A Comparison between SARIMA Models and Feed Forward Neural Network Ensemble Models for Time Series Data

Daniel Alba-Cuéllar, Angel Eduardo Muñoz-Zavala

Universidad Autónoma de Aguascalientes,
Aguascalientes, México

albacd@cimat.mx, aemzmx@gmail.com

Abstract. In this paper, we investigate the robustness of Feed Forward Neural Network (FFNN) ensemble models applied to quarterly time series forecasting tasks, by comparing their prediction ability with that of Seasonal Auto-regressive Integrated Moving Average (SARIMA) models. We obtained adequate SARIMA models which required statistical knowledge and considerable effort. On the other hand, FFNN ensemble models were readily constructed from a single FFNN template, and they produced competitive forecasts, at the level of well-constructed SARIMA models. The single template approach for adapting FFNN ensembles to multiple time series datasets can be an economic and sensible alternative if fitting individual models for each time series turns out to be very time consuming. Additionally, FFNN ensembles were able to produce accurate interval estimations, in addition to good point forecasts.

Keywords: time series forecasts, SARIMA models, artificial neural networks, model ensemble, particle swarm optimization, ensemble channel.

1 Introduction

A central objective pursued by people working in diverse scientific and engineering fields is to predict, as accurately as possible, short-term future behavior from relatively well-understood, but out of control time-evolving variables, like birth rates in a municipality, employment rates in a national economy, enrollments at a certain school, influenza cases detected in a community, blood pressure measured from an individual, rainfall at a specific geographic region, etc. Reasonable short-term predictions for variables such as these enable people to make informed decisions; for instance, good 24-hour-ahead predictions for electrical energy consumption are of fundamental importance when deciding about the actions to take in order to guarantee the optimal management and operation of an energy grid supplying electricity to clients.

A time series is a sequence of chronologically ordered values, sampled from a time-evolving variable; this conceptual tool plays a central role in the scientific

search for good short-term predictions (also called forecasts). Typically, time series values are measured from a real-world process, or variable, at equally-spaced points in time. In order to predict future time series values, mathematical models are typically employed; these models extract information about the characteristic behavior of consecutive past time series values (historical data), and then use this extracted information to project (extrapolate) time series behavior into the near future.

We can identify two broad classes of mathematical methodologies for building time series models: traditional methods based on parametric statistical modeling, and newer techniques based on nonparametric modeling. One potential limitation of parametric models is that they make strong assumptions about the true nature of the time series generating mechanism (e.g., temporal data are generated by a linear, time-invariant process). This limitation, coupled with the advent of powerful and cheap computing devices, motivated researchers to consider more flexible modeling strategies: enter the nonparametric approach, which includes support vector regression models [5]; artificial neural network (ANN) models [10], [11]; and wavelet methods [6].

Although ANNs are often employed for supervised classification and pattern recognition tasks, it was soon realized that ANNs are also a good alternative to regression problems, and more specifically, to time series forecasting tasks. Lapedes reports the first attempt to model nonlinear time series with artificial neural networks [9]. ANNs achieve universal functional approximation; it has been shown that an artificial neural network can approximate any continuous function (linear or nonlinear) to any desired accuracy [4]. Time series modeling, in particular, is a function approximation problem, so ANNs seem to be a natural alternative to this kind of problem. Feed Forward Neural Network (FFNN) models belong to a particular class of ANNs; they combine sigmodal activation functions in order to achieve nonlinear mappings.

In this paper, we'll investigate the robustness of FFNN ensembles applied to quarterly time series forecasting tasks, by comparing their prediction ability with that of SARIMA models (SARIMA is a class of ARIMA model which considers seasonal fluctuations present in temporal data). The rest of this document is organized as follows: Sect. 2 describes briefly the main theoretical concepts used in this experiment to construct our models: Time Series modeling basics, ARIMA models, FFNNs, and the basic PSO algorithm; Sect. 3 describes the experimental methodology; Sect. 4 describes the results of our experiment, and Sect. 5 summarizes our findings and conclusions.

2 Conceptual Tools

2.1 Time Series Modeling: the Problem of Predicting the Future

Extending backwards from time t, we have a time series $\{y_t, y_{t-1}, y_{t-2}, \ldots\}$. Employing this information, we now want to estimate y at some future time $t + s$ (s is called the prediction horizon; typically, $s = 1$). We can accomplish

this task if we assume a model in which y_{t+s} is generated from a functional relationship f involving past y values $y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-d+1}$; i.e.,

$$y_{t+s} = f(y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-d+1}). \tag{1}$$

This is a function approximation problem. To solve it, we'll typically go through the following steps: 1. Assume a generative model f; 2. For every time point t_p in the past, train f using y_{t_p}, y_{t_p-1}, y_{t_p-2}, ..., y_{t_p-d+1} as `inputs` and y_{t_p+s} as the `target` (this step is called *training phase*); 3. Now run the trained generative model f to predict y_{t+s} from $y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-d+1}$. This procedure, characterized by functional relationship (1), is called *auto-regressive approach to time series modeling*. The objective of the training phase is to adjust the behavior (parameters) of f until all of its predictions $\hat{y}_{t_p+s}$ get sufficiently close to corresponding target values y_{t_p+s}. A properly trained generative model f will produce forecasts $\hat{y}_{t+s}$ reasonably close to future time series values y_{t+s}; we say in this case that the model f generalizes well to unknown future time series values.

