Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)

Opposite Transfer Functions and Backpropagation Through Time

Mario Ventresca and Hamid R. Tizhoosh

Abstract— **Backpropagation through time is a very popular discrete-time recurrent neural network training algorithm. However, the computational time associated with the learning process to achieve high accuracy is high. While many approaches have been proposed that alter the learning algorithm, this paper presents a computationally inexpensive method based on the concept of opposite transfer functions to improve learning in the backpropagation through time algorithm. Specifically, we will show an improvement in the accuracy, stability as well as an acceleration in learning time. We will utilize three common benchmarks to provide experimental evidence of the improvements.**

Index Terms— **Backpropagation through time, opposition-based learning, opposite transfer functions.**

I. Introduction

Recurrent neural networks behave as a dynamic system that evolves according to some set of nonlinear equations. Due to their dynamic nature, recurrent neural networks have been applied to various time-dependant problems in control, communication and signal processing to name a few [1].

The Backpropagation through time (BPTT) [2], [3] algorithm is a popular approach for training discrete time recurrent neural networks. Many other gradient-based learning algorithms have also been proposed to accelerate the convergence time [4], [5]. However, the computational time required to successfully complete training for these gradient-based algorithms may take some time. Other approaches such as Real-Time Recurrent Learning [4] and truncated backpropagation through time [2] have also been developed (for a good review recurrent networks see [1] or [6]).

This paper presents a computationally inexpensive method called OBPTT (Opposition-based Backpropagation Through Time) based on the concept of opposite transfer functions to improve learning in the BPTT algorithm. Opposite transfer functions are an idea based on the concept of opposition-based learning [7]. Specifically, we will show an improvement in the accuracy, stability as well as an acceleration in learning time.

The remainder of this paper is organized as follows: Section II will briefly discuss the concept of opposition-based learning and Section III will explain the notion of opposite transfer functions. The OBPTT algorithm is then outlined in Section IV. In Section V we will discuss the benchmark problems used. The experimental results will be given in Section VI and conclusions and future work will be provided in Section VII.

This work has been supported in part by Natural Sciences and Engineering Council of Canada (NSERC).

M. Ventresca is a student member of the Pattern Analysis and Machine Intelligence (PAMI) laboratory in the Systems Design Engineering Department, University of Waterloo, Waterloo, ONT, N2L 3G1, CANADA (email: mventres@pami.uwaterloo.ca)

H. R. Tizhoosh is a faculty member of the Pattern Analysis and Machine Intelligence (PAMI) laboratory in the Systems Design Engineering Department, University of Waterloo, Waterloo, ONT, N2L 3G1, CANADA (email: tizhoosh@uwaterloo.ca)

II. Opposition-based Learning

Recently, the concept of opposition-based learning (OBL) has been introduced by Tizhoosh [7]. Here we will provide a brief introduction to the motivating principles of this idea. Then, the next section will expand on this to include the concept of opposite transfer functions.

Opposition-based learning is a rather simple concept that aims to improve the convergence rate and/or accuracy of computational intelligence algorithms. The fundamental idea behind this concept is taken from the world around us, where we observe particles/anti-particles, up/down, left/right, on/off, male/female, and so on. Furthermore, opposites exist in social contexts as well, for example via a social revolution, which typically occurs rather rapidly. In terms of the computational idea, this manifests itself as a pair $< x, \breve{x} >$, where the guess x has some function determining what its opposite $\breve{x}$ is.

Consider the problem of discovering the minima (or maxima) of some unknown function $f\colon \Re \rightarrow \Re$. Initially, we can make some guess x as to what the solution s may be. It is possible that this guess is based on some a priori knowledge, but is not a requirement. In either situation, we will either be satisfied with x or not. For the latter case, we will try to further reduce the difference between the estimate x and s whether or not s is known. In general, we are trying to discover some solution x^* such that $|f(x^*) - f(s)| < \epsilon$ for some level of accuracy $\epsilon \in \Re$. Associated with this process is some computational complexity, which due to the curse of dimensionality, is usually beyond permissable application limits. Additionally, some algorithms may converge at a similar rate/time, yet yield rather different solutions.

The concept behind OBL can be applied to this problem. After we choose x_1, we need to select some other solution at the next iteration, say x_2. However, this value may or may not be close to s. Assuming the solution is far away from the optimal, s, then it may take some time before the algorithm reaches that value, if at all. Under the OBL scheme, when we consider x_2 we simultaneously consider its opposite $\breve{x}_2$ and continue learning with the more desirable of the two. For a minimization problem this would be $min(f(x_2), \breve{f}(x_2))$, for some evaluation function f. In order to avoid convergence to a local optima, it is possible to probabilistically accept this value.

To date OBL has been utilized to improve the convergence rate and/or accuracy of differential evolution (by anti-chromosomes) [8], [9], [10], reinforcement learning (by opposite actions/states) [11], [12], [13] and backpropagation in feed-forward neural networks (by opposite transfer functions) [14]. This paper will expand on the work by Ventresca and Tizhoosh [14] to examine opposite transfer functions for the backpropagation-through-time algorithm for recurrent neural networks.

