Reinforcement Learning with Neuroevolution: Final Report

G109 (s1879066, s1886694)

Abstract

This project explores neuroevolution designed for application to reinforcement learning tasks. A genetic algorithm (GA) is applied to a population of neural networks that control an agent acting within an environment. Building on recent work, we apply neuroevolution to agents in Atari and Mujoco environments. We explore smaller population sizes, crossover and hierarchical genetic algorithms. In our experiments smaller population sizes have not been successful. This might be caused by a comparably complex GA implementation. Furthermore we did not find evidence for crossover impacting performance significantly. We obtained promising results with using hierarchical genetic algorithms. These results have been similar to some of the most advanced algorithms in reinforcement learning. This should be expanded in future work.

1. Introduction

Our project aims to apply neuroevolution to solving tasks that Reinforcement Learning (RL) may more traditionally be applied. Neuroevolution refers to using a Genetic Algorithm (GA) to evolve weights and/or the architecture of a neural network (Mitchell, 1998; Such et al., 2017; Montana & Davis, 1989). A Deep Neural Network (DNN) (Goodfellow et al., 2016; Nielsen, 2015) is, in very simplistic terms, a map from an input to an output. This mapping is achieved by matrix multiplication of weights. These weights are optimised subject to a loss function and are traditionally optimised with gradient descent methods. Those methods are usually computationally expensive. The motivation of neuroevolution is to instead evolve weights towards an optimal set of parameters. While the idea of evolving weights was introduced in 1989 by Montana and Davis, it is only recently that significant progress has been made in applying these techniques to the area of RL. For example Such et al. from Uber Research made significant progress in early 2018(Such et al., 2017). Many recent RL algorithms train one or more neural networks as a method of approximating the optimal action to take at each step of a RL problem (so as to maximise future reward). More information about RL, Genetic Algorithms and neuroevolution are given in the background section 2. We aim to evolve a neural network designed for this task as opposed to optimising through gradients. We also compare the results to traditional RL algorithms. Our main project goals are to answer the following research hypotheses throughout our project:

1. Many current applications of neuroevolution to RL problems use a large population size (Such et al., 2017; Salimans et al., 2017). Many research papers such as these have state of the art computing resources available. We explore if similar results can be obtained with smaller population sizes.
2. Much of the current literature does not use a crossover operator when evolving new networks(Such et al., 2017). We explore if crossover can improve results.
3. Hierarchical genetic algorithms have the ability to learn optimal hyperparameters for an individual genetic algorithm, similar to applying meta-learning in deep learning. During experimentation, we found that our algorithm often got stuck in local optima, and hypothesised that a change in learning rate would be required to escape. However, different learning rates appeared to be more optimal depending on the progress of the algorithm (for example a small learning rate to refine current good solutions as much as possible, and a larger learning rate to escape local optima). Therefore we hypothesised that using a hierarchical genetic algorithm to evolve the learning rate may result in higher performance.

Initially, we began our project by using a simple environment. We gradually expanded to more complex environments using Atari games. Subsequently we expanded our research to continuous control problems from the MuJoCo environment suite(Todorov et al., 2012).These experiments are further described in Section 4. In the background section we will give a brief introduction to concepts of reinforcement learning, genetic algorithms, neuroevolution and convolutional neural networks. Following from this, we will introduce the environments and implementations that we use. This is then followed by an outline of the experiments that we have undertaken in Section 4. Finally we draw conclusions from our results and suggest directions for future work.

2. Background

In this Section we introduce the key concepts of reinforcement learning, genetic algorithms, neuroevolution and convolutional neural networks. We additionally give a background of recent research in this area.

Reinforcement Learning (RL) (Sutton & Barto, 2018) aims to learn from interaction with the environment similar to human beings. In order to evaluate taken actions, favourable choices are rewarded. The actor is called agent and aims

to maximise the accumulated rewards. An illustration of how an agent interacts with its environment can be found in Figure 11. More formally, an agent interacts in an environment through timesteps t by taking actions A_t and observing states S_t and possible rewards R_t. The agent receives a representation of the state S_t and selects an action A_t by using S_t. In the subsequent time step after taking A_t, the agent observes a new state S_{t+1} and obtains reward R_{t+1}. The goal of the agent is to maximize the cumulative reward it obtains (Sutton & Barto, 2018). For our purposes, the simple sum of rewards $G = \sum_{t=0}^{T} R_t$ is sufficient, where T marks the final time step of the agent interacting with the environment. The time steps $t = 0, \ldots, T$ are referred to as a full episode. A common method of choosing optimal actions at each time step (with the goal of maximising future reward) is to optimise a value function. A value function takes state information as input, and the output can be used to choose the best action. Deep RL uses a neural network to approximate this value function. Usually gradients are used to determine the associated weights, such as using backpropagation.

