Evolving Neural Network Agents to Play Atari Games with Compact State Representations

Adam Tupper
University of Canterbury
Christchurch, New Zealand
adam.tupper@pg.canterbury.ac.nz

Kourosh Neshatian
University of Canterbury
Christchurch, New Zealand
kourosh.neshatian@canterbury.ac.nz

ABSTRACT

Recent success in solving hard reinforcement learning problems can be largely credited to the use of deep neural networks which can extract high-level features and learn compact state representations from high-dimensional inputs, such as images. However, the large networks required to learn both state representation and policy using this approach limit the effectiveness and benefits of neuroevolution methods, in particular, methods that evolve both weights and topology. Such methods have proven effective at solving simpler problems in the past. One potential solution to this problem is to separate state representation and policy learning and apply neuroevolution only to the latter. We investigate the plausibility of this approach by assessing the ability to evolve small policy networks for Atari games, from compact, high-quality state representations similar to those we might expect to be learned via state representation learning. Specifically, we evolve policy networks using NEAT that take, as input, compact state representations provided by the recently released Atari Annotated RAM Interface (Atari ARI). Our results show that it is possible to evolve agents that exceed expert human performance from such compact state representations. Furthermore, we show that for some games, successful policy networks can be evolved that contain only a few or even no hidden nodes.

KEYWORDS

neuroevolution, reinforcement learning, video games

1 INTRODUCTION

Prior to the use of deep neural networks for reinforcement learning, what is now commonly referred to as "deep reinforcement learning", the only solvable problems were those for which low-dimensional, high-quality state representations could be constructed. One example is the classic pole balancing problem [5]. Now, deep reinforcement learning has led to success in solving more complex problems, with higher-dimensional state representations. Common benchmarks for new algorithms include videos games [3] and learning motor skills using the MuJoCo physics simulator [21]. Deep reinforcement learning methods solve such tasks by combining feature extraction and state representation learning with policy learning. This end-to-end learning approach, where the state representation and policy are learnt simultaneously, removes the need for feature engineering, but not without missing some of the benefits offered by neuroevolution approaches that have been successful in the past.

One distinct advantage of neuroevolution is the ability to optimise both the weights and topology of neural networks [22]. This removes the need for time-consuming architecture design and can lead to finding more compact and creative architectures than if they were human-designed. Such networks would have the advantage of being more memory efficient and faster at inference time, which are particularly useful advantages for embedded reinforcement learning applications, such as robotic control.

Another benefit of neuroevolution is greater exploration of the policy space. When using gradient-based methods, the reward signal may discourage the agent from learning behaviours that lead to an optimal policy [19]. Furthermore, the common use of shaped rewards to alleviate the credit assignment problem [20] can often result in a policy that does not correspond to the problem we actually want the agent to solve [19]. With gradient-free evolutionary algorithms, we can avoid these issues altogether. Since we do not use the gradient of the reward signal, we do not risk following it to locally optimal behaviour. We also avoid the credit assignment problem and the need for potentially deceptive shaped reward functions by evaluating agents only at the end of each episode.

Unfortunately, the large networks required for end-to-end learning rule out many topology and weight evolving neuroevolution algorithms, such as NEAT [18], that are very effective at evolving solutions for simpler domains. Although deep neuroevolution algorithms have been shown to be competitive with gradient-based approaches for some tasks [8, 17, 19], it is apparent that the increased size of the networks required for end-to-end learning limits their effectiveness.

One potential solution that could allow us to exploit the benefits of deep neural networks and evolutionary reinforcement learning is to separate end-to-end learning into two components: state representation learning and policy learning. With this approach, gradient-based methods, such as auto-encoders, might be used for feature extraction and for learning condensed state representations, enabling evolutionary methods to be used for policy learning.

In this paper, we investigate the plausibility of evolving agents that are able to perform tasks from sufficiently small and high-quality representations, similar to those we might expect to be learned via state representation learning. To assess this, we use NEAT to evolve policies for playing Atari 2600 games from the recently released Atari Annotated RAM Interface (Atari ARI) [2]. This interface provides RAM annotations that identify the specific bytes that are useful in learning to play the game, reducing the size of the input space by up to 93% when compared to using the entire contents of RAM.

2 RELATED WORK

Prior work on evolving game playing agents can be broadly grouped into two categories: those that attempt to learn from low-level features, and those that attempt to learn from high-level features. Although our focus is on approaches that have been tested on the

suite of Atari 2600 games, we discuss work on other games where appropriate.

