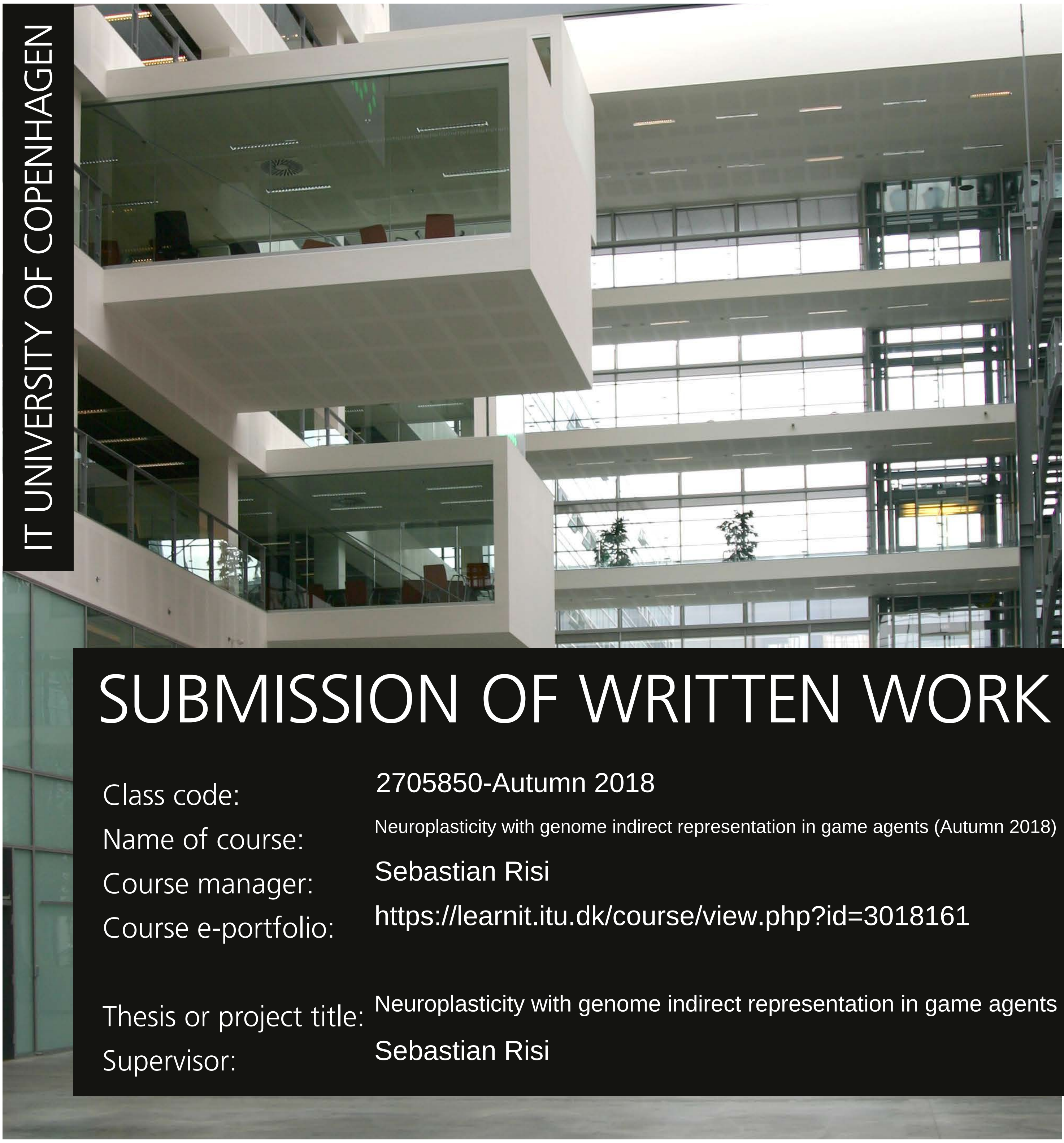

IT UNIVERSITY OF COPENHAGEN

SUBMISSION OF WRITTEN WORK

Class code:	2705850-Autumn 2018
Name of course:	Neuroplasticity with genome indirect representation in game agents (Autumn 2018)
Course manager:	Sebastian Risi
Course e-portfolio:	https://learnit.itu.dk/course/view.php?id=3018161
Thesis or project title:	Neuroplasticity with genome indirect representation in game agents
Supervisor:	Sebastian Risi

