Financial Forecasting Through Unsupervised Clustering and Evolutionary Trained Neural Networks

N. G. Pavlidis, D. K. Tasoulis, M. N. Vrahatis
Department of Mathematics,
University of Patras Artificial Intelligence Research Center (UPAIRC),
University of Patras, GR–26110 Patras, Greece.
{npav,dtas,vrahatis}@math.upatras.gr

Abstract- This paper presents a time series forecasting methodology and applies it to generate one–step–ahead predictions for two daily foreign exchange spot rate time series. The methodology draws from the disciplines of chaotic time series analysis, clustering, artificial neural networks and evolutionary computation. In brief, clustering is applied to identify neighborhoods in the reconstructed state space of the system; and subsequently neural networks are trained to model the dynamics of each neighborhood separately. The results obtained through this approach are promising.

1 Introduction

One of the central problems of science is forecasting: "Given the past how can we predict the future?" The classic approach is to build an explanatory model from first principles and measure initial data [4]. In the field of financial forecasting we still lack the first principles necessary to build reliable models of the underlying market dynamics. Currently, foreign exchange markets are the most active of all financial markets with average daily trading volumes in traditional (non–electronic broker) estimated at $ 1.2 trillion [9]. Although the precise scale of speculative trading on spot markets is unknown it is estimated that only around 15% of the trading is driven by non–dealer/financial institution trading. Approximately, 90% of all foreign currency transactions involve the US Dollar [9].

In this paper we develop a time series forecasting methodology that draws from the disciplines of chaotic time series analysis, clustering, artificial neural networks, and evolutionary computation, and apply it to forecast the time series of the spot daily exchange rate (interbank rate) of the Japanese Yen against the US Dollar and that of the US Dollar to the British Pound. The proposed methodology is closely related to the notion of local approximation as set out by Farmer and Sidorowich [4].

The remaining paper is organized as follows: Section 2 describes analytically the proposed forecasting methodology; Section 3 is devoted to implementation details and the numerical results obtained. The paper concludes in Section 4.

2 Proposed Methodology

In the study of a single time series the first step is to embed it in a state space. Instead of constructing a global model for a chaotic time series, Farmer and Sidorowich in [4] proposed to construct models for neighborhoods of the state space, an approach known as *local approximation*. In brief, to predict $x(t+T)$ primarily, the m nearest neighbors of the state vector $\mathbf{x(t)}$, i.e. the m states $\mathbf{x(t')}$ that minimize the distance $\|\mathbf{x(t)} - \mathbf{x(t')}\|$, are found. Then, a *local predictor* is constructed using $\mathbf{x(t')}$ as points of the domain and $x(t'+T)$ as points of the range. Finally, the local predictor is used to forecast $x(t+T)$. In our approach instead of using the heuristic parameter m to set the size of neighborhoods we use the novel k–windows clustering algorithm [16] to automatically detect neighborhoods in the state space. This algorithm, with a slight modification, has the ability to endogenously determine the number of clusters present in the dataset. Once the clustering process is complete, an evolutionary trained feedforward neural network acts as the local predictor for each cluster. In synopsis, the proposed methodology consists of the following five steps:

1. Determine the minimum embedding dimension for phase–space reconstruction.
2. Identify the clusters present in the training set.
3. For each cluster in the training set train a different feedforward neural network using for training patterns, patterns from that cluster solely.
4. Assign the patterns of the test set to the clusters previously detected.
5. Use the trained feedforward networks to obtain the forecasts.

2.1 Determining the Embedding Dimension

State space reconstruction is the first step in nonlinear time series analysis of data from chaotic systems including estimation of invariants and prediction. The observations $x(n)$, are a projection of the multivariate state space of the system onto the one–dimensional axis of the $x(n)$'s. By utilizing time–delayed versions of the observed scalar quantity: $x(t_0 + n\Delta t) = x(n)$ as coordinates for phase space reconstruction, we create from the set of observations, multivariate vectors in d–dimensional space:

$$y(n) = [x(n), x(n+T), \ldots, x(n+(d-1)T)]. \quad (1)$$

Doing this we hope that the points in $\mathbb{R}^d$ form an attractor that preserves the topological properties of the unknown original attractor. The fundamental theorem of reconstruction, introduced first by Takens [15], states that when the original attractor has fractal dimension d_A, all self-crossings of the orbit will be eliminated when one

0-7803-7804-0 /03/$17.00 © 2003 IEEE

chooses $d > 2d_A$. These self-crossings of the orbit are the outcome of the projection and embedding seeks to undo that. The theorem is valid for the case of infinitely many noise–free data. Moreover, the condition that $d > 2d_A$ is sufficient but not necessary. In other words it does not address the question: "Given a scalar time series what is the appropriate minimum embedding dimension, d_E?" This is a particularly important question when computational intelligence techniques like neural networks are used, since overspecification of input variables produces sub–optimal performance. To determine the minimum embedding dimension we applied the popular method of "False Nearest Neighbors" [8].