2.2 ARIMA Time Series Modeling

Perhaps the most popular and traditional of all statistical methods for time series forecasting is *auto-regressive integrated moving average* (ARIMA) modeling. The general ARMA model (without the I) was described in [19], and it was popularized in the 1970 book by [3]; it has the form

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j w_{t-j} + w_t. \tag{2}$$

Auto-Regressive (AR) parameters ϕ_i and Moving Average (MA) parameters θ_j in (2) are estimated from the time series historical part (training data), usually via maximum likelihood; in this way, we obtain maximum likelihood estimates $\hat{\phi}_i$ and $\hat{\theta}_j$. White noise terms w_t are independent, identically distributed random variates which come from a normal distribution $N(0, \sigma_w^2)$. To estimate future values of y_t, we generally assume that w_t at time t is small relative to y_t. We can obtain estimates of past values of w_t at time $t-i$ from past true values of y_t and past values of $\hat{y}_t$: $\hat{w}_{t-i} = y_{t-i} - \hat{y}_{t-i}$; the estimate for y_t is then

$$\hat{y}_t = \sum_{i=1}^{p} \hat{\phi}_i y_{t-i} + \sum_{j=1}^{q} \hat{\theta}_j \hat{w}_{t-j}. \tag{3}$$

Extensions to the basic ARMA model include ARIMA models for dealing with non-stationary time series with trend, and SARIMA models, which are useful to identify seasonal patterns in temporal data; such seasonal patterns are characterized in a SARIMA model by Seasonal Auto-Regressive (SAR) and Seasonal Moving Average (SMA) parameters. For more information, see [13].

2.3 Feed Forward Neural Networks

An artificial neural network can be graphically represented as a group of nodes interconnected by arrows; such representation helps us visualize how information is processed inside a system of artificial neurons. We now describe the structure of a feed forward neural network (FFNN) adapted to our experiment. Figure 1 depicts a FFNN with inputs $x_1, x_2, \ldots, x_d$ and output $\hat{y}$.

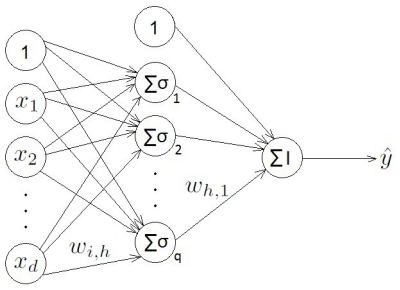

Fig. 1. Feed Forward Neural Network (FFNN)

In the structure from Fig. 1, information flows from left to right. Each arrow inside of the neural network represents a weight w; a signal with value s at the left of arrow w, flows through it, and exits at the right side with value $w \cdot s$. A feed forward neural network (also called a multilayer perceptron) typically contains three layers: input layer, hidden layer, and output layer. The input layer consists simply of the network's inputs; i.e., $x_1, x_2, \ldots, x_d$ (predictor values), plus a constant value of one called intercept. In Fig. 1, our input layer consists of the nodes (circles) aligned vertically on the left side. The column of nodes at the immediate right of the input layer is the hidden layer, which consists of q neurons, or hidden units, placed at the nodes, plus an intercept. Each hidden unit sums all of its inputs, and then transforms this sum via its activation function $\sigma(z) = \frac{1}{1+e^{-z}}$ (sigmoidal logistic function). Mathematically, the functionality of hidden neuron h ($h = 1, 2, \ldots, q$) is described by $\sigma\left(\sum_{i=0}^{d} w_{i,h} x_i\right)$, where $x_0 = 1$ is the intercept coming from the input layer; $w_{i,h}$ is the weight that corresponds to the arrow connecting input node x_i to hidden unit h. The output layer in Fig. 1 consists of one neuron only (since we are interested in forecasting a single time series value), with similar functionality to that of hidden neurons; the difference is that now the activation function has the form $I(z) = z$ (identity function). The functionality of the output layer (and of the whole feed forward neural network) is described by

$$\hat{y} = w_{0,1} + \sum_{h=1}^{q} w_{h,1} \sigma\left(\sum_{i=0}^{d} w_{i,h} x_i\right), \tag{4}$$

where $w_{0,1}$ is the weight that corresponds to the arrow connecting the hidden layer intercept to the output unit, and $w_{h,1}$ is the weight that corresponds to the arrow connecting hidden neuron h to output unit.

