Neuro-Evolution and Transfer Learning on OpenAIs Atari Environments

Submitted May 2022, for the fulfillment of
the conditions of the module: **Designing Intelligent Agents.**

Thomas Cotter
20160230

School of Computer Science
University of Nottingham

Contents

Chapter 1

Introduction

Neuroevolution is a practice where Deep Neural Networks (DNNs) are optimized by evolutionary algorithms, rather by the usual gradient-based learning algorithms. A paper by Such et al (2017, [1]), states that Genetic Algorithms (GAs) are a competitive alternative for training deep neural networks for reinforcement learning. Neuroevolution can be used to train the weights of a DNN. We can test how well neuroevolution performs in reinforcement learning by using OpenAI Atari environments; in particular, Space Invaders [1]. Furthermore, due to Space Invaders immense popularity there have been other games on the Atari which one could refer to as Space Invader "clones", which means the basic premise of the game is to move left and right, shooting the enemies above you to gain points. Two example of this that have identical controls would be Demon Attack [2] and Carnival [3]. Therefore, the question that arises is, would it be possible to train a DNN on Space Invaders and use that learning to produce a "good" performance on a Space Invader clone? Neuroevolution can help to solve this question as we can train the model without the need for a large labelled dataset, and we can use the final model on different enviroments.

1.1 Aim of the Project

The aim of this project is to answer the question: **Can we transfer the learning of Deep Neural Networks trained with neuroevolution techniques on one environment to another similar environment?**. This aim can broken down into some goals which are detailed here:

[1]https://gym.openai.com/envs/SpaceInvaders-v0/
[2]https://gym.openai.com/envs/DemonAttack-v0/
[3]https://gym.openai.com/envs/Carnival-v0/

- Train a DNN using a genetic algorithm to consistenly get high scores on Space Invaders.
- Use this DNN to measure the score of running it on a Space Invader clone.
- Measure how well the DNN performed by comparing it to other ways of selecting an action from a state.

For this project, I will be using Python to use the OpenAI enviroments, and TensorFlow to implement the DNN.

Chapter 2

Related Work

2.1 Neuro-Evolution

Agents in reinforcement learning (RL) tasks need to transform high-dimensional sensory inputs from the environment into an action that they should take. The most logical way to do this is with deep neural networks (DNNs) due to the number of parameters that can be trained to produce an output. The most prolific use of a DNN in RL was by Mnih et al (2015, [2]), in which a deep Q-network was created that could learn policies from high-dimensional sensory inputs. In recent years, hardware has improved drastically allowing for the weights of a neural network to be tuned with a genetic algorithm (GA). GAs were initially proposed by John Holland in a paper called Adaptation in Natural and Artificial Systems (1975, [3]). It is a search heuristic that is inspired by Charles Darwin's theory of natural selection. They involve 3 main processes: selection, crossover and mutation. Selection is the process of choosing the fittest (most suited to solve the task) individuals in a population and let them pass their genes to the next population. Crossover is the act of combining parent individuals into children. Mutation is when certain genes in certain offspring are mutated before being added to the next generation. These 3 processes simulate natural selection. The act of using GAs to train deep neural networks is called neuroevolution.

There has been a lot of work on neuroevolution in the machine learning field. Such et al demonstrated that the weights of a DNN can be evolved with a GA and it performs well on reinforcement learning tasks (2017, [1]). In this paper, the deep neural network had over four million parameters. These results suggest that following the gradient is not always the best choice for optimizing performance in reinforcement learning tasks. To evolve the neural network truncation selection was used, in which the top T individuals become the parents of the next generation. Gaussian mutation was also used, in which Gaussian noise was added to the parameter. Crossover was not included for simplicity in this case. With a population size of 1,000 the GA quickly converged on high performing individuals. Gomez et al (2006, [4]) also looked into the uses of neuroevolution in reinforcement learning tasks, in which the weights of the neural network were evolved to solve the Cart-Pole balancing task. Whilst this is a simple RL task, the neural network performed well. In a paper by Pawelczyk et al, titled Genetically Trained Deep Neural Networks (2018, [5]), is it stated that the DNNs trained with GAs consistently outperformed those trained with a classical method within the same time budget. This was tested on the computer vision task of classifying the MNIST dataset, and the models weights were updated via a GA. This suggests that genetic algorithms perform better than following the gradient in certain image recognition tasks too.