In an embedding dimension that is too small to unfold the attractor of the system not all points that lie close to each other will be neighbors because of the dynamics. Some will actually be far from each other and simply appear as neighbors because the geometric structure of the attractor has been projected down onto a smaller space ($d < d_E$). In going from dimension d to $d+1$ an additional component is added to each of the vectors $y(n)$. A natural criterion for catching embedding errors is that the increase in distance between $y(n)$ and its closest neighbor $y^{(1)}(n)$ is large when going from dimension d to $d+1$. Thus the first criterion employed to determine whether two nearest neighbors are false is:

$$\left[\frac{R_{d+1}^2(n) - R_d^2(n)}{R_d^2(n)}\right]^{1/2} = \frac{|x(n+Td) - x^{(1)}(n+Td)|}{R_d^2(n)} > R_{tol}, \qquad (2)$$

where $R_d^2(n)$ is the squared Euclidean distance between point $y(n)$ and its nearest neighbor in dimension d, and R_{tol} is a threshold whose default value is 10. If the length of the time series is finite, as is always the case in real–world applications, a second criterion is required to ensure that nearest neighbors are in effect close to each other. More specifically, if the nearest neighbor to $y(n)$ is not close [$R_d(n) \simeq R_A$] and it is a false neighbor, then the distance $R_{d+1}(n)$ resulting from adding an extra component to the data vectors will become $R_{d+1}(n) \simeq 2R_A$. That is even distant but nearest neighbors will be stretched to the extremities of the attractor when they are unfolded from each other, if they are false neighbors. This observation gives rise to the second criterion used to identify false neighbors:

$$\frac{R_{d+1}(n)}{R_A} > A_{tol}. \qquad (3)$$

As a measure for R_A the value:

$$R_A^2 = \frac{1}{N}\sum [x(n) - \bar{x}]^2$$

is suggested. The default value for A_{tol} is 2. If the time series is not contaminated with noise, then the appropriate embedding dimension is the one for which the number of false neighbors, as estimated by applying jointly the two criteria (2) and (3) drops to zero. If the data is contaminated with noise, as is the case for foreign exchange rate time series, then the appropriate embedding dimension corresponds to an embedding dimension with a low proportion of false neighbors. Once the minimum embedding dimension sufficient for phase space reconstruction is identified, time–delayed state space vectors are subjected to clustering through the unsupervised k–windows algorithm.

2.2 Unsupervised k–windows Clustering Algorithm

Clustering, that is the partitioning of a set of patterns into disjoint, homogeneous and meaningful groups is a fundamental process in the practice of science. In this subsection, we briefly describe the recently proposed k–windows clustering algorithm and its unsupervised version.

Intuitively, the k-windows algorithm proposed in [16] tries to place a d-dimensional window (frame, box) containing all patterns that belong to a single cluster; for all clusters present in the dataset. At a first stage, the windows are moved in the Euclidean space without altering their size. Each window is moved by setting its center to the mean of the patterns currently included (sea solid line squares in Fig. 1). This process continues iteratively until further movement does not increase the number of patterns included. At the second stage, the size of each window is enlarged in order to capture as many patterns of the cluster as possible. The process of enlargement terminates when the number of patterns included in the window no longer increases. The two processes are exhibited in Fig 1, where the initial 2-dimensional window $M1$, is successively moved, and then subjected to two phases of enlargement that result to the final window $E2$.

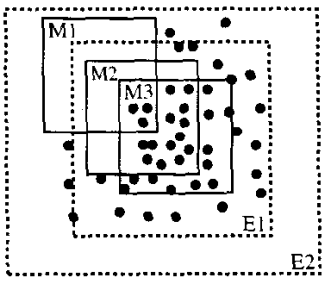

Figure 1: Sequential Movements (solid lines) and subsequently, enlargements (dashed lines) of the initial window $M1$ that result to the final window $E2$.

The unsupervised k-windows algorithm generalizes the original algorithm [16], by considering a number of initial windows greater than the expected number of clusters. After the clustering procedure is terminated, the windows that are suspected to capture patterns that belong to a single cluster are merged. The merging procedure is illustrated in Fig. 2. Windows W_1 and W_2 share a sufficiently large number of patterns between them, and thus, are merged to form a single cluster. On the other hand W_5 and W_6 are not merged, beacause the patterns in their intersection are insufficient to consider them as part of one cluster.

In more detail; at first, k points are selected (possibly in a random manner). The k initial d–ranges (windows), of size

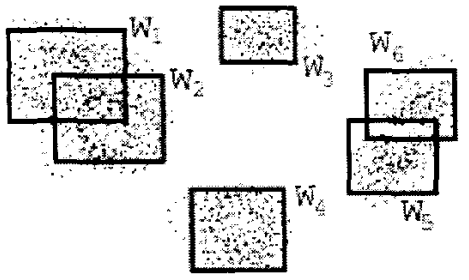

Figure 2: The merging procedure. W_1 and W_2 have many points in common thus they are considered to belong to the same cluster.