FFNNs are trained by using supervised machine learning algorithms, Back-Propagation (BP) being the most representative of such algorithms for FFNN training purposes. BP attempts to minimize a loss function which involves all weights in the FFNN; a typical loss function employed is the mean squared error loss function

$$E(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} - \hat{y}^{(i)}(\boldsymbol{W}) \right]^2, \tag{5}$$

where $y^{(i)}$ is the target (observed value) and $\hat{y}^{(i)}(\boldsymbol{W})$ is the FFNN output for the i-th example input pattern in the training data ($i = 1, 2, \ldots, n$); $\boldsymbol{W}$ is a vector whose components are all the weights in the FFNN. The objective of BP is to adjust the weights in $\boldsymbol{W}$ iteratively, using a stochastic gradient descent technique, until the FFNN output gets sufficiently close to all target values in the training data set. For more details, see [2].

2.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [8] is an optimization algorithm inspired by the motion of a bird flock; any member of the flock is called a "particle". All particles move across a real valued D-dimensional search space, so each particle has three attributes: position $\boldsymbol{x}$, velocity $\boldsymbol{v}$, and best position visited after the first iteration $\boldsymbol{p}$, according to a cost function E. The best of all $\boldsymbol{p}$ values is called global best $\boldsymbol{g}$; this global best position is communicated to all particles such that, at the time of the next algorithm iteration, all the particles are aware of the best position visited. PSO stated goal is to minimize function $E : \mathbb{R}^D \to \mathbb{R}$; i.e., find $\boldsymbol{a} \in \mathbb{R}^D$ such that $E(\boldsymbol{a}) \leq E(\boldsymbol{x})$ for all $\boldsymbol{x}$ in the search space. Figure 2 lists a pseudo-code for the basic PSO algorithm; here, ω is the inertia weight, and φ_p, φ_g are called acceleration coefficients; these meta-parameters must be adjusted by the practitioner.

Recently, PSO has been employed as a FFNN training algorithm; it is easy to adapt the basic PSO algorithm in order to minimize mean squared error loss function (5); simply identify vector $\boldsymbol{W}$, whose components are all the weights in a FFNN, with basic PSO's position vector $\boldsymbol{x}$. Numerous particular applications involving some sort of FFNN-PSO hybrid model for time series prediction can be found in the literature; see for example [1], [12] and [20]. In this paper, we adapt FFNN ensemble models to quarterly time series data, training individual FFNN elements with either BP or basic PSO; individual predictions from an ensemble are averaged out, producing a final prediction for the whole ensemble. Individual predictions taken together, but not combined, form an ensemble channel. See Sect. 3 for more information.

Algorithm 1: basic PSO algorithm

```
for each particle i = 1, 2, ..., N in the swarm do
   initialize position: x_i ← uniform random vector in ℝ^D
   initialize best known position: p_i ← x_i
   if E(p_i) < E(g) update swarm's best known position: g ← p_i
   initialize velocity: v_i ← uniform random vector in ℝ^D
until a termination criterion is met, repeat:
   for each particle i = 1, 2, ..., N in the swarm do
      for each dimension d = 1, 2, ..., D do
         pick random numbers r_p, r_g ~ U(0, 1)
         update velocity: v_{i,d} ← ω v_{i,d} + φ_p r_p(p_{i,d} − x_{i,d}) + φ_g r_g( g_d − x_{i,d})
      update position : x_i ← x_i + v_i
      if E(x_i) < E(p_i) do
         update best known position: p_i ← x_i
         if E(p_i) < E(g) update swarm's best known position: g ← p_i
Now g holds the best found solution
```

Fig. 2. Basic Particle Swarm Optimization (PSO) algorithm.

3 Experimental Methodology

The objective of this experiment is to investigate the robustness of FFNN ensembles applied to quarterly time series forecasting tasks, by comparing their prediction ability with that of SARIMA models. For constructing our models, we employed eleven time series datasets taken from the 2010 Neural Network Grand Competition (NNGC) [16]. The time series datasets employed in this experiment are listed in Table 1. Next we'll describe the conditions under which the experiment was conducted.

Table 1. Time series datasets employed in this experiment. Series with a mark * are seasonally adjusted.

Time series name	from	to	Quarterly values
1. Total operating expenses for US domestic airlines	1993-I	2002-IV	40
2. Personal Expenditures: transportation	1990-I	1997-III	31
3. Motor vehicle output (Autos)	1967-I	2003-IV	148
4. Motor vehicle output (Trucks)	1967-I	2003-IV	148
5. New motor vehicles (Trucks)	1967-I	2003-IV	148
6. Net exports (Autos)	1977-I	2003-IV	108
7. Net exports (Trucks)	1977-I	2003-IV	108
8. Final sales of vehicles to domestic purchasers	1967-I	2003-IV	148
9. Domestic output of new autos	1967-I	2003-IV	148
10. Real Vehicle Output-Private fixed investment *	1967-I	2003-IV	148
11. Real Vehicle Output-Imports *	1967-I	2003-IV	148

For the SARIMA models we employed the basic Box-Jenkins methodology, fitting models of the form ARIMA$(p, d, q)\times(P, D, Q)_4$ to each time series dataset. On the other hand, for FFNN ensembles, we fit a single fixed FFNN structure to each FFNN element in any given ensemble, across all time series datasets.

