A New Supervised Learning Algorithm Based on Genetic Inheritance for Spiking Neural Networks

A New Supervised Learning Algorithm Based on Genetic Inheritance for Spiking Neural Networks

Jie Yang, Ning Wang and Tingting Pan

School of Mathematical Sciences, Dalian University of Technology, Dalian, Liaoning, China.
Email: yangjiee@dlut.edu.cn

Abstract. Spiking neural networks (SNNs) can perform complex spatio-temporal information computations in precise temporal coding. These networks differ from previous models in that spiking neurons convey information by time rather than rate of spikes. Most existing training algorithms are based on gradient descent with inherent defects, such as local optimum and over-fitting. In this paper, we investigate the performance of the Genetic Algorithm Involving Mechanism of Simulated Annealing, as a supervised training algorithm for SNNs. The key idea is to adopt global search, which effectively avoid local optima and over-fitting. According to the experiment results, this approach has higher accuracy than other learning algorithms on well-known classification problems.

1. Introduction

SNNs fall into the third generation of artificial neural network models, increasing the level of realism in a neural simulation [1]. It is theoretically shown that SNNs, where coding information resides in individual spike trains, are computationally more powerful than neural networks with sigmoid activation functions [2]. The main motivation behind the SNNs model is the fact that computation in the brain is primarily carried out by spiking neurons. In addition to the neuronal synaptic state, SNNs incorporate time into their operating model.

The Error Backpropagation (BP) algorithm based on gradient descent is a general method of training Artificial Neural Networks (ANNs). The first BP algorithm for multi-layer feed forward spiking neural network is called SpikeProp algorithm [3]. After that, lots of BP family methods were applied to train SNNs. [4] [5] There are other methods training networks by adjusting the internal structure of SNNs. The normalized perceptron based learning rule trains only the selected misclassified time points and the target ones [6]. In [7], there is an unary spiking circuit for a serial multiplier with variable dendritic delays. The application of the Widrow-Hoff learning rule to train networks of multiple spiking neurons is proposed [8]. For large SNNs, NeuCube SNNs architecture is used to train [9].

Genetic algorithm (GA) is a kind of computational algorithm to solve the optimization algorithm in mathematics, which belongs to evolutionary algorithms. It was originally developed learning from some phenomena in evolutionary biology, including inheritance, mutation, natural selection, and hybridization, etc. GA is usually implemented as a computer simulation. For an optimization problem, a certain number of candidate solutions (called individuals) can be abstractly expressed as chromosomes, allowing the population to evolve to better solutions [10]. The application of GA is also very extensive. The nondominated sorting genetic algorithm II was presented to find much better spread of solutions and better convergence in solving multi-objective problems [11]. And there is an algorithm based on GA which can guarantee a rigorously proven upper bound on its optimization time [12].

In the present paper, we consider the Genetic Algorithm Involving Mechanism of Simulated Annealing (SAGA) to the problem of supervised training of SNNs. The threshold function is approximated linearly to get around of the discontinuous nature of spiking neurons. The paper is organized as follows; Section 2 provides a brief introduction to the neural network structure and spike-response model. Section 3 describes the genetic algorithms involving mechanism of simulated annealing. The experimental results are reported and discussed in Section 4. The paper ends with conclusions and ideas for future research.

2. Network Architecture

The spiking nature of biological neurons has recently led to explorations of the computational power associated with temporal information coding in single spikes. The structure of SNN is similar to traditional neural network, except the synaptic delay and the number of synaptic terminals between of neurons in adjacent layers [13]. This structure is not a single wire connected between the neurons, but a complex multi-line network connection. The specific neural network structure is shown in figure 1. The **I**, **H** and **O** in part ***A*** respectively represent the input layer, the hidden layer and the output layer. Part ***B*** demonstrates connection between any two neurons i and j in adjacent layers.

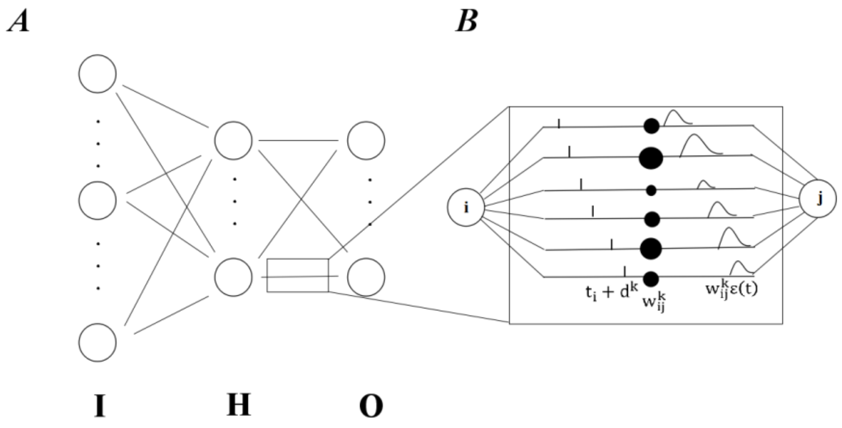

Figure 1. A) Feed-forward spiking neural network. B) Connection consisting of multiple delayed synaptic terminals.

