |
|---|---|---|---|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 2.57 | 2.26 | 2.09 | 2.57 | 2.58 | 2.55 |
| Annualized volatility (excluding costs) | 11.6% | 11.6% | 11.6% | 11.6% | 11.6% | 11.6% | 11.6 |
| Annualized return (excluding costs) | 21.3% | 29.7% | 26.3% | 24.2% | 29.8% | 29.8% | 29.5% |
| Maximum drawdown (excluding costs) | −9.1% | −9.1% | −7.8% | −12.4% | −13.8% | −9.2% | −5.9% |
| Positions taken (annualized) | 109 | 118 | 143 | 162 | 124 | 129 | 133 |

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 with 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 in appendix A.3. The trading strategy is that followed for the MLP. A summary of our findings is presented in table 9.

Once again, our results are similar to those obtained by Dunis and Williams (2002, 2003) with a MLP. As the Psi Sigma and HONN models are able to capture higher order correlations, we expected that their performance should be significantly better than those for the MLP and RNN models, but this was not confirmed by our empirical results. 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.

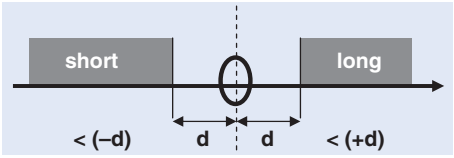


Figure 8. Filtered trading strategy with one single parameter.

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 EUR/USD exchange rate 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

†We needed about 3 minutes to train the Psi Sigma network, about 6 minutes to train the HONN and about 30 minutes to train the RNN with an Intel Core 2 Duo T7300 Fujitsu Amilo Laptop.

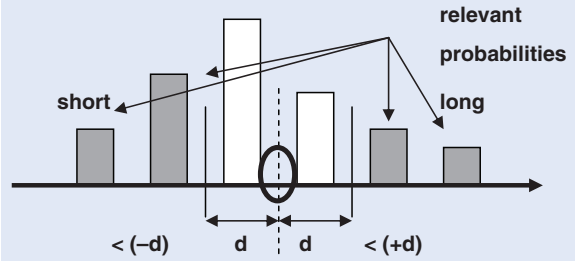


Figure 9. Filtered trading strategy for the SCE model.

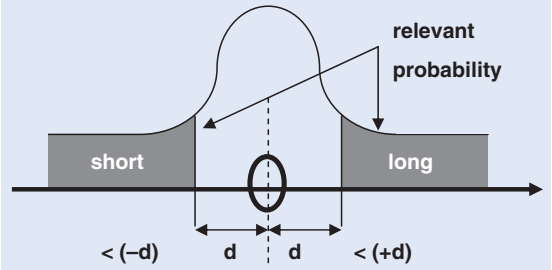


Figure 10. Filtered trading strategy for the GM model.

below zero. In the following, we examine how the models behave if we introduce a threshold $d$ around zero (see figure 8) and what happens if we vary that threshold. The filter rule for the MLP, RNN, HONN and Psi Sigma models is presented in figure 8.

Since the forecasts 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 (figures 9 and 10).

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

Table 10. Parameters chosen by Lindemann *et al.* (2004).

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

Table 11. Chosen parameters for each trading strategy.

| Model | Threshold ($d$) |
|---|---|
| RNN | = 0.00 |
| HONN | = 0.00 |
| Psi Sigma | = 0.00 |

**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 table 11 while the detailed results leading to their choice are documented in appendix A.4.

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.

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

### 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 three 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%† 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 table 13 while an insight about our selection process can found in appendix A.4.

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

†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.

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

| | 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 |
| *Annualized volatility* (excluding costs) | 11.6% | 11.6% | 8.5% | 11.6% | 11.6% | 11.6% | 11.6% |
| Annualized 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 (annualized) | 109 | 118 | 120 | 162 | 124 | 129 | 133 |
| Transaction costs | 3.6% | 3.9% | 3.9% | 5.3% | 4.0% | 4.3% | 4.4% |
| Annualized return (including costs) | 17.7% | 25.8% | 18.8% | 18.9% | 25.7% | 25.6% | 25.1% |

Table 13. Parameters for the leveraged trading strategies.

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

Table 14. Parameters for the leveraged trading strategies.

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

gives the highest profit and Sharpe ratio on the test dataset and filters out of only three trades per year.

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

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 day†). Our final results are presented in table 15.

