Experiments on Neuroevolution and Online Weight Adaptation in Complex Environments

Francisco J. Gallego-Durán, Rafael Molina-Carmona, and Faraón Llorens-Largo

Departamento de Ciencia de la Computación e Inteligencia Artificial
{fgallego, rmolina, faraon}@dccia.ua.es,
Universidad de Alicante

Abstract. Neuroevolution has come a long way over the last decade. Lots of interesting and successful new methods and algorithms have been presented, with great improvements that make the field become very promising. Concretely, HyperNEAT has shown a great potential for evolving large scale neural networks, by discovering geometric regularities, thus being suitable for evolving complex controllers. However, once training phase has finished, evolved neural networks stay fixed and learning/adaptation does not happen anymore. A few methods have been proposed to address this concern, mainly using Hebbian plasticity and/or Compositional Pattern Producing Networks (CPPNs) like in Adaptive HyperNEAT. This methods have been tested in simple environments to isolate the effectiveness of adaptation from the Neuroevolution. In spite of this being quite convenient, more research is needed to better understand online adaptation in more complex environments. This paper shows a new proposal for online weight adaptation in neuroevolved artificial neural networks, and presents the results of several experiments carried out in a race simulation environment.

Keywords: Neuroevolution, Online Adaptation, Complex Environments

1 Introduction

Artificial Intelligence is always seeking to mimic human brain processes in one way or another. The more powerful and developed computers and algorithms are, the more we appreciate the intrinsic complexity and inherent generality of the human brain and its learning capabilities. Attempting to replicate the unique capabilities of the brains into software algorithms, the field of Artificial Neural Networks (ANNs) [13] came into live. The initial euphoria about what ANNs would be able to do soon vanished as the intrinsic complexities of neurons came along. Training ANNs is a difficult task, but even more difficult is finding effective and efficient topologies.

Neuroevolution [12][7] (i.e. the use of evolutive algorithms to search for ANNs topologies and weights) is one of the best known ways to generate and train complex, recurrent ANNs. One of the most prominent Neuroevolutive (NE) algorithms of the past decade was Neuroevolution of Augmenting Topologies (NEAT)

[9][7][10]which proposed a way of minimizing the problem of Competing Conventions and the use of complexification to traverse the search space looking for effective and efficient ANNs topologies. Yet, it remained very difficult to find and train large scale topologies of ANNs, which are believed to be necessary for complex tasks. Then Compositional Pattern Producing Networks (CPPNs) and Hyper-cube Based NEAT (HyperNEAT) [8][1][5] discovered a way to use NEAT as a form of indirect encoding for producing large scale ANNs with topological regularities (as in the human brain). Then on, HyperNEAT has shown great ability to produce large scale Neural Networks able to perform efficiently in complex domains where an intelligent controller is required.

However, as ANNs increase in complexity and capabilities, an important subject remains unsolved. Neuroevolution algorithms have shown as a great way to discover topologies and weight sets that exhibit complex behaviours but, once the training phase is over, the produced ANNs stay fixed from then on. This greatly limits the use of resulting ANNs to static domains or domains where variance is not a problem. In contrast, our brains show what is called Neural Plasticity, which enables them to learn and adapt constantly to changing situations. This means that further research on online learning and adaptation for ANNs is required, to let us develop systems able to perform well on changing domains. In this sense, adaptive Evolving-Substrate HyperNEAT (Adaptive ES-HyperNEAT) [6][5] has done a first step by adding Hebbian ABC Plasticity [3] as patterns of local rules to ANNs, and also using CPPNs to continuously adapt weights over time. Nevertheless, there is no theoretical or empirical evidence about the performance of these approaches in complex environments.

Our contribution in this paper is focused on two points: presenting a new proposal for online adaptation of previously evolved ANNs, and giving some empirical evidence on how these approaches perform in a complex environment. In section 2, a succinct background on previous developments that conform the base for our contribution is presented. Section 3 describes our proposal for online adaptation and its motivation. Experimental results are shown in section 4, along with a description of the simulation environment used. Finally, section 5 summarizes our conclusions and further work.

