International Journal of Neural Systems, Vol. 19, No. 6 (2009) 437–448

World Scientific
www.worldscientific.com

FASTER TRAINING USING FUSION OF ACTIVATION FUNCTIONS FOR FEED FORWARD NEURAL NETWORKS

MD. ASADUZZAMAN and MD. SHAHJAHAN*
Khulna University of Engineering & Technology (KUET)
Department of Electrical & Electronic Engineering
Khulna-9203, Bangladesh
**jahan@eee.kuet.ac.bd*
www.kuet.ac.bd

KAZUYUKI MURASE†
University of Fukui, Department of Human & Artificial Intelligence Systems
Bunkyo 3-9-1, Fukui-910-8705, Japan
murase@synapse.his.fukui-u.ac.jp
www.fukui-u.ac.jp

from the drawback of slow convergence. To make the learning faster, we propose 'Fusion of Activation Functions' (FAF) in which different conventional activation functions (AFs) are combined to compute final activation. This has not been studied extensively yet. One of the sub goals of the paper is to check the role of linear AFs in combination. We investigate whether FAF can enable the learning to be faster. Validity of the proposed method is examined by performing simulations on challenging nine real benchmark classification and time series prediction problems. The FAF has been applied to 2-bit, 3-bit and 4-bit parity, the breast cancer, Diabetes, Heart disease, Iris, wine, Glass and Soybean classification problems. The algorithm is also tested with Mackey-Glass chaotic time series prediction problem. The algorithm is shown to work better than other AFs used independently in BP such as sigmoid (SIG), arctangent (ATAN), logarithmic (LOG).

Keywords: Neural network; training; activation function; convergence; combination of activation functions.

1. Introduction

BP algorithm is a famous and well practiced supervised learning technique used for training Multi-Layer Perceptrons (MLPs).[1–4] In this algorithm, one needs to calculate the gradient of the error function of the network with respect to weights of the network. It attempts to minimize the difference (or error) between the desired and actual outputs in an iterative manner. In conventional BP, weights in the network are adjusted by the algorithm to make the error decrease along a descent direction.[5]

It is important to speed up the learning algorithm for a number of reasons. (a) The convergence time depends on the software to be used to make an algorithm. If the software is slow (such as MATLAB), it needs long time to converge. The case is more severe if someone uses slow machine, (b) There are some real world problems which take very long time to converge although the number of patterns are smaller such as XOR problem, (c) There are some problems which have very large number of attributes. They need long time even to see the output of a trial.

†Corresponding author.

Quick convergence may resolve these problems. Hence, efficient learning technique is required.

The gradient calculation often requires an AF in the hidden layer and sometimes in the output layer for a MLP. The conventional AFs such as sigmoid and tan hyperbolic functions are often used in the MLP training algorithms. Unfortunately, they suffer from the limitation of saturation at around $0, 1$ or $-1, 1$. The value of derivative becomes approximately zero around this zone. As a result the amount of weight update becomes negligible. Therefore, the convergence slows down. It was shown that the convergence rate is extremely slow especially for the networks with more than one hidden layer.[6] Recently, the modification of gradient function to improve the performance of the algorithm has attracted much attention of researchers.[7,8] To improve the convergence of the algorithm we propose a hybrid activation function by combining the conventional AFs and linear functions. Therefore, we call it 'Fusion of Activation Function' (FAF).

There have been a long series of improvements of standard BP algorithm using different cost functions, different activation function, and heuristic strategies. Menke[9] proposed a method to adapt the problem with the learner to improve the supervised learning. The learning rate was adapted to speed up the BP.[10,11] A drawback of standard BP is the existence of local minima resulting from the saturation behavior of the activation function. Zweiri et al had proposed a three term BP, which is a proportional factor to overcome the problems of standard BP.[12] A modified error function having fourth power instead of two was proposed for faster training.[13] An exponential cost function was proposed to improve the standard BP.[14] Tawel[15] proposed an adaptive neural net by introducing the temperature of the sigmoid activation function. To train the network both the weights and temperatures are updated. Kamruzzaman proposed an arctangent activation function to improve the convergence speed of BP.[16] A logarithmic activation function was proposed by Bilski[17] in lieu of sigmoid AF commonly used in BP.

However, most of the AFs proposed above used independently in BP algorithm often cannot escape from local minima. They are faster with flat error. The performances of them become poor for large and difficult problems due to 'quick flat error'. This means convergence is faster but error becomes flat after quick convergence. As a result, unnecessary training may produce bad generalization. The effect of using combined AFs was not investigated extensively in the literature. The use of different activation functions was first motivated by the work of Fahlman.[18] This algorithm is specially designed for constructing network structure in a nice fashion – one hidden unit in each hidden layer. However, combining of AFs are not well studied as of our knowledge. In this paper, we report the results of combining conventional AFs to achieve quick convergence of BP algorithm with different network sizes. Simulations with challenging problems have been carried out to investigate the learning characteristics of the proposed FAF.

The paper is organized as follows. Section 2 explains a brief of BP algorithm. The proposed FAF is described in Sec. 3. The experimental studies including characteristics of benchmark problems and results on them are reported in Sec. 4. A brief discussion is presented in Sec. 5. The paper is concluded in Sec. 6.

