

Learning Functions Generated by Randomly Initialized MLPs and SRNs

Ryan Cleaver, *Student Member, IEEE* and Ganesh Kumar Venayagamoorthy, *Senior Member, IEEE*

Abstract— In this paper, nonlinear functions generated by randomly initialized multilayer perceptrons (MLPs) and simultaneous recurrent neural networks (SRNs) and two benchmark functions are learned by MLPs and SRNs. Training SRNs is a challenging task and a new learning algorithm – PSO-QI is introduced. PSO-QI is a standard particle swarm optimization (PSO) algorithm with the addition of a quantum step utilizing the probability density property of a quantum particle. The results from PSO-QI are compared with the standard backpropagation (BP) and PSO algorithms. It is further verified that functions generated by SRNs are harder to learn than those generated by MLPs but PSO-QI provides learning capabilities of these functions by MLPs and SRNs compared to BP and PSO.

I. INTRODUCTION

THE efficient and accurate training of neural networks (NN's) to approximate functions has been an open topic for many years. In fact, they are known as universal approximators [1]. Many forms of neural networks exist, but this paper examines a popular form: Multilayer Perceptrons (MLPs), and a powerful form: Simultaneous Recurrent Neural Networks (SRNs). Whereas MLPs are basic feedforward networks, SRNs are made more computationally complex by the addition of a recurrence between the layers of the network. This simultaneous recurrence allows the SRN to approximate more complex functions than an equivalent MLP [2], but training is significantly harder and time consuming. Many training algorithms have been developed and tested for use in approximating functions, even down to methods originally intended for filter design, such as Extended Kalman Filter (EKF) [3].

In order to study new learning algorithms for neural networks, nonlinear functions are generated by randomly initialized MLPs (Case 1) and SRNs (Case 2) in this paper, as in [4]. Binary algorithms for the training of neural networks are compared in [4], whereas this paper investigates real-valued based algorithms. Three algorithms are studied, the first of which is standard Backpropagation (BP). This method is one of the earliest training algorithms for developed for neural networks by Werbos [5]. Backpropagation is a gradient descent method and so can only be used effectively on differentiable functions.

The second algorithm tested is an adaptive inertia version of the canonical Particle Swarm Optimization (PSO), developed by James Kennedy and Russell Eberhart in 1995 [6]. This algorithm models social animal behavior such as flocks of birds and schools of fish, causing

populations of agents to wander through a hyper-dimensional search space hunting for target points through communication and competition with other members of the swarm.

Finally, a variant of PSO known as PSO with Quantum Infusion (PSO-QI) is implemented. This algorithm utilizes a quantum update of one or more particles in the swarm to guide the swarm to faster convergence [7]. Quantum-inspired behavior is an interesting development in the field of Computational Intelligence (CI). Several algorithms have been developed to either enhance the behavior of existing algorithms, such as in PSO-QI or Quantum Evolved PSO (QEPSO) [8], or even to create an entirely new algorithm based on this property, as in the Quantum-Inspired Evolutionary Algorithm (QEA) [9]. The difference of PSO-QI, however, is that it is performed directly in real space instead of binary. It also takes a novel approach to the quantum influence. It performs on the PSO particles an interesting expansion into a probability distribution based on the convergence of the swarm.

II. MLPs and SRNs

While there are many types of neural network, this study focuses on learning associated with two types: multilayer perceptron feedforward neural networks, and simultaneous recurrent neural networks. All inputs and outputs are linear activation functions in this study whereas all hidden layer activation functions are sigmoidal, given below:

$$f_{act} = \frac{1}{1 + e^{-\lambda(net-\theta)}} \quad (1)$$

θ = threshold value, derived from a bias input into the network. It is used to influence the output strength of a neuron [10]. λ is a slope coefficient, used to modify the slope of the sigmoid.

MLPs are the oldest and most popular form of neural networks used today. They consist generally of three layers: an input layer, hidden layer, and output layer. The specific one used in Cases 1 and 2 (randomized functions) of this study is shown in Figure 1. The input layer is composed of the inputs to the neural network, along with a bias of 1. This network of size $3 \times 5 \times 1$ has a total of 20 weights: 15 for the input-hidden layer, and 5 for the hidden-output layer. The network for Cases 3 and 4 (the benchmark functions) is $2 \times 5 \times 1$, due to having only one input besides the bias.

This work is supported by the US National Science Foundation under the EFRI #0836017 CAREER grant ECS #348221.