In [14] Gerstner introduced the spike-response model (SRM) to describe the contact between the input spiking and the interval state variable. This model can be adapted to reflect the dynamics of different spiking neurons if proper spike response functions are selected. In the context of spiking neural network, the current activation level (modeled as some differential equation) is normally considered to be the neurons' state, with incoming spikes pushing this value higher, and then either firing or decaying over time.

Consider a neuron j, having a set D_j of immediate feedforward pre-synaptic neurons, receiving a set of spikes with firing times t_i, i ∈ D_j. It is assumed that any neuron can generate at most one spike during the simulation interval and discharge when the internal state variable reaches a threshold. The dynamics of the internal state variable $x_j(t)$ are described by the following function

$$x_j(t) = \sum_{i \in D_j} w_{ij} y_j(t) \tag{1}$$

where ω_{ij} is synaptic weight and $y_j(t)$ is the effect of a feedforward neuron on state variable.

Consider the synapses' every connection, it is assumed that there are m synaptic terminals between two neurons, the effect of a single synaptic terminal on the state variables is recorded as $y_j^k(t)$. The neuron j will receive spikes, which are released by all neurons of last layer. The dynamic process of interval state variables x_j affected by all neurons is represented by the following equation

$$x_j(t) = \sum_{i \in D_j} \sum_{k=1}^{m} w_{ij}^k y_j^k(t) \tag{2}$$

The post synaptic potential $y_j^k(t)$, t_i and d^k have the following functional relationship

$$y_j^k(t) = \varepsilon\left(t - t_i - d^k\right) \tag{3}$$

where t_i is the excitatory time of the former neuron and d^k is synaptic delay of synaptic terminals. The delay d^k of a synaptic terminal k is defined as the difference between the firing time of the pre-synaptic neuron and the time when the post-synaptic potential starts rising.

The spike response function ε(t), comes from the α function, is given by

$$\varepsilon(\mathrm{t}) = \frac{\mathrm{t}}{\tau} \mathrm{e}^{\frac{1-\mathrm{t}}{\tau}}, \tag{4}$$

where τ is the membrane potential decay time constant, that describes the rise and decay of synaptic potential. It is assumed that $\varepsilon(\mathrm{t}) = 0$ for $\mathrm{t} \leq 0$.

3. Genetic Algorithm Involving Mechanism of Simulated Annealing

The SAGA is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of Evolutionary Algorithm (EA). SAGA is commonly used to generate high-quality solutions to optimization and search problems relying on bio-inspired operators such as mutation, crossover and selection [15]. Simulated annealing is a meta-heuristic to approximate global optimization in a large search space. Involving mechanism of simulated annealing can prevent Genetic Algorithm (GA) premature and make its global search ability better.

To apply the SAGA to train SNNs, we start with a population having a specific size P, called parental generation consisting of P, D-dimensional weight vectors, and evolve them over time. P is fixed throughout the training process. All the weights are randomly initialized in the interval$[0, 1]$, following the uniform probability distribution. At each iteration, called generation, all the individuals in the population are evolved independently in parallel, until a genetic operation is decided. Genetic manipulation includes selection, crossover, and mutation. Our selection operation bases on the value of fitness by the roulette rules. Two-point crossover method is used in the crossover operation

$$\mathrm{I}_1 = \alpha \mathrm{I}_2 + (1-\alpha)\mathrm{I}_1, \mathrm{I}_2 = \alpha \mathrm{I}_1 + (1-\alpha)\mathrm{I}_2, \tag{5}$$

where I_1, I_2 are the selected individuals in the population and α is randomly generated in the interval $[0, 1]$. Mutation operation takes a simple single point method. After the evolution of the best individuals, the offspring population continues to evolve as before.

The emphasis of SAGA is fitness function, but for randomly generated population we can only calculate the error function of each individual. This paper presents a new type of relationship formula about fitness value and the sum of squared errors

$$\mathrm{F(i)} = e^{\left(\frac{E(i) - E_{min}}{E_{max} - E_{min}}\right)^{-2}} \tag{6}$$

where E_{min} is the minimum value of the error function in this generation, E_{max} is the maximum value. It makes the individual with greater error less adaptive and thus be eliminated, improving the efficiency of genetic algorithm.