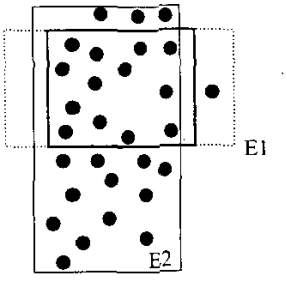

Figure 3: The enlargement process. Enlargement in the horizontal dimension ($E1$) is rejected, while in the vertical dimension it is accepted. After subsequent movements and enlargements the window becomes $E2$. Further enlargement in the horizontal dimension is rejected.

a, have as centers these points. Subsequently, the patterns that lie within each d-range are identified.

Next, the mean of the patterns that lie within each d–range (i.e. the mean value of the d–dimensional points) is calculated. The new position of the d–range is such that its center coincides with the previously computed mean value. The last two steps are repeatedly executed as long as the increase in the number of patterns included in the d–range that results from this motion satisfies a stopping criterion. The stopping criterion is determined by a variability threshold θ_v that corresponds to the least change in the center of a d–range that is acceptable to recenter the d–range.

Once movement is terminated, the d–ranges are enlarged in order to capture as many patterns as possible from the cluster. Enlargement takes place at each dimension separately. The d–ranges are enlarged by θ_e/l percent at each dimension, where θ_e is user defined, and l stands for the number of previous successful enlargements. After the enlargement in one dimension is performed, the window is moved, as described above. Once movement terminates, the proportional increase in the number of patterns included in the window is calculated. If this proportion does not exceed the user–defined coverage threshold, θ_c, the enlargement and movement steps are rejected and the position and size of the d–range are reverted to their prior to enlargement values. If, on the other hand, the proportional increase in the patterns included in the window exceeds θ_c, then the new size and position are accepted. In the case that enlargement is accepted for dimension $d' \geqslant 2$, then for all dimensions d'', such that $d'' < d'$, the enlargement process is performed again assuming as initial position the current position of the window. This process terminates if enlargement in any dimension does not result in a proportional increase in the number of patterns included in the window beyond the threshold θ_c. An example of this process is provided in Fig. 3. In the figure the window is initially enlarged horizontally ($E1$). This enlargement is rejected since it does not produce an increase in the number of patterns included. Next the window is enlarged vertically, this enlargement is accepted, and the result of the subsequent movements and enlargements is the initial window to become $E2$. Next enlargement in the horizontal direction is reconsidered but it is rejected again.

The key idea to automatically determine the number of clusters, is to apply the k-windows algorithm using a sufficiently large number of initial windows. The windowing technique of the k-windows algorithm allows for a large number of initial windows to be examined, without any significant overhead in time complexity. Once all the processes of movement and enlargement for all windows are terminates, all overlapping windows are considered for merging. The merge operation is guided by a merge threshold θ_m. Having identified two overlapping windows, the number of patterns that lie in their intersection is calculated. Next the proportion of this number to the total patterns included in each window is calculated. If the mean of these two proportions exceeds θ_m, then the windows are considered to belong to a single cluster and are merged, otherwise not. This operation is illustrated in Fig. 2; the extent of overlapping between windows W_1 and W_2 exceeds the threshold criterion and the algorithm considers both to belong to a single cluster, unlike windows W_5 and W_6, which capture two different clusters.

The remaining windows, after the quality of the partition criterion is met, define the final set of clusters. If the quality of a partition, determined by the number of patterns contained in any window, with respect to all patterns is not the desired one, the algorithm is re-executed. The user defined parameter u serves this purpose. The output of the algorithm is a number of sets that define the final clusters discovered in the original dataset.

2.3 Artificial Neural Networks and Selected Training Algorithms

Artificial Feedforward Neural Networks (FNNs) are parallel computational models comprised of densely interconnected, simple, adaptive processing units, characterized by an inherent propensity for storing experiential knowledge and rendering it available for use. FNNs resemble the human brain in two fundamental respects; firstly, knowledge is acquired by the network from its environment through a learning process, and secondly, interneuron connection strengths, known as synaptic weights are employed to store the acquired knowledge [6].

Two critical parameters for the successful application of FNNs are the appropriate selection of network architecture and training algorithm. The problem of identifying the optimal network architecture for a specific task remains up to date an open and challenging problem. For the general problem of function approximation, the *universal approximation theorem* proved in [7, 18] states that:

Theorem 2.1 *Standard Feedforward Networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accuracy.*

An immediate implication of the above theorem is that any lack of success in applications must arise from inadequate learning, insufficient number of hidden units, or the lack of a deterministic relationship between the input and the target. A second theorem proved in [11] provides an upper bound for the architecture of an FNN destined to approximate a continuous function defined on the hypercube in $\mathbb{R}^n$.

