Spiking Neural Network Training Using Evolutionary Algorithms

N.G. Pavlidis[1,3], D.K. Tasoulis[1,3], V.P. Plagianakos[2,3] and M.N. Vrahatis[1,3]
[1]Department of Mathematics, University of Patras, GR-26110 Patras, Greece
[2]Department of Information and Communication Systems Engineering,
University of the Aegean, GR-83200 Samos, Greece
[3]University of Patras Artificial Intelligence Research Center
E-mail: {npav,dtas,vpp,vrahatis}@math.upatras.gr

***Abstract*— Networks of spiking neurons can perform complex non–linear computations in fast temporal coding just as well as rate coded networks. These networks differ from previous models in that spiking neurons communicate information by the timing, rather than the rate, of spikes. To apply spiking neural networks on particular applications, a learning process is required. Most existing training algorithms are based on unsupervised Hebbian learning. In this paper, we investigate the performance of the Parallel Differential Evolution algorithm, as a supervised training algorithm for spiking neural networks. The approach was successfully tested on well–known and widely used classification problems.**

I. Introduction

Artificial Neural Networks (ANNs) are parallel computational models comprised of densely interconnected, simple, adaptive processing units, characterized by an inherent propensity for storing experiential knowledge and rendering it available for use. ANNs resemble the human brain in two fundamental respects; firstly, knowledge is acquired by the network from its environment through a learning process, and secondly, synaptic weights are employed to store the acquired knowledge.

The building block of ANNs is the model of the artificial neuron. In [7] three generations of artificial neuron models were distinguished. The first generation of neurons gave rise to multilayered perceptrons, Hopfield nets, and Boltzmann machines. These networks can compute any boolean function, as well as, all digital functions. The second generation neurons employ activation functions with a continuous set of possible output values to a weighted sum of the inputs (e.g. sigmoid functions, linear saturated functions, piecewise exponential functions). From this emerged feedforward and recurrent sigmoidal neural nets and networks of radial basis function units. These networks can further approximate, arbitrarily closely, any continuous function defined on a compact domain, and support learning algorithms based on gradient descent. All these models require the timing of individual computation steps to adhere to a global schedule that is independent of the values of the input parameters. Incoming spikes induce a postsynaptic potential according to an impulse response function.

Spiking neurons are considered to comprise the third generation of neurons. The main motivation behind the spiking neuron model was the fact that computation in the brain is primarily carried out by *spiking neurons*. A spiking neuron *fires* at certain points in time, thereby sending an electric pulse which is commonly referred to as *action potential*, or *spike*. Incoming spikes induce a postsynaptic potential according to an impulse response function. The size and shape of a spike is independent of the input, but the *time* when a neuron fires depends on input of that neuron. Thereby, information in SNNs is propagated by the timing of individual spikes. The latter fact renders SNNs capable of exploiting time as a resource for coding and computation in a much more sophisticated manner than typical computational models. Substantial evidence indicates that the time structure of sequences of neuronal spikes (spike trains) is relevant in neural signal processing. Experimental results coupled with theoretical studies suggest that temporal correlation of activity can be exploited by the brain to bind features of one object and to differentiate objects [2], [17]. For practical applications of temporally encoded SNNs, a learning process is required. Recently, a modification of the classical backpropagation algorithm has been proposed for the training of SNNs [1].

In the present paper we consider the application of a parallel version of the Differential Evolution [16] algorithm to the problem of supervised training of SNNs. The paper is organized as follows, Section II provides a brief introduction to the Spike–Response Model and the theoretical results that underpin the computational power of SNNs. Section III describes the Parallel Differential Evolution algorithm. Next the experimental results are reported and discussed. The paper ends with conclusions and ideas for future research.