Different from standard genetic algorithm (SGA), the value will be used to calculate the cost function value f after the new individual in the offspring generation has calculated the error function value E, population size is SamNum and the smallest error of the previous generation is defined as be

$$\mathrm{f(i)} = \frac{\mathrm{E(i) - be}}{\mathrm{SamNum}}, \tag{7}$$

The new individual is accepted with a certain probability related to the cost function value.

4. Experimental Results

For the test problems considered, we made no effort to tune the cross probability and mutation probability, Cr and Mr respectively, to obtain optimal or at least nearly optimal convergence speed. Instead, the fixed values $\mathrm{Cr} = 0.94$ and $\mathrm{Mr} = 0.05$, were used. The weight population were initialized with random numbers in the interval $[0, 1]$. Regarding total population size, experimental results have shown that a good choice is $30 < \mathrm{P} < 50$. 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 P render the algorithm inefficient and more generations are required to converge to the minimum. The simulated annealing initial temperature was set to 1 and the termination temperature was set to 0.1^{30}, current temperature, $T_{now} = T_{before} \times 0.999$.

Regarding the set of constants that are employed in the equations that describe the spike-response model equation (1) - equation (4), the following values were used. The value of the post synaptic potential threshold v was set to 5 and the number of synapses between two neurons m was set to 16, the time constant, $\tau = 7$. These values of parameters set were selected experimentally. Proper fine-tuning of these parameters can further improve the performance of the SNNs.

4.1. The Exclusive-OR Problem with Gaussian Disturbance

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 $3 - 4 - 1$ SNN has been used for the simulations. In order to make the experiment more comparable, this paper, replicated original data into 160 samples, then added Gaussian disturbance to the input pattern $\{1,1\}$ of them.

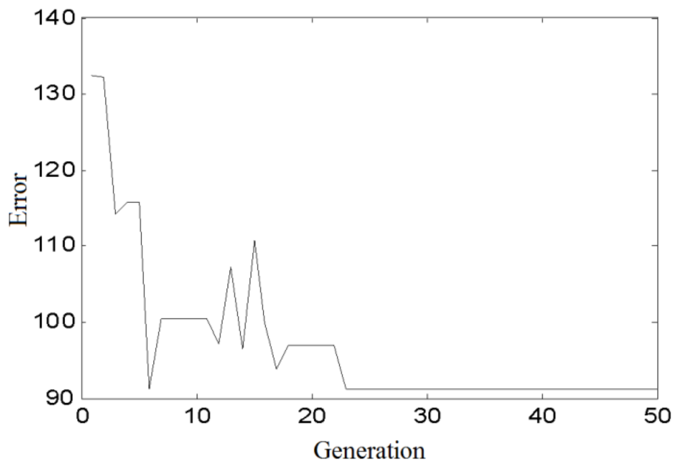

Figure 2. SNN with SAGA for the XOR problem with Gaussian disturbance.

In the SNNs, all information is encoded and processed with precise spike time. To encode the data of this problem in spike times the values of the input variable were normalized in the range $[0, 10]$. The normalized values represent the number of uniformly distributed spikes over the simulation period. In this simulation, SAGA iterative process depicted in figure 2. The performance of the trained SNNs is compared with that of SNNs trained through BP algorithm and SGA. The maximum number of generations was 50 for SAGA trained SNNs. The learning rate was set to 0.1 and the maximum number of iterations was 1000 for BP algorithm. The hyperparameters of SGA were same as SAGA. We divided the data sets into groups A and B on average, respectively, to calculate their classification error rate. The results of 100 independent experiments for the Gaussian Perturbation XOR problem are exhibited in table 1. The mean error of group B of the SNNs trained with the BP algorithm is 11.62%. With the SAGA the error reduces to 7.75%.

Table 1. XOR problem with Gaussian disturbance test set classification error (%).

Algorithm	Group	Mean	Max	Min
SNN with BP	A	8.00	8.75	7.50
	B	11.62	12.50	11.25
SNN with SGA	A	8.25	8.75	7.50
	B	9.75	11.25	7.50
SNN with SAGA	A	7.55	8.75	6.75
	B	7.75	8.75	6.25

4.2. 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 atype 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, 13] the input pattern was classified to the first class, if the number of spikes was in (13, 16] then the pattern was classified to the second class. Otherwise, the input pattern was assigned to the third class. In this simulation, SAGA iterative process depicted in figure 3. These algorithms can achieve higher accuracy in more complex network structures.