The authors are with the Real-Time Power and Intelligent Systems Laboratory at Missouri University of Science and Technology, Rolla, MO 65409, USA (email: gkumar@ieee.org)

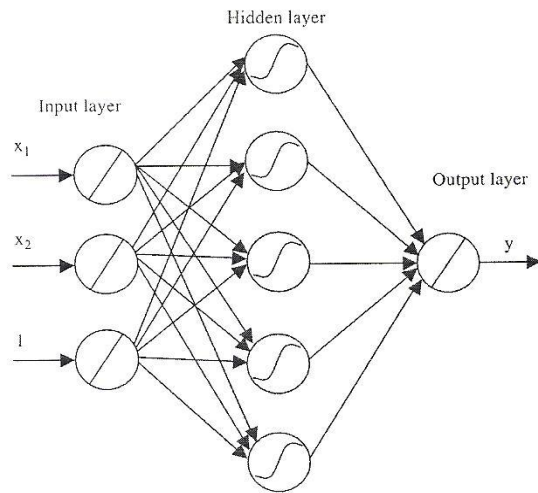


Fig. 1. MLP network structure ($3 \times 5 \times 1$) for Cases 1a and 2a

The SRN is a slightly more complicated form of a neural network. The one shown in Figure 2 and used in this study is of a Jordan recurrent type. This means that there is a feedback loop from the output to the input layer, resulting in the “Context” input layer [10]. The simultaneous nature arises from the inputs and output being held constant for a certain period of internal oscillations. This means that for a specified number of runs the inputs are not changed and the output is not sampled. Only after the specified number of internal iterations is the output sampled. This allows the output to settle down before being sampled. Figure 3 is an example of how the context layer tends to settle down over time. The SRN in Figure 2 of size $4 \times 5 \times 1$ has a total of 25 weights: 20 for the input-hidden layer and 5 for the hidden-output layer. The feedback connection between the output and input layers is not weighted.

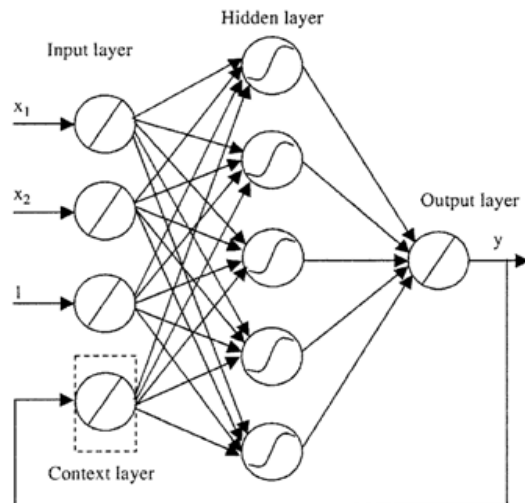


Fig. 2. SRN network structure ($4 \times 5 \times 1$) for Cases 1b and 2b

However, there is also a small portion of the time with certain network weights and inputs where the context layer of an SRN does not settle and instead shows positive feedback with sustained oscillations. An example of this is

in Figure 4. As can be seen from this plot, the variations drawn from the context layer can be quite extreme (-9 to 1).

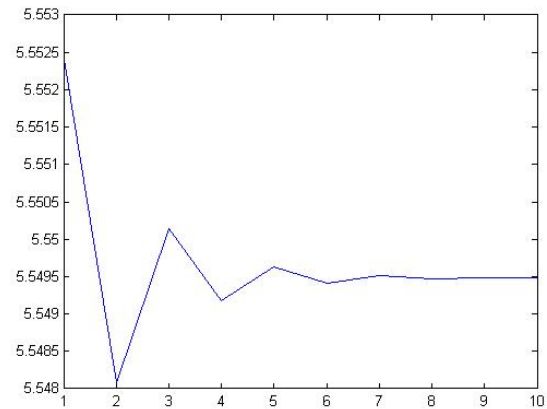


Fig. 3. Sample of output layer over 10 internal iterations of SRN with context layer settling

Generating functions with an SRN exhibiting the positive feedback or sustained oscillations behavior can have interesting consequences. Figure 5 shows two functions generated by the neural networks shown in Figures 1 and 2, respectively. The SRN function looks very different despite having the exact same 15 weights as the MLP plus 5 extra weights to connect the context input to the output layer. Sharp features can also be seen, making this function difficult to learn.

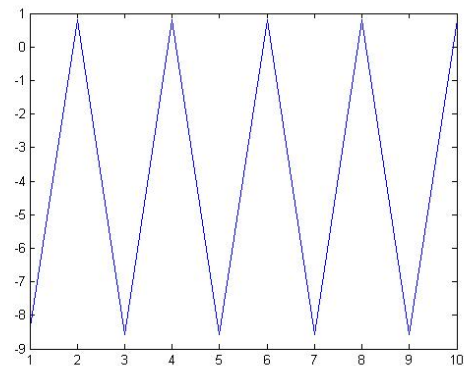
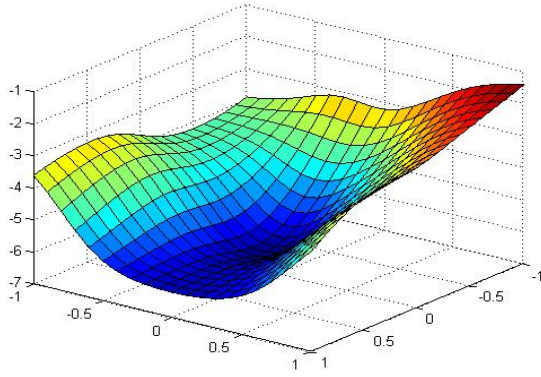