II. Spiking Neural Networks

The spiking nature of biological neurons has recently led to explorations of the computational power associated with temporal information coding in single spikes. In [6] it was proven that networks of spiking neurons can simulate arbitrary feedforward sigmoidal neural nets and can thus approximate any continuous function. In fact, it has been shown theoretically that spiking neural networks that convey information by individual spike times are computationally more powerful than neurons with sigmoidal activation functions.

A. Spike–Response Model

For completeness purposes, we provide some background material on the basic spike–response model [3], [5]. The state of neuron i is described by the state variable u_i. The neuron is said to fire if u_i crosses from below a predefined threshold θ. The moment of threshold crossing defines the firing time $t_i^{(f)}$. The set of all firing times of neuron i is denoted by,

$$\mathcal{F}_i = \{t_i^{(f)} \,:\, 1 \leqslant f \leqslant n\} \equiv \{t \,:\, u_i(t) \geqslant \theta\}. \tag{1}$$

Two factors contribute to the value of the state variable u_i. First, immediately after firing an output spike at time $t_i^{(f)}$, the state variable u_i is reduced. Second, the neuron receives input from a set of presynaptic neurons $j \in \Gamma_i$, where

$$\Gamma_i = \{j \,:\, j \text{ presynaptic to } i\}. \tag{2}$$

A presynaptic spike at time $t_j^{(f)}$ affects the state $u_i(\cdot)$ of neuron i for $t \geqslant t_j^{(f)}$ by the amount $w_{ij}\epsilon\left(t - t_j^{(f)}\right)$. The weight w_{ij} is a factor which accounts for the strength of the connection, while $\epsilon(s)$ is a kernel function. A kernel function is a piecewise continuous, bounded, symmetric around zero, concave at zero, real valued, function. For convenience, kernel functions often integrate to one.

The state $u_i(t)$ of a neuron i at time t is given by the linear superposition of all contributions,

$$\begin{aligned} u_i(t) \;&=\; \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta\left(t - t_i^{(f)}\right) + \\ &\quad + \sum_{j \in \Gamma_i} w_{ij} \left(\sum_{t_j^{(f)} \in \mathcal{F}_j} \epsilon\left(t - t_j^{(f)}\right) \right). \end{aligned} \tag{3}$$

The terms on the right–hand side of Eq. (3) admit a straightforward interpretation. The first sum captures the impact the previous spikes of neuron i exert on its state. The sum over the $\epsilon(\cdot)$ kernels model the neuron's response to all presynaptic spikes. Eqs. (1)–(3) define the generic *Spike Response Model*. To be more specific, we consider the following kernel functions for $\eta(\cdot)$ and $\epsilon(\cdot)$.

The kernel $\eta(s)$ is usually nonpositive for $s > 0$ and in our implementation it assumes the form [5],

$$\eta(s) = -\eta_0 \exp\left(-\frac{s}{\tau}\right) \mathcal{H}(s), \tag{4}$$

where η_0 is the amplitude of the relative refractoriness; τ is a decay time constant; and $\mathcal{H}(s)$ is the well–know heavyside function which vanishes for $s \leqslant 0$ and assumes the value of 1 for $s > 0$. Assuming continuous time, at the moment of firing the value of the state variable $u_i(t)$ is equal to the threshold θ. The effect of Eq. (4) therefore, is to set $u_i(t)$ to the value $(\theta - \eta_0)$ after each firing instance. Note that if $\eta_0 = \theta$, then the state variable, $u_i(t)$ is reset to zero after a firing instance.

The kernel $\epsilon(\cdot)$ models the unweighted postsynaptic potential (PSP) of a single spike of a neuron $j \in \Gamma_i$ impinging on neuron i. The mathematical formulation used is,

$$\epsilon(s) = \exp\left(-\frac{s}{\tau_{ij}}\right) \mathcal{H}(s), \tag{5}$$

where τ_{ij} is an another decay time constant. The amplitude of the PSP is modulated by the synaptic weight factor w_{ij} (in Eq. (3)). For inhibitory synapses the kernel can have a negative sign in front of the right–hand side of Eq. (5). Alternatively, the sign of the weight w_{ij} can determine the excitatory or inhibitory nature of the postsynaptic potential. The latter approach is adopted in this contribution. A positive weighted PSP is referred to as excitatory PSP (in abbreviation EPSP), while a negative weighted PSP is called inhibitory PSP (henceforth IPSP).

