EqPropNEAT Extending NEAT with equlibrium propagation

Cs. Karikó, M. Kovács, Zs. Tornai

Writing Exercise Fictive Results

***Abstract*—Artificial neural networks have been employed successfully to solve a great variety of tasks over the past decade. In most applications, network architecture is usually decided with a trial-and-error process, relying on empirical experience. The neuoroevolution of augmenting topologies algorithm (NEAT) [1] proposes a solution to finding well fitting network architectures to a given task, however for learning weights, it only uses a simple genetic mutation rule.**

This paper proposes an extension of NEAT with an additional learning step employing equilibrium propagation [2] for learning edge weights. We claim that the resulting novel technique needs less iterations for convergence, as weights are adjusted by a method which was specifically invented for such task. Our experimental results show a significant reduction in training times on common neural network benchmark tasks, while achieving similar prediction quality.

***Index Terms*—neuroevolution, neat, equilibrium propagation.**

I. Introduction

THE process of designing neural network models always involves deciding on a particular network architecture. In order to avoid both overfitting as well as subpar prediction quality, the right amount of network complexity has to be found for a particular learning task. Over the years, a great amount of experience has accumulated regarding this decision, however these rules are more about empirical know-how, than proven theorems. For our experiment we took well tested and proven algorithms NEAT and Equlibrium Propagation(EP) [2] and tried to design a neural network which best represents a biological solution to a given task. For this reason EP was chosen rather than Backpropagation(BP) [3] to further fine tune the network in a way which would be plausible in a biological network. Due to the similarities between BP and EP we hope to capture all the positive aspects of BP while still maintaining the properties of a biological network. Due to the nature of EP our networks learns slower than a comparable network would utilizing BP, but as BP is considered ”biologically implausible” we accept this shortcoming. In chapter **N** we will discuss the performance compared to BP and we will explain why the difference in real-world applications would not be significant.

II. Background

As mentioned in the previous chapter, the NEAT algorithm uses a simple genetic mutation rule for evolving edge weights. With a sufficiently large population this method can evolve correct connection weights, however this process usually requires lot of generations. A number of extensions have already been proposed for improving weight search by incorporating BP to the original algorithm [4]. Such attempts include Learning-NEAT [5], DeepNeat and CoDeepNeat [6]. A common property of these techniques is that s limited number of training epochs are run during evaluation for adjusting edge weights and determining network performance.

Equilibrium propagation proposes a new approach to training neural networks. Unlike conventional artificial neural network models, equilibrium propagation models the network as a continuos-time dynamic system. Training and evaluation is accomplished with a numerical simulation of changing energy states converging to an equilibrium. After the simulation has arrived at a fixed point, the results can be read from the state of output nodes. Training is achieved by ”nudging” the states of the output nodes towards the correct prediction. This change than propagates throughout the network and weight updates can be computed from the two states of a given node during the evaluation and the training phase.

III. EqPropNEAT

Our method substitutes BP with equilibrium propagation. During the mutation stage of each generation, instead of assigning random weights to some connections, we employ equilibrium propagation iterations in order to adjust edge weights.

As the network's complexity increases with more and more generations, the search space for weights increases rapidly. In order to get the most out of each network architecture proposed by the NEAT algorithm, we suggest adjusting the number of equilibrium propagation steps run between generations according to network complexity. Note that this means that the number of iterations varies even between species of the same generation. During the evolution process, networks only change graudally between each generation. This allows us to split the weight learning process between generations, so that the number of EP iterations does not have to be enough for convergence as the training process will continue in the next generation. Too many EP iterations could also cause overfitting which limits the network's ability to generalize, It even might limit the adaptibility of species in future generations.

We have tested a number of different formulae for determining the number of necessary simulation iterations:

$$N_1 = V \tag{1}$$
$$N_2 = V + E \tag{2}$$
$$N_3 = V \cdot E \tag{3}$$

disclaimer: TODO

Where V and E denote the number of nodes and edges respectively.