Fig. 4. Sample of output layer over 10 internal iterations of SRN with context layer showing signs of positive feedback

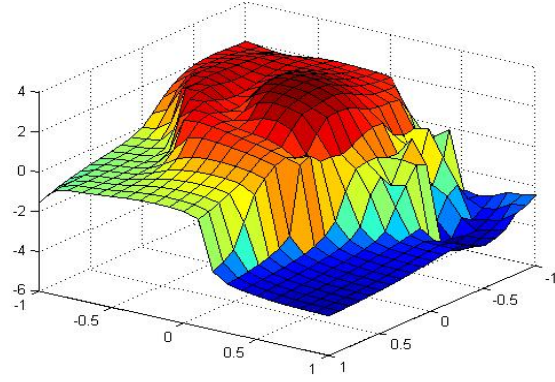
III. BACKPROPAGATION ALGORITHM

Backpropagation is the earliest form of algorithm developed for the training of neural networks, specifically MLPs. It is based on gradient descent mechanism, propagating error backward through the network and using it in the weight update equations, in what is effectively the exact opposite of the feedforward equations.

Knowing the error identifies this and all other algorithms tested here as supervised learning algorithms, where weights are adjusted based on how much the output varies from the target. The first step in the algorithm is to



(a) MLP Generated



(b) SRN Generated

Fig. 5. Random function generated by neural networks using identical weights

calculate the error of the actual output of the network with the desired output. After this, the output error is backpropagated to the hidden layer to find the error emanating from this layer. For the final error calculation, the error of the activation functions is calculated from the backpropagated hidden layer error. Once all the backpropagated errors are calculated, the weight change is calculated. Finally, the weight is altered and the next input pattern is fed through to begin the cycle again until a stopping condition is reached.

IV. PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a classic swarm intelligence algorithm. In it, particles fly through a multi-dimensional search space guided by constraints, the best particles in the swarm, and their own previous positions. The mechanism used for this movement is velocity. This makes the exploration of continuous space efficient and thorough with the correct parameters for the problem. The quantification of the velocity and position updates are shown here in (2) and (3) respectively and the tunable parameters are explained thereafter [6]:

$$v_{id}(t) = w * v_{id}(t-1) + c_1 * U(0,1) * (p_{id}(t) - x_{id}(t-1)) \dots \quad (2)$$

$$+ c_2 * U(0,1) * (g_d(t) - x_{id}(t-1))$$

$$x(t) = x(t-1) + v(t) \quad (3)$$

w = inertia weight

c_1 = cognitive acceleration constant

c_2 = social acceleration constant

$U(0,1)$ = random uniform distribution number between 0,1

The inertia weight here enhances exploration of the search space by not having an instant response to changes in the other velocity factors. It can be static or adaptive. The adaptive version of the inertia weight has the advantage of being able to avoid the problem of excessive overshooting (causing a particle to have difficult converging) while still giving sufficient search space exploration [11]. This adaptive weight is used in this study as shown in Table 1. The cognitive and social constants, c_1 and c_2 , determines

how much influence the particles' own experience and the best experience of the entire swarm have on the particles' exploration, respectively. The dimensions are parameters to be optimized (in this research these are the weights of the neural networks), and the particles themselves are potential solutions to the network.

p_{id} in (2) is the best position particle i has seen in dimension d . All the dimensions together produce the p_{best} position for that particle, known as the particle best. The constant c_1 associated with the p_{best} is known as the cognitive acceleration constant because it serves as the particle's memory, and is only influence by the particle's own past experience. The overall best p_{best} position is the g_{best} position, known as the global best. The constant c_2 associated with g_{best} is known as the social acceleration constant because it acts as the hive mind of the swarm. All particles have access to the g_{best} particle in the star topology and all particles have the opportunity to alter it [12].

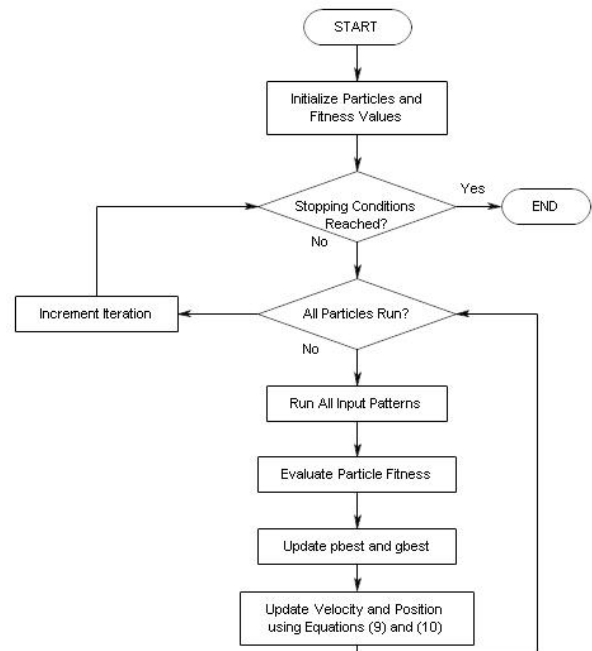


Fig. 6. PSO Flowchart

The basic error equation used to determine particle fitness is the same as that used in (2). It is the mean-squared error (MSE) of the outputs as compared to the target values over all input patterns. There is a flowchart of PSO as applied to a neural network in Figure 6.