2 Background

Our contribution represents one more step in the way that has been followed by lots of researches previous to us. Next we describe some previous work on which our contribution is based.

2.1 Neuroevolution of Augmenting Topologies (NEAT)

NEAT is a direct-encoding neuroevolution algorithm with speciation that evolves populations of ANNs starting from the most simple possible topologies and increasingly complexifying them. This way of traversing the search space is aimed

at getting the simplest possible ANNs that solve a given task (i.e. the topologies with least possible hidden neurons). It does so by adding new neurons and links by means of mutation and protecting innovations through speciation (i.e. different species have their own evolutive niche, not competing with each other). The key concept that makes NEAT a powerful algorithm is innovations tracking. The algorithm maintains a registry of all the innovations that have happened across all the populations of the different epochs. Each new link or neuron that appears in a new position is given a registration Id. and, from then on, each individual that gets a neuron in the same position is referred with the same Id. This mechanism effectively tracks innovations and permits the creation of an effective crossover operator that overcomes part of the consequences of the Competing Conventions problem.

Although NEAT represented a great breakthrough, it suffered from the same problem that all direct-encoding algorithms suffer: they are inherently not scalable and not modular. For instance, closely observing the human brain, there are lots of regularities and patterns that repeat everywhere. If a direct encoding was to discover that topology, it should repeatedly discover each one of the regularities again and again. Direct Encoding has no mechanism for replicating structures or patterns of structures across the phenotype, because it is a direct low-level map between genes and neurons/links.

2.2 Compositional Pattern Producing Networks (CPPN)

Compositional Pattern Producing Networks (CPPNs) [11] were created as an indirect encoding scheme to overcome the impossibility of modularization and pattern repetition that NEAT had due to its direct encoding. CPPNs are a kind of networks similar to ANNs, but with an important difference: each node, instead of being a neuron, represents a mathematical function (e.g. sine, cosine, gaussian...). Therefore, a CPPN is a composition of functions that can produce outputs full of symmetries, patterns and regularities.

Describing this composition of functions as a network instead of a formal math composition, the model can profit from existing neuroevolutive algorithms to produce CPPNs. In particular, a modification of NEAT, called CPPN-NEAT can evolve increasingly complex CPPNs that are suitable for indirectly encoding links and weights of ANNs.

2.3 Hypercube-based NeuroEvolution of Augmenting Topologies

HyperNEAT[1] takes NEAT and CPPNs as indirect encoding scheme and produces large scale ANNs with regularities, patterns and symmetries. HyperNEAT takes a population of CPPNs as genotypes of the final large-scale ANNs, and uses CPPN-NEAT to evolve these genotypes. For a CPPN to produce an ANN, a geometric *substrate* is required. A substrate is a collection of nodes (i.e. neurons) placed in a N-dimensional space, thus having a vector of coordinates $\mathbf{x^i} = (\mathbf{x_1^i}, \mathbf{x_2^i}, ..., \mathbf{x_n^i})$ for each node i. Typically, in a 2D-space neurons would be scattered in $[-1, 1] \times [-1, 1]$. Once a substrate is defined, the next step is

to add links and weights between neurons. This is done iteratively querying the CPPN with the coordinates of each possible pair of neurons $(\mathbf{x^i}, \mathbf{x^j}) \forall i, j$, where the output value from the CPPN represents the weight of the link from $\mathbf{x^i}$ to $\mathbf{x^j}$.

Taking into account that links and their weights are produced as a function of the relative location of neurons in space, it follows that the resulting topology of the ANN is related to the actual geometry of the substrate. This is an interesting characteristic of HyperNEAT, because it can produce ANNs with the ability of understanding geometry relations in their inputs. For instance, if we think of a chess controller with 64 inputs (one for each square of the board), the CPPNs will produce ANNs with intrinsic knowledge of the board structure. This characteristic is not present in traditional ANNs, which have to discover this information by themselves during training phase.