2. Standard BP

The MLP[5] consists of one input layer, one output layer, and one or more hidden layers. Each layer contains a set of neurons and is fully connected with an adjacent layer. Each connection link is represented by a weight. The goal of MLP learning is approximation of an objective function. It adjusts the weights such that the discrepancies between the network outputs and the target values are minimized. This process is called supervised learning. The most popularly used error function is the mean squared error (MSE) that is given by:

$$E = \frac{1}{n}\sum_{p=1}^{n}\sum_{i=1}^{m}(Y_{pi} - t_{pi})^2$$

where the notations are: n is the number of training samples; m output vector dimension; Y_{pi} is the network output of ith neuron for pattern p; t_{pi} is the target value of ith neuron for pattern p.

BP applies a gradient descent procedure to minimize the error function E as follows,

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}}$$

where the notations are: w_{ij} connection weight between neurons i and j; η learning rate.

3. Fusion of Activation Functions

The conventional AFs that are often used in the BP learning algorithm are sigmoid, tan hyperbolic function. A linear function is sometimes used in output layer. Other AFs which are also used are logarithmic, arctangent, tan(sig), log(sig) etc. We proposed a hybrid activation function at hidden and output layers of a MLP. This is nothing but combining the conventional AFs in the hidden and output layers of a MLP. In the hidden layer we combine sigmoid and linear function and in the output layer, sigmoid, tan hyperbolic and linear functions are combined to make a hybrid one. The usual targets of benchmark problems are belonging to 0 or 1. The output of the output layer was computed by adding half of the both sigmoid and tan hyperbolic transfer function. This is only to make the activation between 0 and 1. In addition, a linear transfer function is added with this sum as shown in Fig. 1.

A brief calculation of activation in FAF is described here. The output of the hidden layer of a neural network is computed as

$$f(x) = \frac{1}{1+e^{-x}} + \lambda_1 x. \tag{1}$$

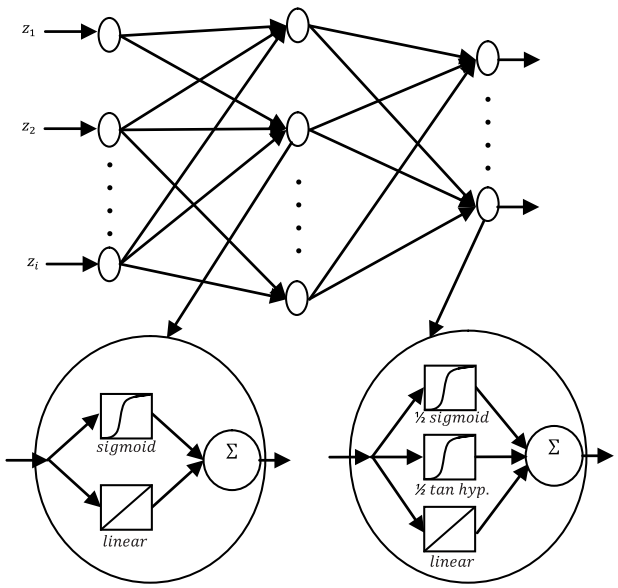

Fig. 1. A neural network with FAF in hidden and output layers.

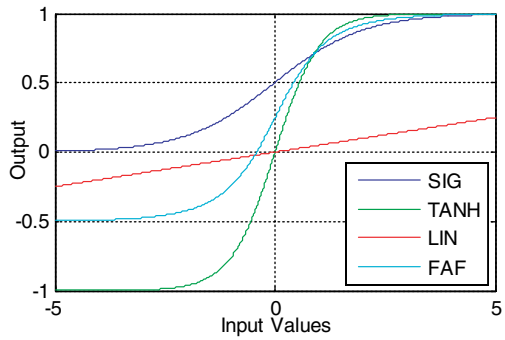

Fig. 2. Different activation functions.

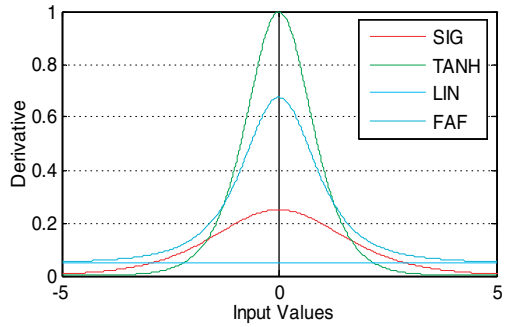

Fig. 3. Derivatives of AFs shown in Fig. 2.

The derivative of the Eq. (1) is

$$f(x) = f(x)(1 - f(x)) + \lambda_1 \tag{2}$$

The output of the output layer is calculated as,

$$f(x) = \frac{1}{2}\frac{1}{1+e^{-x}} + \frac{1}{2}\frac{e^x - e^{-x}}{e^x + e^{-x}} + \lambda_2 x$$

$$f(x) = \frac{1}{2}[f_s(x) + f_{th}(x)] + \lambda_2 x \tag{3}$$

The derivative of the Eq. (3) is,

$$f(x) = \frac{1}{2}[f_s(x)(1 - f_s(x)) + (1 - f_{th}(x)^2)] + \lambda_2 \tag{4}$$

The weight update rule of standard BP requires a derivative of activation functions in the hidden and output layers. Therefore one needs just to use these derivatives as computed with Eqs. (2) and (4) in the update rule of BP. Graphically the AF in FAF and its derivative are shown in Figs. 2 and 3, respectively.

