Particle swarm optimization versus gradient based methods in optimizing neural network

B Warsito¹, H Yasin¹, A Prahutama¹

¹Department of Statistics, Diponegoro University
Jl. Prof. Soedharto, SH, Tembalang, Semarang 50275, Indonesia
E-mail: budiwrst2@gmail.com

Abstract. Neural network modelling has become a great interest for many statisticians to be utilized in various types of data as classification, regression, and time series. It also has been applied in many fields as environment, financial, medical, agriculture and climate change. A lot of parametric methods have been developed to predict time series data such as ARIMA and exponential smoothing. However, requirement of residual assumptions causes limitedness of the models. Time series prediction by using neural network been an interesting study in the forecasting problem. In this model, one of the most interesting discussion is about how to get the optimal weights. Various gradient and non-gradient based methods have been applied in obtaining the network weights. Particle swarm optimization is one non-gradient based algorithm inspired by the behaviour of birds and fish flocks, which move to form certain formations without colliding to get the best position in a multi-dimensional space. In neural network modelling, the number of input and hidden unit give influence to the network architecture. The more complex an architecture, the more network weights must be estimated. In this study, a comparison of particle swarm optimization and some gradient based methods on the optimizing neural network was studied. Comparative studies were performed on both stationary and non-stationary data. Experiments were conducted several times to obtain optimal accuracy and stability of results, through statistics of mean and variance of MSE values.

1. Introduction

Recently, neural network modelling has been undergoing very rapid development. Neural network model has been applied to various problem areas such as time series [1], regression and classification [2]. In the neural network, there are three processing layers, namely input, output, and hidden layer. The neurons in each layer are interconnected through synaptic weights. Connection weights between layers indicate the strength of the relationship. The neurons in the input layer send signals to hidden layers in the form of weighted summation. Furthermore, the hidden layer transfers the signal through the nonlinear activation function. The transfer results are sent to the output layer also in the form of weighted summation. Thus, output can be defined as a form of function from input. This form of function is expected to produce output that resembles the target. During the learning process, synaptic weights are updated to obtain optimal weights. The learning process will continue until the maximum epoch is reached or the minimum error is obtained. This is referred to as the iteration stopping criteria. The problem is whether the weights obtained have reached the global optimum or only reached the local optimum. The choosing of optimization methods then becomes very important to obtain the optimum weights. Gradient-based optimization methods will require that the activation function must
be continuous and differentiable because the updating process of weights is done through calculations using derivative functions.

Various classic gradient based algorithms to train neural network have been proposed and developed in the recent years [3-5]. However, many of them can stay trapped in non-desirable solutions. Gradient based optimization methods oftentimes incapable of getting the global optimum and often been trapped in local optimum at that early stage. They will be far from the optimum or the best solution [6]. Therefore, other kinds of techniques are developed for training neural network. Many heuristic algorithms were then addressed to solve the problem. Heuristic techniques make use of problem specific knowledge to find efficient solutions [7]. In this research, we applied the Particle Swarm Optimization (PSO), a heuristic optimization, for training neural network and compare the performance with some gradient based methods.

2. Material and Methods
Neural network is a modeling algorithm inspired by biological neural. The three layers of the interconnected neural network are input layer, hidden layer and output layer. The connections between neurons in the layers represent the strength of relationship between each unit and explained as weights. More complex the network, more weights should be obtained. Architecture of neural network for time series prediction is described in figure 1. Input layer contains lagged data in the past whereas the current data is at the output layer. The activation function contains a function that transfers signals sent from the input layer in the form of weighted summation. The signals resulting from this layer are then sent to the output layer, also in the form of weighted summation. The output produced at this layer is the predictive value.

![Figure 1. Architecture of neural network for time series prediction](image)

Backpropagation is a standard algorithm for obtaining the optimal weights of a network. The term backpropagation refers to the manner in which the gradient is computed for nonlinear multilayer network. Because of the non-closed form term, the solution of the optimization problem should use a numerical approach. A method based on the second derivative has been widely used for solving the problem. On the other hand, the utilizing of heuristic optimization methods also has been evolved. In this research, the weights are optimized by the two class of optimization methods. The first method is the three gradient based methods including Conjugate Gradient, Levenberg-Marquardt and BFGS, whereas the second is particle swarm optimization as a class of heuristic algorithm.