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Christian Carvelli	23/03-1993	chca@itu.dk
2.	Mads Falkenberg Sønderstrup	30/03-1991	mfso@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk
8.			@itu.dk

Hyper Neuroevolution

Christian Carvelli, Mads Falkenberg Søonderstrup

Abstract—Backpropagation has been proven to be a valuable tool with which to train artificial neural networks (ANNs) for a given task. One issue with backpropagation on its own is that information is lost about the ANN every time backpropagation takes place as no information is stored about the ANN's previous state. This essentially means that the ANN is merely a function optimizer and will be hard to fit for similar tasks let alone different tasks because it does not retain information about general rulesets. The demand for general purpose ANNs is increasing, so learning to learn is essential. Meta-learning has emerged as the solution, where ANNs have to learn the rules for learning. In this paper we aim to lay down a stepping stone for meta-learning in neuroevolution, using the theory of hyper networks, a supervised learning technique, for a reinforcement learning task. We go over how we achieved indirect representation in the genome for game agents, what this means for neuroevolution and how this opens up venues for further experimentation in the future in regards to meta-learning.

Index Terms—Neuroevolution, Reinforcement Learning, Meta-Learning, Indirect encoding

1 Introduction

For certain problem domains it is sufficient enough to employ supervised learning and unsupervised learning techniques when training neural networks. The issue with these approaches is that the environment they work on are static. The content the neural networks have to process changes but the environment stays the same meaning that a network trained for a particular task will prove inefficient and unusable for another task.

Reinforcement Learning (RL) is trying to get around this by only being given the action space and what constitutes a reward, more akin to how humans learn. This is better for tasks that have dynamic environments with a static action space, however networks trained this way can often take thousands of hours to attain a level of performance comparable to or better than humans, whereas humans only need a fraction of the time. Adding to that, the possible tasks that RL can take on are still relatively limited compared to what humans can do.

Deep artificial neural networks (DNNs) are usually trained using backpropagation which is based on gradient descent. Evolutionary Strategies (ES) have shown that it can compete with backpropagation algorithms such as Q-learning and policy gradients in some RL problems. There is one issue with this however; ES can be considered a gradient-based algorithm and thus the question becomes whether it is possible to make use of non-gradient-based evolutionary algorithms which can work at DNN levels of complexity or not.

Genetic algorithms (GAs) have have been proved competitive in RL tasks, specifically in the Atari and human locomotion domains. Experimentation shows that they perform similarly to ES, A3C and DQN while taking advantage of high performant parallel computing optimizations, while showing again that there is no absolute winner but that is important to pick the right algorithm for the given task.

We believe that, using indirect representation, Deep Neuroevolution (DNE) can be improved achieving comparable results to other RL optimizers with less parameters while also possible open up venues for additional experiments with meta-learning and neuroplasticity.

- *chca@itu.dk*
- *mfso@itu.dk*

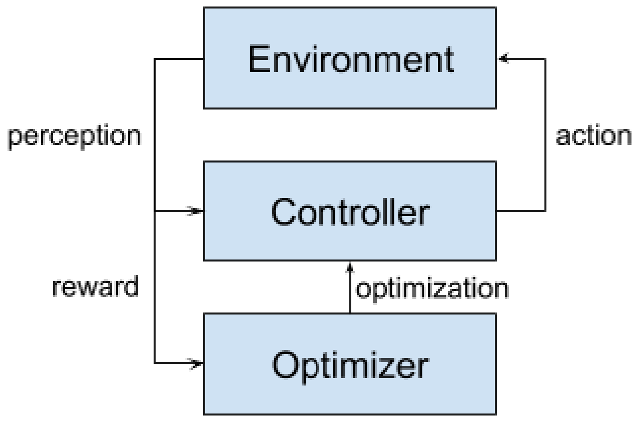

Fig. 1. Typical loop of an optimization process. The optimizer receives the reward from the environment and uses it to tweak the parameters of the controller (or controllers, in case of evolutionary optimizer), but it doesn't change the controller's architecture

We will achieve this by making a Hyper Neural Network (HNN) that is responsible for the generation of a much larger network, the policy network, which does the decision making for the provided input by the environment. With this approach the HNN will be responsible for generating the weights for the policy network. The HNN achieves this by having full knowledge of the policy network topology. This will result in having more efficient networks as less parameters are required, thus less computing time.