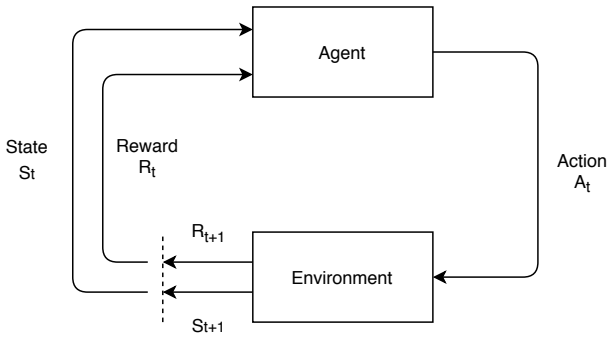

Figure 1. The Agent and its environment. Credit: (Sutton & Barto, 2018)

Genetic Algorithms (GA) (Mitchell, 1998) imitate the evolutionary process in genes. From a given population with certain characteristics, the fittest have the highest probability in being transferred to the next generation. In order to explore further favourable characteristics, certain genes are slightly changed. As Mitchell points out, there is a lack of a rigid definition of the algorithm. However, we will follow her definition. As a starting point a randomly generated population of bit strings, representing genes, is generated. Examples of such strings could be `010001001` or `100100011`. These strings represent potential solutions to a problem. The degree to which a string is 'optimal' is measured by a fitness function, such that strings that have higher fitness values are preferred. In order to evolve to a new generation (of bitstrings) a GA applies three operators:

- Selection: Select bitstrings such that the likelihood of selecting the fittest is highest.
- Crossover: Cut two strings at a point and combine the first half with the string of the second half of the other. For example `1111` and `0000`, can be crossed over to `1100` and `0011`. This is known as single point crossover. Alternative methods of crossover exist.
- Mutation: Flip certain bits in a string. For example `1111` will flip the third position to `1101`. Similarly to crossover, alternative methods for mutation exist.

The algorithm evolves a new generation by applying those operators and replacing some/all of the strings in the previous generation with the newly created strings. The number of iterations of this process equals the number of generations. This imitates the genetic process of inheritance (Mitchell, 1998).

Neuroevolution uses the principles of a GA to evolve the weights and/or architecture of a Neural Network. For the purposes of this project, we will focus on evolving weights, which is sometimes referred to Conventional Neuroevolution (CNE) (Hausknecht et al., 2014). Instead of bitstrings as individuals of our population, we use neural networks. The key idea is to apply the genetic operators to the values of the weights within the networks in an attempt to converge to optimal weights. As a result, we intend to find optimal weights without using gradients. Gradient-based methods are notoriously costly, and so the motivation behind neuroevolution is to optimise a neural network in a faster period of time. Additionally, neuroevolution does not require a set of pre-labelled input-output pairs as deep learning does. This extends the application area compared to gradient-based neural networks.

Combining both of the aforementioned topics together, we aim to use neuroevolution to evolve a network. This network takes state information as input, and outputs an action to take. Formally: $f(S_t, W) = A_t, t \in \{0, \ldots, T\}$ where W represents the weight tensor of all weights (including hidden layers). For the neuroevolutionary approach, we have a set of weights as tensors. This set of tensors represents a generation: $W_1, \ldots, W_N$, where N represents the population size. We begin with a random initialisation of these weight tensors. Our fitness function will use the cumulative reward of running episodes. Following from this, we will apply genetic operators at each generation to evolve the weights, resulting in convergence towards higher fitness values.

A convolutional Neural Network (CNN)(Goodfellow et al., 2016; Nielsen, 2015) consists of *convolutional* layers, in which the weights are given as sets of matrices known as filters. These filters aim to perform a convolution operation on the input. CNNs usually also contain fully connected layers. The most common application for CNNs is image analysis, where each convolution layer aims to extract key information using receptive fields. This information is passed through to the next layer. This helps to recognise context and reduces the amount of information passed. In this project we apply neuroevolution to both feedforward/linear DNNs and to CNNs.