B. Theoretical Aspects of SNNs

On the basis of the computational mechanism described above networks of spiking neurons can be created to approximate any bounded continuous function in the temporal domain. Theorem 1 stated below provides the basis for the proof of the previous proposition [6].

Theorem 1: Any feedforward or recurrent analog neural network, consisting of s sigmoidal neurons that employ the piecewise linear gain function can be simulated arbitrarily closely by a network of $(s + c)$ spiking neurons (where c is a small constant) with analog inputs and outputs encoded by temporal delays of spikes. This holds even if the spikes are subject to noise.

It is well–known that feedforward sigmoidal neural networks with piecewise linear gain function are universal approximators. More specifically, in [4], [18] the following Theorem is proved.

Theorem 2: Standard Feedforward Networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accuracy.

Hence Theorem 1 in conjunction with Theorem 2 implies the following Corollary:

Corollary 1: Any given continuous function $F : [0,1]^n \rightarrow [0,1]^m$ can be approximated arbitrarily closely by a network of spiking neurons with inputs and outputs encoded by temporal delays.

III. Parallel Evolutionary Training Algorithm

Differential Evolution (DE) is a novel minimization method [16], capable of handling nondifferentiable, nonlinear and multimodal objective functions. DE has been designed as a stochastic parallel direct search method, which utilizes concepts borrowed from the broad class of evolutionary algorithms (EAs). The method typically requires few, easily chosen, control parameters. Experimental results have shown that DE has good convergence properties and outperforms other well known evolutionary algorithms [15], [16].

DE, like other EAs, is easily parallelized due to the fact that each member of the population is evaluated individually [14].

The only phase of the algorithm that requires communication with other individuals is in reproduction. This phase can also be parallelized for pairs of individuals [10], [14]. Generally, there are two typical models for EA parallelization. The first employs fine grained parallelism, in which case, each individual is assigned to a different processor. This approach is problematic when the number of available processors is limited, or when the computation of the fitness function requires information from the entire population. The second model, which is used in this paper, maps an entire subpopulation to a processor. Thus each subpopulation evolves independently toward a solution. This allows each subpopulation to develop its own solution independently. To promote information sharing, the best individual of each subpopulation is moved to other subpopulations, according to a predefined topology. This operation is called "migration". The topology of the proposed scheme is a ring, i.e. the best individuals from each subpopulation are allowed to migrate to the next subpopulation of the ring. This concept reduces the migration between the subpopulations and consequently the messages exchanged among the processors [12].

The migration of the best individuals is controlled by the migration constant, $\phi \in (0,1)$. At each iteration, a random number from the interval $(0,1)$ is uniformly chosen and compared with the migration constant. If the migration constant is larger, then the best individuals of each subpopulation migrate and take the place of a randomly selected individual (different from the best) in the next subpopulation; otherwise no migration is permitted. We have experimentally found that a migration constant, $\phi = 0.1$, is a good choice, since it allows each subpopulation to evolve for some iterations before the migration phase actually occurs.

To apply the PARallel DE (PARDE) to train spiking neural networks, we start with a specific number of subpopulations, each one consisting of *NP*, D–dimensional weight vectors, and evolve them over time. *NP* is fixed throughout the training process. All the weight subpopulations are randomly initialized in the interval $[-1,1]$, following the uniform probability distribution. At each iteration, called *generation*, all the subpopulations are evolved independently in parallel, until a migration phase is decided. After the migration of the best individuals, the new subpopulations continue to evolve as before.