2.1. Gradient Based Methods
In a highly nonlinear problem as neural network, there is no such direct method to find the weights, so they should be established iteratively. Trying all of the possible combinations of weights randomly would be prohibitively costly in time and effort [8]. In backpropagation, it is important to calculate the derivative of any activation functions used. Before training neural network, the weights and biases
must be initialized. During the training process, the weights and biases are iteratively adjusted to minimize the network performance. Gradient based algorithms use the gradient of the performance function to determine how to adjust the weights to minimize the performance.

2.1.1. Conjugate Gradient. The basic backpropagation adjusts the weights in the negative of the gradient. In the conjugate gradient algorithm, a search is performed along conjugate directions [6]. It starts out by searching in the negative of the gradient on the first iteration, \( d^{(0)} = -g^{(0)} \). A line search is then performed to determine the optimal distance to move along the current search direction, \( w^{k+1} = w^k + \alpha_k d^{(k)} \). The next search direction is determined so that it is conjugate to previous search direction, \( d^k = -w^k + \beta_k d^{(k-1)} \). There are various versions of conjugate gradient distinguished by the manner in which the constant \( \beta_k \) is computed. For example, the Fletcher-Reeves update is \( \beta_k = \frac{g^{(k)T}g^{(k)}}{g^{(k-1)T}g^{(k-1)}} \).

2.1.2. Quasi-Newton BFGS. The basic step of Newton’s method is \( w^{k+1} = w^k + H_k^{-1}d^{(k)} \), where \( H_k \) is the Hessian matrix of the performance index at the current values of the weights. This method requires a second derivative of the performance and the consequence is too complex and expensive for neural network. A class Newton’s method which don’t require calculation of second derivative is quasi-Newton [6]. It updates an approximate Hessian matrix at each iteration of the algorithm. One kind of Quasi-Newton is the Broyden, Fletcher, Goldfarb and Shanno (BFGS update). The approximate Hessian must be stored and its dimension is \( n^2 \times n^2 \), where \( n \) is equal to the number of weights and biases.

2.1.3. Levenberg-Marquardt. Similar to Quasi-Newton, the Levenberg-Marquardt method was designed to approach second-order training without having to compute the Hessian matrix. The Hessian matrix can be approximated as \( H = J^TJ \) and the gradient is \( g = J^Te \), where \( J \) is the Jacobian matrix which contains first derivatives of the error with respect to the weights and \( e \) is a vector of error. Levenberg-Marquardt uses the approximation to the Hessian matrix in the update term, \( w^{(k+1)} = w^{(k)} - (J^TJ + \mu I)^{-1}J^Te \), where \( \mu \) is a scalar.

2.2. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is inspired by social behavior of bird flocking or fish schooling without colliding with each other [9]. It initialized with a population of random solutions and searches for optima by updating generations. Unlike Genetic Algorithm, PSO has no evolution operators [10]. This algorithm optimizes problems by moving particles (prospective solutions) in the problem space using certain functions for the position and velocity of the particles. The movement of particles is affected by the best solution of the particle, and the best solution is generally obtained from other particles. This set of particles is called a swarm, and in the end this swarm will move towards the best solution. Each solution (particle) is then evaluated for its quality using the fitness function. The particles fly through the problem space by following the current optimum particles. PSO has two main tasks, namely velocity updates and position updates. PSO starts by initializing a random set of particles as the solutions. It is including the initial position and initial velocity of each particle. The fitness value is then determined by using the initial set. The particles used to update the velocity and position of the particles will continue to accelerate near the position of the previous best particles (pBest) and the best globally (gBest), until the minimum error conditions are reached. The update of velocity and position of a particle is formulated as:

\[
\begin{align*}
\nu_{i}^{t+1} & = \rho \cdot \nu_{i}^{t} + c_1 \cdot r_1(pBest_{i}^{t} - x_{i}^{t}) + c_2 \cdot r_2(gBest_{i}^{t} - x_{i}^{t}) \\
x_{i}^{t+1} & = x_{i}^{t} + \nu_{i}^{t+1}
\end{align*}
\]
where $\rho$ is inertia weight, $c_1$ and $c_2$ are acceleration coefficients, $r_1$ and $r_2$ are random values of continue uniform distribution. Symbols of $pBest^t_i$ describe the best position of particle-$i$ at iteration-$t$, whereas $x^t_i$ is the particle position $i$ at iteration $t$. In each stage of the process, the inertia weight also updated by using the formula:

$$\rho = \rho_{\text{max}} - \frac{\rho_{\text{max}} - \rho_{\text{min}}}{t_{\text{max}}} \times t$$  \hspace{1cm} (3)

where $\rho_{\text{max}}$ is the upper bound of $\rho$, $\rho_{\text{min}}$ is the lower bound of $\rho$, $t_{\text{max}}$ is the maximum of iteration and $t$ is the current iteration. The iteration process is done by comparing fitness at the current position with the previous, it will be obtained $pBest$ and $gBest$. The process will occur until the maximum number of iterations is reached.