Uber Research has recently made progress in application of neuroevolution to RL problems(Such et al., 2017). They compared their approach to traditional RL methods, and show that neuroevolution is competitive to algorithms such as Deep Q Learning (DQN)(Mnih et al., 2015) and Aysyn-

crhonous Actor Critic (A3C)(Mnih et al., 2016). However they did not compare their results to more recent state of the art RL techniques, for example Proximal Policy Optimisation (PPO)(Schulman et al., 2017). The innovation of Such et al. is to use a unique encoding method for the networks as a vector of initialisation seeds. Each mutation is then appended to the encoding structure, so that a network can quickly be instantiated using the initialisation seed and sequence of mutations. This enables a larger quantity of networks to be stored in memory. Although applying this representation was considered for our project, we decided to keep a more standard representation (storing all networks directly in GPU memory) so as to keep our GA as traditional as possible. We aim to compare our results primarily with those produced by Such et al. Another similar neuroevolutional approach has been used by (Hausknecht et al., 2014). Additionally, we observe that a gap in the research area is application of hierarchical genetic algorithms, and as such we use this as our final hypothesis question as given in the introduction.

3. Environment and Implementation

In this section, we present our design choices and how we implemented neuroevolution. Firstly, we require an environment for our agent to interact with. Secondly we need to define the network architecture we are using. Finally, we need to define how we evolve each generation of weights using genetic operators.

We are using OpenAI Gym environments (Brockman et al., 2016), which provides simple interfaces for accessing state and reward information for a range of environments. Additionally, this enables a direct comparison with results found in both RL literature and other literature attempting neuroevolution. In OpenAI Gym environments, the reward is a numeric value and the state is either an image or a numerical vector, depending on the chosen environment. As mentioned previously, we perform experiments in three different environment areas. (i) a simple baseline experiment, (ii) Atari games, and (iii) MuJoCo. For the simple baseline experiment and MuJoCo environments, the states are represented as numerical vectors and so no preprocessing is applied. In Atari games the state is given by 210x160 RGB images (more specifically NumPy arrays of shape 210x160x3). We preprocess these to be 84x84 grayscale images. This reduces the number of parameters required in the network, thus increasing efficiency. We additionally used parts of the OpenAI Baseline[1] implementations in order to use a vectorized representation of multiple parallel environments. This was used to reduce computing time.

For the network we use either a feedforward DNN of sequential linear layers or a CNN. We used feedforward DNNs where the state was represented as vector. We used CNNs where the state was represented as image. We use the CNN architecture proposed by (Mnih et al., 2015) for Atari games. This architecture was also used by (Such et al., 2017). The architecture conists of three convolutional layers with (i) 32 8x8 filters with stride 4, (ii) 64 4x4 filters with stride 2, and (iii) 64 3x3 filters with stride 1 respectively. These layers are followed by two final fully connected linear layers. The first of which having output size of 512, and the final layer having an output size of the number of possible actions. The chosen action is the argument of the maximum. We explore a variety of different feedforward architectures for MuJoCo environments, as is justified in section 4.3.

For our GA we implemented the three operators introduced in Section 2: selection, crossover and mutation. We list our methodologies and parameters for each of these operators below. In certain cases these parameters may be deviated from, however any such occurrence will be explicitly stated.

- Selection. We decided on a tournament selection algorithm. Tournament selection randomly samples a subpopulation of t (the tournament size) candidates from the population. From this sub-population, the best candidate (of highest fitness) is chosen. This enables us to directly control the level of selection pressure by changing the parameter of tournament size(Mitchell, 1998; Miller et al., 1995; Blickle & Thiele, 1996). Selection pressure refers to the likelihood of the fittest candidates to be transferred to the next generation. The tournament size was set to around a third of the total population size.
- Crossover. Any time crossover is used, we select two networks from the current population. Those are selected using tournament selection. The layer of weights of the new candidate is randomly composed of those. A crossover was applied with a probability 50% to each generation.
- Mutation. Normal distributed noise was added to the weights with a probability of 90%. The added noise is $\Theta(t+1) = \Theta(t) + \alpha \cdot \epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$ and learning rate α

We additionally implemented concepts such as ageism(Real et al., 2018) and a hierarchical genetic algorithm, both of which are explained in the sections that discuss them.

4. Experiments

In this section we describe all performed experiments. These include our initial baseline experiments on the CartPole environment, experiments on a variety of Atari environments, and experiments on a continuous control MuJoCo environment.

4.1. Simple Baseline Experiment

We designed our baseline experiment around a very simple environment in order to create an initial working implementation that can be modified for more complex tasks. The chosen environment was CartPole, contained within the environments of the OpenAI Gym module. The goal of

[1] https://github.com/openai/baselines

this environment is to balance a pole upright on a moving cart which can be moved either left or right at each time step. A positive reward signal is given for each time step that the pole remains balanced upright, and each episode finishes once the pole is beyond a critical angle from vertical. The environment also caps the maximum reward at 200, at which point the episode is considered 'solved'. The state information of the environment is a short vector of 4 floating point values, and therefore the design of a neural network to control an agent using this state information could be simple. We used a neural network with a single hidden layer that consisted of 20 hidden units. We used no crossover function, and additionally applied ageism. Population members that survive for 25 generations are reinitialised to random weights. Additionally a mutation rate of 0.5 was used. The results can be seen in figure 2. Very few generations were required to reach the maximum reward of the environment, which took roughly 30-40 seconds. Fluctuations in maximum fitness over time can be attributed to either the ageism or to variations within the stochastic environment. The average fitness varies significantly, given that mutating good solutions quite often results in less desirable weights.