The backpropagated error signals include the factors $o(1-o)$ and which are the derivatives of the sigmoid function of the form $y = 1/(1+e^{(-x)})$. The main reason for the slow convergence of BP

is the behavior of these error signals. Similarly the derivative of the tan hyperbolic function is $(1 - o^2)$. When the actual output o approaches extreme values (i.e., 0 or 1 or -1), the error signals will become so small that they cannot actually reflect the error signal and weight update process slows down.

To alleviate this problem, we just mix the AFs together. There are several explanations why we have done this. Firstly, there is less possibility that all of the AFs saturate together in the training if AFs are different. Therefore, the training may have flexibility to avoid saturation. Secondly, even though they saturate altogether the linear constants (λ_1 and λ_2) will contribute to make the update possible in both input and output layers as derived in Eqs. (2) and (4). This is the reason why the learning is faster in FAF. An experimental evidence will be discussed in Sec. 4.4.

4. Experimental Studies

This section describes about the benchmark problems used in this report and the results obtained with this method. Nine classification problems and one time series prediction problem are used to investigate the effectiveness of the proposed FAF.

4.1. *Characteristics of benchmark classification data*

These benchmark data are collected from PROBEN1[19] and the UCI machine learning data repository. The characteristics of the benchmark data are summarized in Table 1. For example, diabetes problem is a classification problem having a total examples or patterns of 768 with 8 attributes and 2 classes. Other problems are arranged in a similar fashion in the table.

4.2. *Experimental results*

In order to investigate the convergence speed of BP learning with different AFs mentioned above, simulations are carried out with wide range of benchmark problems. Since no analytical techniques are available to study the learning speed of BP algorithm, simulation with different problems and comparison are the usually adopted means to evaluate the effectiveness of a modification. In the present work, investigation has been done with nine different problems.

Table 1. Characteristics of benchmark data.

Data set	Number of		
	Examples	Input attributes	Output classes
2-parity	4	2	2
3-parity	8	3	2
4-parity	16	4	2
Breast Cancer	699	9	2
Diabetes	768	8	2
Heart Disease	303	13	2
Iris	150	4	3
Wine	284	13	3
Glass	214	9	6
Soybean	683	82	19

The error function was multiplied by 100 when it is reported.

(a) Parity problem

The first is the parity problems which include 2-bit, 3-bit and 4-bit. A 2-2-2 network (two inputs, two hidden and two output neuron) network was trained with different AFs for 2-bit problem. Four methods (FAF, SIG, ATAN, and LOG) are started at the same initial weights in each run. The network was initialized with small random weights. Training was allowed until sum-squared error reduces to 1. Twenty trials with different initial weights were carried out and the average training cycle is presented in Table 2. For example, FAF needs only 94 iterations on average for $\eta = 0.5$. Other results show that the FAF improves the learning speed of BP learning in terms of number of iterations. The results of FAF are compared with the other activation functions in Table 3. It is seen that for $\eta = 0.1$, average iteration required for FAF is 322 which is much smaller than that of sigmoid (17994), arctangent (4865), and logarithmic (6940). A typical run for all AFs including FAF is shown Fig. 4. It is clear that FAF is faster than others, although there is a slight increase (Fig. 4) in error at the starting stage. This is allowable if someone knows there is drastic down stairs in the next few cycles.

A 3-3-2 and 4-4-2 networks are used for 3-bit and 4-bit parity problems, respectively. The results and comparisons for 3-bit parity problems are listed in

Table 2. Summary of results for benchmark classification problems. Mean, Min, Max and SD represent respectively average, minimum, maximum and standard deviation.

	Last error	Learning rate	Constants		Iteration			
Problems	LE	η	λ_1	λ_2	Mean	Min.	Max.	SD
2 Bit parity	1	0.1	−0.15	2.01	322	238	467	62.04
		0.3	−0.14	1.05	170	132	227	26.77
		0.5	−0.12	0.80	94	78	114	9.16
3 Bit parity	1	0.1	−0.175	2.50	197	138	286	48.47
		0.3	−0.125	1.25	105	88	135	13.40
		0.5	−0.125	0.75	92	72	117	14.42
4 Bit parity	1	0.1	−0.08	2.0	812	678	934	72.91
		0.3	−0.07	0.8	731	626	817	54.14
		0.5	−0.06	0.6	514	438	694	67.37
Breast cancer	2	0.10	−0.10	0.20	6.15	6	7	0.36
		0.15	−0.08	0.12	5.40	5	6	0.50
		0.20	−0.06	0.08	5.05	5	6	0.22
Diabetes	8	0.050	−0.005	0.13	34.5	33	36	0.97
		0.075	−0.004	0.10	26.6	26	28	0.74
		0.100	−0.003	0.08	17	16	19	0.86
Heart	6	0.1	−0.01	0.210	9.2	9	10	0.40
		0.2	−0.01	0.040	8.6	8	10	0.41
		0.3	−0.01	0.002	9.1	8	10	0.61
Iris	0.3	0.1	−0.04	0.35	42.2	27	50	8.10
		0.2	−0.03	0.25	39.4	29	50	6.45
		0.3	−0.02	0.15	38.8	35	43	1.82
Wine	1	0.1	−0.011	0.52	45.1	34	64	8.21
		0.2	−0.005	0.25	31.0	25	38	3.81
		0.3	−0.001	0.16	26.4	21	32	3.34
Glass	4	0.10	−0.12	0.001	438	383	500	36.8
		0.15	−0.10	0.001	335	270	390	36.6
		0.20	−0.07	0.001	275	215	355	43.1
Soybean	0.15	0.1	−0.015	0.40	119	97	142	13.45
		0.2	−0.010	0.30	65	58	82	5.09
		0.3	−0.008	0.20	53	51	58	2.52

Table 3. Comparison in terms of number of iterations for 2-bit parity problem. SIG, ATAN, and LOG indicate sigmoid, arctangent, and logarithmic activation functions respectively.