Let us now focus on a subpopulation. In each subpopulation, new weight vectors are generated by the combination of randomly chosen weight vectors from the population. This operation in our context can be referred to as *mutation*. Specifically, for each weight vector W_g^k, $k = 1,\ldots,$*NP*, where g denotes the current generation, a new vector V_{g+1}^i (mutant vector) is generated according to the following equation:

$$V_{g+1}^k = W_g^k + \mu\left(W_g^{\text{best}} - W_g^k\right) + \mu\left(W_g^{r_1} - W_g^{r_2}\right), \quad (6)$$

where W_g^{best} is the best member of the previous generation; $\mu > 0$ is a real parameter, called mutation constant, which controls the amplification of the difference between two weight vectors so as to avoid the stagnation of the search process; and $r_1, r_2, \in \{1,2,\ldots,k-1,k+1,\ldots,NP\}$, are random integers mutually different and different from the running index k.

To increase further the diversity of the mutant weight vector, the outcoming vectors are combined with another predetermined weight vector – the *target* weight vector – this operation is called *recombination*. Specifically, for each component l ($l = 1,2,\ldots,D$) of the mutant weight vector V_{g+1}^k, we randomly choose a real number r from the interval $[0,1]$. Then, we compare this number with ρ (recombination constant), and if $r \leqslant \rho$ we select, as the l-th component of the trial vector U_{g+1}^k, the corresponding component l of the mutant vector V_{g+1}^k. Otherwise, we pick the l-th component of the target vector W_{g+1}^k. This operation yields the *trial* weight vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the error function. This is the *selection* operation. In [11], it has been shown that this methodology can efficiently train Feedforward Neural Networks with arbitrary, as well as, constrained integer weights.

IV. Experimental Results

For the test problems considered, we made no effort to tune the mutation, recombination and migration constants, μ, ρ and ϕ respectively, to obtain optimal or at least nearly optimal convergence speed. Instead, the fixed values $\mu = 0.8$, $\rho = 1.0$, and $\phi = 0.1$, were used. Smaller values of ϕ can further reduce the messages between the processors and thus reduce computational time, but this may result in rare and insufficient migrations. It is obvious that one can try to fine–tune the μ, ρ, ϕ and *NP* parameters to achieve better results, i.e. less error function evaluations and/or better success rates. The weight subpopulations were initialized with random numbers in the interval $[-1,1]$. The total population size NP was divided equally to the subpopulations. Regarding total population size, experimental results have shown that a good choice is $2D \leqslant NP \leqslant 4D$. Clearly, using a larger population size promotes a finer exploration of the weight space, but this practice usually increases the number of function evaluations performed. On the other hand, small values of *NP* render the algorithm inefficient and more generations are required to converge to the minimum.

Regarding the set of constants that are employed in the equations that describe the spike–response model (Eq. (1)–Eq. (5)), the following values were used. The value of the threshold θ was set to 0.1, the amplitude of the relative refractoriness, $\eta_0 = 1.0$, both decay time constants $\tau = \tau_{ij} = 1.0$. This set of parameter values was selected experimentally. Proper fine–tuning of these parameters can further improve the performance of the SNNs.

Numerical experiments were performed using a Differential Evolution and a Neural Network C++ Interface developed under the Fedora Linux 1.0 operating system using the GNU compiler collection (gcc) version $3.3.2$.

A. The Exclusive–OR Problem

The first test problem we considered is the eXclusive–OR (XOR) Boolean function problem, which historically has been considered as a good test of a network model and learning algorithm. A 2–2–1 SNN (six weights, dimension of the problem $D = 6$) has been used for the simulations. The encoding of this problem in spike–time patterns, was performed by substituting 1 with 10 uniformly distributed spikes in the simulation interval, and 0 with no spikes at all. The desired output of the networks is *at least one* spike at the output neuron for the $\{0, 1\}$ and $\{1, 0\}$ input patterns and zero spikes for the $\{0, 0\}$ and $\{1, 1\}$ input patterns. The following fitness function was used to evaluate network performance:

$$E_p = \begin{cases} 0, & \text{if classification is correct,} \\ |o_p - t_p|, & \text{otherwise,} \end{cases} \quad (7)$$

where p indicates the index of the current pattern, o_p stands for the actual response of the network (in terms of the number of output spikes), and t_p represents the target value for pattern p (number of output spikes as before).