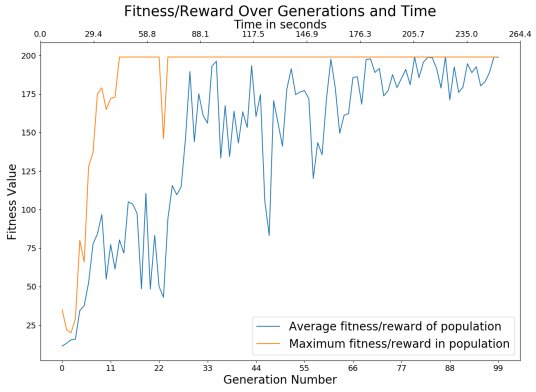

Figure 2. Average fitness/reward of the population and highest fitness/reward within the population against both time elapsed and number of generations.

However, it is worth mentioning that our baseline experiments did not perform consistently since randomness is involved in mutations. Some trials performed better and others worse than our initial results seen in 2, however for the purposes of a baseline to build upon, we were satisfied with our results.

4.2. Atari Games

(Such et al., 2017) tested neuroevolution on a variety of Atari games. We chose two successful environments from this selection, in order to test our hypothesis that smaller population sizes could be similarly successful. Additionally we tested if applying crossover has any impact. The selected environments are Frostbite and Kangaroo. Additionally we used an environment which has not been evaluated by the authors, namely Breakout. We used Breakout as a starting point for our Atari games experiments. Our assumption was that Breakout might be a good starting point, given its simplicity. Breakout has 4 possible actions, compared to 18 in Frostbite and Kangaroo. We assumed that this simplicity would be beneficial for results. We used CNNs and the GA as described in Section 3. In our experimentation we varied the learning rate (synonymously the mutation power) and observed the difference between including or excluding crossover. Other parameters included a population size of 64 and 500 generations. We stopped the algorithm early when the maximum fitness did not change for 100 generations. Given the stochastic nature of neuroevolution, we ran certain set-ups multiple times. Results for all three environments are given in Table 1.

Environment	Total Runs	CO	Score
Breakout	3	no	11 / 11 / 12
Breakout	3	yes	11 / 11 / 11
Frostbite	12	no	150 / 170 / 180
Frostbite	12	yes	140 / 170 / 190
Kangaroo	5	no	1000 / 1400 / 1400
Kangaroo	5	yes	1200 / 1400 / 1400

Table 1. Results Atari Games. Learning Rate has been set to 0.1. CO indicates if crossover has been applied. Score is shown as minimum / median / maximum in accumulated reward points returned by the OpenAI Gym environment for the best performer. Maximum of 500 generations. The number of generations was subject to early-stopping when the maximum fitness did not improve for 100 generations.

Breakout showed little to no sign of improvement to the initial accumulated reward values. A graphical representation of this result is shown in Figure **??**. Despite our assumption that a lower number of actions might be beneficial there was no improvement visible. Breakout was stuck on the same level, regardless of any parameter changes. We additionally increased the population size to 100 for some runs and also tried to lower the selection pressure in the GA. Furthermore, we disabled the early-stopping condition. However, we obtained similar behaviour despite any changes. This result is consistent with Zhou and Feng (Zhou & Feng, 2019). However, their approach was not similar to ours, using an approach that is not gradient free. They used neuroevolution with an Variational Autoencoder and a Recurrent Neural Network. The autoencoder was trained with a convolutional network capturing the spatial component. Recurrent Neural Networks consider temporal components with having an additional (hidden) input from the preceding time step. They used smaller population sizes (32) with up to 30 evolution steps. Considering the breakout environment, they argue that their autoencoder fails to recognise the small ball (which is only a few pixels in width). Given that their autoencoder is based on convolutional networks this might be a reasonable explanation for the poor performance in our case. The result is also consistent with (Salimans et al., 2017). Their performance on Breakout is similar to what we observed. They used evo-

lutionary strategies (ES) for weight evolution. Evolutionary Algorithms apply selection and mutation, similar to a GA.