V. PARTICLE SWARM OPTIMIZATION WITH QUANTUM INFUSION

PSO with Quantum Infusion (PSO-QI) is the adaptive PSO in Section III with the addition of one step. This step effectively turns a particle into a probability density function based on Schrödinger's Equation from quantum mechanics shown in (4):

$$i\hbar \frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m} \nabla^2 \psi + V\psi \quad (4)$$

This equation, commonly known as the "wave equation," shows the wave behavior of a quantum particle in 3-dimensional space with potential V [13]. In quantum mechanics, these are known as "particle waves," and lead to behavior which can be exploited by the PSO algorithm to enhance performance. The potential model used in this research is the delta potential well [7]. This is a potential function with particle collapse limits a distance Δ from the center defined in (5).

The particle to undergo the quantum operation is chosen at random. The center of the probability distribution is located at some point between the particle's current position and the g_{best} particle's position, as shown in (5).

The edges of this distribution are defined by parameter L comprised of a tunable parameter β and the difference between the center of the distribution and a mean best position among all the particles known as m_{best} , defined in [14]. This will have the effect of reducing the area the particle could tunnel to when the algorithm begins to converge (the particles begin to get closer together over time).

$$P_d = \frac{rand_1 p_{id} + rand_2 p_{gd}}{rand_1 + rand_2} \quad (5)$$

P_d = center of probability distribution in dimension d
 p_{id} = the i^{th} particle's current position in dimension d
 p_{gd} = the g_{best} particle's position in dimension d
 $rand_1$ and $rand_2$ = uniform random number between 0 and 1

Overall, the quantum operation on random particle x is as follows:

$$x = P + \beta |m_{best} - x| * \ln\left(\frac{1}{rand}\right) \quad (6)$$

$rand$ = random uniform distribution number between 0 and 1

The quantum modified particle is then put through a tournament with the g_{best} particle and the best of these two is

selected as the new g_{best} . The complete flowchart for PSO-QI is shown in Figure 7.

The random nature of the quantum wave behavior introduced in (6) ensures proper search exploration. The probability being centered on a point between the current location and the g_{best} location enhances convergence of the algorithm by giving two potential g_{best} updates.

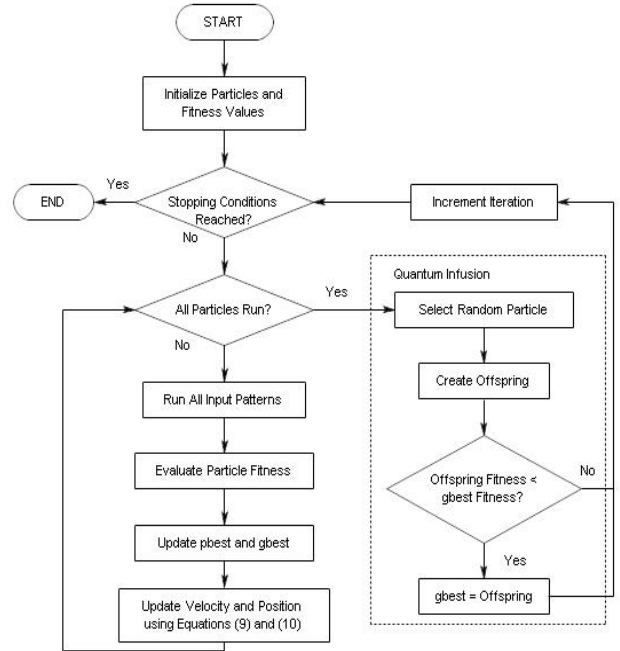


Fig. 7. PSO-QI Flowchart

VI. RESULTS

Four case studies are presented. In Case 1, MLP neural networks with fixed randomly initialized weights are fed the standard input patterns and the corresponding outputs composed the target function. As these weights are different on every run, the algorithm is approximating a different nonlinear function each time. The function approximated is 3-dimensional, having inputs x_1 and x_2 , and a bias of 1. These inputs, as well as the range of the random weights in the target function and all other algorithm parameters, are given in Table 1.

Table 2 shows that in Case 1, MLP (case 1a) and SRN (case 1b) performed at nearly the same level, with MLPs taking a slight advantage. Also, it can be seen that BP performs at a level much worse than either PSO algorithm with PSO-QI performing at a higher level.

Table 3 shows that PSO-QI takes approximately 5% more time than PSO while giving a 25-30% improvement in MSE. While Table 3 also shows a slight increase in time taken to complete the algorithm, the MSE decrease from Table 2 outweighs the time increase from Table 3, 1.134:1. The two PSO algorithms also take nearly ten times as long to complete as BP, but is far outweighed by a nearly 100-fold improvement in the MSE. Figure 8 confirms the results given in Table 2. Figure 8a is a function generated by an MLP, with Figures 8b and 8c showing the approximations of

this function learned by an MLP and an SRN respectively with PSO-QI.