Theorem 2.2 *On the unit cube in $\mathbb{R}^n$ any continuous function can be uniformly approximated, to within any error by using a two hidden layer network having $2n+1$ units in the first layer and $4n+3$ units in the second layer.*

The efficient supervised training of FNNs is the subject of considerable ongoing research and numerous algorithms have been proposed to this end. Supervised training amounts to the global minimization of the network error function E. The rapid computation of a set of weights that minimizes this error is a rather difficult task since, in general, the number of network weights is large and the resulting error function generates a complex surface in the weight space, characterized by multiple local minima and broad flat regions adjoined to narrow steep ones. Next, a brief exposition of the training algorithms considered is provided.

2.3.1 Differential Evolution Training Algorithm

In a recent work, Storn and Price [14] have presented a novel minimization method, called Differential Evolution (DE), designed to handle non–differentiable, nonlinear and multimodal objective functions. DE exploits a *population* of NP potential solutions, that is L–dimensional vectors, to probe the search space. At each iteration of the algorithm, called *generation*, g, three steps, *mutation*, *recombination* and *selection*, are performed in order to obtain more accurate approximations to a solution [12]. Initially, all weight vectors are initialized by using a random number generator. At the mutation step, for each $i = 1, \ldots, NP$ a new mutant weight vector v^i_{g+1} is generated by combining weight vectors, randomly chosen from the population, and exploiting one of the variation operators (4)–(8) included in the C code provided by Storn at his web page [1]:

$$v^i_{g+1} = \omega^{best}_g + \mu(\omega^{r1}_g - \omega^{r2}_g), \quad (4)$$

$$v^i_{g+1} = \omega^{r1}_g + \mu(\omega^{r2}_g - \omega^{r3}_g), \quad (5)$$

$$v^i_{g+1} = \omega^i_g + \mu(\omega^{best}_g - \omega^i_g) + \mu(\omega^{r1}_g - \omega^{r2}_g), \quad (6)$$

$$v^i_{g+1} = \omega^{best}_g + \mu(\omega^{r1}_g - \omega^{r2}_g) + \mu(\omega^{r3}_g - \omega^{r4}_g), \quad (7)$$

$$v^i_{g+1} = \omega^{r1}_g + \mu(\omega^{r2}_g - \omega^{r3}_g) + \mu(\omega^{r4}_g - \omega^{r5}_g), \quad (8)$$

where ω^{r1}_g, ω^{r2}_g, ω^{r3}_g, ω^{r4}_g and ω^{r5}_g are randomly selected vectors, different from ω^i_g, ω^{best}_g is the best member of the current generation. Finally, the positive mutation constant μ controls the magnification of the difference between two weight vectors.

The resulting mutant vectors are mixed with a predetermined weight vector, called *target* vector. This operation is called *recombination*, and it gives rise to the *trial* vector. At the recombination step, for each component $j = 1, 2, ..., L$ of the mutant weight vector a random number $r \in [0, 1]$ is generated. If r is smaller than the predefined recombination constant p, the j-th component of the mutant vector v^i_{g+1} becomes the j-th component of the trial vector. Otherwise, the j-th component of the target vector is selected as the h–th component of the trial vector. Finally, at the selection step, the trial weight vector obtained after the recombination step is accepted for the next generation if and only if it yields a reduction of the value of the error function relative to the previous weight vector; if not the previous weight vector is retained.

2.4 Particle Swarm Optimization

PSO is a swarm–intelligence optimization algorithm. Each member of the swarm, called *particle*, moves with an adaptable velocity within the search space, and retains in its memory the best position it ever encountered. In the *local* variant, each particle is assigned to a neighborhood consisting of a prespecified number of particles. At each iteration, the best position ever attained by the particles of the neighborhood is communicated among them [3]. In the *global* variant, the neighborhood of each particle is the entire swarm.

Assume a d–dimensional search space, $\mathbf{S} \subset \mathbb{R}^d$, and a swarm of N particles. Both the position and the velocity of the i–th particle are in effect d–dimensional vectors, $x_i \in \mathbf{S}$ and $v_i \in \mathbb{R}^d$, respectively. The best previous position ever encountered by the i–th particle is denoted by p_i, while the best previous position attained by all particles of the neighborhood is denoted by p_g. The velocity of the i–th particle at the $(t+1)$–th iteration is obtained through Eq. (9), for the constriction factor variant [2], or through Eq. (10), for the inertia weight variant [3]. The new position of this particle is determined by simply adding the velocity vector to the previous position vector, Eq. (11).