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 annualized 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 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 to 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 well.

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 the other network architectures, a very 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.‡

†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.

‡Because this paper is a forecasting competition with the models developed by Dunis and Williams (2002, 2003), we use their databank which ends in 2001. In Dunis *et al.* (2009) the use of EUR/USD data until August 2008 demonstrates the robustness and superior performance of the novel models presented here.

Table 15. Trading performance—final results.[a].

| | NAIVE | MLP | SCE | GM | RNN | HONN | Psi Sigma |
|---|---|---|---|---|---|---|---|
| Sharpe ratio (excluding costs)[b] | 1.83 | 2.30 | 2.67 | 3.80 | 2.57 | 2.58 | 2.55 |
| Annualized volatility (excluding costs) | 11.9% | 13.4% | 12.5% | 12.2% | 11.9% | 12.3% | 11.9% |
| Annualized return (excluding costs) | 21.8% | 30.8% | 33.2% | 46.4% | 30.7% | 31.7% | 30.4% |
| Maximum drawdown (excluding costs) | −9.3% | −10.3% | −8.5% | −11.3% | −14.3% | −9.8% | −6.1% |
| Leverage factor | 1.03 | 1.62 | 1.46 | 3.99 | 1.03 | 1.03 | 1.03 |
| Positions taken (annualized) | 109 | 89 | 120[c] | 68[c] | 121 | 129 | 133 |
| Transaction and leverage costs | 3.7% | 6.1% | 7.1% | 12.5% | 4.0% | 4.3% | 4.6% |
| Annualized return (including costs) | 18.1% | 24.7% | 26.1% | 33.9% | 26.7% | 27.0% | 25.8% |

[a]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 therefore invest only a few percent of the capital available but apply the leverage factor (>1), the additional capital does not have 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.

[b]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).

[c]The SCE and GM committees have actually taken more trades than reported in the table (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 68 positions).



## References


Adam, O., Zarader, L. and Milgram, M., Identification and prediction of non-linear models with recurrent neural networks. Laboratoire de Robotique de Paris, 1994.

Austin, M., Bates, G., Dempster, M., Leemans, V. and Williams, S., Adaptive systems for foreign exchange trading. *Quant. Finan.*, 2004, **4**, 37–45.

Connor, J. and Atlas, L., Recurrent neural networks and time series prediction, In *Proceedings of the International Joint Conference on Neural Networks*, 1993, pp. 301–306.

Dunis, C. and Huang, X., Forecasting and trading currency volatility: an application of recurrent neural regression and model combination. *J. Forecast.*, 2002, **21**, 317–354.

Dunis, C., Laws, J. and Evans, B., Trading futures spreads: an application of correlation and threshold filters. *Appl. Financial Econ.*, 2006a, **16**, 1–12.

Dunis, C., Laws, J. and Evans, B., Modelling and trading the gasoline crack spread: a non-linear story. *Deriv. Use, Trad. Regul*, 2006b, **12**, 126–145.

Dunis, C., Laws, J. and Sermpinis, G., The robustness of neural networks for modelling and trading the EUR/USD exchange rate at the ECB fixing. *J. Deriv. Hedge Funds*, 2009, forthcoming.

Dunis, C. and Williams, M., Modelling and trading the EUR/USD exchange rate: do neural network models perform better? *Deriv. Use, Trad. Regul.*, 2002, **8**(3), 211–239.

Dunis, C. and Williams, M., Applications of advanced regression analysis for trading and investment. In *Applied Quantitative Methods for Trading and Investment*, edited by C. Dunis, J. Laws, and P. Naïm, 2003 (Wiley: Chichester).

Elman, J.L., Finding structure in time. *Cogn. Sci.*, 1990, **14**, 179–211.

Fulcher, J., Zhang, M. and Xu, S., The application of higher-order neural networks to financial time series. *Artificial Neural Networks in Finance and Manufacturing*, 2006 (Idea Group: London).

Ghazali, R., Hussain, A. and Merabti, M., Higher order neural networks for financial time series prediction, in *The 10th IASTED International Conference on Artificial Intelligence and Soft Computing*, Palma de Mallorca, Spain, 2006, pp. 119–124.

Ghosh, J. and Shin, Y., Efficient higher-order neural networks for classification and function approximation. *Int. J. Neural Syst.*, 1992, **3**, 323–350.