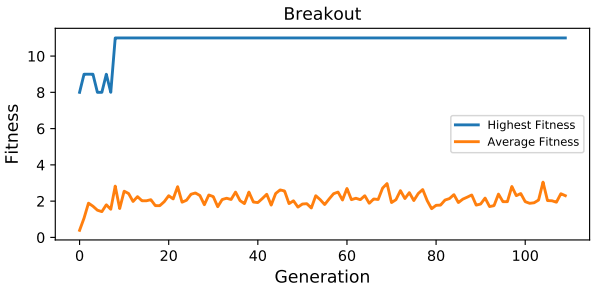

Figure 3. Breakout with population size 64. Number of generations stopped early. Learning rate 0.1 with Crossover

Experiments on the Frostbite environment began as more promising. Results improved in most cases only within the first 50 generations, and then remained at a score of approximately 170. An example of these results is shown in Figure 4. Similarly to Frostbite, experiments on the Kangaroo environment gave initial improvements in early generations. Results did not improve after a certain level, in the range of approximately 1200-1400. Figure 5 demonstrates an example of this behaviour. Once again, crossover did not show any significant impact on performance. Running the algorithm beyond 100 generations did not have any affect on this behaviour. When modifying the learning rate to above 0.1, both environments were quite invariant to learning rate changes. However, modifying the learning rate to below 0.1 resulted in observably worse performance. Additionally, pushing the learning rate to high and low extremes gave no change in the aforementioned behaviour.

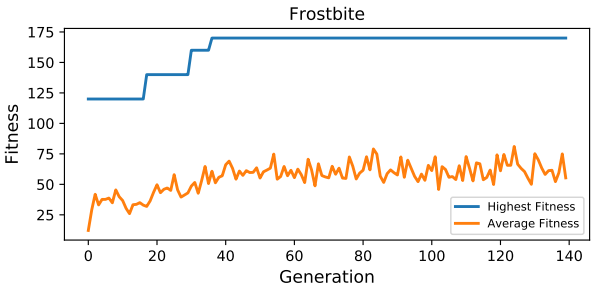

Figure 4. Frostbite with population size 64. Number of generations stopped early. Learning rate 0.1 with Crossover

Our results for Frostbite and Kangaroo were surprising, given that these environments produced much better results in the literature (Such et al., 2017; Salimans et al., 2017; Zhou & Feng, 2019). The explicit numbers are difficult to compare, given that those authors measured their performance with number of frames seen. Implementing the (Such et al., 2017; Salimans et al., 2017; Zhou & Feng, 2019) algorithms as baseline has not been feasible, given the overall time and computing power needed. As mentioned (Salimans et al., 2017; Zhou & Feng, 2019) had a

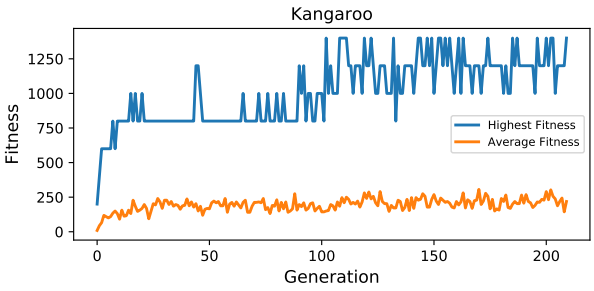

Figure 5. Frostbite with population size 64. Number of generations stopped early. Learning rate 0.1 with Crossover

different approach to (Such et al., 2017) and the approaches explored in this paper. So we therefore focus on comparing our results to (Such et al., 2017). They achieved breakthrough results for applying neuroevolution to RL. In Frostbite they achieved 4,536 points (1b frames) and 6,220 (6b frames). In Kangaroo they achieved 3,790 (1b frames) and 11,254 (6b frames). Our implementation does not perform as well. We hypothesise that the reason is related to our GA implementation and less related to the population size. However, the population size should not be discounted as a factor. A higher population size increases the amount of search space being covered at each generation. Such et al. used a simple GA by selecting the a single elite in the population. They further used a mutation operator with a fixed learning rate (chosen dependent on the environment). Furthermore, they did not apply crossover. Conversely, we used a slightly more complex version as described in Section 3. In consequence, we have been faced with a large parameter space. This is a common issue in GAs (Mitchell, 1998).

4.3. MuJoCo

We extended our algorithm to be applied to continuous control problems. In continous control problems the action space is not discrete as in Atari games. Such problems are more pertinent to application to real world problems. The MuJoCo environment suite(Todorov et al., 2012) contains many continuous control problems. An interface to these environments is provided by OpenAI Gym. Many recent state of the art RL algorithms use performance in MuJoCo environments as a metric for comparison. Compared to Atari games, the size of the vector state space of many MuJoCo environments is relatively small. The state vector has a length that ranges between 10 and 400 depending on the specific environment. We selected the Walker2d environment for our experiments. It provides a small state space of size 17, and an action vector of length 6. We used feedforward linear neural networks. Our initial experiments used an architecture of one hidden layer, consisting of 64 units. These values were chosen based on an estimate on the flexibility required, using the state space size and length of the action vector.