$$v^{(t+1)}_i = \chi[v^{(t)}_i + c_1 r_1 (p^{(t)}_i - x^{(t)}_i) + c_2 r_2 (p^{(t)}_g - x^{(t)}_i)], \quad (9)$$

$$v^{(t+1)}_i = w v^{(t)}_i + c_1 r_1 (p^{(t)}_i - x^{(t)}_i) + c_2 r_2 (p^{(t)}_g - x^{(t)}_i), \quad (10)$$

$$x^{(t+1)}_i = x^{(t)}_i + v^{(t+1)}_i, \quad (11)$$

where $i = 1, \ldots, N$; c_1 and c_2 denote the *cognitive* and *social* parameters respectively; r_1, r_2 are random numbers uniformly distributed in $[0, 1]$; χ is the constriction factor;

[1] http://www.icsi.berkeley.edu/~storn/code.html

w is the inertia weight; and t, stands for the counter of iterations. The proposed default value for c_1 and c_2 is 2. The constriction factor χ can be calculated through the formula proposed in [2]. For w a common approach is to initialize it at a large value (so as to promote global exploration) and to gradually decrease it (to allow fine tuning) [13]. In general, PSO has proved to be very efficient and effective in tackling various difficult problems [10].

3 Numerical Results

We have applied the methodology previously described to two daily (interbank rate) foreign exchange time series; that of the the Japanese yen against the United States dollar, and that of the United States Dollar against the British Pound. The datasets are freely provided by the website `www.oanda.com`. Both series extend over a time period of five years, from the 1st of January 1998 until the 1st of January of 2003, and consist of 1827 observations. The time series of daily exchange rates are illustrated in Figs. 4 and 5.

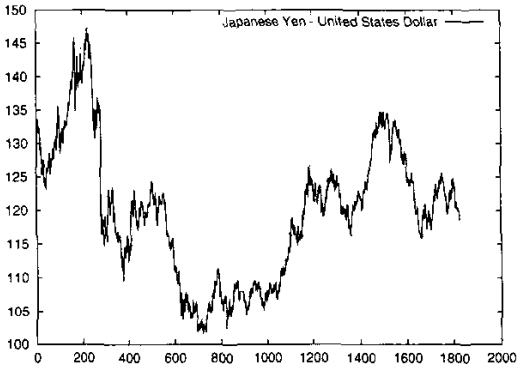

Figure 4: Time series of the daily exchange rate of the Japanese Yen to the U.S. Dollar.

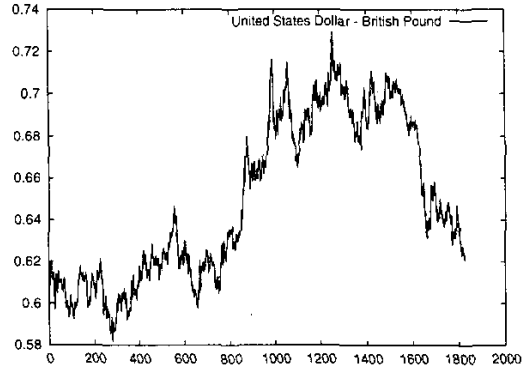

Figure 5: Time series of the daily exchange rate of the U.S. Dollar to the British Pound.

Numerical experiments were performed using a Clustering C++ and a Neural Network C++ Interface built under the Red Hat Linux 7.3 operating system using the GNU compiler collection (gcc) version 2.96. The time series data were split into two sets, a train set containing the first 1500 patterns and a test set containing the remaining 321 patterns, covering approximately the final year of data.

Applying the method of "False Nearest Neighbors" on the two training sets produced the results illustrated in Figs. 6 and 7. For the time series of the Japanese Yen against the US Dollar, the proportion of false nearest neighbors drops sharply to the value of 0.334% for an embedding dimension of $d = 5$ (in terms of the notation used in Eq. (1). For larger values of d the proportion of false nearest neighbors lies in the neighborhood of 0.067%, up to an embedding dimension of $d = 19$ for which the number of false nearest neighbors drops to zero. For the time series of the US Dollar against the British Pound, the proportion of false nearest neighbors falls sharply to the value of 1.337% for $d = 4$. For $d = 5$ the proportion drops further to the value of 0.267%, and finally, it becomes zero for $d = 6$. Since both series are contaminated by noise the embedding dimension chosen was one for which the proportion of false nearest neighbors was close to, but not precisely, zero. Thus, for the time series of the Japanese Yen against the US Dollar, an embedding dimension of 5 was chosen, while for the time series of the US Dollar against the British Pound, the embedding dimension was chosen to be 4.

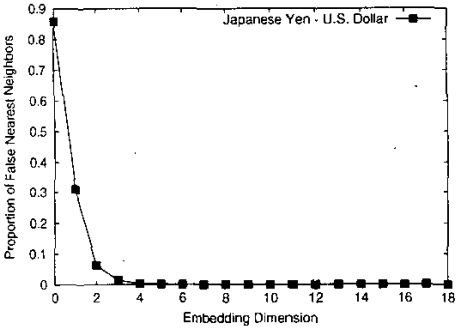

Figure 6: Proportion of false nearest neighbors per embedding dimension for the Japanese Yen against the US Dollar time series.