2.1 Learning from Low-Level Features

The most common low-level features are raw or pre-processed versions of the images that would ordinarily be displayed on screen. This has the benefit of presenting information in the same or a similar fashion to how it is displayed to humans, allowing for the ultimate test of the agents ability to extract important features, understand, and reason about the environment. However, this poses a problem when using neuroevolution because large, deep neural networks are required for learning relevant features from such a high-dimensional feature space. Despite this, both simple genetic algorithms [19] and evolution strategies [17] have been shown to produce agents that perform competitively against gradient-based learning methods, such as Deep Q-Learning (DQN) [14] and Asynchronous Advantage Actor-Critic (A3C) [13], when evolving weights alone. In both cases, the weights of the original convolutional neural network designed for Deep Q-Learning were evolved.

As far as we are aware, the only method that has successfully evolved both the weights and topology of neural network agents that can play Atari games from low-level pixel inputs is HyperNEAT [8]. HyperNEAT achieves this using an indirect encoding scheme, connectivity pattern producing networks (CPPNs). Instead of evolving large game-playing networks directly, the more compact CPPNs that encode them are evolved. The use of a more compact indirect encoding greatly reduces the search space and number of parameters that need to be tuned, making it feasible to evolve the weights and topology of large networks. However, using NEAT to evolve smaller networks from more compact representations has been shown to produce more performant networks than using HyperNEAT to evolve larger networks from larger representations [8] (explained in more detail in the next section).

The greatest limitation of learning from high-level features is, with the exception of HyperNEAT, the inability to effectively evolve both weights and topology. As mentioned in the introduction, by searching for policies in the weight space alone, we limit our policy search and rely on hand-crafted architectures that may hinder our search. Furthermore, although HyperNEAT is able to find good policies when using down-sampled pixel inputs or a higher-level representation, NEAT can find better polices when using a higher-level representation. These results motivate our decision to use NEAT to evolve agents in this work.

2.2 Learning from High-Level Features

Learning from high-level features removes some of the complexity of learning, by providing a more concise set of features that are immediately useful for learning the game. Such features might be manually constructed, or ground-truth state information might be obtained from the game engine. Evolving agents from high-level features has been successfully applied to Atari games, as well as others, including NERO [10] and Super Mario [9].

Prior to the release of the Atari ARI, the highest-level state representation for Atari games was the object class representation developed by Hausknecht et al. [8]. This was constructed by manually capturing images of different objects in each game (e.g. the paddles and ball in Pong) and matching these to the objects displayed in the images during game play. The location of each object within the same class is represented by a position on a 10×8 grid, referred to as a substrate. These substrates are then fed as input to the network, with the number of substrates equal to the number of types (classes) of objects in the game. Across a subset of 61 Atari games, NEAT was found to perform better at learning from this representation than HyperNEAT, CMA-ES and a simple genetic algorithm, highlighting the usefulness of topological evolution [8].

Despite the improvement over raw images, there are several limitations of the object class representation. First, for some games, the representations are still relatively high-dimensional. For example, the objects in Pong belong to a minimum of two object classes, one for paddles and the other for the ball. Therefore, two substrates are required to encode all of the object locations, totalling 160 dimensions. However, this information could be encoded in as little as six dimensions using x and y coordinates. Second, useful information that is not classified as gameplay objects, such as agent and enemy scores, or the number of lives remaining, are not included. Such information could influence the strategy of agents. For example, for the optimal strategy, an agent may want to play conservatively or aggressively depending on the number of lives it has remaining, to maximise the reward. Third, if objects are partially occluded or slightly different than the stored versions due to animation, they will not be detected. Finally, for some games, new objects, such as enemies are introduced during the latter stages of the game. Once the agent surpasses the level achieved by the humans who complied the object database, new objects appearing from this point onward will not be identified and displayed to the agent, and the agent will act unaware of their presence. These issues are addressed by the Atari ARI, particularly for games with *good* representations, as we discuss later.