III. Opposite Transfer Functions

In order to incorporate OBL into the BPTT algorithm, opposite transfer functions [14] will be utilized. In this section we will provide the idea behind opposite transfer functions and how they can be of use for improving BPTT.

A. What is an Opposite Transfer Function

A neural network [15] is represented as a graph structure where edges represent real-valued weights between vertices or neurons. Each neuron utilizes a transfer function of the form $f\colon \Re \to \Re$ and is typically sigmoid in shape. The logistic function,

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

is a very common transfer function used in backpropagation learning. We will define the opposite of a transfer function and utilize the logistic function to illustrate the definition.

Definition 1 (Opposite Transfer Function): Given some transfer function $f(x)$ its corresponding unique opposite function is given by $\breve{f}(x) = f(-x)$. These functions intersect at some point $x = x^*$, which defines a plane which the functions are symmetric about.

With respect to the logistic function, we can define its opposite as

$$\breve{f}(x) = \frac{1}{1+e^{x}} \tag{2}$$

This function is also plotted in Figure 1 against the traditional logistic function. The two curves intersect at $x = 0$, which defines the point they are symmetric about. These functions obey the constraint outlined above, that is $\breve{f}(x) = f(-x)$.

B. Opposite Transfer Functions and Neural Networks

Let a recurrent neural network have L hidden layers, each having k_l hidden neurons each of which can utilize the traditional or opposite transfer functions. Then, for any weight and threshold initialization there exists

$$M = \prod_{i=1}^{L} 2^{k_l} \tag{3}$$

possible network configurations, each uniquely associated with a different combination of transfer functions. Furthermore, it can be experimentally verified that the probability Pr that any

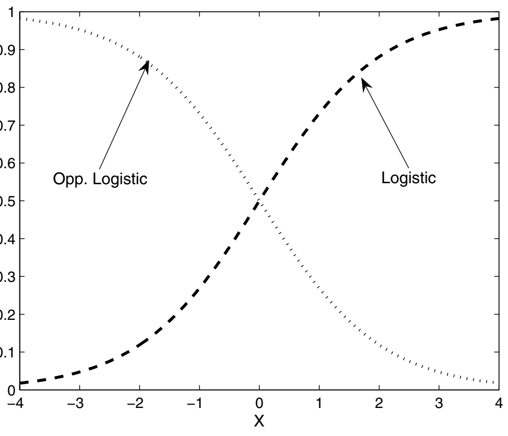

Fig. 1: Logistic and opposite logistic functions.

one of these M networks has the lowest error as calculated by some error function $Er\colon \Re \to \Re$, can be given by

$$Pr(Er(m_i) < Er(m_j)\ \forall i \neq j) = \frac{1}{M}. \tag{4}$$

That is, the choice of initial network with respect to its transfer functions is independent of the initial weights and threshold values.

However, as the learning algorithm (not limited to BPTT) iterates it specializes the weights/thresholds for the some network architecture. Consequently, reducing the probability other architectures yield more favorable results at the respective point in weight space. Therefore, opposite transfer functions will prove to be more useful during the early learning stages. The rate at which this usefulness degenerates is based on properties of the error surface, the learning algorithm, the training data, etc.

Before we can propose an opposition-based neural learning algorithm, it is necessary to first define the notion of an opposite network w.r.t. transfer functions.

Definition 2 (Opposite Transfer Function Network): Given some network N, its opposite $\breve{N}$ w.r.t. its transfer functions is some network with any other combination of hidden neuron transfer functions. That is, there is at least one different transfer function between N and $\breve{N}$. In total there are $M-1$ opposite networks relative to N.

IV. The OBPTT Algorithm

This section will describe the opposition-based backpropagation through time (OBPTT) algorithm, which is based on the concept of opposition-based learning. Specifically we are dealing with Elman [16] recurrent topologies where only neurons of the hidden layer exhibit recurrent connections (although the technique is not limited to this topology). These recurrent values are stored in a context layer which is really a copy of the hidden layer, as shown in Figure 2.

The proposed opposition-based backpropagation through time algorithm is presented in Algorithm 1. The decision as to

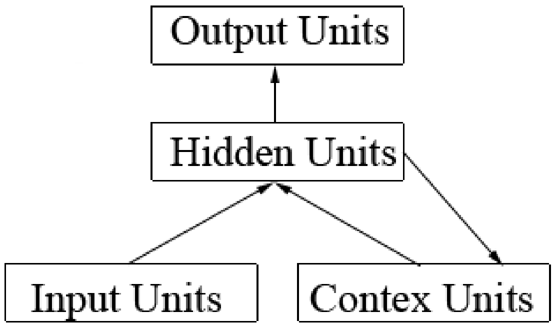

Fig. 2: The Elman recurrent network topology.

which set of transfer functions will be utilized is based on a probabilistic estimation procedure, where each hidden neuron n_i has an associated probability that is represented by $Pr(n_i)$.