2.2 Transfer Learning

Transfer Learning is an important process in all aspects of machine learning. It is commonly used in deep convolution neural networks (CNN) which are used for computer vision tasks. The weights in the first few layers a CNN trained for one task can be reused for a second task. The CNN can be used to recognise high level features such as edges in images in the second task without the need to be trained again. Transfer learning can also be used in RL, Lazaric (2013, [6]) showed that whenever the tasks are similar, the transferred knowledge can be used by a learning algorithm to solve the target task and significantly improve it's performance. The question that arises is how similar do those environments need to be and are the environments selected for this project too different?

Chapter 3

Design

OpenAI Gym is a toolkit for reinforcement learning research. It includes a growing collection of benchmark problems that expose a common interface (2016, [7]). OpenAI Gym focuses on the episodic setting of reinforcement learning, where the agent's experience is broken down into a series of episodes. In each episode the agent's initial state is randomly sampled from a distribution, and the interaction proceeds until the environment reaches a terminal state. OpenAI Gym does not include a built-in agent, so this will have to be built from scratch.

The agents used to solve this task will be deep neural networks. These will be built using TensorFlow, and end-to-end open source platform for machine learning. The agents will follow a similar architecture to the ones used in the paper by Mnih et al (2015, [2]). The agents DNN will contain 2 convolutional layers, a fully connected layer and a dropout layer in order to reduce overfitting on the environment which it was trained on. The exact parameters of each of the layers will be tuned during the implementation into to optimize the learning.

The selection, crossover and mutation processes will also have to be designed. For selection, tournament selection will be use, as well as elitism. Tournament selection is when a sample of the entire population is chosen to perform a tournament, i.e the fitness of the individual, and the best individual is returned. Half the population will be sampled and tournament selection performed on the sample in order to generate the parents for crossover. Elitism is where the top N individuals in terms of fitness will be automatically passed to the next generation. Different values of N will be experimented with to determine the most optimal value, however it will likely be between 2 and 4. For crossover, uniform crossover will be used. This can be performed in two ways, shown in Table 3.1.

Option	**Algorithm**
Option 1	For each weight in a layer, check if crossover should occur (chance < crossover_rate) and crossover
Option 2	For each entire layer, check if crossover should occur (chance < crossover_rate) and crossover

Table 3.1: Possible CR Operators

Problems could arise could arise by choosing option 2. This is because by swapping an entire layer, too much change has occurred and the model is likely to not perform well. Therefore, option 1 will be used as the crossover operator in this project, The crossover rate will be experimented with, however the range of experimentation will be [0.1,0.4]. Mutation is used to maintain and introduce diversity into the population. The mutation rate should be initially large, to increase the exploration across the search space before gradually decreasing in order to maximise the exploitation on the found local minima. This local minima should hopefully be the global minima as the exploration rate was high at the start of the GA. To implement this, the mutation rate will follow this proportionality: $mr \propto \frac{1}{\sqrt{g}}$, where g is the current generation number. The possible set of mutation operators to be experimented with can be seen in Table 3.2. The operation would be, for each weight if the a randomly generated number between 0 and 1, is less than the mutation rate, then perform the mutation operator on the weight.

Option	**Algorithm**
Option 1	Each weight is replaced with a random value (within certain constraints)
Option 2	Each weight is changed by a random percentage (performed by multiplying each weight by a random value between 0.5 and 1.5)
Option 3	Each weight has its sign flipped.

Table 3.2: Possible Mutation Operators

The population should converge on weights that consistently score high scores in Space Invaders. Then, these networks can be taken and tested on the other two environments. In order to test the transfer learning, the resulting score from running the model on the new environment could be used. This score can be compared against taking steps at random in the environment until termination. The time constraints on this project might result in a smaller improvement than expected, due to the time it takes to train a RL DNN model, but if positive results can be produced in a small time-frame then even better results could be produced with more time

Chapter 4

Implementation

4.1 Agent

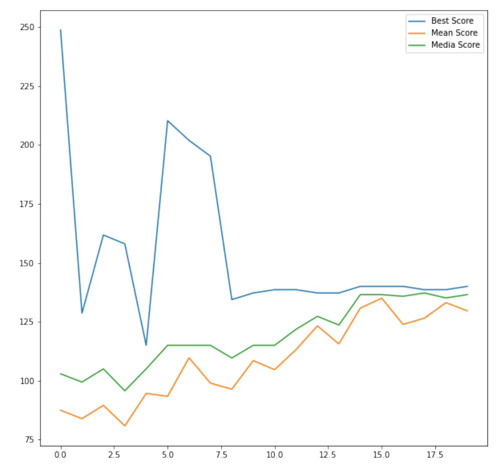
The agent was constructed in a similar manner to Mnih et al agents (2015, [2]). However, the agents in this project had fewer parameters due to time constraints. The agents contained 2 convolutional layers, 1 dense layer and a single dropoout layer.