Prior to commencing model construction, we standardized each time series dataset (i.e., we subtracted dataset mean from each observation and divided resulting difference by dataset standard error); we did this in an initial attempt to minimize the occurrence of convergence problems inside model training procedures. During the construction of SARIMA models, we realized that series 4, 5, 7 and 8 show growing variability (heteroscedasticity), so first we applied a logarithmic transformation to the original series in those cases, and then we standardized the log transformed series. The forecasts produced by all constructed models were compared against transformed target values. In this experiment we considered two types of ensembles: FFNN-BP ensembles, and FFNN-PSO ensembles; all individual models in a given ensemble were trained using just one of the two training algorithms considered in this experiment: Back-Propagation, and PSO.

Model implementation. All models in this experiment were implemented by using the `R` programming language [17]. SARIMA models were implemented with the help of the `astsa` package [14]; individual FFNN-BP models in an ensemble were implemented via the `nnet` package [15], and FFNN-PSO models were programmed from scratch, using a combination of `R` and `Java` code. For FFNN-PSO ensembles, the programming of the FFNN part, was done in `R`, while the PSO part was done in `Java`, in order to improve computing speed. `R` and `Java` code communicate inside the `R` environment via the `rJava` package [18]. The basic PSO algorithm, according to Kennedy and Eberhart [8] was the algorithm of choice in our experiment. For FFNN models, we chose a 4-2-1 FFNN architecture (i.e., 4 input variables, 2 hidden sigmoid units and 1 output linear unit). This choice of FFNN size was made taking into account the parsimony principle for model construction: we chose the minimal number of input nodes capable of capturing annual seasonal patterns for quarterly time series, and a small number of hidden units so as to allow for a moderate amount of nonlinearity. One output unit is all we need, because we are producing 1-step ahead forecasts for univariate time series.

Notational conventions for model building. Suppose a given time series dataset contains N (normalized) values $x_1, x_2, \ldots, x_N$ equally spaced in time. For all FFNN models considered in this experiment, inputs are time series values $x_t, x_{t-1}, x_{t-2}, x_{t-3}$, while for all models (including SARIMA models), the corresponding output $\hat{x}_{t+1}$ is considered to be an estimate of value x_{t+1}. Now we'll describe how we split the time series datasets into training and test sets.

Training set. It consists of all values in a given time series dataset, except for the 4 most recent, which are reserved for testing purposes. For FFNN models, it is necessary to rearrange training set values into a rectangular array like the one shown in Table 2. Columns x_{t-3}, x_{t-2}, x_{t-1}, x_t represent predictor variables, while column x_{t+1} contains target values.

Test set. It consists of the 4 most recent observations $x_{N-3}, x_{N-2}, x_{N-1}, x_N$ in a given time series dataset. These values are to be compared against model predictions $\hat{x}_{N-3}, \hat{x}_{N-2}, \hat{x}_{N-1}, \hat{x}_N$, so we can measure forecast ability for any model; see Equation (6).

Table 2. Training data for FFNN models.

Example	x_{t-3}	x_{t-2}	x_{t-1}	x_t	x_{t+1}
1	x_1	x_2	x_3	x_4	x_5
2	x_2	x_3	x_4	x_5	x_6
3	x_3	x_4	x_5	x_6	x_7
⋮	⋮	⋮	⋮	⋮	⋮
$N-8$	x_{N-8}	x_{N-7}	x_{N-6}	x_{N-5}	x_{N-4}

In order to generate forecast values for any trained model, we proceed as follows: First, we compute $\hat{x}_{N-3}$ by using values x_{N-7}, x_{N-6}, x_{N-5}, x_{N-4} as model inputs; then, we generate test forecast value $\hat{x}_{N-2}$ by using values x_{N-6}, x_{N-5}, x_{N-4}, $\hat{x}_{N-3}$ as model inputs; $\hat{x}_{N-1}$ is generated from inputs x_{N-5}, x_{N-4}, $\hat{x}_{N-3}$, $\hat{x}_{N-2}$, and finally $\hat{x}_N$ is generated from inputs x_{N-4}, $\hat{x}_{N-3}$, $\hat{x}_{N-2}$, $\hat{x}_{N-1}$; notice that each succeeding forecast employs more and more predicted values as inputs, so forecast uncertainty grows as we advance into the distant future.