TABLE 1
PARAMETERS FOR TRAINING ALGORITHMS

	BP	PSO	PSO-QI
Target Weights	-6 to 6	-6 to 6	-6 to 6
Inputs	$x_1 = -1$ to 1 $x_2 = -1$ to 1 Step = 0.1	$x_1 = -1$ to 1 $x_2 = -1$ to 1 Step = 0.1	$x_1 = -1$ to 1 $x_2 = -1$ to 1 Step = 0.1
Error Threshold	0.001	0.001	0.001
Max. Iterations	600	600	600
Population	30	30	30
No. of Trails	50	50	50
Algorithm-Specific	$\gamma_0 = 0.2$ $\gamma_m = 0.1$	$w = 0.9$ to 0.4 $c_1 = 2$ $c_2 = 2$	$w = 0.9$ to 0.4 $c_1 = 2$ $c_2 = 2$ $\beta = 0.5$ to 1.0

The SRN shows a slightly less accurate approximation than the MLP, but the shape of the nonlinear function is maintained. It can also be seen that the random function generated by the MLP is of a less complex nature with slower dynamics than that in Figure 11 generated for Case 2. Figures 9 and 10 also show that PSO-QI is consistently at an MSE equal to or less than BP and PSO through time. Figure 10 specifically shows the algorithms over an equal period of time, measured at equal intervals. It shows that PSO-QI holds a large lead over PSO and BP in the early time period, due to the convergence-hastening effect of the quantum influence. PSO, does, however, catch up late in the time period when the quantum tunneling of PSO-QI has less affect on the algorithm.

Case 2 is exactly the same as Case 1, except that instead of an MLP generating the target function, an SRN takes on this task. This leads to a more challenging task, as the SRN's context layer makes it able to produce functions with more nonlinearity, and even sharp points.

This is due to the tendency of SRNs to go into positive feedback under certain conditions as is demonstrated by Figure 4. In fact, the points visible in Figure 5b are the result of this instability. Since MLPs do not have this feature, they are less able to approximate functions of this nature. This is evidenced in Figure 11, which shows in 11a a function generated by the SRN which shows dynamics below the z-axis precision. The MLP in Figure 11b is not able to model this behavior very well at all, whereas the SRN in Figure 11c is able to model it with surprising accuracy. Table 2 shows that, aside from BP, SRNs (Case 2b) outperform MLPs (Case 2a) by a large margin. The poor performance of BP in the SRN's case is not surprising, as BP is not the preferred algorithm for SRN training. As for the timing, Case 1 and Case 2 computational times are not significantly different because, with the MSE error limit being 0.001, the algorithms nearly always went to maximum iterations in both Cases, thus creating equivalent runtimes.

Case 3 utilizes the benchmark function (7), while Case 4 uses benchmark (8). These choices were inspired by [15].

Table 2 shows mixed results for these benchmarks. Both PSO algorithms outperform BP by a large margin, but the difference between the two is minimal. The only pattern is that PSO-QI tends to perform better with SRNs, but PSO itself seems to perform better for MLPs. This is confirmed by the graphical results in Figures 12 and 13.

Obviously PSO-QI used here is not a significant improvement for simple single-input, single-output problems. The same can be said of SRNs, which actually show slightly degraded performance when compared with MLPs in Cases 3 and 4. Both PSO-QI and SRNs shine through, however, in the more difficult problems demonstrated in random, nonlinear, multiple input cases such as Case 1 and Case 2.

$$y(x) = \sin(\pi x) \quad (7)$$

$$y(x) = (1 + x + 2x^2) e^{-x^2} \quad (8)$$

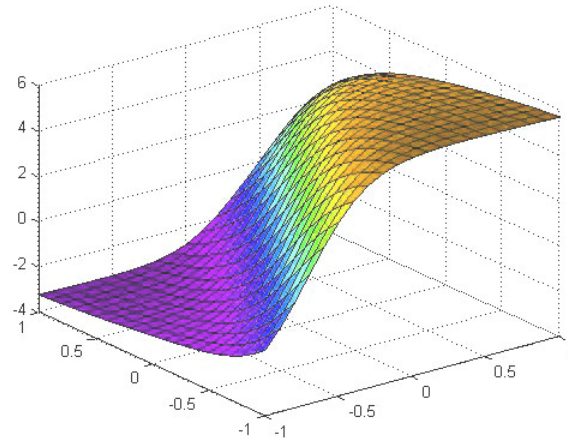
VII. CONCLUSION

This paper has presented a new learning algorithm, especially for simultaneous recurrent neural networks. Cases 1 and 2 both show that SRNs are better able to approximate complex functions, and results further show that SRNs are very exact in their ability to model detail system dynamics. While the PSO runtimes are far longer than backpropagation, when compared to the results obtained, the mean square error decrease outweighs the increase in runtime. PSO is a much better algorithm in all but time-sensitive applications. The increase in runtime incurred by PSO-QI is outweighed by its decreased MSE, providing it better results at a rate slightly better than PSO. When the network is a simple single-input, single-output case, however, the two PSO algorithms perform at equal rates, lending PSO-QI more toward the more complex multi-input problems.