	Convergence time			
η	FAF	SIG	ATAN	LOG
0.1	322	17994	4865	6940
0.3	170	5335	1373	1879
0.5	94	3080	807	1040

Tables 2 & 4, respectively and that for 4-bit parity problem are listed in Tables 2 and 5 respectively. It is clear that FAF is faster than others AFs. Usually, BP (with SIG, ATAN, and LOG) needs very long time to converge for 4-bit parity problem. Especially, BP with SIG function cannot even converge within 5000 and 30,000 iterations for 3-bit and 4-bit problems. The test set is the same as training set, the testing accuracy is same with one exception in case of 4-bit parity problem. In this case, conventional sigmoid

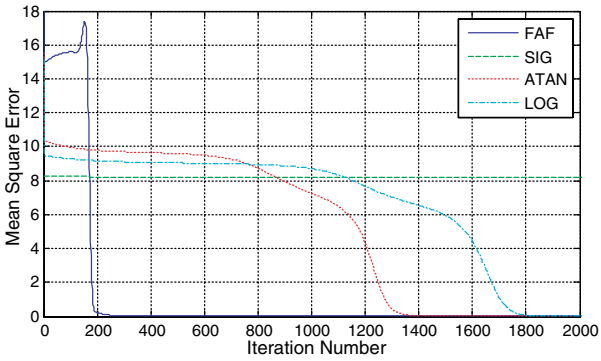

Fig. 4. Convergence curve for 2-bit parity problem.

Table 4. Comparison in terms of iteration for 3-bit bit parity problem.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	198	31200	6600	12000
0.3	106	8270	3100	3800
0.5	92	9227	1800	2600

Table 5. Comparison in terms of number of iteration for 4-bit bit parity problem.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	812	110000	15000	26000
0.3	731	80700	10300	12800
0.5	514	75300	5860	8450

function can classify 15 patterns and it cannot reach even less than a sum square error of approximately 0.04. ATAN and LOG AFs often classify 15 patterns for most of the trials out of 16 patterns and they need long training time to classify all the patterns. But FAF can easily classify all the 16 patterns for all 20 runs.

(b) Breast cancer

The first 350 patterns belong to train, and the last 174 patterns are used for testing. A NN size of 9-4-2 (9 inputs, 4 hidden neurons, and 2 outputs) is taken. An error of 2 is taken to stop the training. This algorithm can classify 172.5 patterns on an average out of 174 patterns for 10 runs. Table 6 shows the average

Table 6. Comparison in terms of number of iteration for Breast Cancer problem.

	Convergence time			
η	FAF	SIG	ATAN	LOG
0.10	6.2	14.5	34.1	31.5
0.15	5.4	10.5	24.9	27.2
0.20	5.1	7.40	24.1	22.4

results for all AFs starting with different initial conditions. The training is started with same initial conditions in a run for all AFs. The result proved that the FAF needs smaller average iteration than others. Since this is an easy problem there is less variation in the average number of iterations at different learning rate as observed in Table 6. The accuracies for all the AFs including FAF are approximately the same ($172.5/174 = 99.13\%$) since there is less noise in this problem.

(c) Diabetes

The first 384 patterns belong to training set, and the last 192 patterns are used for testing. A NN size of 8-4-2 is taken. An error of 8 is taken to stop the training. The FAF has attained average number of iterations as 34.5, 26.6, and 25.5 for learning rate 0.05, 0.75, and 1.0 respectively as listed in Tables 2 & 7. They are smaller than other AFs used independently in the training as shown in Table 7. It is important to note that the ATAN and LOG functions need larger number of average iterations at increasing learning rate while FAF needs smaller number of iterations. The error at which the training was stopped was chosen as 8, because ATAN and LOG AF cannot reach a significant amount lower than 8. Also it is easy to compare FAF with others as well. The testing accuracy (TA) is approximately 76% for FAF on average.

Table 7. Comparison in terms of number of iterations for Diabetes problem.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.05	34.5	86.1	63.4	63.0
0.075	26.6	43.6	103.1	89.6
0.10	17	43.7	93.9	68.4

Table 8. Comparison n terms of number of iterations for heart disease problem.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	9.2	20	10.5	12.2
0.2	8.6	15.9	16.0	14.1
0.3	9.1	10.9	22.1	16.0

The TA is 77.5% for SIG on average and 76% for both ATAN and LOG. This is the problem where FAF gives comparable results. Diabetes is one of the hard problems in machine learning. There is much noise in the data set. Therefore, classification performance is limited. This is not inherent problem of FAF.

(d) Heart Disease

The first 151 patterns are taken as training set and a NN size of 13-4-2 is used in the experiment. The average results in terms of number of iterations are shown in Tables 2 and 8. It is clear from the Table 8 that FAF is better than others at different learning rates. The average number of iterations are 8.6, 15.9, 16, and 14.1 obtained with FAF, SIG, ATAN and LOG respectively at $\eta = 0.2$. Therefore, FAF is faster than other AFs. Last 76 patterns are used for testing. The TA is approximately 83% on average for all AFs. The average values are approximately same since relatively less distinct values are observed in errors and numbers of iterations across different trials.