As well as manually-created high-level representations, researchers have also attempted to evolve agents from *learned* high-level representations. While these methods have yet to be tested on a general game-playing benchmark, such as the suite of Atari games, they provide evidence of the successful separation of state representation and policy learning. Poulsen et al. [16] trained a convolutional neural network (CNN) to extract the positions of the agent and a target in a first-person shooter (FPS) game in a supervised manner, using information from the game engine. They then used NEAT to evolve an agent from this representation. Similarly, for a racing game, Koutnik et al. [11] learned high-level features using a CNN that was trained to maximise the variation in the representation of different images, before using CoSyNE [6] to evolve the weights of a simple recurrent neural network for policy learning. Using a different unsupervised approach, Alvernaz and Togelius [1] used an auto-encoder to learn compressed representations of the game state from images for a health pack gathering task in the FPS game Doom. Both policy and feature learning were performed simultaneously, using images selected from the experiences of agents evolved using CMA-ES [7] to refine the auto-encoder.

2.3 Our Motivations and Rationale

Although evolutionary algorithms have been successfully used to optimise the weights of deep end-to-end learning networks, this has

been at the cost of abandoning topology optimisation. An exception to this is HyperNEAT, yet NEAT has been shown to produce better agents when using more compact state representations. Furthermore, separated state representation and policy learning, using evolutionary algorithms to evolve policy networks, has proven successful for individual domains other than Atari games.

This work distinguishes itself by evaluating the ability to evolve Atari agents from a new high-level, low-dimensional state representation, provided by the recently released Atari Annotated RAM Interface (Atari ARI). As far as we are aware, this is the most compact, high-quality set of features available, and the closest to the intrinsic dimensionality of the games themselves. By removing the need to learn lower level features, our work provides a closer assessment of the ability of to evolve agents based on the difficultly of the games themselves, independent of the state representation learning technique used.

3 ATARI ANNOTATED RAM INTERFACE

The Atari ARI [2] provides RAM annotations for 22 of the Atari 2600 games supported by the OpenAI Gym toolkit[1]. These annotations identify which of the 128 bytes of RAM store values related to information displayed on screen, such as the position of the player. These annotations were created by analysing commented disassemblies or source code for each game. Through a wrapper for the existing OpenAI Gym interfaces for each supported Atari game, the values of the RAM for each state variable are made available at each time step. For one of these games, Pong, the values provided at each time step are shown in Fig 1.

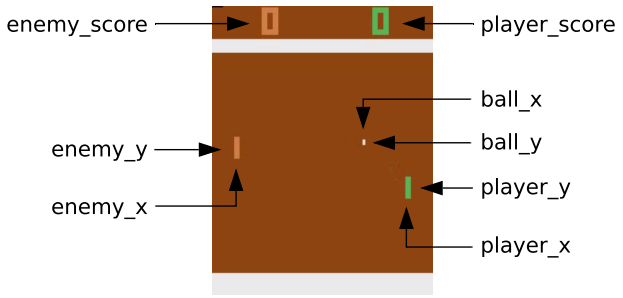

Figure 1: The RAM annotations provided by the Atari Annotated RAM Interface (Atari ARI) for the game of Pong.

The interface was originally designed as a benchmark for assessing the performance of state representation learning methods that try to learn condensed representations of the game state, usually from the images displayed on screen. Such methods are assessed using the Atari ARI by measuring how well they are able to learn representations that linearly separate the identified variables. As such, it is interesting to evaluate how well policies can be learnt from these "ground-truth" variables. By filtering out unimportant RAM values, the input space for each game is substantially reduced, removing excess noise that might otherwise make the task of policy learning more challenging. For instance, for Pong, the input space is reduced to only eight inputs, a reduction of 93%. To our knowledge, the Atari ARI provides the most condensed, high-quality state representation for many of the supported Atari 2600 games, and learning from this representation has not previously been evaluated.

[1] At the time that this experimental work was conducted.

4 METHOD

This section describes several important aspects of our method for evolving agents to play Atari games using the Atari ARI. We discuss our implementation of NEAT and our rationale behind our game selection. Details of our experimental setup, including the choice of hyperparameters, follows in Section 5.

4.1 Proposed Neuroevolution

As discussed previously, the algorithm chosen for evolving neural networks is Neuro-Evolution of Augmenting Topologies (NEAT) [18]. This algorithm evolves both the weights and topology of networks, starting from minimal structures consisting only of the input and output nodes. Our Python implementation of NEAT conforms to the NEAT-Python interface [12], to allow for the reuse of existing reporting and logging modules. To parallelise the fitness evaluations of the population, our implementation uses Ray [15].

Our implementation of NEAT follows the original description of the algorithm. However, we make several modifications to better suit the domain of Atari games. Additionally, there are several aspects where our implementation may diverge from the original due to differences in interpretation. These differences are briefly described below.