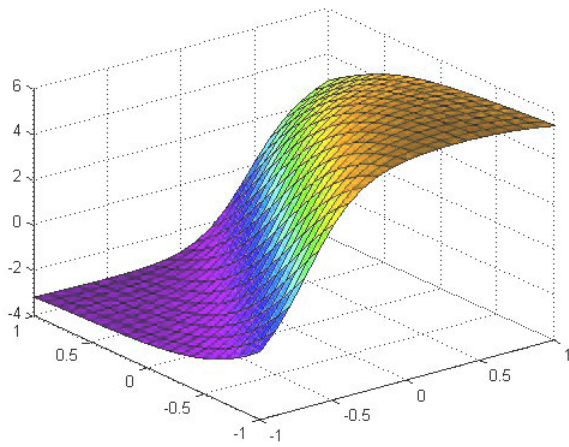
PSO-QI as a learning algorithm for SRNs remains to be investigated on real-world complex problems such as multi-step prediction of a large power system states.

TABLE 2
MSE RESULTS FOR BP, PSO, AND PSO-QI

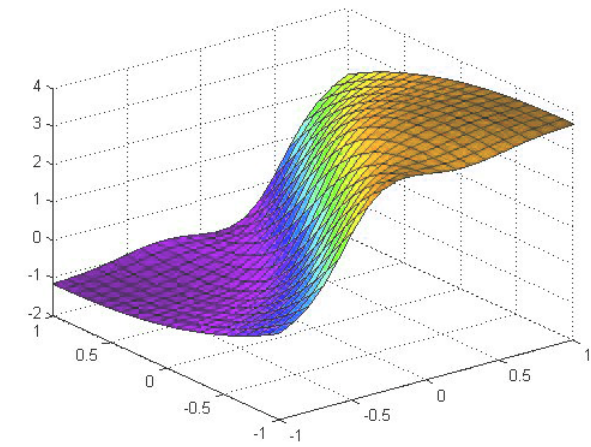
Case	Topology	BP	PSO	PSO-QI
1a	MLP	2.9226 ± 4.3398	0.0223 ± 0.0500	0.0171 ± 0.0191
1b	SRN	2.5167 ± 4.4720	0.0650 ± 0.8690	0.0489 ± 0.0510
2a	MLP	4.4158 ± 9.1680	2.0540 ± 3.2671	1.1600 ± 1.8315
2b	SRN	4.4598 ± 8.6800	0.6033 ± 1.3148	0.2624 ± 0.3886
3a	MLP	0.1176 ± 0.0048	4.5886x10 ⁻⁵ ± 5.279x10 ⁻⁵	5.7071x10 ⁻⁵ ± 6.9493x10 ⁻⁵
3b	SRN	0.1137 ± 0.0047	4.3532x10 ⁻⁵ ± 4.46x10 ⁻⁵	4.2628x10 ⁻⁵ ± 6.8145x10 ⁻⁵
4a	MLP	0.1961 ± 0.0033	4.7184x10 ⁻⁴ ± 4.908x10 ⁻⁴	5.0568x10 ⁻⁴ ± 5.129x10 ⁻⁴
4b	SRN	0.1938 ± 0.0029	0.0011 ± 8.875x10 ⁻⁴	8.3919x10 ⁻⁴ ± 9.8956x10 ⁻⁴



(a)



(b)



(c)

Fig 8. (a) is the target function developed by an MLP, and (b) and (c) are approximations of this function learned by an MLP and SRN respectively with PSO-QI