Our initial results were immediately promising, as evi-

denced in figure 6. We included a comparison between using crossover and without. Similarly to our experiments on Atari games, we found that crossover made little difference and even appeared to cause instability. It should also be noted that the average fitness is omitted in all figures from this point forward. This enables for easier comparison between different experiments in one graph.

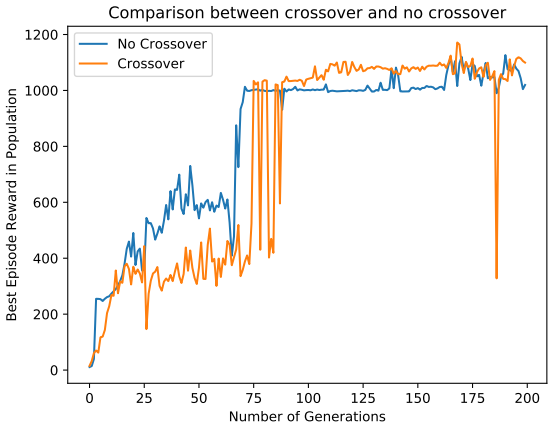

Figure 6. Walker2D using crossover and no crossover

Following from these initially promising results, we failed to make any significant progress when aiming to improve the highest score. A list of modifications and experiments we performed are as follows:

- We tried integrating the concept of ageism as previously described. Recent research in (Real et al., 2018) has shown success in using this concept in neuroevolution for classification tasks. The principle behind ageism is to continue incorporating diversity in the population. We included this in our MuJoCo implementation and tested using various values of the maximum age threshold. The results showed slower initial learning, and getting stuck at the same local optima of around 1000-1100 maximum score. Although further diversification was being included within the population, it did not seem to have any major benefit. The graph of performance can be seen in figure 9 in the appendix.
- We then experimented with different network architectures, aiming to introduce more flexibility. These results can be seen in figure 7. It can be seen that more flexibility did not improve the highest score, only the time taken to run the algorithm. This is due to a higher number of parameters. We additionally tried a less flexible network with one layer of 32 hidden units, producing poor results. This less flexible network failed to improve past a total reward of approximately 10.
- We then hypothesised that the learning rate may be the cause for the genetic algorithm failing to escape local optima. We initially tested a variety of fixed learning rates with no success. We then tested a dynamic learning rate, in which the learning rate decreases over time. The motivation behind this is based on the idea that past a certain point only small changes are needed. The strictly decreasing learning rate learned slightly faster than a static one. But the dynamic learning rate did not escape the local optima around 1000-1100 episode reward. This can be seen in figure 10 in the appendix.
- Although our aim was to experiment if lower population sizes could achieve similar results to existing literature, we tested a higher population. We increased the population size from 10 to 50. Although the initial learning (in terms of generations) was faster, again, the highest score did not improve. A graph of this can be seen in figure 10 in the appendix.

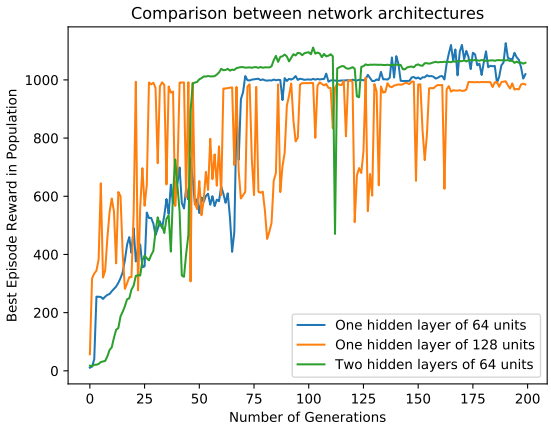

Figure 7. Walker2D using a variety of network architectures

Following from these unsuccessful modifications, we remained convinced that the learning rate was an important variable to investigate further. We implemented a hierarchical genetic algorithm to control an optimal value of the learning rate, since the optimal value may change throughout the course of learning. It is possible that the algorithm may require a high learning rate to escape a local optimum. On the other hand, a small learning rate might be beneficial to improve already superior candidates. Within the hierarchy, we evolved 10 sub-populations independently for a fixed number of generations. Each of those had its own learning rate. At each generation of the hierarchy, the learning rates of each sub population are mutated. We used the highest fitness of a network within a sub-population as the fitness of the sub-population itself. A tournament selection was used to select higher quality population with a higher probability. The mutation of the learning rate is then performed in a similar fashion as mutation of networks (adding normally distributed noise). This hierarchical procedure can be considered as analogous to meta-learning in deep learning. The results of this described hierarchical Genetic Algorithm are shown in figure 8. This algorithm took significantly more time to run than others, caused by 10 sub-populations evolving independently. In consequence, we chose to stop the algorithm after 100 generations. We observe significant improvement. The highest total reward was nearly doubled compared to previous results. Observa-