The original work on Hypernetworks experimented with supervised learning tasks and, to our knowledge, to this day no attempt have been made to use Hypernetworks on RL tasks, nor to use them with neuroevolution.

2 Background

2.1 Theory

Neuroevolution as a mechanism to generate neural networks was first approached by Yao in the late 1990's [22], addressing the possibility of evolving weight distribution, topology, or both. After Deepmind's breakthrough in end-to-end Atari controllers with DQN [10], many optimization methods from the classic literature have been adapted to DNNs [9], [17], including evolutionary algorithms like ES [16].

While evolutionary optimizers focus solely on generating the weights of the networks, efforts have also been made into evolving network topologies [2], [19]. NEAT was later used as a base for an indirect-encoding version of the algorithm called HyperNEAT [18] that got a great deal of attention [12], [13], [14], [15] and generated interest in indirect-encoded networks.

2.2 Related work

Such et al. [20] use DNE to train agents using a Genetic Algorithm (GA) as a gradient-free method. Their work shows that GAs are able to compete with other gradient-based optimizers and with other evolution based methods such as Evolution Strategies (ES) in the Atari domain.

Salimans et al. [16] shows a compression method, also used in the DNE paper, that allows them to efficiently scale up their ES methods across multiple networked workers. The method consists of storing only the random seeds used for mutation, as well as the random seed used to initialize the model's first ancestor. This way, a model can be reconstructed simply by initializing it as its first ancestor and then replicating all the mutations that led to the current version.

Ha et al. [7] develops a method they called Hypernetworks as an indirect-encoding method reminiscent of HyperNEAT [18] but completely trained with backpropagation. They focus their work on convolutional and residual networks (with the version they call static hypernetwork), managing to achieve results comparable to fully connected networks but reducing the amount of parameters by several orders of magnitude, and on LSTM (dynamic hypernetwork) reaching state-of the-art results in language modelling and other tasks. Our work is focused on static hypernetworks, Figure 2 explains how they work.

2.3 Tools

PyTorch [1] is an open-source machine learning library that provides many high-level features like GPU-accelerated tensor computation, modularized deep network architecture and API for common ML tasks as natural language processing and computer vision (torchvision module).

OpenAI Gym [4], [11] is a toolkit for developing and comparison RL algorithms. It provides a collection of test problems implementation (environments) as well as a common interface for RL problems. Many environments are available, ranging from classic controls to Atari games based on the ALE framework [3] to robotics.

3 Methods

Based on Ecofett's DNE [5] we tested the results obtained in the DNE paper, obtaining comparable results on the deterministic version of the Frostbite Atari game environment. We used the originally proposed networked architecture with model compression based on random seeds. This first experiment was stopped after three days of computation, using three to eight computers running four workers each at any given point, after almost reaching the peak performance obtained in the original DNE paper using c.a. 365 million frames (out of the one billion used in the original work). We then proceeded to add an HNN to the model used for the controller, using the implementation provided by Ha [6]. The preliminary runs showed very poor results, generated mostly by bugs that we introduced editing the original architectures to make them a) compatible with each other and b) flexible enough to allow easy experimentation and change of parameters and architectures.

Since Ha's architecture consisted simply of two linear layers, we initially tried to follow their own simple implementation with simple matrix multiplications. This wasn't compatible with Pytorch's Module pattern and our code wasn't hooked up properly to the framework's code, causing problems during evolution. Due to our inexperience at the time we decided, instead of adapting that particular implementation, to follow Pytorch's pattern and use Linear layers, that solved the problem.

In the meantime we tried many different ways of analyzing the relationships between the HNN and the policy network, both at initialization time and during evolution.The most useful one was to statistically compare the initialization of the policy network layers of the DNE setup with the ones generated by an HNN. We saw that our first HNN setup was yielding weights that were on average in the hundreds and with a different standard deviation with respect to the parameters used to initialize the policy network during the DNE tests. Tweaking the HNN initialization parameters and adding a multiplication constant to the output before applying it to the network helped reach the first comparable results with DNE's original paper [20].