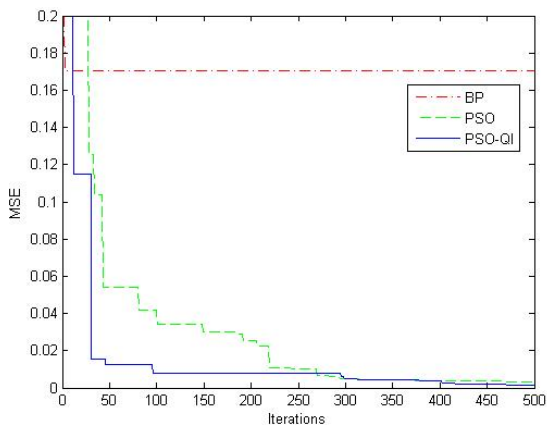


Fig. 9. MSE convergence of algorithms over 600 iterations for Case 1a

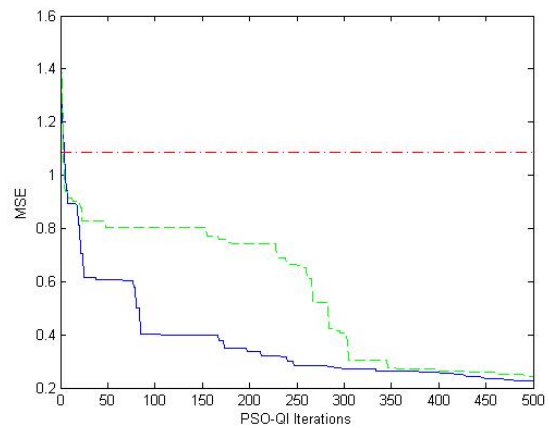
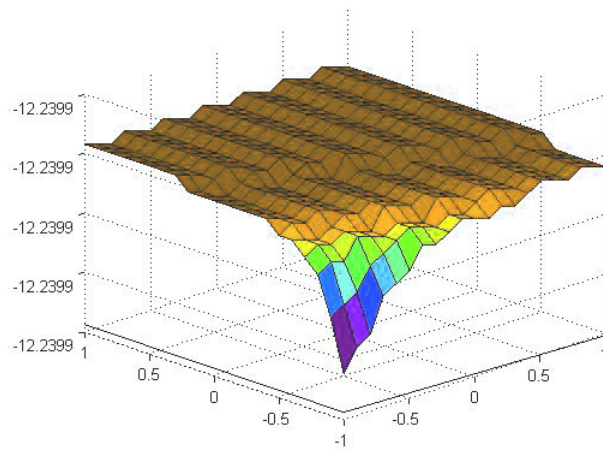
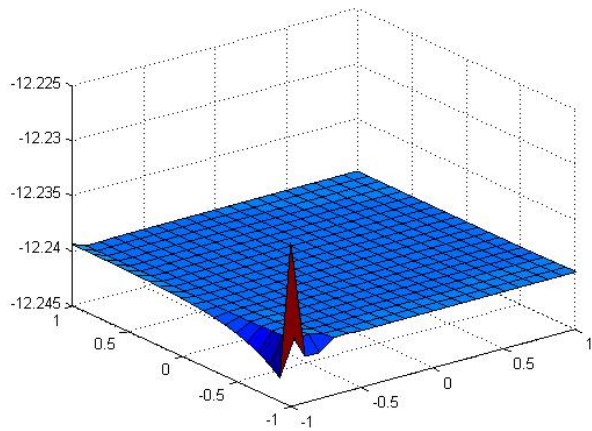


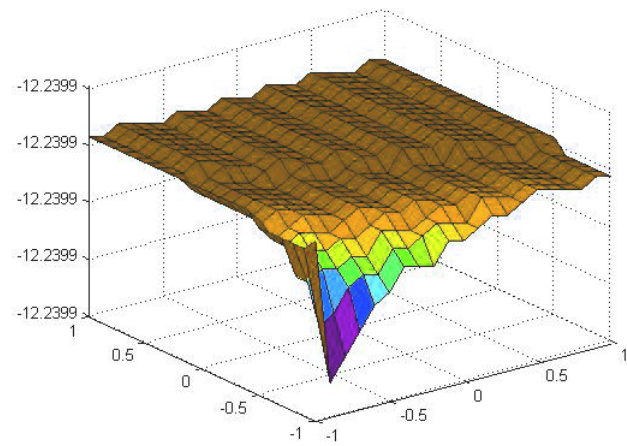
Fig. 10. MSE graphed for Case 2b over equal periods of time



(a)



(b)

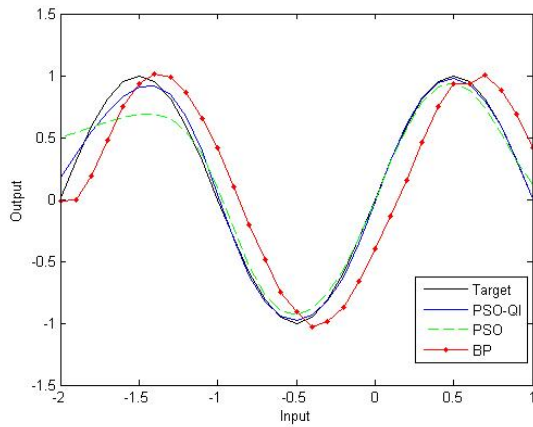


(c)

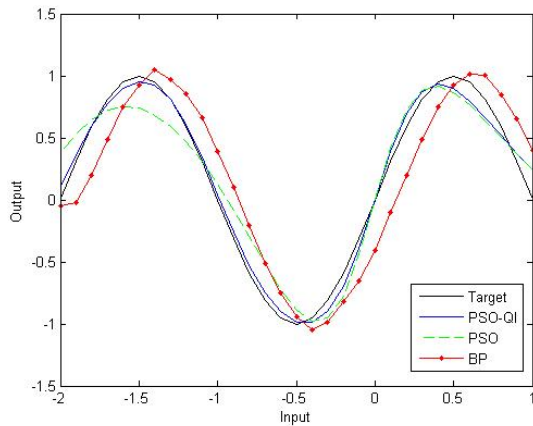
Fig 11. (a) is the target function developed by an SRN. (b) and (c) are approximations of this function developed by PSO-QI on an MLP and SRN respectively