Giles, L. and Maxwell, T., Learning, invariance and generalization in higher order neural networks. *Appl. Optics*, 1987, **26**, 4972–4978.

Hryshko, A. and Downs, T., Development of machine learning software for high frequency trading in financial markets. *Business Applications and Computational Intelligence*, 2006 (IGI: Hershey).

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

Hussain, A., Ghazali, R. and Al-Jumeily, D., Dynamic ridge polynomial neural network for financial time series prediction, in *IEEE International Conference on Innovation in Information Technology*, IIT06, Dubai, 2006.

Jeng, D., Watada, J., Wu, B. and Wu, J., Fuzzy forecasting with DNA computing. *DNA Computing*, 2006 (Springer: Berlin).

Kaastra, I. and Boyd, M., Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 1996, **10**, 215–236.

Kamijo, K. and Tanigawa, T., Stock price pattern recognition: a recurrent neural network approach, In *Proceedings of the International Joint Conference on Neural Networks*, 1990, pp. 1215–1221.

Karayiannis, N. and Venetsanopoulos, A., On the training and performance of high-order neural networks. *Math. Biosci.*, 1994, **129**, 143–168.

Knowles, A., Hussein, A., Deredy, W., Lisboa, P. and Dunis, C.L., Higher-order neural networks with Bayesian confidence measure for prediction of EUR/USD exchange rate. CIBEF Working Papers, 2005. Available online at: www.cibef.com.

Kosmatopoulos, E., Polycarpou, M., Christodoulou, M. and Ioannou, P., High-order neural network structures for identification of dynamical systems. *IEEE Trans. Neural Netw.*, 1995, **6**, 422–431.

Lindemann, A., Dunis, C. and Lisboa, P., Level estimation, classification and probability distribution architectures for trading the EUR/USD exchange rate. *Neural Netw. Comput. Applic.*, 2004, **14**, 256–271.

Lindemann, A., Dunis, C. and Lisboa, P., Probability distributions and leveraged trading strategies: an application of Gaussian mixture models to the Morgan Stanley technology index tracking fund. *Quant. Finan.*, 2005, **5**, 459–474.

Lisboa, P.J.G. and Vellido, A., Business applications of neural networks. In *Business Applications of Neural Networks: The State-of-the-Art of Real-World Applications*, edited by P.J.G. Lisboa, B. Edisbury, and A. Vellido, pp. vii–xxii, 2000 (World Scientific: Singapore).

Ozun, A., Using new information technologies for modelling data on global markets: an efficient interaction between 'artificial' human brain and economics, In *Proceedings of the Conference on Human and Economic Resources*, 2006, pp. 349–359.

Psaltis, D., Park, C. and Hong, J., Higher order associative memories and their optical implementations. *Neural Netw.*, 1988, **1**, 149–163.

Redding, N., Kowalczyk, A. and Downs, T., Constructive higher-order network algorithm that is polynomial time. *Neural Netw.*, 1993, **6**, 997–1010.

Shapiro, A.F., A hitchhiker's guide to the techniques of adaptive nonlinear models. *Insurance, Math. Econ.*, 2000, **26**, 119–132.

Shin, Y. and Ghosh, J., The Pi-Sigma network: an efficient higher-order neural network for pattern classification and function approximation, in *Proceedings IJCNN*, Seattle, 1991, pp. 13–18.

Tenti, P., Forecasting foreign exchange rates using recurrent neural networks. *Appl. Artif. Intell.*, 1996, **10**, 567–581.

Tino, P., Schittenkopf, C. and Doffner, G., Financial volatility trading using recurrent networks. *IEEE Trans. Neural Netw.*, 2001, **12**, 865–874.

Ullrich, C., Seese, D. and Chalup, S., *Foreign Exchange Trading with Support Vector Machines, Advances in Data Analysis*, 2007 (Springer: Berlin).

Vecci, L., Piazza, F. and Uncini, A., Learning and approximation capabilities of adaptive spline activation neural networks. *Neural Netw.*, 1998, **11**, 259–270.

Zhang, M., Xu, S.X. and Fulcher, J., Neuron-adaptive higher order neural-network models for automated financial data modelling. *IEEE Trans. Neural Netw.*, 2002, **13**, 188–204.



## Appendix A

### *A.1. Performance measures*

The performance measures are calculated as follows.†

### *A.2. Results of alternative benchmark models*‡

### *A.3. Networks characteristics*

Table A3 presents the characteristics of the networks for the different architectures that presented the best statistical performance on the training and on the test sub-period used in this paper.