Lines 2-6 utilize the current probability of a network using the "normal" transfer function, or its opposite. This probability approaches 1 or 0, representing the "normal" or opposite functions, respectively. Therefore, as the number of epochs $\rightarrow \infty$ the number of opposite networks under consideration approaches 0. In line 9 we then choose the best network of the $Q + 1$ (+1 for current network) networks under consideration.

The probabilities are updated in lines 12-17. If, according to the error function Er we have discovered a network with a lower error, we then increase the probability of the same network being selected by a factor of $0 < \tau^{ampl} < 1$, where the function $trans(n_i)$ is defined as

$$trans(n_i) = \begin{cases} 1 & \text{if neuron } i \text{ uses "normal" function} \\ 0 & \text{if neuron } i \text{ uses "opposite" function} \end{cases}.$$

In the event the network error was not improved, the probabilities of the transfer functions in network N will be reduced according to $0 < \tau^{decay} < 1$.

This algorithm also has the inherent behavior of selected networks similar to the best known, which arises as a result of these probabilities. The assumption being that as learning progresses, the probability of a similar network (where only a small number of transfer functions differ) of achieving a higher accuracy is higher than for very dissimilar networks (where many transfer functions differ).

V. OBPTT Test Problems

In order to determine the quality of the proposed algorithm, we have chosen three common benchmark problems (embedded Reber grammar, symbolic laser and Mackey-Glass). Each of these problems represents time series data exhibiting different degrees of difficulty.

A. Embedded Reber Grammar

A Reber grammar [17] is a deterministic finite automaton capable of generating some language, L. An element, or string $x \in L$ is generated according to the rules of the Reber grammar. Furthermore, another (invalid) string can also be generated that does not obey the grammar. Both strings can be used to train the neural network such that it can distinguish whether some unknown string z is an element of L or not. In this work, we only utilize positives example during training. Figure 3 shows a finite automaton for a Reber grammar, where $L = \{B, P, T, S, X, V, E\}$

Algorithm 1 An epoch of Opposition-based Backpropagation Through Time

```
Require: current network N, and error function Er.
Ensure: $\tau_{ampl} > 0$ and $\tau_{decay} > 0$
Ensure: $0 < Pr(n_i) < 1$
1:  {Examine opposite networks}
2:  select $Q \subseteq N$ {$Note: |Q| \rightarrow 0$ as epochs $\rightarrow \infty$ }
3:  backpropagate_one_epoch(N)
4:  for all $q \in Q$ do
5:      backpropagate_one_epoch(q)
6:  end for
7:
8:  {Let N be the network with minimum error}
9:  $N$=network with $\min(Er(N), Er(q))\ \forall q \in Q$
10:
11: {Update probabilities}
12: for all $hidden\ neurons\ n_i \in N$ do
13:     if $Er(N) - Er(N_{i-1}) \leq 0$ then
14:         $Pr(n_i) = \begin{cases} \tau_{ampl} \cdot Pr(n_i) & \text{if } trans(n_i) = 1 \\ 1 - \tau_{ampl} \cdot Pr(n_i) & \text{if } trans(n_i) = 0 \end{cases}$
15:     else
16:         $Pr(n_i) = \begin{cases} \tau_{decay} \cdot Pr(n_i) & \text{if } trans(n_i) = 0 \\ 1 - \tau_{decay} \cdot Pr(n_i) & \text{if } trans(n_i) = 1 \end{cases}$
17:     end if
18: end for
```

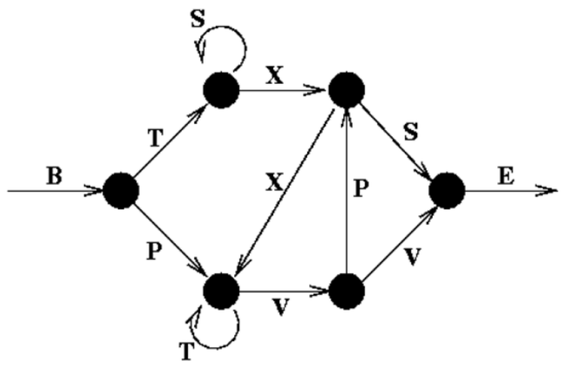

Fig. 3: A finite automaton representation of a Reber grammar (Image was taken from [18]).

A more difficult task is an extension of this problem, known as an embedded Reber grammar. These grammars are of the form $nLm|aLb$, where n, m, a and b are unique strings also in L. Thus, this task is much more difficult because the neural network must remember the initial sequence for some number of previous time steps.

For the experiments in this paper we utilize a language of 6 symbols which are then converted to binary strings. The new representation is a 6-bit binary string that contains a '1' at each position which corresponds to each of the 6 symbols. In total our training data consists of 8000, 6-bit embedded Reber grammar strings. The goal of this learning task is to be able to predict the next symbol in the training data.

B. Symbolic Laser Data

In this section we describe the common deterministic chaotic benchmark known as the Laser problem [19]. The data represents 8,000 readings from a Far-Infrared-Laser in a chaotic state. Each reading is separated by an 81.5 micron 14NH3 laser and the intensity is recorded by an oscilloscope.