A brief attempt has been made using a different gym environment called Bipedal Walker, a simple robot controller with the objective to reach the furthest distance. We don't have relevant data from these experiments, but we got first-hand experience with continuous control problems. That helped us figure out that our controller got an initially good improvement, but failed to overcome some of the earliest plateaus due to a great instability in the optimization process.

Another problem, discovered after the main set of experiments had to be terminated due to external factors, was our handling of the z vector. It turned out that during compression we saved the current evolved version of it, while the evolution method expected the original z vector so it could regenerate it in a similar fashion to the HNN weights.

Initially we setup a networked architecture, but we ran into issues with workers not deploying correctly after changes were made to the code. After removing the networking code during development, we didn't reintegrate it

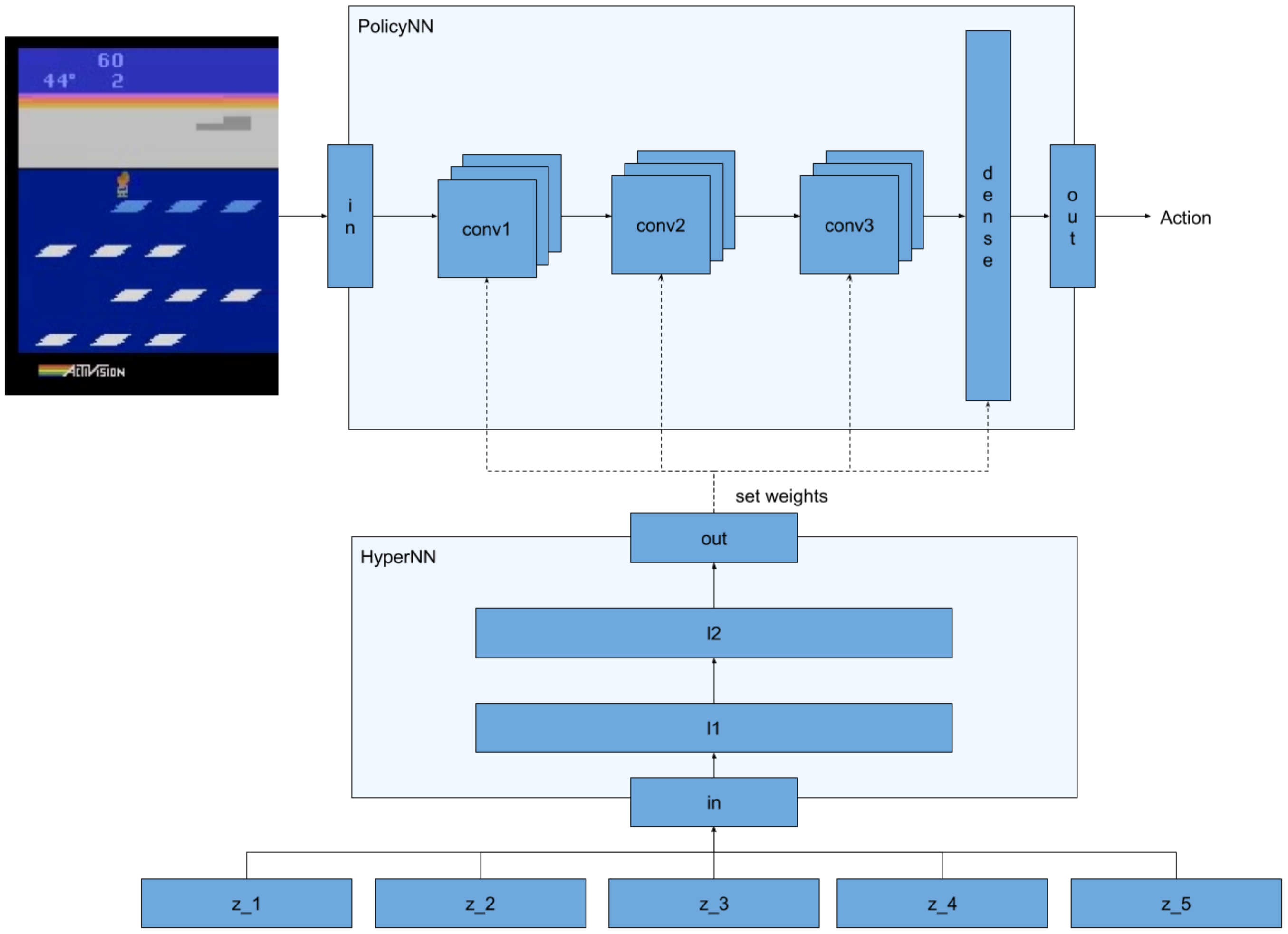

Fig. 2. Controller network setup. When the controller is initialized, a vector of floating point numbers is fed to the hypernetwork for each layer in the policy network. The output of the network has the size of the biggest layer in the policy network and it's resized and reshaped properly for each layer.

Population size	(N)	1000
Mutation rate	(σ)	0.005
Truncation size	(T)	10
Number of trials		1
z_v evolution P		0.5

TABLE 1
Hyperparameters used for both DNE and HNE runs

for the final test suite as we thought it was a bottleneck for our experiments due to long waiting times when a job failed, but also becase of a decrease in hardware availability.

3.1 Hyperparameters and architecture

The hyperparameters used for the runs reported here are shown in Table 1. Those are the ones used in [5] and deviate slightly from the ones used in the original paper. For our HNN setup, we also needed an additional hyperparameter used to determine when to evolve the z vector and when to evolve the HNN. We hypothesized that simultaneusly evolving the HNN and the z vector could be problematic for the GA, but this should be investigated further. A proper exploration of the optimal hyperparameters for the HNN, and how they differ from the ones used for the non-augmented controller, could explain a lot about how the two networks relate to each other, but currently we lack the resources to do that. What we could do is to try the hyperparameters of the original work and see which one fits best for the given task[1].