4.1.1 Recurrent neural network behaviour. Due to the presence of moving objects, most Atari games, including Pong and Breakout, are partially observable Markov decision processes (POMDPs) when only information from the current frame is provided with each observation. This is because the speed and direction of moving objects cannot be determined from a single observation. To alleviate this problem, a common solution is to include information from the four most recent frames in each observation and train feed-forward networks [14]. However, this increases the dimensionality of the input space and does not solve the partial observability problem for games that require a memory of more than four time steps. Instead, we evolve recurrent neural networks (RNNs) to enable the development of good policies in partially observable environments.

We evolve time-delayed RNNs that at each time step propagate the outputs of each neuron forward. This behaviour is illustrated in Fig 2. Compared to some RNN implementations, the initial outputs are not immediately influenced by the inputs, but they do not require there to be topological ordering of nodes or the specification of whether lateral connections are recurrent. For situations like ours, where the network topology is evolved, rather than handcrafted, this reduces the constraints that need to be specified and overhead in implementation.

4.1.2 Negative fitness values. To better fit the Atari domain, we modify NEAT to support negative fitness values. The original specification of NEAT does not allow for negative values, because positive values are required for fitness sharing. To avoid this issue, fitness sharing is implemented using an adjusted fitness value,

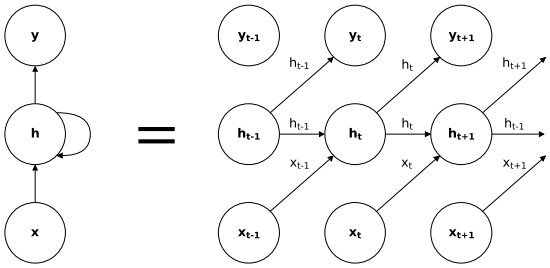

Figure 2: An unrolled example of the behaviour of our time-delayed RNN implementation.

specified as the difference between the individual's fitness and the lowest fitness in the population. This ensures positive values and preserves the fitness differences between individuals.

4.1.3 Explicit biases. The original version of NEAT does not include bias parameters for each node. Instead, evolution decides which nodes need biases by connecting them to "bias" nodes. Although this approach reduces the search space, we include bias parameters for all nodes as this approach suits our RNN implementation. The node biases are mutated following the same mutation scheme as connection weights. It should be noted that while all biases are present to begin with, they can be mutated to zero during evolution to effectively remove them if it is found to be beneficial to the network.

4.2 Game Selection

The Atari ARI provides gym wrappers for 22 different Atari games. However, inspection of the objectives of different games reveals inadequacies in some of the state representations provided. Due to this, we use only a subset of the supported games in our evaluations.

We categorise the supported games into three categories: *poor*, *fair* and *good*; based on the perceived quality and completeness of the state representations provided by the Atari ARI. Representations that lack key information relevant to the objective of the game, that is so crucial that we do not reasonably expect an agent to be able to learn a good strategy from, are classified as *poor*. Representations that lack some information about the state of the game, but appear to include enough information to develop an adequate strategy from, are classified as *fair*. Finally, games for which the representation appears to include all information required to learn an optimal strategy, generally speaking information about all game play components displayed on screen, are classified as *good*. Each game is classified by examining game play and comparing the knowledge required to play the game against the information provided by the Atari ARI.

One example of a *poor* game is Riverraid. The Atari ARI representation is classified as *poor* for this game, because only information regarding the state of the player (e.g. x position, remaining fuel) is provided at each time step. The representation does not include any information about the state of enemies and obstacles, or the locations of fuel tanks for refuelling. Because this information is not provided, we find it unreasonable to expect the agent to develop an adequate strategy, given that they must avoid the obstacles and refuel to survive. An example of a *fair* game is Video Pinball, which includes information on the location of the ball and position of the paddle, allowing the agent to keep the ball in play, but it does not include information on the targets the agent needs to aim for to score points. Finally, an example of a *good* representation is the representation provided for Pong, which includes information about all of the objects displayed in each frame. The state representation for Pong is illustrated in Fig 1.

Of the 22 supported games, eight games have *poor* representations (Hero, Montezuma's Revenge, Pitfall, Private Eye, Qbert, Riverraid, Venture and Yars Revenge), eight games have *fair* representations (Berzerk, Breakout, Demon Attack, Frostbite, Ms. Pacman, Seaquest, Space Invaders and Video Pinball) and six games have *good* representations (Asteroids, Bowling, Boxing, Freeway, Pong and Tennis).