4.2 Genetic Algorithm

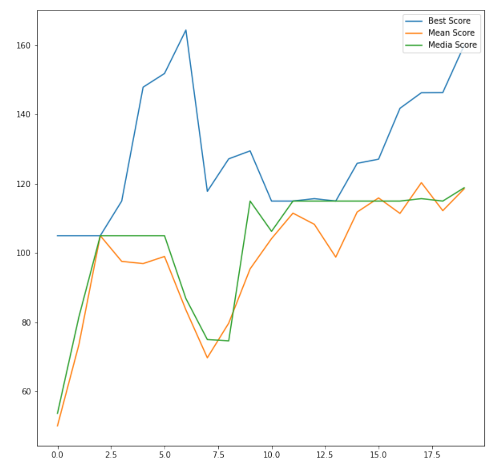
The genetic algorithm was performed by a Runner class, which could take an environment and the current generation as inputs to __init__. The runner class also applied the OpenAI function WarpFrame to the input environment. This means that all the observations on an environment within the runner class will have the shape (84,84,1). This means we can use the same deep neural network for each environment, allowing transfer learning to be utilised.

The score for each agent per generation was not just it's reward from a single play-through of Space Invaders. Instead, per generation, each agent "played" Space Invaders N times. N varied throughout the testing process to speed up testing, however N was always in the range [5,20] and N will be specified whenever a figure is displaying a set of results. These "play-throughs" can also be defined as episodes. The score for each episode was recorded, and then final score for the agent can be calculate using this equation: $0.3(median(r)) + 0.7(mean(r))$, where r is the set of scores for the agent. The median was used as the scores were often skewed by a large outlier which greatly affected the mean. This means the algorithm targets more consistent game playing. However, the mean is still useful to try to exploit certain large outliers if possible, or perhaps an agent produced good scores and bad scores at an even ratio, so the mean was used to continue evolving some of these agents. Just using the median often lead to slow improvements as the models would converge on the median score for the game, rather than trying to get a higher score.

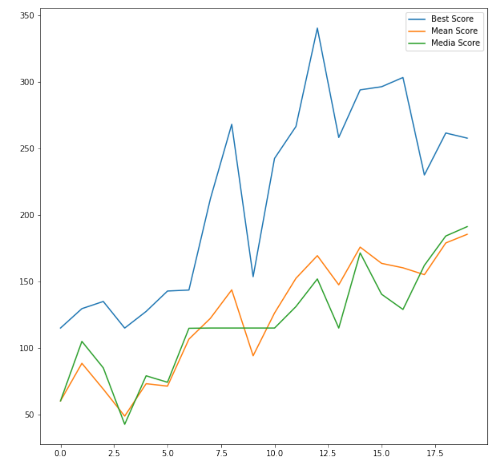
Experiments were then performed to determine the optimal crossover and mutation rates. To perform these, the selected crossover operator was Option 1, and the selected mutation operator was Option 2. Figure 4.1[1] shows these results. It shows the best score from the set agents, as well as the mean and median of the set of scores per generation. For all of these results, there was 10 episodes per agent per generation. Each episode was limited to 1 life to speed up testing. From these results, CR was chosen to be 0.4, and then Initial MR was chosen to be 0.3. Furthermore, the best score and median score did not converge after 20 generations, so that suggests that there is still improvements to be made.



