Developing Architectures of Spiking Neural Networks by Using Grammatical Evolution Based on Evolutionary Strategy

Andrés Espinal[1], Martín Carpio[1], Manuel Ornelas[1], Héctor Puga[1], Patricia Melín[2], and Marco Sotelo-Figueroa[1]

[1] Instituto Tecnológico de León, Av. Tecnológico S/N. León, Gto, México
andres.espinal@itleon.edu.mx
[2] Instituto Tecnológico de Tijuana, Calz. del Tecnológico S/N. Tijuana, B.C., México

Abstract. The Artificial Neural Networks (ANNs) have been used for solving problems in many theoretical and practical areas. Advances on the field of ANNs have derived in Spiking Neural Networks (SNNs); which are considered as the third generation of ANNs. SNNs receive/send the information by timing of events (spikes) instead by the spike rate; as their predecessors do. Although SNNs are capable to solve some functions with fewer neurons than networks of previous generations, there aren't rules to set the architecture of any kind of ANN for solving a specific task; usually the architecture is set empirically based on the designer's experience and the neural network's performance over the problem. Recently, metaheuristic algorithms are being implemented to optimize some aspect on ANNs such as weight, connections and even the architecture. This work proposes a generic framework for automatic construction of Fully-Connected Feed-Forward Spiking Neural Networks through an indirect representation by means of Grammatical Evolution (GE) based on Evolutionary Strategy (ES) algorithm. Two well-known benchmarks datasets of pattern recognition were used for testing the proposal of this paper.

1 Introduction

Artificial Neural Networks (ANNs) are mathematical models inspired in biology, which have been successfully applied to solve problems in fields such as pattern recognition, segmentation, regression, etc. Advances on ANNs field have derived in Spiking Neural Networks (SNNs), which are considered as the third generation of ANNs [17]. The SNNs are formed by spiking neurons, which deal with information encoded in timing of events (spikes), are computationally stronger than sigmoid neural networks [16]; in fact, there is evidence that fewer spiking neurons are required for solving some functions than neurons of previous generations [17].

Several topologies have been proposed for ANNs, which are defined around three aspects: computing nodes (neuron models), communications links (synapse connections) and message types (coding schemes) [13]. Although the aspects for

J.F. Martínez-Trinidad et al. (Eds.): MCPR 2014, LNCS 8495, pp. 71–80, 2014.

defining an ANN with a particular topology can be chosen, there isn't a rule to set how many computing nodes are required to solve an specific task, this last is an important criteria due to the fact that they have a strong impact on ANNs' performance [5].

Recently, methauristic algorithms have been used for dealing with the problem designing ANNs for solving some given problem [5] [22]. Metaheuristics can design ANNs by either direct or indirect representation; with a direct representation the algorithm tries to modified some aspects of pre-set ANNs [7] in order to improve the performance of it and with an indirect representation, aspects of an ANN are codified, and the algorithm tries to design with that an ANN for a given problem [3].

In this work a generic framework is presented for designing SNNs to solve pattern recognition problems. The paper is organized as follows: section 2 gives all the fundamentals required for this work, section 3 explains the proposal of this work, in section 4 some experiments and their results are explained and showed, and finally, section 5 gives the conclusions of this work and proposes future work.

2 Backgrounds

2.1 Spiking Neural Networks

The Spiking Neural Networks (SNNs) are formed by the interconnection of spiking neurons, which handle the information by timing of events (spikes). When a SNN is applied to solve pattern recognition problems, original patterns can not be fed into the SNN; they need to be transformed as spikes in an interval of time by using an encoding scheme. Several encoding schemes have been proposed, such as the Gaussian Receptive Fiels (GRFs); this encoding scheme has been extensively used in [2], [1], [21]. Basically, this encoding scheme requires for encoding a variable, m neurons (with Gaussian functions) used for covering the whole range of the variable, γ as a coefficient for setting the width of Gaussian functions and the encoding simulation time τ . For detailed information about GRFs and other encoding schemes authors suggest to check [12].

After transforming original vectors into spikes in an interval of time, they can be fed into the SNN. The architecture of the SNN can vary depending on the kind of the problem to solve. This work focuses on SNNs with architecture known as Fully-Connected Feed-Forward as used in [2], [1] and [11]. There are several spiking neuron models that can be implemented in a SNN, this work use the Spike Response Model which is detailed in next section.

Spike Response Model

The Spike Response Model (SRM) [8] [9] is an approximation of the dynamics of the integrate-and-fire neuron. For this work, the spiking neurons use the time-to-first-spike as coding scheme for sending/receiving messages. Due to the coding scheme being used, a reduced version of the SRM is implemented, which has been used in [2], [1] and [21].