2.4 Adaptive, Evolvable-Substrate HyperNEAT

The two most recent improvements on HyperNEAT address two important issues. On the one hand, HyperNEAT requires the user to design an a priori substrate. While this could be very interesting in some problem domains, it normally is a matter of concern. Most of the time there are no clues on how to design the substrate, how many neurons to use, how to distribute them, etc. For this issue, Evolvable Substrate HyperNEAT (ES-HyperNEAT) [5][6] has developed a way to automatically configure a suitable substrate. The main idea behind is to measure variance in the function that the CPPN encodes: spatial areas of high variance in the function are considered to encode more information and, therefore, to require more density of nodes. On the other hand, HyperNEAT produces trained ANNs that, like almost any other neuroevolution algorithm, do not learn and/or adapt outside the evolution phase. Therefore, produced ANNs have a sort of fixed, hardcoded behaviour that will not change even if it is required. In order to address this issue, weight adaptation has been added to HyperNEAT, encoded as a pattern of local rules that modulate each weight. Concretely, three alternatives have been explored in [4]: Hebbian plasticity, Hebbian ABC plasticity and a modification of CPPN to make them able to update weights at each iteration. These three approaches were tested in a T-Maze where there were two rewards that switched position sometimes. Two different experiments were set up: in the first one, there was one big reward and a small one that sometimes switched positions. In the second one, rewards were relative to a color graduation scheme, which was designed explicitly to be non linearly-separable. Simulated agents were required to traverse the T-Maze and find the best reward repeatedly, thus requiring online adaptation to the different reward schemes and changes. They found that standard Hebbian Plasticity was not able to adapt, whereas there was a trade-off between the other two alternatives, considering bests results for the modification of CPPNs but with higher computational costs, as CPPNs needed to be queried every time-step for each weight.

3 Proposed method for online weight adaptation

The approaches for ANNs online weight adaptation proposed by [4] are quite interesting and open up a new field for discussion. Results shown up to date encouraged us to further test with these approaches but in a more complex scenario, where online adaptation and learning is required. For our experiments we used a modification of The Open Racing Car Simulator (TORCS) game, as in the annual car racing competition [2]. The experimental setup and results are discussed in section 4.

The results in [4] have shown that the most promising approach seams to be the modification of CPPNs to accept pre and post synaptic activation information in order to output a new weight at each time-step. Thinking of this approach, it seams a reasonable hypothesis that good weight adaptation comes from non-linear functions (and, most probably, from continuous and derivable ones). However, thinking of the human brain, it is also reasonable to think that delays between activations, frequency and strength seam to be the most important factors in the modulation of neural connections. Therefore, adding up all these ideas, we hypothesized and constructed a new prospective model for weight adaptation. Our model considers that the updated weight of a link connecting neuron i to neuron j depends on the activations of i and j on the n previous time-steps. From these n previous time-steps we will consider the post-synaptic activity O of i and j (namely O^i and O^j) and the pre-synaptic activity I of j (I^j). The updated weight is the result of a relation between all these values at the n-th step (the latest one) and the mean of their previous values, pre-processed by three modulation functions $\psi_{\mathbf{m}}, m \in \{1, 2, 3\}$ and post-processed by the update-strength function ζ. This relation is expressed in equation 1.

$$\mathbf{w_{ij}} = \zeta_n\left(\frac{(n-1)\psi_1(O_n^i)\psi_2(I_n^j)\psi_3(O_n^j)}{\psi_1(\sum_{k=1}^{n-1} O_k^i)\psi_2(\sum_{k=1}^{n-1} I_k^j)\psi_3(\sum_{k=1}^{n-1} O_k^j)}\right) \tag{1}$$

This equation is the same for each link, with the exception of the parameter n, the modulation functions $\psi_{\mathbf{m}}$ and the update-strength function ζ. All this functions will be created by the CPPN as a weighted average of a predefined subset of the functions that the CPPN uses as internal nodes $\Omega = \{F_i\}$, at the time the phenotype is created. For this approach to work properly, CPPNs have been modified to include $4M$ new outputs, being $M = |\Omega|$. Each $M_k = \{m_{k1}, m_{k2}, ...m_{kM}\}$ set of parameters is used as coefficients to construct a weighted function, as in equation 2.