### *A.4. Empirical results*

Table A4 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.

Table A5 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.

Table A1. Trading simulation performance measures.

| Performance measure | Description | |
|---|---|---|
| Annualized return | $R^A = 252 * (1/N) \sum_{t=1}^{N} R_t$ with $R_t$ being the daily return | (A1) |
| Cumulative return | $R^C = \sum_{t=1}^{N} R_t$ | (A2) |
| Annualized volatility | $\sigma^A = \sqrt{252} * \sqrt{[1/(N-1)] * \sum_{t=1}^{N} (R_t - \bar{R})^2}$ | (A3) |
| Sharpe ratio | $SR = R^A/\sigma^A$ | (A4) |
| Maximum drawdown | Maximum negative value of $\sum(R_t)$ over the period $MD = \mathrm{Min}_{i=1,\ldots,t,t=1,\ldots,N}\left(\sum_{j=i}^{t} R_j\right)$ | (A5) |

Table A2. Out-of-sample trading performance results for traditional models as reported by Dunis and Williams (2003, table 1.20, p. 35).

| | NAIVE | MACD | ARMA | LOGIT | MLP |
|---|---|---|---|---|---|
| Sharpe ratio (excluding costs) | 1.83 | 0.97 | 1.10 | 1.81 | 2.57 |
| Annualized volatility (excluding costs) | 11.6% | 11.7% | 11.7% | 11.6% | 11.6% |
| Annualized 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 (annualized) | 109 | 22 | 112 | 123 | 118 |

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

‡The results for all the novel models used in this research are given in table 9 and are not reproduce here in order to save space.

Table A3. Network characteristics.

| Parameter | Reccurent | HONNs | Psi Sigma |
|---|---|---|---|
| Learning algorithm | Gradient descent | Gradient descent | Gradient descent |
| Learning rate | 0.001 | 0.001 | 0.5 |
| Momentum | 0.003 | 0.003 | 0.5 |
| Iteration steps | 500 | 500 | 500 |
| Initialization of weights | N(0,1) | N(0,1) | N(0,1) |
| Input nodes | 10 | 10 | 10 |
| Hidden nodes (one layer) | 5 | 0 | 5 |
| Output node | 1 | 1 | 1 |

Table A4. Results for alternative threshold values.

| 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% | 27.8% | 12.9% | 4.36% | 1.43% | −0.6% | −0.5% | −0.1% | 0.2% | 0.3% |
| | (2.93) | (2.39) | (2.87) | (1.81) | (0.98) | (−1.1) | (−0.9) | (−0.2) | (1.28) | (1.29) |
| HONN | 22.5% | 17.0% | 13.9% | 5.99% | −4.8% | −0.1% | −0.2% | 0.5% | −0.9% | 0.6% |
| | (2.31) | (1.94) | (1.76) | (0.88) | (−0.9) | (−0.1) | (−0.1) | (0.2) | (−0.4) | (0.37) |
| Psi Sigma | 24.9% | 21.8% | 14.1% | 4.3% | 1.44% | 0.52% | 0.51% | 0.52% | 0.00% | 0.00% |
| | (2.51) | (2.12) | (3.74) | (2.37) | (1.3) | (0.92) | (0.92) | (0.92) | (0.00) | (0.00) |

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

Table A5. Results for alternative threshold values.

| 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% | 29.7% | 13.2% | 4.5% | 1.47% | −0.6% | −0.5% | −0.1% | 0.3% | 0.27% |
| | (2.93) | (4.3) | (2.87) | (1.81) | (0.98) | (−1) | (−0.9) | (−0.2) | (1.3) | (1.3) |
| HONN | 23.2% | 17.5% | 14.3% | 6.17% | −4.9% | −0.1% | −0.2% | 0.52% | −0.9% | 0.7% |
| | (2.31) | (1.94) | (1.76) | (0.88) | (−0.3) | (−0.2) | (−0.1) | (0.2) | (−0.4) | (0.4) |
| Psi Sigma | 25.1% | 22.4% | 14.3% | 4.45% | 1.48% | 0.52% | 0.52% | 0.52% | 0.00% | 0.00% |
| | (2.5) | (2.1) | (3.74) | (2.5) | (1.3) | (0.93) | (0.93) | (0.93) | (0.00) | (0.00) |

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