Test error. To measure forecast ability for any model considered in this experiment, we employed a Mean Squared Error (MSE) measure, computed according to Equation (6).

$$MSE = \frac{\sum_{t=N-3}^{N}(x_t - \hat{x}_t)^2}{4}. \tag{6}$$

FFNN model training. As previously mentioned, a fixed FFNN architecture 4-2-1 with a linear output unit and sigmoidal hidden units was chosen. FFNN-BP models were implemented and trained adjusting weight decay to 0.01, and maximum number of iterations to 1000. These settings were applied to each individual FFNN-BP element in an ensemble, across the 11 time series datasets.

FFNN-PSO models in this experiment employed 100 particles (each particle conceptually being a FFNN), evolving through 200 iterations; particle position components (which happen to be FFNN weights) were randomly initialized using a uniform distribution $U(-2, 2)$; likewise, particle velocity components were randomly initialized, but using a uniform distribution $U(-0.7, 0.7)$. If any particle velocity component v exceeds 1.4 in absolute value at any iteration of the PSO process, it is set back to 0.7, multiplied by the sign of v; this is done to prevent particles from moving too fast in the search space, thus favoring a stable evolution of the PSO process. As for the rest of PSO meta-parameters, the inertia weight ω was set equal to 1.5, while acceleration coefficients φ_p and φ_g were both set equal to 2.0 (see Fig. 2). Why these choices for our FFNN-PSO models? Position and velocity components were randomly initialized with rather small values, as this provides better convergence behavior. In previous experiments, we observed that 10 or even 50 particles were not sufficient for proper convergence; on the other hand, more than 100 particles take too much computation time and produce no better results than 100 particles. Previous experimentation also

shows that 200 iterations are enough for our PSO algorithm to converge; in this experiment, FFNN-PSO training stops exactly after 200 iterations. Acceleration coefficients were both set to 2.0 and inertia weight was set to 1.5 because we observed that the recommended values of 0.72 for inertia weight and 1.49 for both acceleration coefficients, suggested in the literature (see, for example, [7]), were rather slow for proper convergence.

4 Experimental Results

MSE measures for SARIMA models. Table 3 shows MSE measures, computed as indicated by Equation (6), for constructed SARIMA models, along with their final selected orders $(p, d, q) \times (P, D, Q)_4$; these orders were selected carefully: sample ACF and PACF plots were inspected thoroughly in order to determine if regular and/or seasonal differentiation was required; tentative AR, SAR, MA and SMA orders were tested by fitting several SARIMA models; diagnostics were inspected to make sure residuals from a given model were as uncorrelated and normal as possible; finally, to select a winner from a list of competing SARIMA models adequately fitted to a time series dataset, Akaike and Bayesian Information Criteria were used. Recall from Table 1 that series 10 and 11 are seasonally adjusted, which means that all seasonal influences from these series were already removed; this is why P, Q and D orders for SAR, SMA components and Seasonal differencing all equal zero in these series.

Table 3. MSE measures for constructed SARIMA models. std(x) means original series was standardized; std(log(x)) means log of original series was standardized.

Series	SARIMA order	Transformation	MSE
1	$(0,1,0) \times (0,0,0)_4$	std(x)	0.0239
2	$(0,1,0) \times (0,0,0)_4$	std(x)	0.0435
3	$(1,1,1) \times (0,0,0)_4$	std(x)	0.1578
4	$(0,1,0) \times (0,1,5)_4$	std(log(x))	0.0006
5	$(0,1,3) \times (2,0,0)_4$	std(log(x))	0.0033
6	$(1,1,1) \times (2,0,2)_4$	std(x)	0.0467
7	$(1,1,0) \times (0,0,0)_4$	std(log(x))	0.0030
8	$(1,1,0) \times (0,0,0)_4$	std(log(x))	0.0025
9	$(0,1,1) \times (0,0,3)_4$	std(x)	0.1701
10	$(0,1,2) \times (0,0,0)_4$	std(x)	0.0403
11	$(0,1,0) \times (0,0,0)_4$	std(x)	0.0317

SARIMA vs. individual FFNN-PSO models (MSE for test data). Table 4 shows the distribution of prediction errors associated to the elements in each ensemble of FFNN-PSO models (we constructed one ensemble of 100 elements per time series dataset), along with the prediction error associated to the corresponding SARIMA model. From Table 4, we see that for each one of the 11 time series datasets, there is at least one individual FFNN-PSO model which produces a prediction error smaller than the prediction error associated to the corresponding SARIMA model; this tells us that if we build an ensemble

Table 4. SARIMA vs. individual FFNN-PSO (MSE for test data).