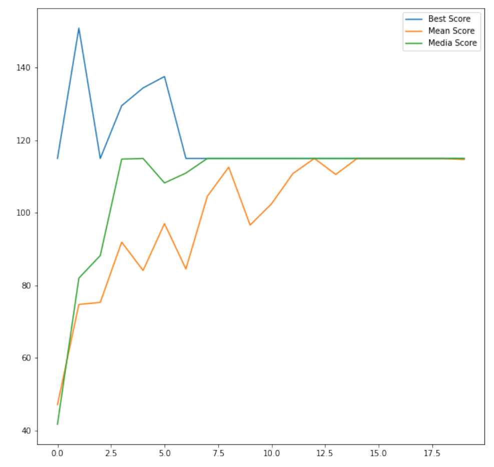
(a) CR = 0.3, Initial MR=0.2



(b) CR = 0.2, Initial MR=0.4



(c) CR = 0.4, Initial MR=0.3

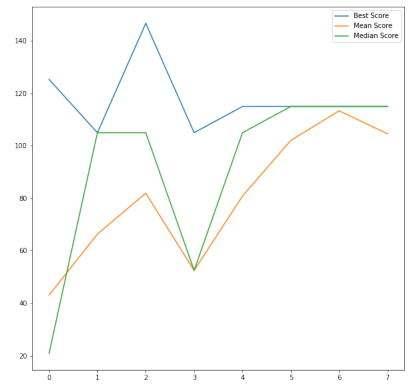


(d) CR = 0.4, Initial MR=0.2

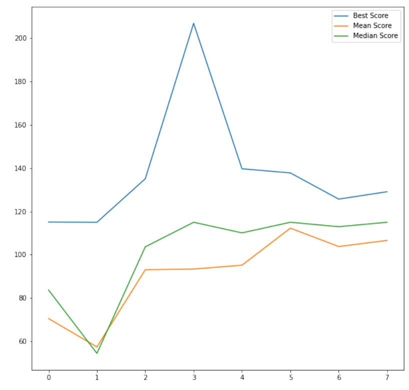
Figure 4.1: CR and MR Testing (x-axis - Current Generation, y-axis - Score)

[1]While this key in this figure states that the green line is the media score, it is in fact the median score. Each of these graphs too a long time to produce and I did not notice till the end of producing all 4.

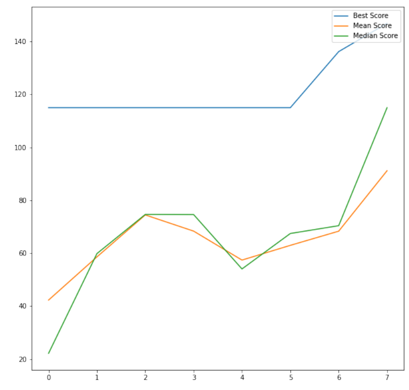
Using these determined rates both the mutation and crossover operators were tested. Initially, Option 2 from Table 3.1 for crossover was tested. This is where an entire layer was chosen to swap with both parents, if the chance was less than the crossover rate. The results of this coincided with my initial thoughts, and the models being used are likely too small for this option to be viable. Therefore, Option 1 from Table 3.1 was chosen, which was to randomly select an individual weight for crossover between two parents. For mutation, I tested the 3 different operators from Table 3.2, the resulting output graphs can be see in Fig. 4.2. For these results, there was 5 episodes per agent per generation, and each episode was limited to 1 life. The total number of generations was limited to 8 due to time constraints. The final mutation operator was chosen to be Option 2 (where each weight is changed by a random percentage), as this produced the most consistent results and largest scores.



(a) Option 1



(b) Option 2



(c) Option 3

Figure 4.2: Best Mutation Operator Testing - (x-axis - Current Generation, y-axis - Score)

4.3 Transfer Learning

The transfer learning can be performed by initiating another instance of the Runner class with a new environment. The best agent from the final execution of the GA can be passed into this Runner to compare this agent with an agent that just takes random moves at each step. We can perfom this over N episodes, where N will be defined depending on the time constraints, and compare the results.

All code for this project can be found in this repository - https://github.com/tomcotter7/OpenAI-GeneticAlgorithms

Chapter 5

Evaluation

5.1 Training on Space Invaders

Using the determined mutation and crossover operators, with the initial mr = 0.3, and cr = 0.35, the final set of models were trained. The CR is slightly decreased from the implementation chapter as this produces slightly better results. In this case, each agent had 14 episodes on the environment per generation, but the number of lives were still limited to 1. This reduced the training time, without too much of a deficit to the quality of the training. This was tested for 25 generations, and the population contained 20 agents. The training curve can be seen in Fig. 5.1, and how well the model performed compared to an agent which performed random action at each step for the Space Invaders game can be seen in Table 5.1. To obtain these results both the best agent and the random agent had 14 episodes with the environment. We can see from this table that neuro-evolution with these parameters resulted in a 400% increase in the score of the agent on Space Invaders, something which could increase with further training and a larger number of parameters in the model.

	Mean Score	Median Score	Best Score
Trained Agent Results	552.0	625.0	670.0
Random Action Agent Results	153.0	155.0	205.0

Table 5.1: Space Invaders - Trained Agent vs Random Action Agent

5.2 Transfer Learning

The comparison between an agent performing random actions and the best agent taken from the final generation from the Space Invaders training for both new environments can be seen in Table.