tions can be made about the inconsistency of the algorithm, as the fitness takes a sharp decrease and does not recover. This algorithm was developed and tested late in the project timeline, so therefore we did not have enough remaining time to optimise the hierarchical GA. A table of results of our implementations on the Walker2D environment is given in table 2. This table also contains a comparison to popular state of the art RL algorithms. The results for RL algorithms are taken from graphical results shown in the paper introducing PPO, (Real et al., 2018)). Additionally, we tested Proximal Policy Optimization (PPO) on the same hardware as we ran the genetic algorithms. We recorded the time taken for PPO to achieve the highest observed score. As far as we can tell, our hierarchical genetic algorithm is outperformed only by PPO on the Walker2d environment.

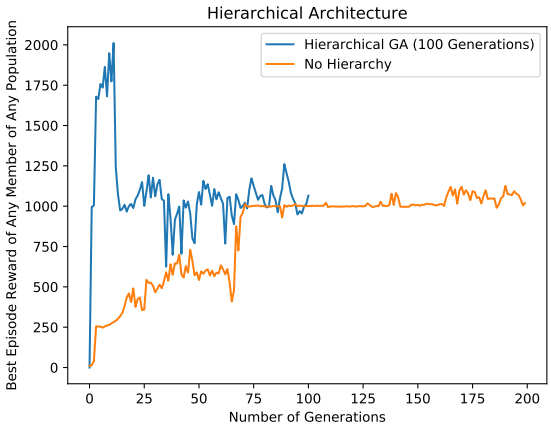

Figure 8. Walker2D using a GA hierarchy, against a single population

Algorithm	Highest Score	Time Taken
PPO (clipped policy change)	3780	~ 30 minutes
Hierarchical Neuroevolution	2012	~ 1 hour 30 minutes
Neuroevolution	1327	17 minutes
Vanilla PG, Adaptive	~1400	Unknown (not tested locally)
TRPO	~1100	Unknown (not tested locally)
A2C	~850	Unknown (not tested locally)

Table 2. Highest achieved episode rewards by a variety of algorithms on the Walker2d environment

5. Conclusions

Our first research question was to ask if a smaller population size could achieve similar results to those achieved in papers attempting similar tasks(Such et al., 2017). Ultimately when tested on Atari environments our results were of lower quality. This is partly due to a higher population size covering a wider area of the search space, but also likely to be caused by the parameters and methodologies chosen in our GA implementation. Our computing resources and available time were limited, hence extensive parameter tuning was not feasible. Our second research question asked if crossover has an effect on results. We discovered that crossover had very little effect either way on the quality of results. We consider our higher level of success with MuJoCo environments as being due to the number lower number of mutated parameters. Lastly, we asked if a hierarchical GA could be used to learn some of the parameters of our GA. This is analogous to how meta-learning can be used to learn parameters in deep-learning. Although we implemented this late into the project timeline and did not have time to further refine this implementation, we saw significant success. We achieved a highest episode reward of almost double of any other method applied previously.

Another result we found was that the neuroevolution algorithm only performed well on some environments. This is consistent with previous work (Such et al., 2017; Salimans et al., 2017; Zhou & Feng, 2019). We believe the No Free Lunch (NFL) theorem(Wolpert et al., 1997) applies here. A summarisation of the NFL algorithm can be given by that "any two optimization algorithms are equivalent when their performance is averaged across all possible problems"(Wolpert & Macready, 2005). In the context of this paper, this can be rephrased to state that no algorithm can perform optimally across all environments. Or, in other words, the neuroevolution algorithm cannot be expected to perform well across all environment domains. It will perform better on some (e.g. Frostbite, Kangaroo) and worse on others (e.g. Breakout).

Our best performing models using the hierarchical GA outperform many state of the art RL methods such as TRPO and A2C. To the best of our knowledge, only PPO outperforms our hierarchical GA for the Walker2d environment. PPO outperformed our neuroevolution in the metrics of highest reward and time taken. Furthermore, our methodologies have been relatively speedy on modest hardware. They are less computationally expensive than many RL methods, for example DQN and A3C.