series	MSE SARIMA	min	1st q	median	mean	3rd q	max	se
1	0.0239	**0.0169**	0.3649	0.4501	0.4455	0.5468	0.7428	0.1552
2	0.0435	**0.0371**	0.0682	0.1014	0.1441	0.1601	0.7871	0.1376
3	0.1578	**0.0042**	**0.0405**	**0.0754**	**0.0821**	**0.1093**	0.2102	0.0501
4	0.0006	**0.0001**	0.0085	0.0298	0.0812	0.0600	2.1110	0.2321
5	0.0033	**0.0012**	0.0160	0.0556	0.1021	0.1428	0.5756	0.1179
6	0.0467	**0.0406**	0.1906	0.3242	0.3690	0.5258	0.9925	0.2287
7	0.0030	**0.0028**	0.0569	0.0872	0.0885	0.1145	0.2231	0.0456
8	0.0025	**0.0005**	0.0052	0.0184	0.0419	0.0504	0.3193	0.0612
9	0.1701	**0.0206**	**0.0682**	**0.1146**	**0.1155**	**0.1510**	0.2696	0.0573
10	0.0403	**0.0184**	0.0762	0.1655	0.2164	0.3051	0.7544	0.1742
11	0.0317	**0.0096**	**0.0221**	0.0346	0.0773	0.0720	0.7600	0.1224

of FFNN-PSO models with at least 100 elements, some of those elements will produce better forecasts than an adequately-built SARIMA model; additionally, all forecasts from the FFNN-PSO ensemble will form an "ensemble channel" which hopefully will contain all time series test values without being too wide (see Fig. 6). Thus, with a single fixed template employed to construct all FFNN-PSO models in this experiment, we were able to produce competitive FFNN-PSO ensembles, at the level of well-constructed SARIMA models; the single template approach could be an economic alternative if fitting individual models for each time series dataset turns out to be a very time consuming task; of course, better results would be achieved by adapting a FFNN-PSO model template to each time series dataset.

SARIMA vs. individual FFNN-BP models (MSE for test data). We also performed an analogous comparison between prediction errors associated to individual FFNN-BP elements in an ensemble, and to SARIMA models; the results are shown in Table 5. We see that, for time series 1, 2, 4, 5, 6, 7 and 10, individual FFNN-BP models never produce better time series forecasts; by contrast, for series 3, 8, 9 and 11, nearly all individual FFNN-BP models produce better time series forecasts with respect to their SARIMA counterparts. Note also from Table 4 and Table 5, that prediction errors for FFNN-PSO ensembles contain more variability than their FFNN-BP counterparts; the resulting variability induced by the PSO process will allow us to estimate prediction limits (see Fig. 3). Unfortunately, small variability of prediction errors from FFNN-BP ensembles does not help in the construction of ensemble channels (see Fig. 4).

FFNN ensemble channels and SARIMA confidence intervals. Figure 3 shows the averages of individual estimations produced by all elements in the 11 FFNN-PSO ensembles built in this experiment (one average per ensemble), along with the corresponding original time series, and the limits for the ensemble channels; the thick vertical lines shown here separate training data from test data. Figure 4 is the FFNN-BP counterpart of Fig. 3, and Fig. 5 is similar to Fig. 3, but shows instead the SARIMA forecasts along with their corresponding prediction limits, built upon the assumption of residual normality. From Fig. 3 and Fig. 5, we see that most test data for the 11 time series datasets are contained

Table 5. SARIMA vs. individual FFNN-BP (MSE for test data).

series	MSE SARIMA	min	1st q	median	mean	3rd q	max	se
1	**0.0239**	0.3192	0.5262	0.5306	0.5202	0.5319	0.5956	0.0370
2	**0.0435**	0.0495	0.0495	0.0521	0.0526	0.0522	0.0859	0.0050
3	0.1578	**0.0884**	**0.1012**	**0.1012**	**0.0991**	**0.1022**	**0.1034**	0.0051
4	**0.0006**	0.0107	0.0109	0.0110	0.0132	0.0139	0.0203	0.0035
5	**0.0033**	0.0085	0.0085	0.0092	0.0140	0.0126	0.0347	0.0089
6	**0.0467**	0.0609	0.0647	0.0647	0.0688	0.0655	0.2049	0.0200
7	**0.0030**	0.0547	0.0599	0.0608	0.0613	0.0643	0.0644	0.0026
8	0.0025	**0.0013**	**0.0013**	**0.0014**	**0.0025**	**0.0019**	0.0070	0.0020
9	0.1701	**0.0776**	**0.0787**	**0.0798**	**0.0835**	**0.0838**	**0.0981**	0.0067
10	**0.0403**	0.0873	0.0938	0.0970	0.0971	0.0978	0.1176	0.0050
11	0.0317	**0.0110**	**0.0112**	**0.0112**	**0.0113**	**0.0113**	**0.0117**	0.0002

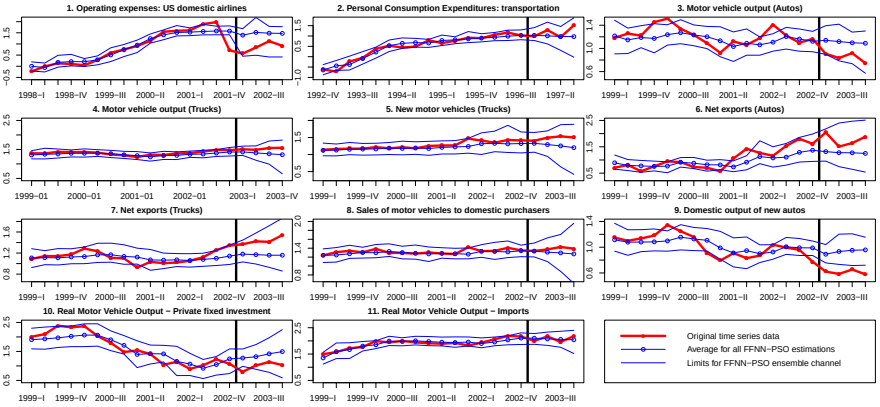

Fig. 3. FFNN-PSO ensemble predictions for all time series datasets.