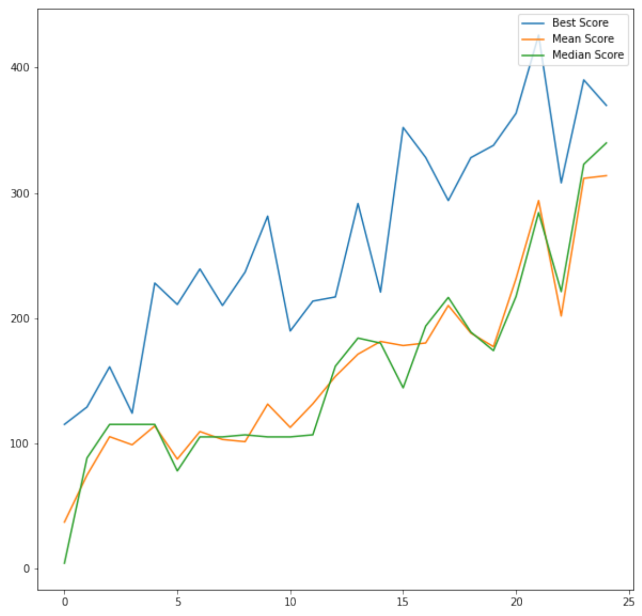


Figure 5.1: Final Neuro-Evolution Training Graph (x-axis - current generation. y-axis - score)

5.2. Again, for both environments both the random agent and pre-trained agent experienced the environment 14 times. From these results we can say that utilising transfer learning for RL does not produce a significant improvement over using a random agent. In both Demon Attack and Carnival, the transfer learned agents did not perform better than a random agent at all. Whilst, in theory the agent should aim to shoot anything floating in the sky, perhaps the intricacies of the games were too different from Space Invaders. It is because of this that the DNN is unlikely to be able to utilise it's previously learnt knowledge to new environments. The initial question posed was **can we transfer the learning of DNNs trained with neuro-evolution techniques on one environment to another similar environment?** The answer is not with these environments. Space Invaders is too different from Carnival and Demon Attack, and in fact, a random agent outperformed the trained agent on the new environments, meaning that the transfer learning actually had an adverse affect.

	Mean Score	Median Score	Best Score
Demon Attack Random Agent	181.786	175.0	385.0
Demon Attack Pre-Trained Model	83.2	70.0	235.0
Carnival Random Agent	717.1	690.0	1120
Carnival Model Results	410.0	410.0	620.0

Table 5.2: Demon Attack and Carnival, Random vs Trained Agent Results

Chapter 6

Summary and Reflections

6.1 Summary of Work

The project successfully tested both the outcome of using neuro-evolution on Space Invaders and the transfer of that learning into other similar Atari game environments. Whilst the project did not produce the results I had hoped for, the confirmation that RL algorithms can in fact transfer their learning, I still produced a detailed and well thought out project. The main restriction for this project was the time-constrains, which proved difficult to stick to especially given the time it takes to train the agents via neuro-evolution. Furthermore, agents trained in this way can only produce outputs that correspond to discrete actions. Many RL tasks nowadays use a continuous action space, such as the ShadowHand environments from OpenAI. These are robotics hands that can perform simple tasks, which could in theory be trained using neuroevolution, but their action space is continuous. Apart from these two limitations, the project was successful, I learnt a lot more about the frameworks for RL tasks and gained more experience in Tensorflow, which I enjoyed.

6.2 Future Work

This project did not show evidence of transfer learning being significantly viable in RL tasks, when neuro-evolution is used. However, when transfer learning is utilised in computer vision tasks, it is often a base understanding of the image that is transferred, and the rest of the layers are then retrained to fit the new problem. This could be viable in RL tasks too. If I had more time, I would like to compare using neuro-evolution on a environment with two initial generations, the first one is a random set of models, and the second is the final generation from neuro-evolution on a second similar environment. We could then see if the two generations converge to an optima at different rates, perhaps the one with the previously learnt knowledge converging faster due to its previous knowledge of how to move about the environment. I would also like to test the transfer learning capabilities of different types of algorithms, such as DDPG or TD3 to compare with neuro-evolution.

Bibliography

[1] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” 2017.

[2] V. Mnih, K. Kavukcouglum, and D. Silver, “Human level control through deep reinforcement learning,” 2015.

[3] J. Holland, “Adaptation in natural and artificial systems,” 1975.

[4] F. Gomez, J. Schmidhuber, and R. Miikkulainen, “Efficient non-linear control through neuroevolution,” pp. 654–662, 2006.

[5] K. Pawelczyk, M. Kawulok, and J. Nalepa, “Genetically-trained deep neural networks,” 2018.

[6] A. Lazaric, “Transfer in reinforcement learning: a framework and a survey,” 2013.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016.