(e) Iris

The entire data set is taken as training and testing sets. A NN size of 4-3-3 is chosen for training. This is a three class problem. The average results are listed in Tables 2 and 9. It is clear that FAF is faster than all other AFs as shown in Table 9. For example FAF is 3 times faster than SIG, 1.6 times than ATAN and 1.5 times than LOG AFs at 0.1 learning rate.

(f) Wine

A 13-4-3 NN is used for training. The results are shown in Tables 2 and 10. The first 178 patterns are used for training. This is also a three class problem. Here, FAF is 1.51 times, 2.77, and 3.30 times faster than SIG, ATAN and LOG AFs respectively at 0.1 learning rate as shown in Table 10. For the same reason as diabetes, we consider last error of 1 so that we can compare the results of FAF with others. The comparison was not possible especially with ATAN and LOG smaller than this error. The last 53 patterns are used for testing. The averages TAs are approximately 100%, 100%, 98%, and 98% for FAF, SIG, ATAN and LOG AFs respectively.

Table 9. Comparison in terms of number of iteration for iris data set.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	42.2	126	69.8	63.7
0.2	39.4	51.3	60.2	51.5
0.3	38.8	44.8	56.7	53.0

Table 10. Comparison in terms of number of iterations for wine data set.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	45.1	68.1	125.2	148.8
0.2	31.0	37.2	60.8	84.4
0.3	26.4	27.8	36.8	47.8

(g) Glass

The first 107 examples are used as training set and a NN size of 9-7-6 is used for training. This is a six class problem. The average results in terms of number of iterations are listed in Table 2 and 11 as well. This is only the problem where approximately similar results (number of iterations) are obtained for all AFs including FAF. The reason behind this may

Table 11. Comparison in terms of number of iterations for glass data set.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.10	438	439	441	439
0.15	334	336	355	347
0.20	275	270	366	347

Table 12. Comparison in terms of number of iterations for soybean data set.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.10	119	147	137	159
0.20	65	80	71	74
0.30	53	64	57	58

be the class distribution. They are not well and uniformly distributed over all classes. Some classes have very few examples. But the error of FAF is much smaller than the others. Last 53 patterns are used for testing. The average accuracies obtained with FAF, SIG, ATAN, and LOG are respectively 70.5%, 66%, 65%, and 65%. Here the average accuracy of FAF is much better than other AFs in terms of TA.

(h) Soybean

The first 342 examples are used as training set and a NN size of 82-40-19 is chosen for training. The results and comparisons are furnished in Tables 2 and 12 respectively. The FAF is 1.23, 1.15, and 1.33 times faster than SIG, ATAN and LOG AFs respectively at 0.1 learning rate. FAF has approximately similar results with others at other learning rates. The averages TAs are approximately 93% for both FAF and SIG. The reason behind this may be the class distribution over all examples as for glass problem. ATAN & LOG achieve TAs of 92% and 92.5% respectively.

4.3. *Mackey-Glass (MG) chaotic time series prediction problem*

We have applied this method to MG chaotic time series prediction problem. This is because most of the literature uses this kind of time series. The MG chaotic time series is generated by the following differential equation.

$$\dot{x} = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{10}(t-\tau)} \tag{5}$$

where $\alpha = 0.2, \beta = -0.1, \tau = 17$. $x(t)$ is a quasi-periodic and chaotic with a fractal dimension 2.1 for the above parameters. The input to an artificial neural network (ANN) consists of four past data points, $x(t), x(t-6), x(t-12)$, and $x(t-16)$. During training, the true value of $x(t+6)$ was used as target value. The data for MG time series was obtained by applying the fourth-order Runge-Kutta method to equation (5) with initial condition $x(0) = 1.2, x(t-\tau) = 0$ for $0 \le t < \tau$, and the time step is one. The training data consisted of first 500 examples. The following 500 data points were used as test set.

This is a continuous problem and different from classification problem. The input and output are continuous. Remarkable results are obtained for this problem as listed in Table 13. An average number of iterations obtained with FAF is 8.4 for learning rate of 0.1, and with two constants of $\lambda_1 = 0.4$ and $\lambda_2 = 0.15$. The results are compared with other AFs in Table 14. FAF is clearly 5, 2, and 2 times faster than SIG, ATAN and LOG AFs respectively for a learning rate of 0.1. It is clear that FAF is faster than others by a wide margin.

4.4. *Learning characteristics*

In this section, we explain two things — the weight update process during training and the effect of varying the number of hidden units in training. We have presented the theoretical reasons why FAF becomes faster in Sec. 3. Here we will describe the experimental evidence of the theoretical explanation using two bit parity problem. Figure 5 explains the update procedure of FAF and BP algorithms for two bit parity problem. It is clear that FAF updates very quickly, while BP does not contribute to update process yet.

Table 13. Results in terms of number of iterations for MG chaotic time series prediction problem.

	Constants		Iteration			
η	λ_1	λ_2	Mean	Mn	Mx	SD
0.1	0.4	0.15	8.4	7	11	0.99
0.2	0.3	0.12	7.7	7	9	0.67
0.3	0.3	0.10	6.3	6	8	0.59

Table 14. Comparison in terms of number of iterations for MG chaotic time series prediction problem.