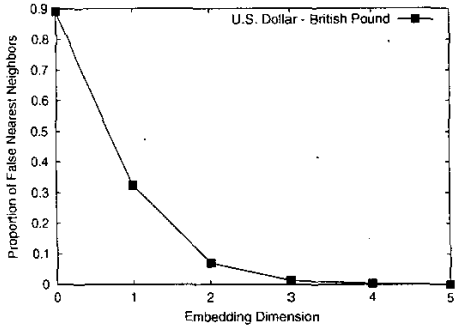

Figure 7: Proportion of false nearest neighbors per embedding dimension for the US Dollar against the British Pound.

Having identified the appropriate embedding dimension

Clustering Results for the Yen–Dollar Series

	Train Set	Test Set
	1500 6d Patterns	321 5d Patterns
Cluster 1	464	133
Cluster 2	32	10
Cluster 3	466	178
Cluster 4	399	0
Cluster 5	199	0

Table 1: Number of clusters identified on the training set and patterns from the test set assigned to each cluster.

Clustering Results for the Dollar–Pound Series

	Train Set	Test Set
	1500 5d Patterns	321 4d Patterns
Cluster 1	128	172
Cluster 2	591	14
Cluster 3	401	79
Cluster 4	216	57
Cluster 5	164	0

Table 2: Number of clusters identified on the training set and patterns from the test set assigned to each cluster.

for both series, the unsupervised k–windows algorithm is employed to compute the clusters present in the training set. Pattern n is of the form:

$$p_n = [x_n, x_{n+1}, \ldots, x_{n+d-1}, x_{n+d}], n = 1, \ldots, 1500.$$

In other words, the value to be predicted (target value) x_{n+d} is also included as an input to the clustering algorithm. Once the clusters present in the training set are computed, the patterns from the test set are assigned to these clusters. To identify the cluster to which a pattern belongs, it is first necessary to find the window whose center is closest to that pattern. The pattern is then assigned to the cluster to which this window belongs. Since the target value for patterns in the test set is unknown distances from window centers are computed by excluding the target component x_{n+d} from the window center vector. In this way the dimensionality of the window center vector is reduced by one and it equals that of the test patterns. The selected distance metric is the Euclidean distance. The clusters identified for each time series, as well as the number of patterns assigned to each cluster, from each dataset, are depicted in Tables 1 and 2.

As illustrated in Tables 1 and 2, for both time series, patterns from the test set were not assigned to all clusters identified in the training set. This implies that information from significant portions of the training set was not considered relevant for the task of forecasting patterns in the test set. Selecting the appropriate input for neural network forecasting systems has been recognized as a determining factor [5, 17] for their success. In our approach the selection process is based on a clustering algorithm with the ability to endogenously determine the number of clusters, thus limiting significantly the number and impact of heuristic factors.

Numerous criteria have been proposed for the evaluation of the forecasting performance of FNNs. The most prominent among them are the (normalized) root squared mean error, the mean relative error, and the correlation coefficient between the actual and the forecasted values [1]. These measures are designed to capture the extent of discrepancy between the actual and the predicted values. Our experience, as well as, that of other researchers [19] suggests that these measures can be misleading when applied to one–step–ahead forecasting of daily foreign exchange rates. The reason is that all the aforementioned measures tend to assume very satisfactory values when the forecasts closely resemble a time–lagged version of the original series. This phenomenon is attributable to the fact that the extent of daily variation rarely represents a significant proportion of the daily values. In these situations the forecasts can be as bad as the naive prediction that has tomorrow's rate to be equal to today's. Therefore, the accuracy of the FNNs was assessed by the percentage of correct *sign* prediction [5, 17]. This measure captures the percentage of forecasts in the test set for which the following inequality is satisfied:

$$(\widehat{x_{t+d}} - x_{t+d-1}) \cdot (x_{t+d} - x_{t+d-1}) > 0, \qquad (12)$$

where, $\widehat{x_{t+d}}$ represents the prediction generated by the FNN, x_{t+d} refers to the true value of the exchange rate at period $t + d$ and, finally, x_{t+d-1} stands for the value of the exchange rate at the current period, $t + d - 1$. Correct sign prediction in effect captures the percentage of profitable trades enabled by the forecasting system employed. To successfully train FNNs capable of forecasting the direction of change of the time series, a modified, nondifferentiable, error function was implemented, Eq.(13).

$$E_k = \begin{cases} 0, & \text{if } (\widehat{x_{t+d}} - x_{t+d-1}) \cdot (x_{t+d} - x_{t+d-1}) > 0 \\ |x_{t+d} - \widehat{x_{t+d}}|, & \text{otherwise} \end{cases} \qquad (13)$$

This function assigns an error value of zero for the kth training pattern as long as the network accurately predicts the direction of change for that pattern; otherwise the error is computed as the absolute value of the discrepancy between the actual and the predicted rate. In addition to the fact that this error function represents more accurately than a standard (mean) squared error function, the goal we wish to achieve through training, it has also proved to discourage overfitting. This is an important advantage since network performance on the training set becomes a reliable measure of generalization ability.