In general, deterministic chaotic dynamical systems tend to organize themselves around attractors that contain various levels of instability. For example, a period of unpredictable behavior can follow a period of predictable behavior, and vice versa. It is possible to transform raw data into a symbolic representation of each level of instability [20]. From this transformed data it is possible to gain insight into basic structure of the system.

We can apply this notion to the 8,000 laser readings. We convert the series into a 4 character symbolic stream $S = \{s_t\}$ where $s_t \in \{1, 2, 3, 4\}$. The four symbols correspond to high/low negative and high/low positive laser activity changes. An example of this process is presented in Figure 4.

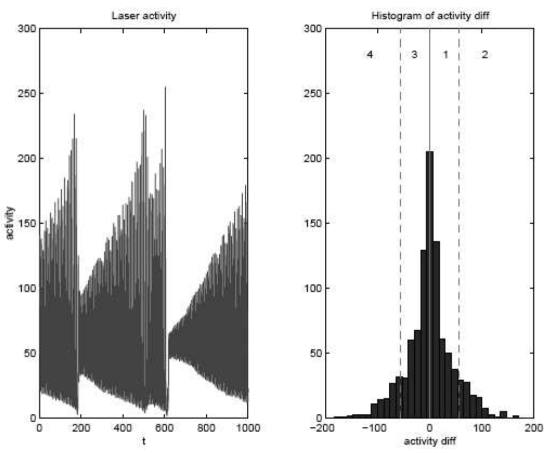

Fig. 4: (Left) First 1000 laser readings, (Right) Histogram of the differences in successive laser readings.

Each of the 4 symbols was converted to a 4-bit binary pattern, where each symbol corresponded to a '1' at the respective index position. The output pattern is also 4-bits, and represents the next laser reading. Therefore the purpose of the network is to predict successive levels of instability.

C. Mackey-Glass

The Mackey-Glass equation [21] is a time-delayed differential equation that models the dynamics of white blood cell production in the human body. The function produces a chaotically evolving continuous dynamic system. We have utilized equation (5) to generate the 200 training patterns:

$$\frac{dx}{dt} = -b \cdot x(t) + \frac{a \cdot x(t-\tau)}{1 + x(t-\tau)^c} \tag{5}$$

In this equation we let $b = 0.1$, $a = 0.2$, $c = 10$ and $\tau = 30$ with initial condition $x(0) = 0.9$ (see Figure 5).

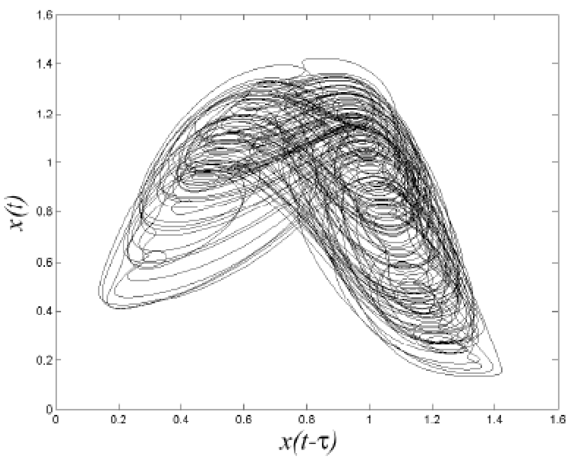

Fig. 5: Phase portrait of the Mackey-Glass system given by equation (5).

The input to the neural network is a single real-valued variable, and the output is the next value of the time series. So, the purpose of the network is to be able to predict the next value in the series.

VI. Results

We examine the behavior of the Elman networks on the above benchmark problems. It has been empirically determined that values of $\tau^{ampl} = 0.75$ and $\tau^{decay} = 0.1$ yield best results. Additionally, to determine sensitivity to the number of hidden neurons we have conducted experiments that vary the number of neurons over $\{3, 5, 7\}$. Also, varying the learning rate and momentum values and their effect is examined. Each of the networks use the logistic (1) and opposite logistic (2) functions in the hidden layer. It should be noted that we also performed experiments using the hyperbolic tangent ($tanh(\cdot)$) and its opposite function ($-tanh(\cdot)$) were conducted however they did not yield significantly different results than those presented.

Each of the network errors is gauged by the mean squared error (MSE) error function and each of the OBPTT and BPTT runs began at the exact initial weight/threshold conditions so as to rule out any bias in initial weight/threshold value generation. All networks weights were initialized according to the Nguyen-Widrow rule [22]. We also experimented with initial weights on the interval $[0, 1]$ and observed similar results, although not presented here due to space limitations. Also, in each case the maximum number of opposite networks considered during a single epoch was empirically set to $|Q| = 3$. All of the results have been averaged over 30 runs, each composed of 30 epochs.

A. Embedded Reber Grammar Results

Table I shows a summary of the results for the embedded Reber experiments where α and β represent the learning rate and momentum parameters, respectively. The table also shows the results of varying the number of hidden layers.

The average error, μ, of the networks trained using the OBPTT technique is lower for nearly all experiments. Furthermore, the network outputs are more reliable as the standard deviation σ tends to be lower for all the examples as well. In many of the cases, the standard deviation is half that of the traditional technique. Nevertheless, these results provide experimental evidence showing that the OBPTT trained networks achieve a lower, more reliable error measure.