5.1. Directions for Further Work

Despite PPO being state of the art for RL problems, neuroevolution remains a promising direction for research. The field is still relatively unexplored, and so further research may be fruitful. We particularly highlight the hierarchical GA as an area that is most promising to explore further. We achieved near state of the art performance without having time to optimally tune our algorithm. Additionally, Cartesian Genetic Programming (CGP)(Wilson et al., 2018) has also been shown to perform competitively to some traditional deep-RL algorithms such as DQN and A3C. Applying a hierarchical structure to CGP may also be an interesting direction for further work.

References

Blickle, Tobias and Thiele, Lothar. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput.*, 4(4):361–394, December 1996. ISSN 1063-6560. doi: 10.1162/evco.1996.4.4.361. URL http://dx.doi.org/10.1162/evco.1996.4.4.361.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hausknecht, Matthew, Lehman, Joel, Miikkulainen, Risto, and Stone, Peter. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

Miller, Brad L., Miller, Brad L., Goldberg, David E., and Goldberg, David E. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9: 193–212, 1995.

Mitchell, Melanie. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262631857.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

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

Montana, David J. and Davis, Lawrence. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pp. 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1623755.1623876.

Nielsen, Michael A. Neural networks and deep learning, 2015. URL http://neuralnetworksanddeeplearning.com/.

Real, Esteban, Aggarwal, Alok, Huang, Yanping, and Le, Quoc V. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

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

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

Such, Felipe Petroski, Madhavan, Vashisht, Conti, Edoardo, Lehman, Joel, Stanley, Kenneth O., and Clune, Jeff. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017. URL http://arxiv.org/abs/1712.06567.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Wilson, Dennis G, Cussat-Blanc, Sylvain, Luga, Hervé, and Miller, Julian F. Evolving simple programs for playing atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 229–236. ACM, 2018.

Wolpert, David H and Macready, William G. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, 2005.

Wolpert, David H, Macready, William G, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Zhou, Bin and Feng, Jiashi. Sample efficient deep neuroevolution in low dimensional latent space, 2019. URL https://openreview.net/forum?id=SkgE8sRcK7.

6. Appendix: Additonal Graphs

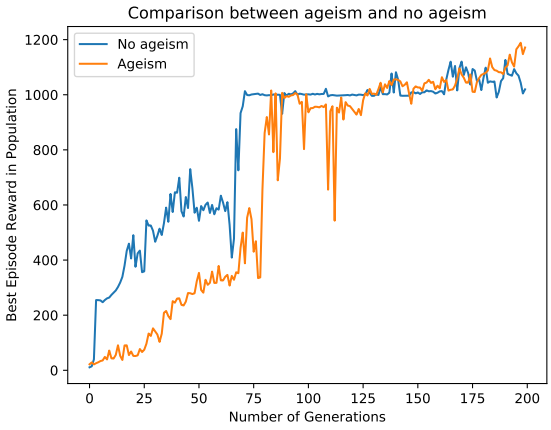

Figure 9. Walker2d environment with and without using ageism.

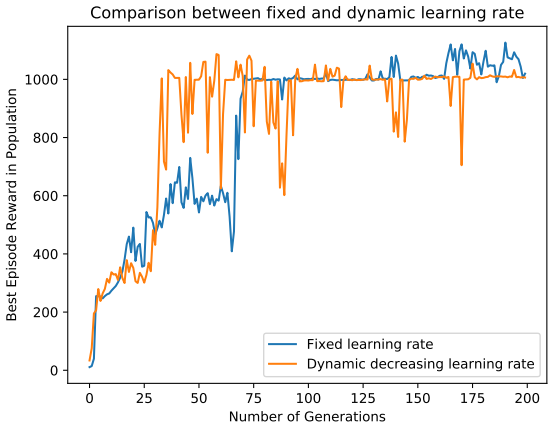

Figure 10. Walker2d environment with a fixed and with a dynamic learning rate (synonymous to learning rate).

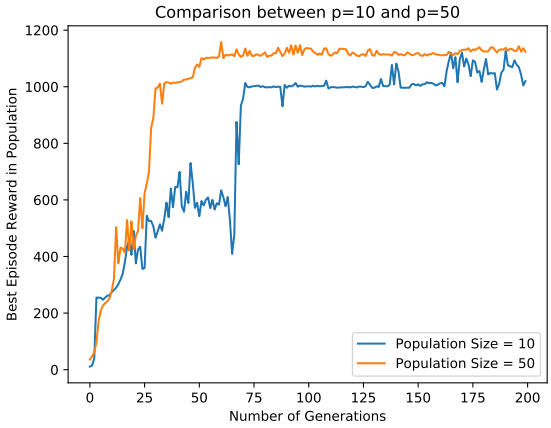

Figure 11. Walker2d environment using a population of size 10 and population of size 50.