As previously mentioned, the issue of selecting the optimal network architecture for a particular task, remains up to date an open and challenging problem. After extensive experimentation with networks with one and more hidden layers, our experience indicates that appropriate network architectures, in terms of performance on the training set and generalization ability, are 5–3–2–1 for the Japanese Yen–US Dollar time series, and 4–3–2–1 for the US Dollar–British Pound time series. All FNNs were trained for 100 epochs on the patterns of the training set and subsequently their performance was evaluated on the test set. This process was repeated for 100 times for each one of the five DE mutation operators and for the two global versions of PSO. The forecasting capability of the trained FNNs with respect to accurate sign prediction per cluster in the test set, is reported in

Table 3, for the Japanese Yen–US Dollar time series, and in Table 4, for the US Dollar–British Pound time series. The overall, mean predictive accuracy for both series is provided in Table 5. As a benchmark for comparison, Table 5, also reports for each time series the mean predictive accuracy achieved by a single FNN over 100 experiments. The two FNNs had the same architecture as the FNNs used by the proposed methodology, and they were trained for 100 epochs using all the patterns of the training set.

Japanese Yen–US Dollar			
Cluster:	1	2	3
	464–133	32–10	466–178
	DE1		
mean	59.90	88.88	85.87
std	1.12	6.04e-14	1.84e-13
min	56.81	88.88	85.87
max	63.63	88.88	85.87
	DE2		
mean	68.47	88.88	85.87
std	1.12	6.04e-14	1.84e-13
min	62.87	88.88	85.87
max	68.93	88.88	85.87
	DE3		
mean	65.88	77.11	83.54
std	1.96	8.45	1.82
min	60.60	55.55	76.27
max	68.93	88.88	86.44
	DE4		
mean	64.91	88.33	84.97
std	4.23	2.42	1.98
min	56.81	77.77	73.44
max	69.69	88.88	86.44
	DE5		
mean	68.41	77.44	84.62
std	1.54	5.98	1.88
min	59.09	55.55	74.57
max	71.96	88.88	86.44
	PSO Constriction Factor		
mean	64.32	76.22	82.49
std	3.46	8.18	3.12
min	52.27	55.55	69.49
max	71.21	88.88	87.00
	PLO Inertia Weight		
mean	64.24	78	82.96
std	3.01	9.04	2.65
min	56.06	55.55	74.57
max	72.72	88.88	86.44

Table 3: Sign prediction performance per cluster on the test set.

US Dollar–British Pound				
Cluster:	1	2	3	4
	128–172	591–14	401–79	216–57
	DE1			
mean	71.17	99.30	80.62	98.10
std	3.61	6.17	4.16	0.49
min	60.81	38.46	61.53	94.64
max	74.85	100	88.46	98.21
	DE2			
mean	66.39	99.84	80.32	98.10
std	2.61	1.07	3.64	0.49
min	60.81	92.30	69.23	94.64
max	74.26	100	88.46	98.21
	DE3			
mean	65.19	99.53	80.23	98.05
std	4.86	2.39	3.84	0.51
min	53.80	84.61	69.23	96.42
max	73.09	100	88.46	98.21
	DE4			
mean	68.11	99.07	80.84	98.12
std	5.20	4.78	5.32	0.46
min	58.47	61.53	55.12	94.64
max	76.60	100	88.46	98.21
	DE5			
mean	64.22	100	82.94	94.2
std	7.28	0	4.99	3.14
min	40.93	100	61.53	87.27
max	75.43	100	89.74	100
	PSO Constriction Factor			
mean	66.64	99.84	80.67	97.96
std	4.51	1.07	3.50	0.67
min	52.04	92.30	69.23	94.64
max	74.26	100	88.46	98.21
	PSO Inertia weight			
mean	66.75	100	80.35	98.10
std	5.22	0	3.09	0.49
min	46.78	100	69.23	94.64
max	74.85	100	85.9	98.21

Table 4: Sign prediction performance per cluster on the test set.

Mean Predictive Accuracy				
Training	Dollar–Pound		Yen–Dollar	
Algorithm:	Proposed Methodology	Single FNN	Proposed Methodology	Single FNN
DE1	79.48	51.23	75.21	51.35
DE2	76.87	51.04	78.76	51.23
DE3	76.18	50.99	76.03	51.25
DE4	77.89	51.19	76.77	51.32
DE5	75.66	50.8	77.68	51.3
PSO Constr.	77.07	48.36	74.77	48.42
PSO Inertia	77.08	47.70	75.05	48.76

Table 5: Mean predictive accuracy for all the considered methods, for both time series. As a benchmark for comparison the performance of a single FNN is provided.

