Efficient Machine Learning through Evolving Combined Deep Neural Networks

Rune Krauss Marcel Merten Mirco Bockholt Saman Froehlich Rolf Drechsler
Cyber-Physical Systems, DFKI GmbH and Group of Computer Architecture, University of Bremen, Germany
{krauss, mar_mer, bockholt, froehlich, drechsle}@uni-bremen.de

ABSTRACT

The usage of *Artificial Neural Networks* (ANNs) with a fixed topology is becoming more popular in daily life. However, there are problems where it is difficult to build an ANN manually. Therefore, genetic algorithms like *NeuroEvolution of Augmented Topologies* (NEAT) have been developed to find topologies and weights. The downside of NEAT is that it often generates inefficient large ANNs for different problems.

In this paper, we introduce an approach called Turbo NEAT, which combines divide and conquer methods with NEAT to allow a symbiosis of specialised smaller ANNs. In addition, we optimise the weights of the ANNs through backpropagation in order to better compare the topologies. Experiments on several problems show that these approaches allow the handling of complex problems and lead to efficient ANNs.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; *Neural networks*; *Cluster analysis*;

KEYWORDS

Genetic Algorithms, Neural Networks, Cluster Analysis, Divide and Conquer, Backpropagation

1 INTRODUCTION

Machine Learning (ML) is becoming an increasingly important topic in computer science, especially *Artificial Neural Networks* (ANNs), whose topology consists of neurons and weighted connections. Usually, the topology of an ANN can be specified according to the problem. However, for some problems it can be difficult to determine the topology of an ANN.

NeuroEvolution of Augmented Topologies (NEAT) [11] is one of the most successful genetic algorithms for dynamic evolution of ANNs represented by genomes within a population. It uses genetic operations (1) Crossover and (2) Mutation to evolve topologies over generations so that the population approximates an optimum. A defined fitness function influences which genomes are allowed to reproduce after a generation.

However, when approaches like NEAT are used to solve large-scale problems, the size of ANNs can grow rapidly [10]. Even for simpler problems NEAT is unstable in performance if the ideal evolutionary parameters have not been found. Although huge research efforts have been made to improve the performance of NEAT to solve problems with large state spaces, existing approaches [5, 7, 8, 13] can still be improved.

In this work, we introduce *Turbo NEAT* (TNEAT), a C++ project that combines *Divide & Conquer* (D&C) methods with NEAT to realise various specialised ANNs for improving runtime and memory usage. Furthermore, we implement *Backpropagation* (BP) as proposed in [2], an algorithm to adjust the weights of the connections in order to better compare the resulting topologies and to optimise the weights for suitable problems. This reduces the runtime of the evolutionary process.

2 TNEAT AND EXPERIMENTAL RESULTS

The main concept in TNEAT is to divide a problem into subproblems that can be solved by specialised ANNs. For this purpose, TNEAT provides the possibility to use the clustering algorithm k-means++ [1] for clustering unlabelled samples, where k is the number of ANNs in a combination of ANNs. For each sample used to evaluate a combination, the distance to the centers of the determined clusters is calculated. The ANN which is responsible for the cluster of the nearest center is evaluated. Due to the simpler task of a specialised ANN, the D&C method will work more efficiently than an ANN, which has to solve the whole problem by itself [12].

TNEAT pursues the idea of using multiple populations to evolve ANN combinations. Each population provides an ANN as part of an ANN combination and is therefore responsible for a cluster. ML problems can be divided into non-sequential and sequential problems. While for non-sequential problems each input can be processed independently, the output of sequential problems depends on previous decisions.

If a problem is non-sequential, the populations are trained sequentially, i. e. the populations evolve independently one after the other. However, if a problem is sequential, the training needs to be interleaved, i. e. all populations of the D&C method are trained together. Thus, every ANN of each population has to be evaluated in combination with every ANN of the other populations. Each different output can influence the next sample sequence, which can cause k-means++ to assign the sequences to the individual populations differently, resulting in different *Control Access Points* (CAPs). A CAP means that a population starts to process one or more samples. If another population is selected by k-means++ for a subsequent sample, control is handed over to this population and it has the CAP for this sample. Because of the dynamic CAP behaviour, it is always necessary to calculate the fitness over the whole combination, resulting in an exponential time expense $O(s^n g)$, where n is the number of populations, s is the number of genomes in a population and g is the number of generations.

In order to control exponential time expense, TNEAT provides a sequential training method with time-dependent clustering for sequential problems. In this method, we cluster the samples in sequences of equal length and each population is responsible for one

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3390055

Table 1: Comparison between NEAT and TNEAT in runtime and ANN sizes to solve different problems

Algorithm	XOR (with BP)				Cart-pole				SMB											
									World 1-1				World 1-2				World 2-2			
	t in s	#G	#E	#V	t in s	#G	#E	#V	t in m	#G	#E	#V	t in m	#G	#E	#V	t in m	#G	#E	#V
NEAT	3.7 (2.7)	165 (91)	75 (42)	37 (15)	3.8	13	37	12	1383	356	3456	1042	1116	283	3269	761	1673	455	4087	1375
TNEAT	0.6 (0.4)	25 (8)	14 (10)	8 (4)	16.4	7	8	14	439	68	391	21	349	58	366	16	570	91	448	23
	t in s	Time in seconds							t in m	Time in minutes										

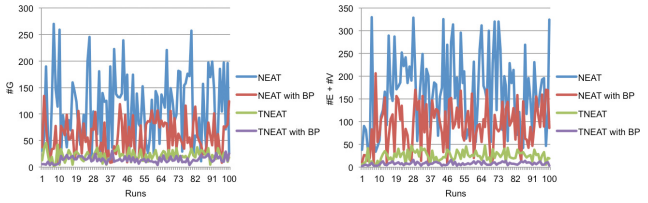
of these sequences. However, this distribution can cause problems with fixed CAPs. A poorly chosen CAP can cause a case where the next population has to fail. Moreover, a population may not be able to find a solution in the search space in a certain time. To solve these described problems, TNEAT can dynamically adjusts the fixed CAPs so that a population takes control of the last samples from previous populations.