TABLE I: Results for Embedded Reber Grammar Experiments

		OBPTT		BPTT	
Hidden Neurons=3					
α	β	μ	σ	μ	σ
0.10	0.00	0.043	0.003	0.047	0.006
0.25	0.00	0.042	0.003	0.045	0.005
0.25	0.10	0.044	0.004	0.049	0.006
0.50	0.00	0.043	0.003	0.049	0.008
0.50	0.10	0.044	0.005	0.049	0.006
0.50	0.25	0.045	0.003	0.051	0.005
0.75	0.00	0.045	0.004	0.050	0.005
0.75	0.10	0.047	0.005	0.051	0.006
0.75	0.25	0.046	0.004	0.052	0.005
0.75	0.50	0.051	0.006	0.052	0.004
Hidden Neurons=5					
α	β	μ	σ	μ	σ
0.10	0.00	0.040	0.002	0.045	0.004
0.25	0.00	0.040	0.002	0.045	0.004
0.25	0.10	0.040	0.002	0.043	0.003
0.50	0.00	0.042	0.003	0.046	0.003
0.50	0.10	0.042	0.003	0.046	0.004
0.50	0.25	0.042	0.001	0.046	0.004
0.75	0.00	0.043	0.002	0.047	0.004
0.75	0.10	0.044	0.003	0.047	0.005
0.75	0.25	0.045	0.004	0.050	0.005
0.75	0.50	0.046	0.003	0.051	0.006
Hidden Neurons=7					
α	β	μ	σ	μ	σ
0.10	0.00	0.039	0.001	0.042	0.001
0.25	0.00	0.040	0.001	0.042	0.002
0.25	0.10	0.040	0.001	0.042	0.002
0.50	0.00	0.041	0.001	0.044	0.004
0.50	0.10	0.042	0.002	0.046	0.004
0.50	0.25	0.042	0.001	0.046	0.003
0.75	0.00	0.043	0.002	0.045	0.004
0.75	0.10	0.043	0.002	0.046	0.004
0.75	0.25	0.044	0.002	0.047	0.006
0.75	0.50	0.045	0.002	0.049	0.005

Figure 6 shows a characteristic plot of a 5 hidden-neuron network trained with $\alpha = 0.1$ and $\beta = 0.0$. Not only does the OBPTT approach converge to a higher quality solution relatively quickly, but it also exhibits a more stable learning trajectory about the convergence point. While the BPTT trained network increases in error after the 8^{th} epoch, the network trained by OBPTT maintains a relatively stable nature. This is important to recurrent networks because it indicates the ability to adequately fit the data over the length of the time series without losing stability of the learning trajectory.

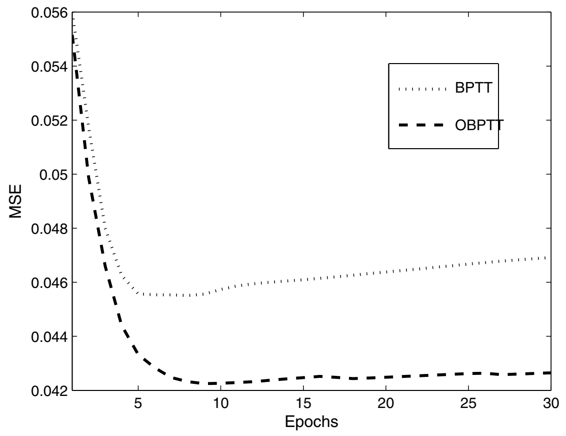

Fig. 6: Sample output for the Reber Grammar problem for a network with 5 hidden neurons. The networks were trained using $\alpha = 0.1$ and $\beta = 0.0$.

The networks trained in Figure 7 further support the above claims regarding relatively higher quality networks, with respect to final error. We can see that the OBPTT trained network achieves a significantly lower error in the same amount of epochs as its BPTT counterpart. These networks were trained with $\alpha = 0.25$ and $\beta = 0.1$ and had 7 hidden neurons.

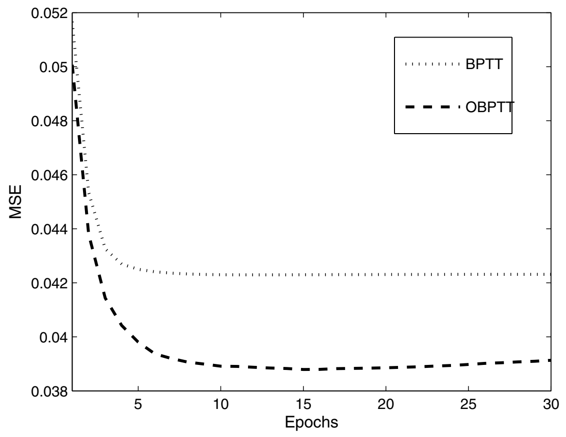

Fig. 7: Sample output for Reber Grammar problem for a network with 7 hidden neurons. The networks were trained using $\alpha = 0.1$ and $\beta = 0.1$.