4 Conclusions

This paper presents a time series forecasting methodology which draws from the disciplines of chaotic time series analysis, clustering, artificial neural networks, and evolutionary computation. The methodology consists of five stages. Primarily the minimum dimension necessary for phase space reconstruction through time delayed embedding is calculated using the method of false nearest neighbors. To identify neighborhoods in the state space, time delayed vectors are subjected to clustering through the unsupervised k–windows algorithm. This algorithm has the capability to endogenously determine the number of clusters present in a dataset. Subsequently, a different feedforward neural

network is trained through the differential evolution algorithm and the particle swarm optimization method, on each cluster. At the fourth stage, the patterns in the test set are assigned to the clusters identified in the training set. Finally, the trained networks are used to generate a prediction for each test pattern.

This methodology was applied to generate one–step–ahead predictions for two, spot foreign exchange time series; that of the Japanese Yen against the US Dollar, and that of the US Dollar against the British Pound. The aim of the trained feedforward networks was to predict the direction of change of the next day. The obtained results were promising for both the differential evolution algorithm and the particle swarm optimizer. Overall mean predictive accuracy for both series was in the neighborhood of 75 to 80 percent.

In future work we intend to investigate the robustness of the proposed methodology as the test set is expanded to cover a larger portion of the dataset. Moreover, we intend to apply this methodology to perform multistep prediction and to incorporate an evolutionary process to determine the topology of the feedforward networks. Finally, the performance of this methodology on alternative economic and chaotic time series will be investigated.

Acknowledgements

The authors wish to thank two anonymous referees for their valuable comments and suggestions.

Bibliography

[1] A.S. Andreou, E.F. Georgopoulos, and S.D Likothanassis, *Exchange–rates forecasting: A hybrid algorithm based on genetically optimized adaptive neural networks*, Computational Economics **20** (2002), 191–210.

[2] M. Clerc and J. Kennedy, *The particle swarm–explosion, stability, and convergence in a multidimensional complex space*, IEEE Trans. Evol. Comput. **6** (2002), no. 1, 58–73.

[3] R.C. Eberhart, P. Simpson, and R. Dobbins, *Computational intelligence pc tools*, Academic Press, 1996.

[4] J. D. Farmer and J. J. Sidorowich, *Predicting chaotic time series*, Physical Review Letters **59** (1987), no. 8, 845–848.

[5] L. C. Giles, , S. Lawrence, and A. H. Tsoi, *Noisy time series prediction using a recurrent neural network and grammatical inference*, Machine Learning **44** (2001), no. 1/2, 161–183.

[6] S. Haykin, *Neural networks: A comprehensive foundation*, New York: Macmillan College Publishing Company, 1999.

[7] K. Hornik, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), 359–366.

[8] M. B. Kennel, R. Brown, and H. D. Abarbanel, *Determining embedding dimension for phase–space reconstruction using a geometrical construction*, Physical Review A **45** (1992), no. 6, 3403–3411.

[9] Bank of International Settlements, *Central bank survey of foreign exchange and derivative market activity in april 2001*, Bank of International Settlements (October 2001).

[10] K.E. Parsopoulos and M.N. Vrahatis, *Recent approaches to global optimization problems through particle swarm optimization*, Natural Computing **1** (2002), no. 2–3, 235–306.

[11] A. Pinkus, *Approximation theory of the mlp model in neural networks*, Acta Numerica (1999), 143–195.

[12] V.P. Plagianakos and M.N. Vrahatis, *Parallel evolutionary training algorithms for 'hardware–friendly' neural networks*, Natural Computing **1** (2002), 307–322.

[13] Y. Shi and R.C. Eberhart, *A modified particle swarm optimizer*, Proceedings IEEE Conference on Evolutionary Computation (Anchorage, AK), IEEE Service Center, 1998.

[14] R. Storn and K. Price, *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces*, Journal of Global Optimization **11** (1997), 341–359.

[15] F. Takens, *Detecting strange attractors in turbulence*, Dynamical Systems and Turbulence (D. A. Rand and L. S. Young, eds.), Lecture Notes in Mathematics, vol. 898, Springer, 1981, pp. 366–381.

[16] M.N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides, *The new k-windows algorithm for improving the k-means clustering algorithm*, Journal of Complexity **18** (2002), 375–391.

[17] S. Walczak, *An empirical analysis of data requirements for financial forecasting with neural networks*, Journal of Management Information Systems **17** (2001), no. 4, 203–222.

[18] H. White, *Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings*, Neural Networks **3** (1990), 535–549.

[19] J.T Yao, H. Poh, and T. Jasic, *Foreign exchange rates forecasting with neural networks*, International Conference on Neural Information Processing (Hong Kong), 1996, pp. 754–759.