$$\psi_{\mathbf{k}}(\mathbf{x}) = \frac{\sum_{q=1}^{M} m_{kq} F_q(x)}{\sum_{q=1}^{M} m_{kq}} \tag{2}$$

4 Experimentation and results

One of our main aims in this work was to do empirical tests in complex environments in order to give more evidence on the previously existing methods, and

have initial measures and evidence to give us a hint on what to expect from our hypothesis (i.e. whether it has a probability of being good or it is plainly wrong). As we stated before, we have used a modification on the TORCS racing game as environment (see figure 1). We have set up a population of 200 individuals and have trained them to be able to drive alone in the circuit "CG Speedway number 1", which comes with the original game package. Drivers are evolved to drive alone until they drive fine enough to finish the track, and then no more evolution is carried and they enter the adaptive test phase. In this second phase, drivers are asked to drive the same circuit but with lots of opponents and starting from the last position. The problem requires the drivers to be able to adapt not to crash against opponents, just by using the local weight adaptation rules.

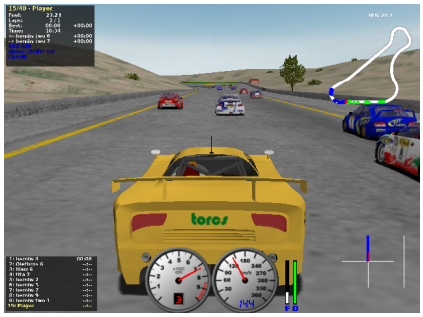

Fig. 1. A screenshot during a test of a previously trained driver. The driver receives 180.000 inputs (600×300) one from each pixel from the viewport in front.

Standard HyperNEAT was the algorithm chosen to evolve the ANNs, and a virtual first person camera, with a resolution of 600×300 pixels and 32 bits of color per pixel, was used as main input. The ANNs also had as input a status vector with this information: amount of damage done, current gear, current RPM, speed in x and y axis, and the 4 wheel rotation speeds. ANNs were required to output the values of acceleration, brake, gear up/down, and steer left/right. Therefore, the substrate was configured with 180.009 input neurons and 4 outputs. Based on experimental research, we set up 540.000 hidden nodes with recurrent connections enabled for CPPNs, and sine, gaussian, sigmoid, absolute value and linear as available activation functions. Neuron output range was set to $[-1, 1]$ and a CPPN output value less than 0.2 was considered 0 (no link). The functions contained in the Ω set where sine, gaussian and sigmoid. The compatibility threshold was set to 5.5, and the compatibility modifier to 0.3. Survival threshold within species was set to 20%, whereas drop-off age was set to 18 and target number of species to 7. There was a 4% chance of adding either a node or a link, and links had 50% chance of mutation.

After approximately 2500 epochs of evolution, drivers were able to satisfactory drive their cars to the finish line (just 1 lap) when asking them to drive

alone. From then on, evolution was finished and drivers were put in the same track, but together with 37 other drivers (standard fuzzy logic drivers coming with the TORCS package). Each driver was repeatedly tested 1500 times with each online adaptation method and population averages were taken (see figure 2). Final results depicted in figure 2 show that our proposal has a decent level

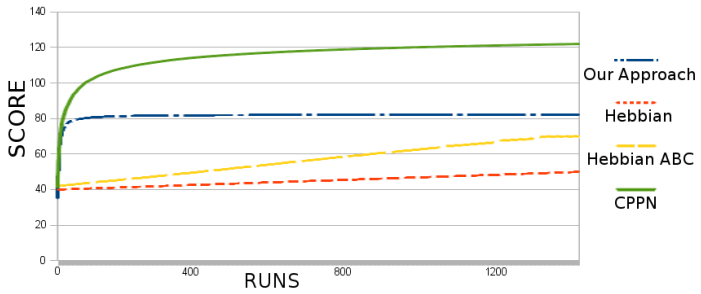

Fig. 2. Comparison of average results over 1500 runs of the track against 37 opponents. 100 points mark means that the driver arrived to the finish line in 70 seconds, more than 100 points means the driver arrived earlier.

of adaptation, whereas CPPN shows the greatest performance. Interestingly, Hebbian approaches show some kind of linear improvements, what seems quite unnatural. We tested both Hebbian approaches with more than 1500 repetitions of the track, (actually, up to 3500 repetitions) and both of them reached a top in the intervals $[47.54, 56.5]$ for standard Hebbian and $[69.9, 75.6]$ for ABC Hebbian.