The reduced SRM is defined according [1] as follows. Let us consider that a neuron j has a set Γ_j of immediate predecessors called presynaptic neurons and recives a set of spikes with firing times t_i, $i \in \Gamma_j$. Neurons fire when their state variable $x(t)$, called membrane potential, reaches a certain threshold θ. The internal state of a neurons is determined by eq. (1), where w_{ji} is the synaptic weight to modulate $y_i(t)$, which is the unweighted postsynaptic potential of a single spike coming from neuron i and impinging on neuron j.

$$x_j(t) = \sum_{i \in \Gamma_j} w_{ji} y_i(t) \tag{1}$$

The unweighted contribution $y_i(t)$ is given by eq. (2), this uses a function $\varepsilon(t)$; which describes the form of the postsynaptic potential and its input parameter are formed by the next three values: t is the current time, t_i is the firing time of the presynaptic neuron i and d_{ji} is the associated synaptic delay.

$$y_i(t) = \varepsilon(t - t_i - d_{ji}) \tag{2}$$

The form of the postsynaptic potential $\varepsilon(t)$ is given by eq. (3), the function has a τ parameter, that is the membrane potential time constant defining the decay time of the postsynaptic potential.

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{1-\frac{t}{\tau}} & \text{if } t > 0 \\ 0 & \text{else} \end{cases} \tag{3}$$

Each neuron fires once at most, the firing time t_j of neuron j is determined as the first time the state variable crosses the threshold from below.

2.2 Evolutionary Strategy

The Evolutionary Strategies (ES) [19], deal natively with problems in real domain. In [1] was designed a Self-Adaptive ES originally for training SNNs, but it could be used to solve other optimization problems. In this ES each population member consists of n-dimensional vectors. The population at any given generation g is denoted as $P(g)$. Each individual is taken as a pair of real-valued vectors, (x_i, η_i), where x_i's are objective variables which depend of the optimization problem, and η_i's are standard deviations for mutations. Each individual generates a single offspring (x'_i, η'_i), where each variable $x'_i(j)$ of the offspring can be randomly defined by either eq. (4) (local search) or eq. (5) (global search) and the standard deviation for mutation of the offspring is defined by eq. (6).

$$x'_i(j) = x_i(j) + \eta_i(j) N_j(0,1) \tag{4}$$

$$x'_i(j) = x_i(j) + \eta_i(j) \delta_j \tag{5}$$

$$\eta'_i(j) = \eta_i(j) exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{6}$$

Algorithm 1. Self-adaptive ES

1: Generate the initial population of μ individuals.
2: Evaluate the fitness score for each individual$(x_i, \eta_i), i = 1, \ldots, \mu$ of the population based on the fitness function.
3: **while** the maximum iteration is not reached **do**
4: Each parent (x_i, η_i) generates a single offspring (x'_i, η'_i)
5: Calculate the fitness of each offspring $(x'_i, \eta'_i), i = 1, \ldots, \mu$.
6: Generate a new population $P(g)$ using tournament selection and elitism to keep track of the best individual at each generation
7: **end while**

where:

- $N(0, 1)$ denotes a normally distributed one dimensional random number with $\mu = 0$ and $\sigma = 1$.
- $N_j(0, 1)$ indicates that the random number is generated anew for each value of j.
- δ_j is a Cauchy random variable, and it is generated anew for each value of j (Scale = 1).
- Factor $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$
- Factor $\tau' = \frac{1}{\sqrt{2n}}$

The Self-adaptive ES is presented in the algorithm 1.

2.3 Grammatical Evolution

Grammatical Evolution (GE) [20] is a grammar-base form of Genetic Programming (GP) [15]. GE joins the principles from molecular biology, which are used by the GP, and the power of formal grammars. Unlike GP, the GE adopts a population of lineal genotypic integer strings, or binary strings, which are transformed into functional phenotypic through a genotype-to-phenotype mapping process, this process is also know as Indirect Representation [6]. This transformation is governed through a Backus Naur Form grammar (BNF). Genotype strings are evolved with no knowledge of their phenotypic equivalent, only use the fitness measure.

Eventhough the GE uses the Genetic Algorithm (GA) [4, 10, 20] as search strategy it is possible to use another search strategy like the Particle Swarm Optimization, called Grammatical Swarm (GS) [18]. In the GE each individual is mapped into a program using the BNF.