TODO: fake figure about network complexity and needed iterations

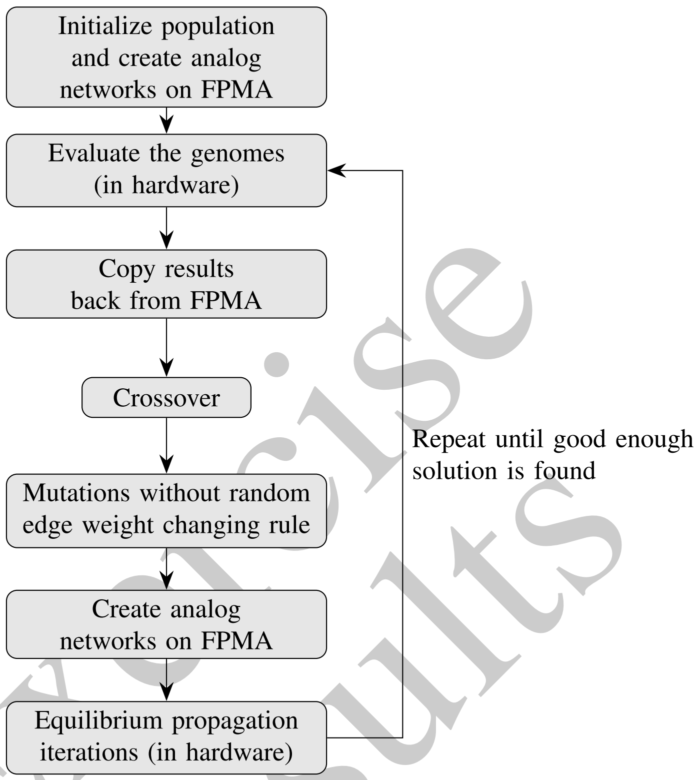
A. Hardware acceleration

As mentioned in the introduction, EP was designed with effective hardware implementations in mind. It has been shown [7], that the EP training of purely analog neural networks is indeed, possible. The missing piece for accelerating most of the EqPropNEAT stages is the capability to dynamically create the analog neural network representations for the current population. This would allow the use of hardware implemented EP for high-speed training.

Fortunately, field-programmable analog arrays (FPAA) provide an off the shelf solution for reprogramming the interconnects between different analog devices, enabling the creation of hardware based analog neural networks while our algorithm is running. However, implementing EP requires the ability to efficiently measure voltages in our network, as well as additional circuitry for digital operations such as computing the gradient. Field-programmable gate arrays (FPGA) are a much more popular digital counterpart to FPAAs. For our use case, a mixture of both systems is needed. While such devices have been already proposed in technical literature [8], to the best of our knoweledge, they have never been manufactured in a meaningful quantity yet. In theory, a mostly hardware accelerated implementation of our algorithm will be possible, once the above mentioned hardware becomes widely available. Figure 1 shows an outline of this algorithm.

Although at first glance the proposed hardware acceleration seems to improve runtimes significantly, we must mention a couple of potential performance limiting bottlenecks. Genetic algorithms in general require a population with thousands of genoms to provide enough diversity. Without this the algorithm does not work, or at least needs a lot more generations to arrive at a solution. It is also even more susceptible to overfitting. However with thousands of neural networks, with each of them potentially having at least hundreds of nodes, the required circuitry becomes problematically large. There is a high chance that even if the required hardware becomes available, it probably will only be able to contain a fraction of tha population at a time. Of course this problem can be somewhat alleviated by splitting the population into groups which are evaluated and trained in separate passes (slower runtime), however this way the number of occasions when the FPMA hardware is reconfigured with the new networks grows. In the case of modern day FPGAs the time it takes to reconfigure the entire board is usually in the realm of seconds and we believe this might not change in the near future since in most use cases this is not considered a bottleneck. As a consequence it is best to avoid reconfiguration as much as possible. At the time of writing this paper it is impossible to tell how much of a limiting factor this will be.

Another potential issue is the lifetime of the hypothetic hardware. Based on similar currently available boards, each reconfiguration lowers life expectancy significantly.



Initialize population and create analog networks on FPMA
Evaluate the genomes (in hardware)
Copy results back from FPMA
Crossover
Mutations without random edge weight changing rule
Create analog networks on FPMA
Equilibrium propagation iterations (in hardware)
Repeat until good enough solution is found

Fig. 1. An outline of the hardware accelerated EqPropNEAT algorithm. Note that the evaluation and weight training of genomes is entierely accomplished on the hardware level.

TODO: slow reconfiguration, hardware lifetime, runtime partial reconfiguration

IV. Performance

V. Conclusions

References

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[4] M. Y. Ibrahim, R. Sridhar, T. Geetha, and S. Deepika, "Advances in neuroevolution through augmenting topologies – a case study," in *2019 11th International Conference on Advanced Computing (ICoAC)*, 2019, pp. 111–116.

[5] L. Chen and D. Alahakoon, "Neuroevolution of augmenting topologies with learning for data classification," in *2006 International Conference on Information and Automation*, 2006, pp. 367–371.

[6] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[7] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, "Training end-to-end analog neural networks with equilibrium propagation," *arXiv preprint arXiv:2006.01981*, 2020.

[8] R. B. Wunderlich, F. Adil, and P. Hasler, "Floating gate-based field programmable mixed-signal array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 8, pp. 1496–1505, 2013.

Writing Exercise Fictive Results