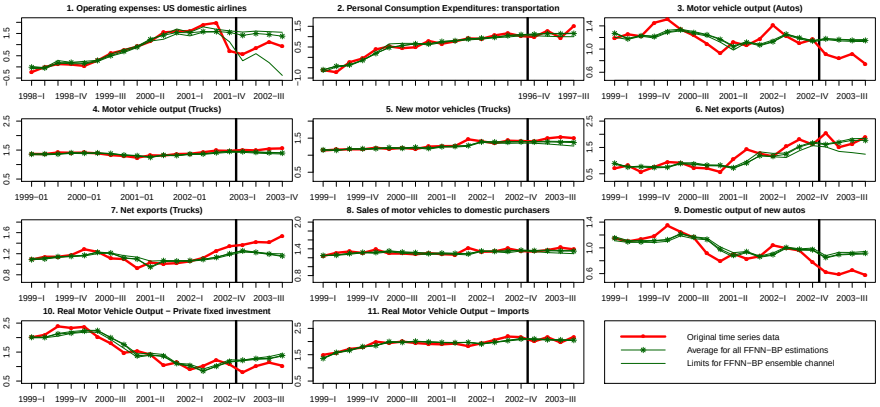

Fig. 4. FFNN-BP ensemble predictions for all time series datasets.

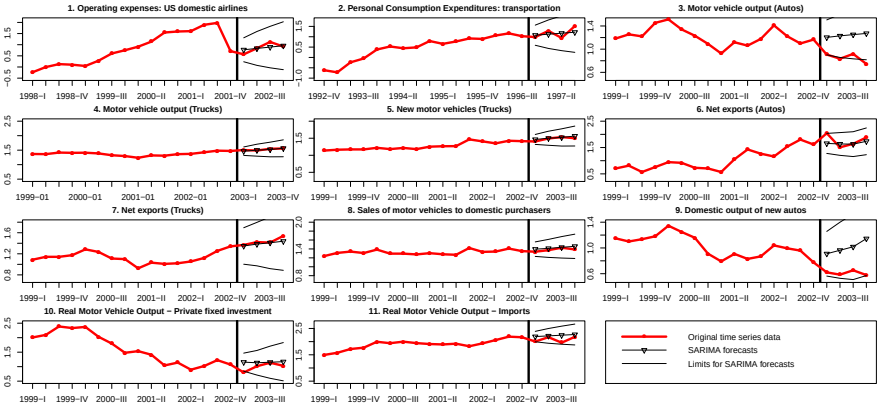

Fig. 5. SARIMA predictions for all time series datasets.

inside the prediction limits defined by the models. To be more precise, SARIMA models enclose 91% of all test data within their prediction limits (indeed getting close to the theoretical expected confidence value of 95%), while FFNN-PSO ensembles enclose 89% of all test data within their ensemble channels; by contrast, FFNN-BP ensembles enclose only 16% of all test data within their prediction limits. Thus, SARIMA models and FFNN-PSO ensembles are able to produce accurate interval estimations, in addition to point forecasts; FFNN-BP ensembles practically give us point forecasts only (except for time series 1 and 6; see Fig. 4). From Fig. 3, we see that FFNN-PSO forecast limits for time series 4, 5 and 8 are too wide; note also that in these cases, ensemble channels enclose training data comfortably, even with a spare margin (by the way, in Fig. 5 we see that SARIMA forecast limits for time series 1, 3, 7 and 9 are also too wide).

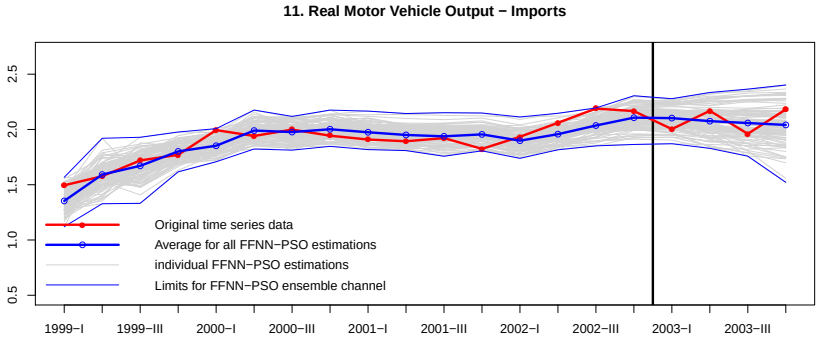

Fig. 6. FFNN-PSO ensemble channel for time series 11.