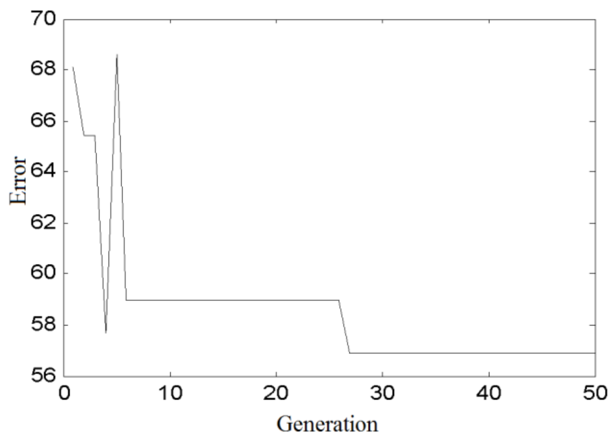

Figure 3. SNN with SAGA for the iris problem

Table 2. Iris problem test set classification error (%).

Algorithm	Mean	Max	Min
BP	5.24	8.00	2.66
MBP	5.70	10.66	4.00
SMBP	5.20	6.66	4.00
SNN with PARDE	4.73	6.75	2.70
SNN with SAGA	4.67	6.67	3.33

The results of 100 runs on this dataset are reported in table 2. The results of other algorithms in Table 2 come from Spiking Neural Network Training Using Evolutionary Algorithms [16]. In this test problem the classification ability of the SAGA trained SNNs compares favorably to that of gradient descent and differential evolution in terms of mean performance.

5. Conclusions

This paper investigates the SAGA on the problem of supervised training of feed forward SNNs. The experimental results on two well-known classification problems indicate that the proposed approach can produce networks having generalization ability comparable to that of standard multilayer perceptrons trained through gradient descent based algorithms. In a future correspondence we intend to apply genetic operation to adjust not only the weights of the network but also the parameters of the spike-response model. The crossover probability and mutation probability in SAGA will be adaptive, and the acceptance probability of new individuals in simulated annealing will be further optimized.

6. References

[1] Maass and Wolfgang 1997 Networks of Spiking Neurons: The Third Generation of Neural network models (England: Neural Networks) 10 (9) 1659 – 71
[2] Maass and Wolfgang 1996 Lower Bounds for the Computational Power of Networks of Spiking Neurons (United States: Neural Computation) 8(1) 1– 40
[3] Bohte S M Kok, Joost N and La Poutré H 2002 Error-backpropagation in Temporally Encoded Networks of Spiking Neurons (Netherlands: Neurocomputing) 48(1) 17-37
[4] Jianguo Xin M J and Embrechts M J 2001 Supervised Learning with Spiking Neural Networks (Washington: International Joint Conference on Neural Networks) 3 177-1777
[5] Yan X, Xiaoqin Z, Lixin H and Jing Y 2013 A Supervised Mult-spike Learning Algorithm Based on Gradient Descent for Spiking Neural Networks (England: Neural Networks) 43(4) 99 –113
[6] Xiurui X, Hong H, Guisong Liu and Malu Zhang 2017 Efficient Training of Supervised Spiking Neural Networks via The Normalized Perceptron Based Learning Rule (Netherlands: Neurocomputing) 241 152–163
[7] Carlos D, Giovanny S, Gonzalo D, Mariko N and Hector P 2016 An Efficient Hardware Implementation of A Novel Unary Spiking Neural Network Multiplier with Variable Dendritic Delays (Netherlands: Neurocomputing) 189 130 – 4
[8] Ammar M, Stefan S, Satoshi M and Nikola K 2013 Training Spiking Neural Networks to Associate Spatio-temporal Input-output Spike Patterns (Netherlands: Neurocomputing) 107(4) 3–10
[9] Anne A, Neelava S and Nikola K 2016 Which Method to Use for Optimal Structure and Function Representation of Large Spiking Neural Networks: A Case Study on The Neucube Architecture (2016 International Joint Conference on Neural Networks) 1367-72
[10] David E G 1989 Genetic Algorithm in Search, Optimization, and Machine Learning (New Jersey: Addison-Wesley)
[11] Deb K, Agrawal S, Pratap A and Meyarivan T 2000 A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II (Parallel Problem Solving from Nature PPSN VI) 1917 849–58
[12] Corus D and Lehre P K 2018 Theory Driven Design of Efficient Genetic Algorithms for A Classical Graph Problem
[13] Ghosh-Dastidar S and Adeli H 2009 Spiking Neural Networks (United States: International Journal of Neural Systems) 19(4) 295–308
[14] Gerstner W 2001 Chapter 12 A Frameworks for Spiking Neuron Models: The Spike Response Model (Handbook of Biological Physics) 4 469–516
[15] Kalyanmoy D 1999 An Introduction to Genetic Algorithms (India: Sadhana) 24(4) 293-315
[16] Pavlidis N G , Tasoulis O K, Plagianakos V P, Nikiforidis G and Vrahatis M N 2005 Spiking Neural Network Training Using Evolutionary Algorithms (England: Neural Networks) 4 2190–4