	Converge time			
η	FAF	SIG	ATAN	LOG
0.1	12	65.5	23.8	24.4
0.2	9	32.7	17.9	16.7
0.3	8	23.2	16.8	15.6

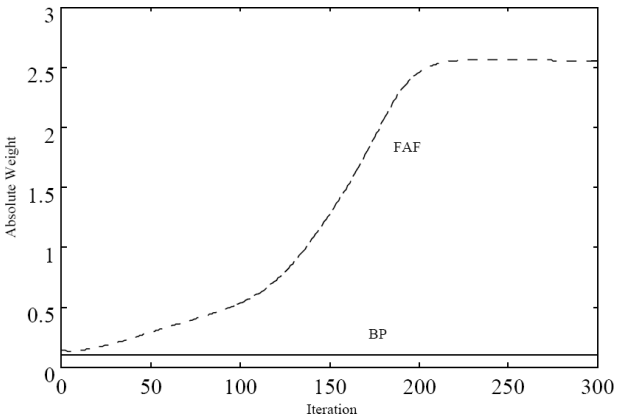

Fig. 5. Weight update process in FAF and BP for 2-bit parity problem.

That is why absolute weight of BP does not start to change. BP takes long time to converge. It shows the efficiency of FAF with respect to standard BP. The absolute value of weight was computed by taking the root mean square of weights in the network. The Fig. 5 shows only the initial stage of training for simplicity sake.

It is interesting to observe the impact of varying the number of hidden units in FAF. We consider two problems – one from parity i.e., two bit parity and another from disease diagnosis i.e., diabetes problems. This is because entire picture of the impact of varying hidden unit may be come out together.

Tables 15 and 16 describe the variation of hidden units with the variation of learning rates. The network size 2-3-2 indicates the two input-three hidden unit-two output network. We observe that the number of iterations is slightly decreases with the increase of number of hidden unit for both the problems. This is because the network now has the more flexibility (more weights) to map the given function. For example, the average number of iterations was 312, 310 and 297 for 2-3-2, 2-4-2, and 2-5-2 network respectively as listed in Table 15. Similar observation was found for diabetes problem as shown in Table 16. The BP with higher number of hidden does not significantly speed up the training. The main point here is that the speed of training was not significantly improved with the increase of hidden units.

Table 15. Variation of hidden unit with learning rate for two-bit parity problem.

		Network size		
η		2-3-2	2-4-2	2-5-2
0.1	Mean	312	310	297
	Min	241	247	238
	Max	412	390	383
	SD	32.76	22.58	12.64
0.3	Mean	169	164	162
	Min	126	121	116
	Max	210	194	187
	SD	22.64	18.78	11.21
0.5	Mean	91	92	90
	Min	81	86	83
	Max	99	105	101
	SD	11.90	10.60	6.59

Table 16. Variation of hidden unit with learning rate for diabetes problem.

		Network size		
η		8-5-2	8-6-2	8-7-2
0.05	Mean	26.35	26.35	25.8
	Min	24	24	23
	Max	29	30	29
	SD	1.31	1.57	1.61
0.1	Mean	15.45	15.1	14.9
	Min	15	14	14
	Max	16	16	16
	SD	0.51	0.45	0.72
0.2	Mean	13.35	13.2	13.2
	Min	13	13	13
	Max	14	14	14
	SD	0.48	0.41	0.41

4.5. *Further comparisons*

In this section, we first report the comparative timings of FAF, SIG, ATAN and LOG AFs to converge. Secondly, we compare FAF with some existing algorithms. We have compared the time required to complete a training session for all AFs. We have just computed the time in a training session and multiplied with the average iteration as listed in Table 17. A programming language executes each instruction within nanosecond depending upon the machine. Our program adds several instructions in a backpropagation algorithm to form FAF. Since FAF requires less iteration, the total execution time in fact is relatively less than other algorithms as depicted from the table. The execution time depends upon the machine and software used for the algorithm. Therefore it may differ from machine to machine,

Table 17. Comparison in term of execution time (seconds).

Problems	FAF	SIG	ATAN	LOG
2-bit	0.89	30.71	7.83	11.38
3-bit	1.11	115.54	24.71	41.46
4-bit	11.43	1347.44	139.28	214.96
Cancer	2.03	3.17	8.27	7.83
Diabetes	13.17	23.15	27.45	29.44
Heart	1.45	1.73	1.79	1.59
Iris	5.48	8.37	7.61	7.45
Wine	6.68	6.78	11.73	13.65
Glass	38.42	38.94	37.02	45.13
Soybean	54.51	54.92	52.95	57.25
Time series	6.87	25.08	9.20	9.61

Table 18. Performance comparison of FAF with other existing algorithms in terms of number of iteration.

Algorithm	XOR	IRIS
FAF	90 (100%)	38.8 (100%)
BP	3080 (100%)	44.8 (100%)
Quick prop	62.88 (56.67%)	671.14 (96.67%)
RPROP	82.36 (46.67%)	402.30 (90%)
SARPROP	243.60 (90%)	—
LM	12 (35%)	—
MGFPROP	562.87 (100%)	430.20 (100%)
DWM	275.43 (100%)	813.70 (100%)
MDPROP	83.93 (100%)	394.38 (100%)

software to software. We have run the program in MATLAB environment.