To demonstrate the capabilities of TNEAT and to allow a comparison with NEAT, we have configured three problems with evolutionary parameters according to their complexity: (1) XOR, (2) cart-pole [6] and (3) *Super Mario Bros.* (SMB) [4]. We have adjusted the fitness threshold accordingly for better comparability. For TNEAT, we also have collected sufficiently large datasets for clustering and configured corresponding parameters for D&C. Regarding the parameters, we first have oriented ourselves on the designs proposed in [9] and [3]. We then adapted the parameters experimentally and by observation. Due to the characteristics of the problems, we have set the sequential training method for XOR, interleaved training for cart-pole and the sequential training method with fixed CAPs for SMB. Furthermore, we additionally configured BP for XOR, whereby less precision is required for the fitness threshold. Consequently, the topology was evolved with NEAT and TNEAT, while the weights were subsequently readjusted with BP.

For the sake of clarity, the average number of required generations (column #G) as well as the sum of the average number of hidden neurons (column #V) and connections (column #E) of all (combined) ANNs are shown in Table 1. The results clearly confirm that the proposed methods satisfy the objectives of this work. For XOR and SMB, TNEAT requires significantly fewer generations, hidden neurons and connections to solve these problems. Similar results can be observed for cart-pole. Regarding the described complexity of the training method used here, it should be noted that for cart-pole TNEAT needs more time to evolve the ANNs compared to NEAT. However, it can be argued that the time loss is a reasonable compromise compared to the resulting ANN size. In addition, the results of NEAT fluctuate quite strongly for certain parameter settings and a requirement for near-perfect precision, which is illustrated by XOR in Figure 1a and Figure 1b. It is also shown that the use of BP has accelerated the evolutionary process in NEAT and TNEAT.

3 CONCLUSION

In this paper, we presented TNEAT, which is a product of combining D&C methods with NEAT. Based on various experiments, we have shown that TNEAT is capable of evolving smaller ANNs in fewer generations than NEAT. Moreover, the XOR problem has shown that BP can accelerate the evolutionary process and lead to smaller generated ANNs.



(a) XOR: Generations **(b) XOR: Connections/hidden neurons**

Figure 1: Average number of required generations and hidden neurons/connections of NEAT/TNEAT to solve XOR

ACKNOWLEDGMENTS

We would like to thank Cornelia Große, Jannis Stoppe and Kenneth Schmitz for countless helpful discussions that have supported our reported research.

REFERENCES

[1] Shalove Agarwal, Shashank Yadav, and Kanchan Singh. 2012. K-means versus k-means ++ clustering technique. (03 2012). https://doi.org/10.1109/SCES.2012.6199061

[2] Lin Chen and Damminda Alahakoon. 2007. NeuroEvolution of Augmenting Topologies with Learning for Data Classification. 367 – 371. https://doi.org/10.1109/ICINFA.2006.374100

[3] Stephen Chen, James Montgomery, and Antonio Bolufé-Röhler. 2015. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence* 42 (04 2015). https://doi.org/10.1007/s10489-014-0613-2

[4] Erik Demaine, Giovanni Viglietta, and Aaron Williams. 2016. Super Mario Bros. Is Harder/Easier than We Thought. (06 2016).

[5] Thomas G. van den Berg and Shimon Whiteson. 2013. Critical factors in the performance of HyperNEAT. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 759–766. https://doi.org/10.1145/2463372.2463460

[6] F Gomez. 2003. *Robust Non-Linear Control Through Neuroevolution. PhD thesis.*

[7] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. 2014. A Neuroevolution Approach to General Atari Game Playing. *Computational Intelligence and AI in Games, IEEE Transactions on* 6 (12 2014), 355–366. https://doi.org/10.1109/TCIAIG.2013.2294713

[8] Yiming Peng, Gang Chen, Harman Singh, and Mengjie Zhang. 2018. NEAT for large-scale reinforcement learning through evolutionary feature learning and policy gradient search. 490–497. https://doi.org/10.1145/3205455.3205536

[9] Kenneth Stanley, Bobby D. Bryant, and Risto Miikkulainen. 2006. Real-Time Neuroevolution in the NERO Video Game. *Evolutionary Computation, IEEE Transactions on* 9 (01 2006), 653 – 668. https://doi.org/10.1109/TEVC.2005.856210

[10] Kenneth Stanley, David D'Ambrosio, and Jason Gauci. 2009. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial life* 15 (02 2009), 185–212. https://doi.org/10.1162/artl.2009.15.2.15202

[11] Kenneth Stanley and Risto Miikkulainen. 2002. Efficient Reinforcement Learning through Evolving Neural Network Topologies. (04 2002).

[12] Wen Wang, P.H.A.J.M. Gelder, J Vrijling, and Jun Ma. 2006. Forecasting Daily streamflow using hybrid ANN models. *Journal of Hydrology* 324 (06 2006), 383–399. https://doi.org/10.1016/j.jhydrol.2005.09.032

[13] Shimon Whiteson and Peter Stone. 2006. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research* 7 (05 2006), 877–917.