The PARDE trained SNN was able to correctly classify the four patterns associated with the XOR problem in all 100 experiments performed. The maximum number of generations in each run was 10. An advantage of the proposed approach over *SpikeProp* [1] is the significant reduction in the number of weights. Specifically, the SNN trained through SpikeProp had 320 weights, whereas the proposed approach required only 6 weights.

B. The Diabetes Problem

The Diabetes Problem is a classification problem from the Proben1 dataset [13]. To encode the data of this problem in spike times the values of the input variable were normalized to integer values in the range $[0, 10]$. The normalized values represent the number of uniformly distributed spikes over the simulation period. For each input pattern the classification decision of the SNN was decided by the output neuron with the maximum number of spikes. The fitness function in Eq. (7) was employed. The SNN architecture used was an 8–4–4–2 feedforward network.

The performance of the trained SNNs is compared with that of MultiLayer Perceptrons (MLP) trained through Back Propagation (BP) algorithm, Backpropagation with Momentum (MBP) [8], [9], as well as, Second Order Momentum (SMBP). The MLP architecture was 8–2–2–2. The stopping error criterion for training was an error goal of 0.15 for the BP algorithm. Since BP performance heavily depends on the learning rate parameter, while the performance of the momentum methods additionally depends on the value of the momentum term, we tried all the values in the range $(0.0, 1.0)$, with a 0.1 step. For the MBP, the best results were obtained for the values 0.9 and 0.6 for the learning rate and momentum term respectively. For SMBP, the best results were produced for 0.9 and 0.6. The maximum number of generations was 100 for PARDE trained SNNs. For the Backpropagation family methods the maximum number of iterations was 1000. The results of 100 independent experiments for the Diabetes1 problem are exhibited in Table I. The mean performance of the SNNs trained through the PARDE algorithm is comparable to that of multilayer perceptrons.

TABLE I
DIABETES1 PROBLEM TEST SET CLASSIFICATION ERROR (%)

Algorithm	Mean	Stdev	Max	Min
BP	36.45	0	36.45	36.45
MBP	28.71	4.8	36.45	23.43
SMBP	28.70	5.12	36.45	21.88
SNN with PARDE	37.95	4.5	38.21	37.69

C. The Iris Problem

The Iris benchmark dataset consists of 150 samples, each having four features. The dataset consists of three classes, two of which are non–linearly separable. Each class contains 50 instances and refers to a type of iris plant. The dataset was temporarily encoded by normalizing all input values to integers in the range $[0, 10]$. As before, the value of the integer was then transformed to uniformly distributed spikes over the simulation period. The SNN topology used was 4–4–1. If the number of output spikes of the network was in the range $[0, 10]$ the input pattern was classified to the first class, if the number of spikes was in $(10, 20]$ then the pattern was classified to the second class. Otherwise, the input pattern was assigned to the third class. The results of 100 runs on this dataset are reported in Table II. For the algorithms employed to train the multilayer perceptrons the following set of values were used. The learning rate was set to 0.6 for all three methods, while the momentum term appearing in MBP and SMBP was equal to 0.6. The maximum number of generations was 100 for PARDE trained SNNs. For the Backpropagation family methods the maximum number of iterations was 1000. In this test problem the classification ability of the PARDE trained SNNs compares favorably to that of multilayer perceptrons in terms of both mean performance and standard deviation.