3. Results and Discussion

The data used in this paper is the monthly data of Suspended Particle Material (SPM) in Station of Climatology Semarang Central Java from January 2009 until November 2017. SPM is a complex mixture of extremely small particles and liquid droplets that get into the air. It is one of the air quality parameters. In this case, the first 100 data are used as training and the 19 remaining as testing. Base on the partial autocorrelation function, we used the 1$^{\text{st}}$ and 2$^{\text{nd}}$ lags as input. In obtaining the better result, the logistic sigmoid is used as the activation function in hidden layer whereas the activation function in output layer is linear. The number of hidden units in the hidden layer is selected by choosing the best of one until eight neurons. In each architecture, the training process is done for 1000 iterations and we repeated 30 times for obtaining the best one. Figure 2 shows plot of the one experiment for PSO optimization with 1000 iterations. It shows that convergence result was reached. After about four hundredth iterations, the optimal value is stable and does not show a declining pattern anymore. The resulting values have formed a straight line that converges to a certain value. The similar results for the other experiments also been obtained. This shows that the 1000 iterations specified are sufficient for experiments of this data. We also found a similar tendency for gradient-based methods that were tried with the same number of iterations.

![Figure 2. Plot of the iteration of PSO for optimizing neural network](image-url)
After 30 experiments, the statistics of mean (average) and variance of MSE were obtained. The least MSE and the most stable result been the base of the comparison for obtaining the best architecture. We used both in-sample prediction using training data and out-sample prediction using testing data as a consideration for selecting the best model and the best optimization method. The results of all of the experiments are shown in table 1.

Table 1. Average and variance of MSE of neural network model for prediction SPM data by using PSO and gradient based methods.

| Hidden units | training data | testing data |
|--------------|---------------|--------------|
|              | average (x10³) | variance (x10⁴) | average (x10³) | variance (x10⁴) |
| PSO          |               |               |               |               |
| 1            | 2.4526        | 1.0550        | 1.9040        | 0.1670        |
| 2            | 2.6226        | 1.2227        | 2.1746        | 0.3186        |
| 3            | 2.5190        | 2.6240        | 2.3119        | 10.4641       |
| 4            | 2.5877        | 2.9495        | 2.1537        | 3.9142        |
| 5            | 2.4616        | 3.2345        | 2.1999        | 2.7193        |
| 6            | 2.6275        | 7.8960        | 2.5163        | 21.1604       |
| 7            | 2.3981        | 3.3636        | 3.2543        | 20.2160       |
| 8            | 2.4086        | 1.2571        | 2.3722        | 6.2759        |
| LM           |               |               |               |               |
| 1            | 3.5076        | 25.0201       | 1.9302        | 16.2332       |
| 2            | 2.6759        | 2.6642        | 11.4640       | 10.7094       |
| 3            | 2.5257        | 2.3211        | 5.1432        | 43.7605       |
| 4            | 2.4000        | 2.0529        | 11.9230       | 31.2874       |
| 5            | 2.1768        | 3.2072        | 6.6565        | 63.8726       |
| 6            | 2.0844        | 4.0233        | 18.1771       | 30.3752       |
| 7            | 1.7479        | 5.2692        | 45.5102       | 79.5574       |
| 8            | 1.6615        | 9.1932        | 13.4202       | 15.4808       |
| CG           |               |               |               |               |
| 1            | 3.9138        | 41.4363       | 2.2573        | 39.0723       |
| 2            | 3.1640        | 18.8316       | 2.0690        | 10.4621       |
| 3            | 2.5684        | 2.6722        | 2.7595        | 35.4216       |
| 4            | 2.4269        | 8.3865        | 3.5313        | 88.1197       |
| 5            | 2.2764        | 2.2651        | 3.4641        | 10.1032       |
| 6            | 2.2254        | 1.7293        | 4.4395        | 11.0978       |
| 7            | 2.1090        | 3.3399        | 5.8489        | 40.6859       |
| 8            | 1.9958        | 1.3423        | 3.5573        | 58.8693       |
| BFGS         |               |               |               |               |
| 1            | 3.9312        | 46.3484       | 2.0238        | 31.1346       |
| 2            | 2.8321        | 5.9851        | 1.8783        | 3.7068        |
| 3            | 2.6151        | 1.7864        | 35.3902       | 17.7642       |
| 4            | 2.4198        | 2.2975        | 11.3446       | 36.5145       |
| 5            | 2.2772        | 2.6530        | 49.9512       | 52.2476       |
| 6            | 2.1082        | 2.9405        | 9.7373        | 12.0195       |
| 7            | 1.9912        | 3.4155        | 36.2853       | 57.8453       |
| 8            | 1.8040        | 3.0970        | 45.5771       | 25.3714       |