5 EXPERIMENTAL SETUP

We follow a similar experimental setup to other works that have trained Atari-playing agents using evolutionary algorithms [8, 17, 19]. The source code for our implementation of NEAT and for replicating our experiments can be found at <URL removed, but an anonymised version is included with the supplementary material>.

5.1 Environment Setup

The OpenAI Gym [4] Atari environments provide an interface for the Arcade Learning Environment [3]. As previously described, the Atari ARI provides wrappers for each supported Gym environment that provide a compact state representation at each time step.

For each game, we normalise the state variables provided by the Atari ARI to values within the range [0, 1]. These values are used as the inputs to the neural network agents. Normalisation is achieved by dividing each state variable by 255. This is guaranteed to produce a value within the range [0, 1] because each state variable represents a byte of RAM, with a value within the range [0, 255].

The number of input and output nodes varies between games due to the size of the state representation provided by the Atari ARI and the set of legal actions the agent is allowed to perform. All other hyperparameters are held constant across all games.

5.2 Hyperparameter Selection

Given the number of hyperparameters, and the time required to perform evolutionary runs, a grid search for optimal hyperparameter values is infeasible. Therefore, our hyperparameters are initialised based on prior studies that have evolved Atari-playing agents using different approaches [8, 17, 19]. We empirically refined these values through informal experimentation on a subset of three games: Asteroids, Boxing and Pong. The single set of values chosen from these games is then used for our formal experiments across all games. This subset of games was chosen because they all have good representations, considerably different gameplay mechanics, and cover the spectrum of initial network sizes (due to the size of their input and output spaces). Considering games that require different network sizes is important to ensure that the speciation mechanism of NEAT adequately speciates the populations for each game.

Table 1: A summary of the experimental parameters used in our experiments.

Evaluation Parameters	
Max Frames per Episode	20,000
Action Set	Legal Actions
Min Action Set Size	3
Max Action Set Size	18
Min Input Set Size	6
Max Input Set Size	41
NEAT Parameters	
Generations	200
Population Size	130
Add Node Probability	0.03
Add Connection Probability	0.05
Mutate Weight Probability	0.8
Activation Function	Sigmoid

The initial population of networks are fully connected with no hidden nodes. All weights and biases are initialised using a uniform distribution with a range of [-3, 3]. Mutations for weights and biases are drawn from a uniform distribution with a range of [-0.05, 0.05]. A summary of important hyperparameter values are listed in Table 1. The full list and assignment of hyperparameters are included with our source code.

5.3 Evaluation Procedure

Although all games use a common set of hyperparameter values, a separate policy is evolved for each. For each game, three evolutionary runs are performed, each for 200 generations. At each generation, the fitness of each individual is calculated as the mean total reward received over three episodes of gameplay. An average over a number of episodes is required to get an accurate estimate of the quality of each individual, due to the randomness in the game environments. Using three episodes was found to be a good compromise between accuracy and computation time. An episode is terminated if it lasts 20,000 frames (approximately five minutes of real-time gameplay), and the score for an episode that reaches this frame cap is recorded as the total reward at that point. The total reward is analogous to the agent's score in the game. To speed up the evaluations of the population at each generation, they are parallelised over 130 CPU cores. To select the final policy for each game, the best policy from each run is evaluated for 100 episodes. The policy with the highest average reward is reported in our results.

To assess the performance of the evolved solutions for each game, they are compared against the expert human scores published alongside DQN [14]. These scores are the mean score achieved over 20 episodes by a professional human games tester, after approximately two hours of practice playing each game. The only exception to this is for Berzerk, which was not included in their set of experiments. Instead, for Berzerk, we compare the agent's performance against the record human score listed on gaming records website www.twingalaxies.com. The reason we do not use records for all games is because they are not indicative of average expert human performance. We also compare the performance of our agents against the published results of NEAT using Hausknecht et al.'s object class representation [8], and DQN using pre-processed images [14]. These provide points of comparison against another high-level state representation and a simple gradient-based approach.

5.4 Human-Normalised Scoring

For Boxing, Pong, and Tennis, the agent plays against a computer-controlled opponent. In these games, the total reward is defined as the player's score minus the opponent's score. Therefore, the agent can achieve a negative total reward. For all other games, the agent's score begins as zero, and points are accumulated during gameplay.