The number of considered opposite networks $|Q|$ is plotted against the number of epochs in Figure 8. The average number of these networks that improved on the network error is also shown. Both $|Q|$ and the number of improving networks decreases to nearly zero within the allotted 30 epochs indicating that one of the networks chosen in the initial epochs (typically the first 2-5 epochs) was quite well suited, and the algorithm quickly discarded the others from consideration.

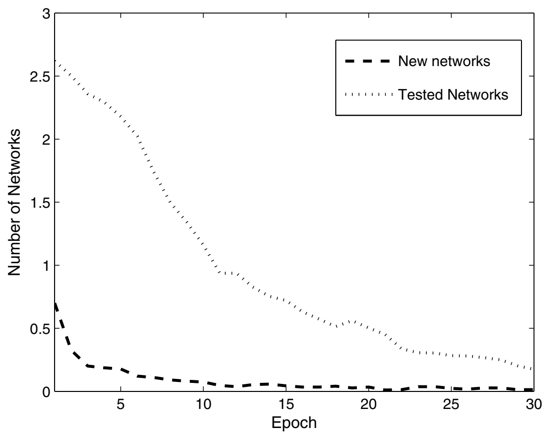

Fig. 8: Usefulness of opposite networks against number of epochs for the Reber grammar problem. These values represent the average over all α and β for 7 hidden neurons.

B. Symbolic Laser Results

A summary of the results for the symbolic laser experiments is provided in Table II. Overall much of these results are very similar, both in average error and standard deviation. However, the OBPTT trained networks seem to be able to achieve slightly lower error levels over most of these experiments.

Figure 9 gives an idea of the similarity of the results. The network trained with OBPTT achieves slightly lower error values early on, but by the 30^{th} epoch training limit, the networks are nearly indistinguishable, in terms of MSE. As with the Reber grammar we observe more stable learning trajectory, although more pronounced for this problem. These networks both had 5 hidden neurons and $\alpha = 0.25$, $\beta = 0.0$.

Figure 10 shows the degree to which opposite neurons are considered and used versus the number of training epochs. Unlike the Reber grammar experiments, the number of networks considered and utilized decreases at a slower rate, implying that OBPTT has not yet determined which network architecture is best for this problem. Additionally, the stability of the learning trajectory can be attributed to this behavior. That is, as the network becomes unstable, a more suitable set of transfer functions is utilized.The results presented represent averages over all α and β values for 7 hidden neurons.

C. Mackey-Glass Results

The final benchmark problem under consideration was the Mackey-Glass data set. As Table III shows, the BPTT and OBPTT approaches are nearly indistinguishable (values of 0.000 for standard deviations are not necessarily 0, but rather not within the number of significant digits). This is due to a smaller data set (200 patters), as well as the problem difficulty. Furthermore, the low dimensionality (1 dimensional input, 1 dimensional output) of the problem also influences the similarity in behavior. We have not presented any graphs for these experiments because of the degree to which the results are similar (i.e. the graphs are nearly identical). This applies to both accuracy and convergence results. However,

TABLE II: Results for Symbolic Laser Experiments

		OBPTT		BPTT	
Hidden Neurons=3					
α	β	μ	σ	μ	σ
0.10	0.00	0.048	0.012	0.046	0.011
0.25	0.00	0.055	0.013	0.054	0.010
0.25	0.10	0.060	0.010	0.065	0.014
0.50	0.00	0.065	0.009	0.069	0.008
0.50	0.10	0.067	0.008	0.068	0.010
0.50	0.25	0.064	0.008	0.071	0.006
0.75	0.00	0.065	0.007	0.072	0.008
0.75	0.10	0.068	0.007	0.073	0.007
0.75	0.25	0.070	0.008	0.074	0.006
0.75	0.50	0.070	0.007	0.074	0.004
Hidden Neurons=5					
α	β	μ	σ	μ	σ
0.10	0.00	0.036	0.006	0.039	0.005
0.25	0.00	0.044	0.010	0.050	0.019
0.25	0.10	0.046	0.009	0.050	0.014
0.50	0.00	0.052	0.011	0.060	0.011
0.50	0.10	0.060	0.015	0.062	0.016
0.50	0.25	0.058	0.013	0.065	0.011
0.75	0.00	0.059	0.014	0.065	0.013
0.75	0.10	0.060	0.016	0.066	0.010
0.75	0.25	0.062	0.010	0.073	0.006
0.75	0.50	0.063	0.009	0.070	0.009
Hidden Neurons=7					
α	β	μ	σ	μ	σ
0.10	0.00	0.029	0.005	0.030	0.004
0.25	0.00	0.038	0.011	0.037	0.009
0.25	0.10	0.039	0.012	0.041	0.011
0.50	0.00	0.047	0.013	0.052	0.013
0.50	0.10	0.051	0.011	0.055	0.013
0.50	0.25	0.050	0.012	0.055	0.017
0.75	0.00	0.054	0.015	0.062	0.016
0.75	0.10	0.053	0.012	0.061	0.014
0.75	0.25	0.055	0.014	0.066	0.013
0.75	0.50	0.063	0.012	0.067	0.014