Table 1 shows that PSO gives a stable result at in-sample prediction, although that is not the best result compared with the other techniques. In various number of hidden units, the averages of replication are almost same, likewise with the results of the out-sample predictions. The average of MSE using the Levenberg-Marquardt method with 8 hidden units gives the best result in training data but not for the variance. It gives a high value and indicates the instability of the estimation result. Similarly, the result of the testing data. In fact, the result of out-sample prediction with this architecture gives a poor result.
The average is very high and so is the variance. The similar result also given by BFGS method with 8 hidden units. Conjugate Gradient method with 8 hidden units gives more precision result in both average and variance of the training data but prediction of the testing data also does not provide satisfactory result, especially on the variance. The different type is obtained from the PSO algorithm. Architecture with one hidden unit gives a stable result in both training and testing data, likewise with two hidden units. In fact, the model with one hidden unit is one that produces the best out-sample prediction. Based on the results obtained, we can state that gradient based methods with the chosen architecture give a better result for in-sample prediction with training data but it needs some replications for obtaining the best one. The disadvantage of this method is that in general, the out-sample prediction obtained is unsatisfactory. Different pattern obtained in the PSO method. We do not need many replications for obtaining the best architecture and there will be a guarantee that the out-sample prediction resulted will be accurate. Plots of the in-sample and out-sample predictions in one experiment of PSO method are shown in figure 3. It can be seen that the results of in-sample and out-sample predictions are approach the actual data.

4. Conclusion
Procedure of choosing optimum architecture of neural network modeling optimized by PSO and gradient based optimization in time series prediction have been developed. The using of statistics average and variance of the repeating process with various architecture for obtaining the best model by observing the least and the most stable results been the effective ways. Simple architecture with few hidden units is the chosen architecture in each optimization method. In this case, gradient based methods give smaller average of MSE, but poor in variance. PSO give more stable prediction. Comparison of Particle Swarm Optimization versus other heuristic optimizations and the using PSO for optimization various neural network models like RBFNN, CFNN and RNN can be done as the future works.

Acknowledgement
This research was funded by Diponegoro University by scheme of Research of Development and Application (RPP) with the contract number 474-29/UN7.P4.3/PP/2018.

References
[1] Gheyas I A and Smith L S 2009 Proceedings of the World Congress on Engineering II July 1-3 London U.K
[2] Bourdes V, Bonnevay S, Lisboa P, Defrance R, Perol D, Chabaud S, Bachelot T, Gargi T and Negrier S 2010 Advances in Artificial Neural Systems vol 2010 article ID 309841
[3] Garro B A and Vázquez R A 2015 Computational intelligence and neuroscience 61.
[4] Khadse C B, Chaudhari M A and Borghate V B 2016 International Journal of Electrical Power & Energy Systems 82 197.
[5] Chattopadhyay S and Chattopadhyay G 2018 *Journal of Atmospheric and Solar-Terrestrial Physics* **179** 202.

[6] Samarasinghe S 2007 *Neural Networks for Applied Sciences and Engineering: from Fundamentals to Complex Pattern Recognition* (New York: Auerbach Publications Taylor and Francis Group).

[7] Potdar G P P and Thool R C 2014 *International Journal of Artificial Intelligence & Applications (IJAIA)* **5** 63.

[8] Demuth H and Beale M 2004 *Neural Network Toolbox for Use with Matlab* User’s Guide Version 4 (The MathWorks)

[9] Cuevas E, Cienfuegos M, Zaldivar D and Cisneros M P 2014 *Expert Systems with Applications* **40** 6374.

[10] Sivanandam S N and Deepa S N 2009 *International Journal of the Computer, the Internet and Management* **17** 1.