TABLE 3
TIME TAKEN BY ALGORITHMS (IN SECONDS) TAKEN TO COMPLETE ONE RUN

Case	Topology	BP	PSO	PSO-QI
1a	MLP	11.7793 ± 0.0984	96.8117 ± 5.6300	102.5510 ± 5.0321
1b	SRN	36.9257 ± 0.2506	1058.7100 ± 18.2194	1118.5900 ± 92.3091

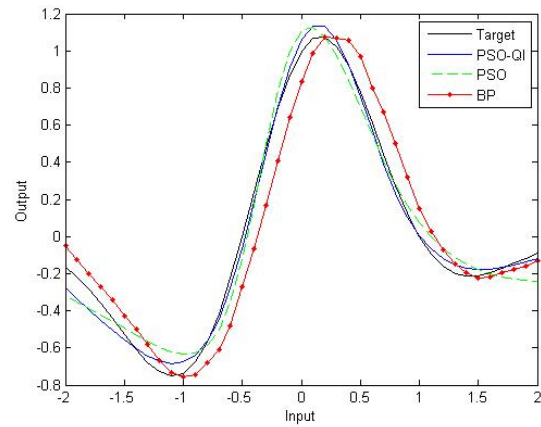


(a)

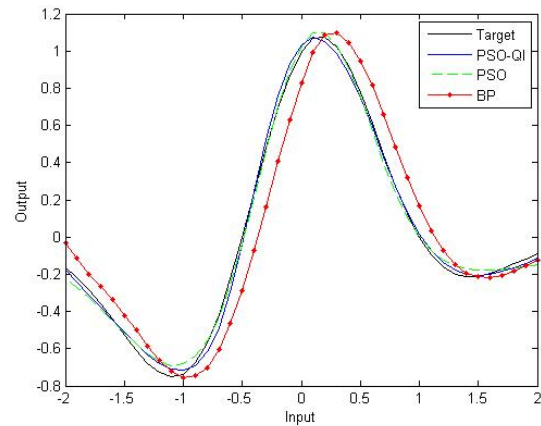


(b)

Fig. 12. Case 3 approximations of function in (7). (a) is MLP and (b) is SRN



(a)



(b)

Fig. 13. Case 4 approximations of function in (8). (a) is MLP and (b) is SRN

REFERENCES

- Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators." *Neural Networks* (1989), Vol. 2, pp. 359-366.
- X. Z. Pang and P. J. Werbos, "Neural Network Design for J Function Approximation in Dynamic Programming," *Math Mode. Sci. Comput. (Principia Scientia J.)*, Vol. 5, no. 2/3, 1996.
- X. Cai, D. Prokhorov, and D. Wunsch II, "Training Winner-Take-All Simultaneous Recurrent Neural Networks," *IEEE Transactions on Neural Networks* (2007), Vol. 18, No. 3, pp. 674-684.
- G.K. Venayagamoorthy and G. Singhal, "Quantum-Inspired Evolutionary Algorithms and Binary Particles Swarm Optimization for Training MLP and SRN Neural Networks." *Journal of Computational and Theoretical Nanoscience* (2005), Vol. 2, pp. 1-8.
- P. J. Werbos, "Backpropagation through time: what it does and how to do it", *Proceedings of the IEEE* (1990), vol. 8, no. 10, pp. 1550-1560.
- J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks*, Australia (1995), Vol. 4, pp. 1942-1948.
- B. Luitel and G.K. Venayagamoorthy, "Particle Swarm Optimization with Quantum Infusion for the Design of Digital Filters," *2008 IEEE Swarm Intelligence Symposium*, (2008).
- P. Moore and G.K. Venayagamoorthy, "Evolving Combinational Logic Circuits Using a Hybrid Quantum Evolution and Particle Swarm Inspired Algorithm," *Proceedings of the 2005 NASA/DoD Conference of Evolution Hardware*, (2005).
- K. Han and J. Kim, "Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization," *IEEE Transactions on Evolutionary Computation* (2002), Vol. 6, pp. 580-593.
- Andries P. Engelbrecht, "Computational Intelligence: An Introduction," John Wiley & Sons, Ltd, University of Pretoria, South Africa (2007).
- F. Han and Q. Ling, "A New Approach for Function Approximation Base on Adaptive Particle Swarm Optimization," *Third International Conference on Natural Computation* (2007).
- P.W. Moore and G.K. Venayagamoorthy, "Empirical Study of an Unconstrained Modified Particle Swarm Optimization," *2006 IEEE Congress on Evolutionary Computation* (2006), pp. 1477-1482.
- Feynman, Leighton, and Sands, "The Feynman Lectures on Physics: Definitive Edition Vol. III," Pearson, San Francisco, (2006).
- J. Sun, W. Xu, and B. Feng, "Adaptive Parameter Control for Quantum-behaved Particle Swarm Optimization on Individual Level," *IEEE Intl. Conf. on Systems, Man and Cybernetics* (2005), Vol. 4, pp. 3049-3054.
- P. Rowcliffe and J. Feng, "Training Spiking Neuronal Networks with Applications in Engineering Tasks," *IEEE Transactions on Neural Networks* (2008), Vol. 19, No. 9, pp. 1626-1640.