Similarly, our architecture is borrowed from Ecoffet and shown in Table 2.

4 Results

The results of our main experiment are shown in Fig. 3, run before we found the z vector bug. The fact that the evolution process used as a start the already evolved z vector turned out to effectively perturb the z vector, acting as noise for the policy network initialization. This of course hindered the performance of our controller, but did not made it completely ineffective, due to the natural neural network's resistance to noise. In fact, the run executed

1. Is worth to mention that, to our knowledge, Ecoffet tried (and we imagine optimized) the hyperparameters for the Frostbite environment, while Such at al. experimented on many Atari games and additional environments

Layer	Type	in-features	out-features	kernel	stride
conv1	conv + relu	4	43	8x8	4
conv2	conv + relu	32	64	4x4	2
conv3	conv + relu	64	64	3x3	1
dense	linear + relu	1024	512		
out	linear	512	18		
l1	linear + relu	32	128		
l2	linear + relu	128	128		
out	linear	512	1204 * 512		

TABLE 2
Network architecture. The first section refers to the policy network architecture for both DNE and HNE. The second section refers to HNE's hypernetwork.

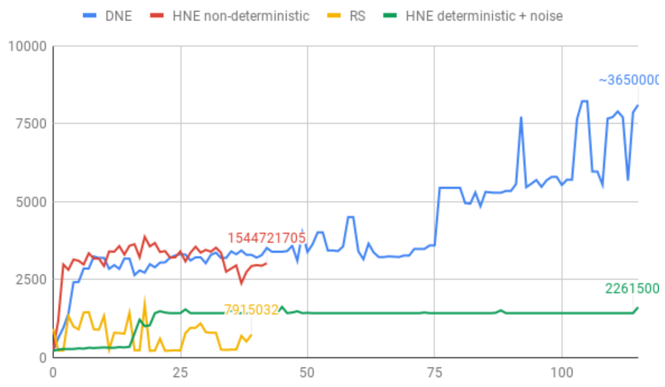

Fig. 3. Results of experiments run with code that used z vectors perturbed by noise. The run executed with deterministic environment and noisy start (Green) got stuck on a very early plateau, with results comparable with RS. The non-deterministic run (Red) got strangely comparable results, but had to be terminated early due to hardware problems. Random Search (yellow) and DNE runs added for comparison.

with a deterministic but noisy environment (Green) showed to be not much better of a RS, even if more stable. The run executed with a non-deterministic environment (Red) seemed to yield comparable results to the original DNE, but we don't have a complete run due to external problems with the hardware. At the time of writing, an experiment that handles the z vector properly is running showing promising (even if still not conclusive) results.