Mapping Process
When approaching a problem using GE, initially a BNF grammar must be defined. This grammar specifies the syntax of desired phenotypic programs to be produced by GE. The development of a BNF grammar also affords the researcher the ability to incorporate domain biases or domain-specific functions.

A BNF grammar is made up of the tuple N, T, P, S; where N is the set of all non-terminal symbols, T is the set of terminals, P is the set of production rules that map $N \rightarrow T$, and S is the initial start symbol where $S \in N$. Where there are a number of production rules that can be applied to a non-terminal, a "|" (or) symbol separates the options.

Using the grammar as the GE input, eq. (7) is used to choose the next production based-on the non-terminal symbol.

$$Rule = c\%r \tag{7}$$

where c is the codon value and r is the number of production rules available for the current non-terminal.

An example of the mapping process employed by GE is shown in Figure 1.

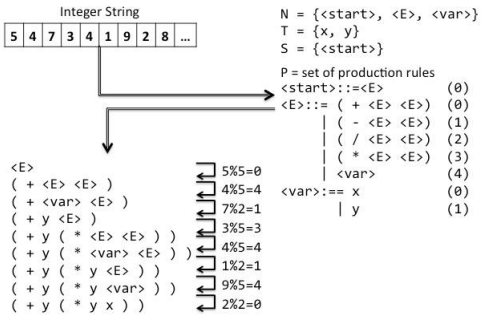

Fig. 1. An example a transformation from genotype to phenotype using a BNF Grammar. It begins with the start symbol, if the production rule from this symbol is only one rule, then the production rule gets instead of the start symbol, and the process begins to choose the productions rules base on the current genotype. It is taking each genotype and the non-terminal symbol from the left to realize the next production using eq. (7) until all the genotypes are mapped or there aren't more non-terminals in the phenotype.

3 Proposal

This paper proposes a generic framework for designing architectures of SNNs to solve pattern recognition problems. The proposed framework (see fig. 2) requires as inputs: training and testing sets from the pattern recognition problem to solve, a BNF grammar as indirect representation of the architectures of SNN and the output firing times for each class in the dataset. The indirect representation is defined by Grammar 1.1, which can specify a Fully-Connected Feed-Forward SNN. The specified SNN architecture will be defined by the number of GRFs for encoding patterns, one or several hidden layers with their respective number of neurons and a single output neuron in the output layer.

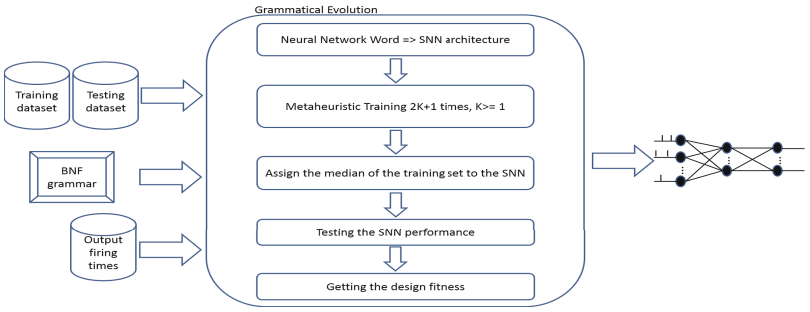

Fig. 2. Generic Framework Diagram for designing SNNs architectures

$$
\begin{aligned}
\langle \text{network} \rangle &::= \langle \text{receptiveFields} \rangle - \langle \text{layers} \rangle \\
\langle \text{receptiveFields} \rangle &::= \langle \text{digit} \rangle \\
\langle \text{layers} \rangle &::= \langle \text{layer} \rangle \mid \langle \text{layer} \rangle, \langle \text{layers} \rangle \\
\langle \text{layer} \rangle &::= \langle \text{digit} \rangle \\
\langle \text{digit} \rangle &::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Grammar 1.1. BNF grammar for SNN design based on the first approach proposed in [3]

The design process is carried out by the GE based on ES (the used ES is that proposed in [1]), which basically by means of the BNF grammar and the mapping process creates architectures of SNNs candidates. The GE evaluates the quality of its candidates by using a fitness function and evolves them trying to improve the candidates by minimizing their fitness values.The eq. (8) is proposed as a fitness function for looking over the search space of SNNs architectures.