TABLE II
IRIS PROBLEM TEST SET CLASSIFICATION ERROR (%)

Algorithm	Mean	Stdev	Max	Min
BP	5.24	0.51	8	2.66
MBP	5.70	1.21	10.66	4
SMBP	5.2	0.48	6.66	4
SNN with PARDE	4.73	0.52	6.75	2.70

V. CONCLUSION

This paper investigates the application of a parallel evolutionary algorithm, namely parallel Differential Evolution, on the problem of supervised training of feedforward spiking neural networks. Spiking neural networks constitute a new form of connectionist model. These networks differ from traditional models in that spiking neurons communicate via delayed spikes. Thereby, information in spiking nets is propagated by the timing of individual spikes. The experimental

results on three well–known classification problems indicate that the proposed approach can produce networks having a generalization ability comparable to that of standard multilayer perceptrons trained through gradient descent based algorithms. In a future correspondence we intend to apply an evolutionary scheme to adjust not only the weights of the network but also the parameters of the spike–response model.

ACKNOWLEDGMENT

This work was supported in part by a "Karatheodoris" research grant, awarded by the Research Committee of the University of Patras, and the "SOCIAL (IST–2001–38911)" research program.

REFERENCES

[1] S.M. Bohte, H.La Poutr, and J.N. Kok, *Spikeprop: Error– backpropagation for networks of spiking neurons*, Neurocomputing **48** (2002), 17–37.

[2] F. Crick and C. Koch, *Towards a neurobiological theory of conciousness*, Seminars in the Neuroscience **2** (1990), 263–275.

[3] W. Gerstner, *Spiking neurons*, Pulsed Neural Networks (W. Maass and C. M. Bishop, eds.), Cambridge, MA: MIT Press, 1999, pp. 3–54.

[4] K. Hornik, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), 359–366.

[5] A. Jahnke, U. Roth, and T. Schoenauer, *Digital simulation of spiking neural networks*, Pulsed Neural Networks (W. Maass and C. M. Bishop, eds.), Cambridge, MA: MIT Press, 1999, pp. 237–258.

[6] W. Maass, *Fast sigmoidal networks via spiking neurons*, Neural Computation **9** (1997), 279–304.

[7] ———, *Networks of spiking neurons: The third generation of neural network models*, Neural Networks **10** (1997), no. 9, 1659–1671.

[8] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis, *Effective backpropagation training with variable stepsize*, Neural Networks **10** (1997), no. 1, 69–82.

[9] ———, *Increasing the convergence rate of the error backpropagation algorithm by learning rate adaptation methods*, Neural Computation **11** (1999), no. 7, 1769–1796.

[10] Z. Michalewicz and D. B. Fogel, *How to solve it: Modern heuristics*, Springer, 2000.

[11] V.P. Plagianakos and M.N. Vrahatis, *Training neural networks with threshold activation functions and constrained integer weights*, IEEE International Joint Conference on Neural Networks (IJCNN'2000) (Como, Italy), 2000.

[12] ———, *Parallel evolutionary training algorithms for 'hardware–friendly' neural networks*, Natural Computing **1** (2002), 307–322.

[13] L. Prechelt, *Proben1: A set of neural network benchmark problems and benchmarking rules*, Tech. Report 21/94, Fakultät für Informatik, Universität Karlsruhe, 1994.

[14] H.-P. Schwefel, *Evolution and optimum seeking*, Wiley, New York, 1995.

[15] R. Storn, *System design by constraint adaptation and differential evolution*, IEEE Transactions on Evolutionary Computation **3** (1999), 22–34.

[16] R. Storn and K. Price, *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces*, Journal of Global Optimization **11** (1997), 341–359.

[17] S. J. Thorpe and M. Imbert, *Biological constraints on connectionist models*, Connectionism in Perspective (R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, and L. Steels, eds.), Elsevier, 1989.

[18] H. White, *Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings*, Neural Networks **3** (1990), 535–549.