Despite the fact that CPPN clearly outperforms our approach in learning results, our approach has the advantage of being much more efficient computationally. For each single decision, CPPN has to query the entire network, whereas our method only has to do a simplified math calculation. Measured in CPU cycles of a Intel Core i7 920, CPPN takes a mean of 2.22Mpf (Million cycles per frame), whereas our approach takes only 1.25Mpf, which represents a 44% improvement.

5 Conclusions and Futher Work

In this paper we have considered Neuroevolution and HyperNEAT algorithms for Machine Learning and, in concrete, we have focused on the addition of different kinds of local adaptation rules (sometimes called neuro-plasticity) to enable the evolved individuals to continue learning online after training.

We also have presented a new proposal for online learning based on a combination of continuous and derivable functions to update weights each time-step. Our approach shows interesting results, as it shows some nice level of adaptation. Despite not achieving the same level of performance than CPPN-based approach, our approach is less computationally expensive, as it does not require to query the CPPN for each link at each time-step.

The first most interesting question to continue this research would be why our approach seams to reach a top so fast. It is our belief that more work on this approach could offer more interesting results, as there is still much room for new approaches on online learning through weight updating or even creation/destruction of neurons and links.

References

1. Gauci, J., Stanley, K.O.: Autonomous evolution of topographic regularities in artificial neural networks. Neural Computation 22(7), 1860–1898 (July 2010), http://eplex.cs.ucf.edu/publications/2010/gauci-nc10
2. Loiacono, D., Cardamone, L., Luca Lanzi, P.: Simulated car racing championship: Competition software manual (4 2013), http://arxiv.org/abs/1304.1672
3. Niv, Y., Joel, D., Meilijson, I., Ruppin, E.: Evolution of reinforcement learning in uncertain environments: a simple explanation for complex foraging behaviors. Adapt. Behav. 10(1), 5–24 (Apr 2002), http://dx.doi.org/10.1177/10597123020101001
4. Risi, S.: Towards Evolving More Brain-Like Artificial Neural Networks. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Central Florida (2012)
5. Risi, S., Lehman, J., Stanley, K.O.: Evolving the placement and density of neurons in the hyperneat substrate. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 563–570. GECCO '10, ACM, New York, NY, USA (2010), http://eplex.cs.ucf.edu/publications/2010/risi-gecco10
6. Risi, S., Stanley, K.O.: An enhanced hypercube-based encoding for evolving the placement, density and connectivity of neurons. Artificial Life p. 54 (2012), http://eplex.cs.ucf.edu/publications/2012/risi-alife12
7. Stanley, K.O.: Efficient Evolution of Neural Networks Through Complexification. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin (2004), http://nn.cs.utexas.edu/?stanley:phd04
8. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial Life 15(2), 185–212 (April 2009), http://eplex.cs.ucf.edu/publications/2009/stanley-alife09
9. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002), http://nn.cs.utexas.edu/?stanley:ec02, [The pdf file is made available by MIT Press]
10. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research 21, 63–100 (2004), http://nn.cs.utexas.edu/?stanley:jair04
11. Stanley, K.: Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines 8(2), 131–162 (2007), http://dx.doi.org/10.1007/s10710-007-9028-8
12. Wagner, G.P., Altenberg, L.: Complex adaptations and the evolution of evolvability. Evolution (1996), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.1524
13. Yao, X.: Evolving artificial neural networks. PIEEE: Proceedings of the IEEE 87, 1423–1447 (Sep 1999), ftp://www.cs.adfa.edu.au/pub/xin/yao_ie3proc_online.pdf/knowledge-extracted-from-trained.pdf