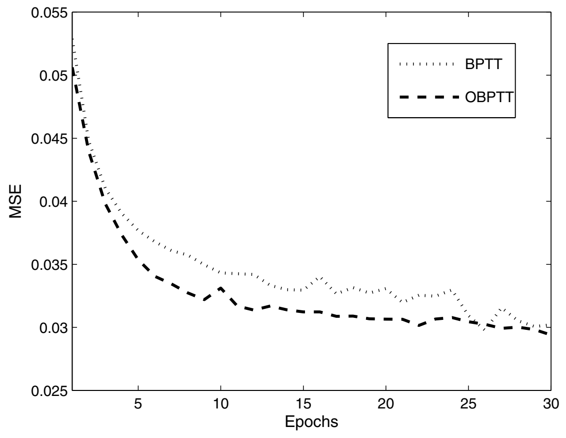

Fig. 9: Sample output for symbolic laser problem for a network with 5 hidden neurons. The networks were trained using $\alpha = 0.25$ and $\beta = 0.0$.

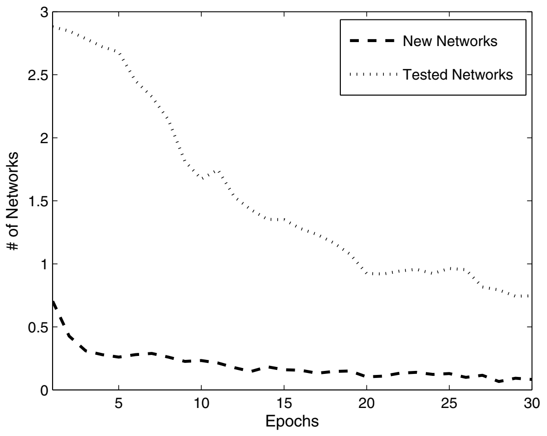

Fig. 10: Use of opposite neurons for a 5-hidden neuron network, averaged over all values of α and β for the Symbolic Laser problem.

it is important to include this information to highlight the fact that OBPTT may not be useful for simple, very low dimensional problems where BPTT already performs very well.

TABLE III: Results for Mackey-Glass Experiments

		OBPTT		BPTT	
Hidden Neurons=3					
α	β	μ	σ	μ	σ
0.10	0.00	0.017	0.000	0.017	0.000
0.25	0.00	0.017	0.000	0.017	0.000
0.25	0.10	0.017	0.000	0.017	0.000
0.50	0.00	0.019	0.000	0.020	0.000
0.50	0.10	0.019	0.000	0.019	0.000
0.50	0.25	0.019	0.000	0.020	0.000
0.75	0.00	0.023	0.000	0.024	0.000
0.75	0.10	0.023	0.000	0.024	0.000
0.75	0.25	0.024	0.000	0.024	0.000
0.75	0.50	0.023	0.000	0.024	0.000
Hidden Neurons=5					
α	β	μ	σ	μ	σ
0.10	0.00	0.017	0.000	0.017	0.000
0.25	0.00	0.017	0.000	0.018	0.000
0.25	0.10	0.017	0.000	0.018	0.000
0.50	0.00	0.019	0.000	0.020	0.000
0.50	0.10	0.019	0.000	0.020	0.000
0.50	0.25	0.020	0.000	0.020	0.000
0.75	0.00	0.023	0.000	0.024	0.000
0.75	0.10	0.023	0.000	0.024	0.000
0.75	0.25	0.024	0.000	0.024	0.000
0.75	0.50	0.024	0.000	0.024	0.000
Hidden Neurons=7					
α	β	μ	σ	μ	σ
0.10	0.00	0.017	0.000	0.017	0.000
0.25	0.00	0.017	0.000	0.018	0.000
0.25	0.10	0.017	0.000	0.018	0.000
0.50	0.00	0.019	0.000	0.020	0.000
0.50	0.10	0.019	0.000	0.020	0.000
0.50	0.25	0.020	0.000	0.021	0.000
0.75	0.00	0.023	0.000	0.024	0.000
0.75	0.10	0.023	0.000	0.024	0.000
0.75	0.25	0.024	0.000	0.024	0.000
0.75	0.50	0.024	0.000	0.024	0.000

D. Summary

From the results of the benchmark problems it can be inferred that in general the results obtained by OBPTT are at least as good as those obtained by traditional BPTT. However, for more difficult problems like the embedded Reber grammar the OBPTT approach yields significantly higher quality results as is depicted in Table IV. We have also seen evidence to indicate a more stable learning trajectory and more reliable results.

Table IV outlines the expected behavior over all values of α and β for both learning algorithms using 3, 5 and 7 hidden layers from the data presented in the previous subsections. Additionally, the significance level $(1 - confidence)$ is provided to test the null hypothesis that the means of the OBPTT and BPTT approaches are indistinguishable. This is especially true for the embedded Reber grammar problem where the it can be said with 99% confidence that the OBPTT algorithm achieves a lower error for any given value of α, β or number of hidden neurons.