There are some existing algorithms involving the fastness such as Quickprop,[20] RPROP,[21] SARPROP,[22] Levenberg–Marquardt (LM),[23] DWM,[7] MGFPROP,[7] and MDPROP[7] algorithms. It is very difficult to compare with the other algorithm unless the setup of all algorithms becomes same. It is usual that each algorithm is different setup from others. However, FAF is compared with the available algorithms found in the literature. The most of the algorithms are found in reference number 3 and XOR and IRIS problems are used. Therefore, we have compared with these two problems as listed in Table 18 (reported in Ref. 3 except FAF). We found that FAF exhibits quite competitive result except Quickprop, RPROP and LM and MDPROP algorithms for XOR problem. Although LM is the fastest algorithm so far, its global convergence is poor as indicated by the parentheses (35%). The percentage inside the parentheses indicates the counts the percentage of runs (over 30 different runs) that successfully converge to the system error of less than 10^{-3} as reported in Ref. 3. RPROP is variable step size algorithm and the weight update does not include the magnitude of error gradient whereas MDPROP include a magnified gradient function and deterministic weight modification (DWM). SARPROP is the modification of RPROP using simulated annealing. Although they are faster algorithm, they are not completely free from local minima. Their global convergences are poor such as Quickprop 56.67% and RPROP 46.67% for XOR problem. FAF outperforms all algorithms in terms of number of average iterations as shown in the table for the IRIS problem. In this case Quickprop and RPROP achieve 96.67% and 90% convergence — not 100%, while FAF achieves 100% convergence with only 38.8 iterations.

5. Discussion

In this technique user needs to select two linear constants λ_1 and λ_2. One can run FAF without making any adaptive technique in η. It is enough to maintain at a small suitable value. We have shown the learning characteristics at different η to make sure about the fastness and performance of FAF. We have three observations on the user specified parameters.

(a) We have observed the inverse relationship between λ_2 and η as shown in Fig. 6. Smaller is suitable for FAF. The value of λ_2 becomes less than 1 for all problems except parity problems. These values were selected by limited number of initial trials. We observed that the small positive values of λ_2 are suitable for faster and better performance. Negative values of λ_2 are not suitable since it slows down training. However, we did not optimize it.

(b) The linear relationship was also observed between — λ_1 and η as shown in Fig. 6 (shown for 4 different problems). The negative values of λ_1 are chosen since the error is reduced quickly in course of training and TA is seen to be good also. However, we suggest that the absolute value of λs should be decreased with the increase of η for better results. We observed an exception for MG time series problem where λ_1 was selected within small positive values since this is a continuous problem – different from classification ones.

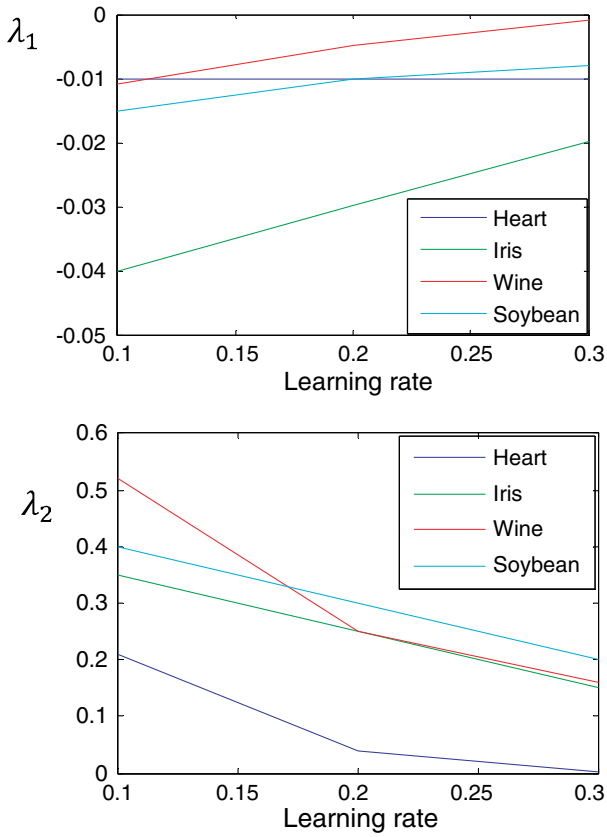

Fig. 6. Dependency of lambda parameter with learning rate.

One can select λ_1 less than 0.25 since the maximum value of derivative of sigmoid AF - O(1-O) is 0.25. So the selection is not critical.

(c) The learning rate should be smaller for acceptable results for FAF. This is true for all other AFs as well. The learning rate varies from 0.05 to 0.3 for all problems except parity problems. The η varies from 0.1 to 0.5 for parity problems.

There is a marginal increase in error at the beginning of training for only 2-bit parity problem. This may be due to the use of steeper linear line in the output layer. Another reason may be the limited number of weights (2-2-2 = 8). Sometimes insufficient weight makes the learning hard. However, this is allowable if error is expected to reduce quickly in the following few cycles.

We found few important observations as below:

(i) The FAF shows faster and better results than other AFs for parity problems. SIG AF requires long training time to converge whereas FAF needs much smaller number of iterations. For 4-bit parity problem, FAF classifies always 16 patterns and other AFs fail for that.