5 Discussion

Before discussing how comparable our results are to the ones of Such et al. it is paramount to underline that our results are only indicative and not conclusive. In the research done on DNE the elite was tested five times for every run with the final score for each elite as the mean of 30 independent episodes. In our experiments the elite of each run was only tested once mainly due to time and hardware constraints. Following the same procedure would allow us to test if our results are statistically relevant. We also only tested on the frostbite game gym making it possible for the HNE to compete much worse or better in other games.

With that in mind our test results do hint at there being potential for an HNN to compete with current implementations of deep neuroevolution using less parameters, for the same task.

Going forward what needs to be looked into further is more experimentation with more rigorous testing to verify that the hyper network augmented controller indeed does perform comparably in not just Frostbite but also in the other Atari gyms using indirect encoding of the genome then later move on to other types of gyms like the Bipedal Walker and possibly other robotics tasks such as the MuJoCo Humanoid Locomotion. Meet the experimentation time and resources required for this kind of rigorous and extensive experimentation could prove problematic. Towards the end of the project we managed to have a reliable setup for networked experiments, but the hardware necessary to run them in reasonable amount of time is not easy to obtain. Even the last, most optimized version of the networking setup managed to obtain some useful results from HNE on an Atari game in roughly 16 hours. Run minimal or non-networked experiments would probably end up in very long iteration iteration times that would probably hinder the research. Then we can move on to looking into meta-learning possibilities using an HNN .

The desired outcome being that the HNN will enable the controller to express neuroplastic properties by utilizing previously learned rules and policies.

Another interesting prospect of future research would be to adapt an HNN to work with ES. It might prove beneficial to introduce the HNN to ES since the HNN reduces the dimensionality of the problem which many versions of ES don't scale well with [8], [21].

This could have rather large implications for continuous learning and one-shot learning as the HNN could act as a facilitator for meta-learning, reducing the time and resources required to train a network for any given task. Another interesting approach would be to try an Hyper ES implementation as it could prove beneficial since the HNN reduces the dimensionality of the problem and many versions of ES don't scale well with it [8], [21]. Finally, future work will surely include tests the setup in different domains such as other Atari games, the Bipedal Walker environment and possibly the MuJoCo Humanoid Locomotion and other Robotics tasks.

6 Conclusion

We managed to combine HNN and DNE creating what we called Hyper Neuroevolution; allowing for a more indirect representation of our agents, resulting in controllers that may be indicative of being comparable to current DNE implementations.

Our results are not conclusive by any means but they do seem to point towards a different approach to DNE that could result in more efficiently packed networks for challenging RL tasks, as well as networks that are able to exploit regularities and symmetries in problems that have regularities and symmetries that can be exploited. More experimentation is required for the results to be conclusive.

With more research, we hope that HNN will prove useful in the meta-learning research.

References

[1] Pytorch an open source deep learning platform that provides a seamless path from research prototyping to production deployment., December 2018.

[2] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 4148–4152. AAAI Press, 2015.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[5] Adrien L. Ecoffet. Paper repro: Deep neuroevolution, 2018.

[6] David Ha. Supercell, 2016.

[7] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[8] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] OpenAI. Openai gym a toolkit for developing and comparing reinforcement learning algorithms., December 2018.

[12] Justin K Pugh and Kenneth O Stanley. Evolving multimodal controllers with hyperneat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 735–742. ACM, 2013.

[13] Sebastian Risi, Joel Lehman, and Kenneth O Stanley. Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 563–570. ACM, 2010.

[14] Sebastian Risi and Kenneth O. Stanley. Enhancing es-hyperneat to evolve more complex regular neural networks. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1539–1546, New York, NY, USA, 2011. ACM.

[15] Sebastian Risi and Kenneth O Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial life*, 18(4):331–363, 2012.

[16] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009. PMID: 19199382.

[19] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[20] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[21] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

[22] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.