This fitness function of design requires of three stages to be calculated. The first stage consists in training $2K + 1$ (with $K \geq 1$) times the candidate architecture using the input training set, the desired firing output times and the supervised-learning based on ES for training SNNs used in [1]. The second stage consist in assigning to the candidate architecture, the median training process from the training set generated in the first stage. Finally, the third stage consist in obtaining the performance of the candidate architecture of SNN (trained) over unseen patterns using the input testing set.

$$f = \frac{\#feat * \#GRFs}{9 * \#feat} + \sum_{i=1}^{\#nhl_i} i * \frac{\#nhl_i}{9} + ((1 - perfTest) * 100) \qquad (8)$$

where:

- $\#feat$ is the number of original features.
- $\#GRFs$ is the number of GRFs selected by the GE.

- $\#nhl_i$ is the number of neurons in the $i-th$ hidden layer set by the GE.
- $perfTest$ is the performance of the trained SNN with the candidate architecture over the input testing set.

The fitness function of design showed in eq. (8) was designed for trying to find compact SNNs architectures with a high classification performance over unseen patterns in the search space of architectures.

4 Experiments and Results

The proposed framework for developing architectures of SNNs was tested over two well-known benchmark datasets form the *UCI Machine Learning Repository*:

Iris plant: The iris plant dataset contains 3 classes (*Iris Setosa*, *Iris Versicolour* and *Iris Virginica*). One class is linearly separable from the other 2; the latter are not linearly separable. There are 50 instance patterns for each class. Each pattern is described by 4 attributes.

Wine: The wine dataset contains 3 classes. There are 59, 41 and 48 instance patterns for classes 1, 2 and 3 respectively. Each patter is described by 13 attributes.

Usually in the pattern recognition area, a classifier is tested by using some accuracy estimation method such as K-folds cross validation [14]. In this work is used the leave-one-out version of this method, not to test a classifer but to obtain a test instances set for each benchmark dataset; each benchmark dataset was splitted with $K = 10$, giving a set of 10 test instance. For each test instance was designed a SNN using the proposed framework.

As both benchmark datasets have three classes, the same configuration was used through all their test instances; some parameters were chosen empirically, others were taken from the checked works for this paper and others parameters were defined by the framework. The configuration of the experiments is presented next.

GE based on ES. The parameters *size of population* $= 30$, *function calls* $= 600$ and *boundaries* $\in [0, 255]$.

Supervised-learning based on ES. The parameters *size of population* $= 30$, *function calls* $= 15000$, *weights* $\in [-1000, 1000]$ and *delays* $\in [0.1, 16]$.

GRFs. m could be from 1 to 9, which is defined by the framework. The parameters $\tau = 4$ *ms* and $\gamma = 1.5$.

SRM. The parameters $\theta = 1.0$ mV, $\tau = 9.0$ *ms*, *simulation start time* $= 5$ *ms* and *simulation end time* $= 19$ *ms*.

Output firing times for the classes 1,2 and 3 where 6.0 *ms*, 10.0 *ms*, 14 *ms* respectively.

The tables 1 and 2 show for each test instance the designed architectures and the performance of the trained SNN over the training and testing patterns. The last row shows an accuracy value of the obtained performance over training and testing patterns through all tests.

Table 1. Classification performance of the designed SNNs over training and testing patterns through 10 tests for Iris Plant Dataset

# test	Architecture	Performance training	Performance testing
1	$4(5)-3-1$	0.8740	1.0
2	$4(4)-7-1$	0.9481	1.0
3	$4(5)-2-1$	0.8592	1.0
4	$4(3)-7-1$	0.9111	1.0
5	$4(9)-6-1$	0.9185	1.0
6	$4(6)-6-1$	0.8666	1.0
7	$4(7)-6-1$	0.9407	1.0
8	$4(5)-3-1$	0.9777	1.0
9	$4(5)-5-1$	0.9333	1.0
10	$4(7)-4-1$	0.9333	1.0
	Accuracy	0.91625	1.0

Table 2. Classification performance of the designed SNNs over training and testing patterns through 10 tests for Wine Dataset

# test	Architecture	Performance training	Performance testing
1	$13(7)-3-1$	0.6037	0.6842
2	$13(4)-8-1$	0.8187	0.9444
3	$13(8)-6-1$	0.8000	0.8333
4	$13(7)-7-1$	0.7687	0.9444
5	$13(7)-9-1$	0.7750	0.8333
6	$13(5)-7-1$	0.7937	0.9444
7	$13(8)-3-1$	0.8062	0.8888
8	$13(4)-8-1$	0.7625	0.8333
9	$13(7)-8-1$	0.8757	0.8823
10	$13(4)-6-1$	0.9012	1.0
	Accuracy	0.79054	0.87884

The proposal was compared against an ANNs trained by means of Backpropagation algorithm only for the Iris Plant dataset. The ANN has an architecture of 4 neurons in the input layer, 7 neurons in the hidden layer and 3 neurons in the output layer; this ANN was trained for each test instance. The accuracy achieved for the SNN over the tests for known patterns was 0.91625 and 1.0 for unknown patterns, againts the accuracy achieved for the ANN for known patterns was 0.99407 and for unkown patterns was 0.96. This comparison shows that the proposal can design SNN with better generalization than an ANN with empirical design.