(ii) The FAF shows better performance for all two-class and more than two-class problems such as Wine, Iris, Glass, and Soybean. Also TA of FAF was better than others except diabetes problem. Since diabetes is a hard problem and percentage of noise added in it is larger than others. The testing accuracies for Glass problem are higher than others by a wide margin.

(iii) Exceptional results were obtained for MG chaotic time series prediction problem. The convergence time and prediction accuracies are better by a wide margin than other AFs.

6. Conclusion

This paper proposes a new hybrid activation function by combining the conventional AFs. The sigmoid, tan hyperbolic and linear AFs are fused together to compute final activation in a NN. This is similar to yet different from other existing algorithms. Promising and interesting results are obtained with different kinds of real world benchmark problems such as parity problem, cancer, diabetes, heart disease, iris, wine, glass, soybean and MG chaotic time series prediction problems. Almost in every cases FAF achieves faster training. One of the main points in this paper is the joint use of two linear constants (λ_1, λ_2) in opposite sign. MG time series problem was an exception and FAF gives faster with it with the same sign of linear constants. FAF is a useful technique for faster training. The testing accuracies are also validated through a series of simulation and they are better than other methods. This may reduce the terrible long training time and tediousness in trial. A nice further study may be the selection of optimal values of user defined parameters $(\eta, \lambda_1, \lambda_2)$ using genetic algorithm.

References

1. S. L. Hung and H. Adeli, Parallel backpropagation learning algorithms on Cray Y-MP8/864 Supercomputer, *Neurocomputing* **5**(6) (1993) 287–302.
2. S. L. Hung and H. Adeli, Object-oriented back propagation and its application to structural design, *Neurocomputing* **6**(1) (1994) 45–55.

3. A. Dharia and H. Adeli, Neural network model for rapid forecasting of freeway link travel time, *Engineering Applications of Artificial Intelligence* **16**(7–8) (2003) 607–613.
4. S. Hooshdar and H. Adeli, Toward intelligent variable message signs in freeway work zones: A neural network approach, *Journal of Transportation Engineering* **130**(1) (2004) 83–93.
5. X. H. Yu and G. A. Chen, Efficient backpropagation learning using optimal learning rate and momentum, *Neural Networks* **10**(3) (1997) 517–527.
6. X. H. Yu, G. A. Chen and S. X. Cheng, Dynamic learning rate optimization of the backpropagation algorithm, *IEEE Transactions on Neural Networks* (1995) 669–677.
7. S.-C. Ng, C.-C. Cheung and S.-H. Leung, Magnified gradient function with deterministic weight modification in adaptive learning, *IEEE Trans. Neural Network* **15**(6) (2004) 1411–1423.
8. A. Cichocki and R. Zdunek, Multilayer nonnegative matrix factorization using projected gradient approaches, *Int. J. Neural Systems* **17**(6) (2007) 431–446.
9. J. Menke and T. Martinez, Improving supervised learning by adapting the problem to the learner, *Int. J. Neural Systems* **19**(1) (2009) 1–9.
10. C. C. Yu and B. D. Liu, A backpropagation algorithm with adaptive learning rate and momentum coefficient, (WCCI 2002). IEEE, (2002) 1218–1223.
11. G. D. Magoulas, V. P. Plagianakos and M. N. Vrahatis, Globally, convergent algorithms with local learning rates, *IEEE Trans. Neural Network* **13**(3) (2002) 774–779.
12. Y. H. Zweiri, J. F. Whidborne, K. Althoefer and L. D. Seneviratne, A three-term backpropagation algorithm, *Neurocomputing* **50** (2003) 305–318.
13. S. Abid and F. Fnaiech, Fast training of Multilayer perceptrons with Least mean Fourth (LMF) Algorithm, *Int. J. Soft Computing* **3**(5) (2008) 359–367.
14. J. Kamruzzaman, Improving convergence of backpropagation algorithm using exponential cost function, *IEEJ Trans. EIS.* **123**(5) (2003) 999–1003.
15. R. Tawel, Does the neuron learn like synapse? in *Advance in Neural Information Processing System*, ed. D. Touretzy, Vol. 1 (Morgan Kaufmann, 1989), pp. 169–176.
16. J. Kamruzzaman, Arctangent activation function to accelerate backpropagation learning, *IEICE Transactions of Fundamentals of Electronics*, Communications and Computer Sciences, (2002) 2373–2376.
17. J. Bilski, The backpropagation learning with logarithmic transfer function, *Proc. Fifth Conf. on Neural Networks and Soft Computing*, (Poland) 2000 71–76.
18. S. E. Fahlman and C. Lebiere, The cascade-correlation learning architecture, in *Advances in Neural Information Processing Systems*, ed. D. S. Touretzky, Vol. 2 (San Mateo, CA: Morgan Kaufmann 1990) 524–532.
19. L. L. Prechelt, PROBEN1-A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Faculty of Informatics, (University of Karlsruhe, 1994).
20. S. E. Fahlman, Fast learning variations on backpropagation: An empirical study, in Proc. Connectionist Models Summer School, D. Touretzky, G. Hinton and T. Sejnowski, (eds.) (San Mateo, CA, 1989) 38–51.
21. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, in *Proc. Int. Conf. Neural Networks*, Vol. 1, (1993) 586–591.
22. N. K. Treadgold and T. D. Gedeon, Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm, *IEEE Trans. Neural Networks* **9** (1998) 662–668.
23. M. T. Hagan and M. B. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Trans. Neural Networks* **5** (1994) 989–993.