However, the confidence that OBPTT outperforms BPTT drops significantly for the symbolic laser and Mackey-Glass data. Nevertheless, these results still support the claim that OBPTT yields at least as good results as those found with BPTT. Furthermore, as was described above the convergence rate and stability of of the learning trajectory of OBPTT seem to also be superior to BPTT. Stability can be a great concern for many dynamic problems [1].

TABLE IV: Summary of Results for All Experiments

		Summary		
	Hidden	Embedded Reber	Laser	Mackey-Glass
OBPTT	3	0.045	0.063	0.020
	5	0.042	0.054	0.020
	7	0.042	0.048	0.020
BPTT	3	0.050	0.067	0.020
	5	0.046	0.060	0.021
	7	0.045	0.053	0.022
Significance	3	0.0007	0.3714	0.7131
	5	0.0006	0.1904	0.6488
	7	0.0062	0.3686	0.4930

VII. Conclusions and Future Work

We have presented an opposition-based learning framework for improving the BPTT algorithm. Our proposed approach uses the concept of opposite transfer functions which are dynamically determined during runtime to achieve more desirable results. Specifically, we have experimentally shown that the accuracy of OBPTT trained networks achieve at least the same final error as those trained with BPTT. However, using OBPTT leads to more reliable results and a more stable learning trajectory which is a very important aspect of recurrent learning. Our findings are based on experiments conducted on the embedded Reber grammar, symbolic laser and Mackey-Glass time series problems.

Directions for future work involve further investigation into properties of the error surface, specifically the influence of dynamic neuron transfer functions. The network's stability and other theoretical properties are also very important to understand as is the impact of OBPTT on the network's generalization ability. Employing the concept of opposite transfer functions, and OBL in general, to improving other types of learning algorithms and types of neural networks is a very important direction.

REFERENCES

[1] D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. John Wiley and Sons Ltd, 2001.

[2] D. Rumelhart, G. Hinton, and R. Williams, *Parallel Distributed Processing*, ch. Learning Internal Representations by Error Propagation, Chapter 8. MIT Press, 1986.

[3] P. Werbos, "Backpropagation Through Time: What it Does and How to do it," in *Proceedings IEEE*, vol. 78, pp. 1150–1160, 1990.

[4] R. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[5] R. Williams and J. Peng, *Backpropagation: Theory, Architectures, and Applications*, ch. Gradient-Based Learning Algorithms for Recurrent Networks and their Computational Complexity. Lawrence Erlbaum, 1992.

[6] P. Baldi, "Gradient Descent Learning Algorithms: A General Overview," *IEEE Transactions on Neural Networks*, vol. 6, pp. 182–195, 1995.

[7] H. R. Tizhoosh, "Opposition-based Learning: A New Scheme for Machine Intelligence," in *International Conference on Computational Intelligence for Modelling, Control and Automation*, 2005.

[8] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "A Novel Population Initialization Method for Accelerating Evolutionary Algorithms," *(to appear) Computers and Mathematics with Applications*, 2006.

[9] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "Opposition-based Differential Evolution Algorithms," in *IEEE Congress on Evolutionary Computation*, pp. 7363–7370, 2006.

[10] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "Opposition-based Differential Evolution Algorithms for Optimization of Noisy Problems," in *IEEE Congress on Evolutionary Computation*, pp. 6756–6763, 2006.

[11] M. Shokri, H. R. Tizhoosh, and M. Kamel, "Opposition-based Q(lambda) Algorithm," in *IEEE International Joint Conference on Neural Networks*, pp. 646–653, 2006.

[12] H. R. Tizhoosh, "Opposition-based Reinforcement Learning," *(to appear) Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 10, no. 4, pp. 578–585, 2006.

[13] H. R. Tizhoosh, "Reinforcement Learning Based on Actions and Opposite Actions," in *International Conference on Artificial Intelligence and Machine Learning*, 2005.

[14] M. Ventresca and H. R. Tizhoosh, "Improving the Convergence of Backpropagation by Opposite Transfer Functions," in *IEEE International Joint Conference on Neural Networks*, pp. 9527–9534, 2006.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation, 2nd Edition*. Prentice Hall, 1998.

[16] J. L. Elman, "Distributed Representations, Simple Recurrent Networks and Grammatical Structure," *Machine Learning*, vol. 7, no. 2, pp. 195–226, 1991.

[17] A. S. Reber, "Implicit Learning of Synthetic Languages: The Role of Instructional Set," *Journal of Experimental Psychology: Human Learning and Memory*, vol. 2, no. 1, pp. 88–94, 1976.

[18] G. Orr, "www.willamette.edu/ gorr/classes/cs449/reber.html," October 2006.

[19] A. Weigand and N. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1994.

[20] A. Katok and B. Hasselblatt, *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press, 1995.

[21] M. C. Mackey1 and L. Glass, "Oscillations and Chaos in Physiological Control Systems," *Science*, vol. 197, pp. 287–289, 1977.

[22] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *IEEE Proceedings of the International Joint Conference on Neural Netowrks*, vol. 3, pp. 21–26, 1990.