5 Conclusions

This work presents a generic framework for designing Fully-Connected Feed-Forward SNNs to solve pattern recognition problems no matter how many features represent the patterns in the dataset. The framework requires few parameters (population size for the GE based on ES and BNF grammar) to do the design process due that uses an ES, which is self-adaptive; the other parameters are dependant of the characteristics for the SNNs to be designed and they aren't cosider to be part of the framework.

Analyzing the BNF grammar, it was possible to define a fitness function for the GE based on ES. The fitness function looks for SNNs with compact architectures over the search space, which have acceptable performance for both known and unknown patterns. The resulting SNNs have a good generalization performance due that the fitness function besides that it tries to looks for compact architectures, it chooses architectures which have maximum performance for unknown patterns.

A remarkable advantage of the proposed framework is that it can handle different grammars as indirect representation for designing architectures of SNNs.

As future work, authors propose to define a grammar, which integrates additional inherent aspects of SNNs for their design. Besides to implement other metaheuristics for the optimization process and compare them.

Acknowledgments. The authors thank to Consejo Nacional de Ciencia y Tecnología (CONACyT) and *Instituto Tecnológico de León* (ITL) for the support to this research.

References

1. Belatreche, A.: Biologically Inspired Neural Networks: Models, Learning, and Applications. VDM Verlag, Saarbrücken (2010)
2. Bohte, S.M., Kok, J.N., LaPoutre, H.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48, 17–37 (2002)
3. De Mingo Lopez, L.F., Gomez Blas, N., Arteta, A.: The optimal combination: Grammatical swarm, particle swarm optimization and neural networks. Journal of Computational Science 3(1-2), 46–55 (2012)
4. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. SCI, vol. 194. Springer, Heidelberg (2009)
5. Ding, S., Li, H., Su, C., Yu, J., Jin, F.: Evolutionary artificial neural networks: A review. Artif. Intell. Rev. 39(3), 251–260 (2013)
6. Fang, H.-L., Ross, P., Corne, D.: A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 375–382. Morgan Kaufmann (1993)
7. Garro, B.A., Sossa, H., Vazquez, R.A.: Design of artificial neural networks using a modified particle swarm optimization algorithm. In: Proceedings of the 2009 International Joint Conference on Neural Networks, IJCNN 2009, pp. 2363–2370. IEEE Press, Piscataway (2009)

8. Gerstner, W.: Time structure of the activity in neural network models. Physical Review E 51(1), 738–758 (1995)
9. Gerstner, W., Kistler, W.: Spiking Neuron Models: Single Neurons, Populations, Plasticity. Cambridge University Press (2002)
10. Holland, J.: Adaptation in natural and artificial systems. University of Michigan Press (1975)
11. Hong, S., Ning, L., Xiaoping, L., Qian, W.: A cooperative method for supervised learning in spiking neural networks. In: CSCWD, pp. 22–26. IEEE (2010)
12. Johnson, C., Roychowdhury, S., Venayagamoorthy, G.K.: A reversibility analysis of encoding methods for spiking neural networks. In: IJCNN, pp. 1802–1809 (2011)
13. Judd, J.S.: Neural Network Design and the Complexity of Learning. Neural Network Modeling and Connectionism Series. Massachusetts Institute Technol. (1990)
14. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI, pp. 1137–1145 (1995)
15. Koza, J.R., Poli, R.: Genetic programming. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pp. 127–164. Kluwer, Boston (2005)
16. Maass, W.: Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons, pp. 211–217. MIT Press (1996)
17. Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks 10(9), 1659–1671 (1997)
18. O'Neill, M., Brabazon, A.: Grammatical differential evolution. In: International Conference on Artificial Intelligence (ICAI 2006), Las Vegas, Nevada. CSEA Press (2006)
19. Rechenberg, I.: Evolutions Strategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog (1973)
20. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–95. Springer, Heidelberg (1998)
21. Shen, H., Liu, N., Li, X., Wang, Q.: A cooperative method for supervised learning in spiking neural networks. In: CSCWD, pp. 22–26. IEEE (2010)
22. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87(9), 1423–1447 (1999)