5 Conclusions

In this experiment, we obtained adequate SARIMA models; their construction required statistical knowledge and good judgment in order to select the best model template for each time series; forecasts produced by these SARIMA models served as a baseline for our model comparison experiment. On the other hand, FFNN ensemble models for all quarterly time series datasets were readily constructed from a single FFNN template, without making any previous assumptions about the data generating mechanism; the point in comparing FFNN ensemble models built from a single template against individual SARIMA models is to test robustness of FFNN ensembles applied to quarterly time series prediction tasks. All individual FFNN elements in an ensemble model were trained using just one optimization algorithm (Back-Propagation or Particle Swarm Optimization) with pre-established meta-parameters, and a fixed training set. Estimations from individual FFNN elements in an ensemble were averaged to form a final ensemble prediction; all individual predictions from the ensemble elements, taken together, but not combined, form an ensemble channel, which suggests multiple possibilities for future time series movements. All FFNN-PSO ensembles generated competitive forecasts, at the level of well-constructed SARIMA models; the single template approach for adapting FFNN-PSO ensembles to multiple time series datasets of quarterly frequency can be an economic and sensible alternative if fitting individual models for each time series dataset turns out to be a very time consuming task. FFNN-PSO ensembles were able to produce accurate interval estimations, in addition to good point forecasts; FFNN-BP ensembles generally produced point forecasts only. In some cases, individual FFNN-BP models produced forecasts more accurate than those produced by SARIMA models or by FFNN-PSO ensembles. Prediction intervals from SARIMA and FFNN-PSO ensemble models covered most test data; in a few cases, prediction intervals for both models were too wide; thus, PSO meta-parameter fine-tuning is necessary in a case-by-case basis. Future work should also include testing FFNNs of varying complexity, to see if forecasting accuracy can be further improved.

Acknowledgments. The first author gratefully acknowledges the financial support from CONACyT to pursue graduate studies at Universidad Autónoma de Aguascalientes. The authors would also like to thank all people involved in the reviewing process for their comments and suggestions which contributed to the improvement of this article.

References

1. Adhikari R., Agrawal R., Kant L.: PSO based Neural Networks vs. traditional statistical models for seasonal time series forecasting. 3rd International Advance Computing Conference (IACC), pp. 719–725. IEEE (2013)
2. Bishop C.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)

3. Box G., Jenkins G.: Time series analysis: forecasting and control. New York, NY: Holden Day (1970)
4. Cybenko G.: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems 2(4):303–314 (1989)
5. Drucker H., Burges C., Kaufman L., Smola A., Vapnik V.: Support vector regression machines. Advances in neural information processing systems, 9, 155–161 (1997)
6. Härdle W.: Nonparametric and semiparametric models. Springer Science & Business Media (2004)
7. Hsieh S., Sun T., Liu C., Tsai S.: Efficient Population Utilization Strategy for Particle Swarm Optimizer. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol 39, Issue 2, IEEE (2009)
8. Kennedy J., Eberhart R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference On Neural Networks, IEEE, vol. 4, pp 1942–1948 (1995)
9. Lapedes A., Farber R.: Nonlinear signal processing using neural networks: Prediction and system modeling. Tech. Rep. LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM (1987)
10. McCulloch W., Pitts W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115–133 (1943)
11. Rosenblatt F.: The perceptron - a perceiving and recognizing automaton. Tech. Rep. 85-460-1, Cornell Aeronautical Laboratory (1957)
12. Sheikhan M., Mohammadi N.: Time series prediction using PSO-optimized neural network and hybrid feature selection algorithm for IEEE load data. Neural Computing and Applications, Volume 23, Issue 3-4, pp 1185–1194, Springer-Verlag London Limited (2013)
13. Shumway R., Stoffer D.: Time series analysis and its applications (with R examples). 3rd edn, Springer Science+Business Media, LLC (2011)
14. Package "astsa", http://www.stat.pitt.edu/stoffer/tsa3/
15. Package "nnet", http://www.stats.ox.ac.uk/pub/MASS4/
16. Artificial Neural Network & Computational Intelligence Forecasting Competition, http://www.neural-forecasting-competition.com/instructions.htm
17. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/ (2014)
18. Package "rJava", http://www.rforge.net/rJava/
19. Whittle P.: Hypothesis Testing in Time Series Analysis. Hafner Publishing Company (1951)
20. Yeh W.: New Parameter-Free Simplified Swarm Optimization for Artificial Neural Network Training and Its Application in the Prediction of Time Series. IEEE Transactions on Neural Networks and Learning Systems, Vol. 24, Issue 4, pp. 661–665, IEEE (2013)