Because each game has a different scoring mechanism, it is difficult to compare agent performance between games directly. Therefore, to compare performance between games and against human performance, we report agent performance using *human-normalised* scores. These scores measure the percentage of the human score that is achieved by the agent and are calculated by:

$$normalised\ score = \frac{agent\ score - min\ score}{human\ score - min\ score} \times 100 \qquad (1)$$

where *agent score* refers to the mean total reward of the best performing solution over 100 episodes of gameplay, *min score* is the minimum total reward possible for the game, and *human score* is the expert human score.

6 RESULTS

In this section, we present the results of our performance analyses and investigate properties and policies of the evolved solutions.

6.1 Overall Performance

Fig. 3 shows the human-normalised performance of each of the best agents; as can be seen, performance varies substantially between games. While the agents for Video Pinball, Boxing and Bowling exceed or match expert human performance, for other games the agents perform far worse. Although most of the best performing games (Boxing, Bowling and Freeway) have *good* state representations, the highest performing agent was found for Video Pinball. This illustrates that for some games, good strategies can still be discovered with imperfect and missing information. The difference between Tennis A and Tennis B is explained in Section 6.2.

Table 2 lists the average scores of the best agents for each game and compares these against the average scores achieved using Hausknecht et al.'s object class representation [8], DQN [14] and the expert human scores [14]. This table highlights a few unexpected outliers in terms of performance; in particular, our evolved agents for Pong and Breakout perform considerably worse than the other methods and human performance. These are further investigated in the following sections.

Despite the more compact state representations provided by the Atari ARI, our agents typically perform worse than the NEAT object class and DQN agents; outperforming each in only two of the 14 and three of the 14 games respectively. However, for Frostbite, our agent outperforms both.

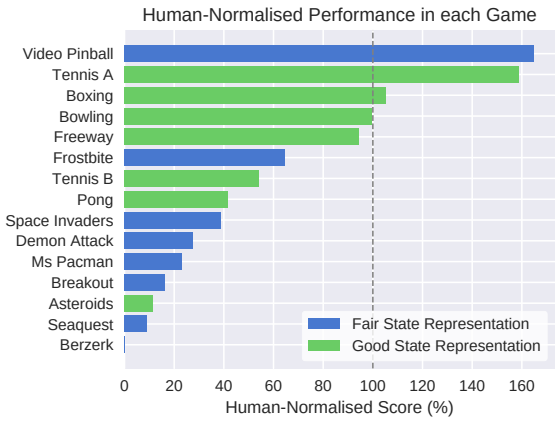

Figure 3: The human-normalised performance of the best agent for each game. The bar colour denotes the quality of state representation provided by the Atari ARI. Agents that surpass the dashed line exceed expert human performance.

Table 2: Scores for NEAT using the ATARI ARI inputs, compared against NEAT using a larger object representation [8], DQN [14] and expert human performance [14]. The top half of the table lists games that have *good* state representations, and the bottom half games that have *fair* representations.

Game	NEAT Atari ARI	NEAT Obj. Class	DQN Pixels	Human
Asteroids	1530.8	**4144.0**	1629.0	13157.0
Bowling	154.3	**231.6**	42.4	154.8
Boxing	9.9	**92.8**	71.8	4.3
Freeway	27.9	**30.8**	30.3	29.6
Pong	-8.3	**15.2**	9.3	18.9
Tennis A	0.0	**1.2**	-2.5	-8.9
Tennis B	-15.8	**1.2**	-2.5	-8.9
Berzerk	1010.0	**1202.0**	-	1057940.0
Breakout	5.2	43.6	**401.2**	31.8
Demon Attack	940.7	3464.0	**9711.0**	3401.0
Frostbite	**2796.8**	1452.0	328.3	4335.0
Ms Pacman	3611.1	**4902.0**	2311.0	15693.0
Seaquest	1865.3	944.0	**5286.0**	20182.0
Space Invaders	640.5	1481.0	**1976.0**	1652.0
Video Pinball	28552.2	**253986.0**	42684.0	17298.0

6.2 Loopholes and Local Optima

In several of the games, the poor performance of the agents is explained by the inability to overcome locally optimal behaviour. Two extreme cases of this are for Pong, due to a loophole in the game, and Tennis, due to the way we calculate fitness.

When playing Pong, the initial trajectory of the ball for each episode is randomly selected from a small set. However, for one of the trajectories, the agent can achieve a perfect score (21-0) in approximately 22% of episodes[2] by merely moving to one particular position. In this position, when the opponent returns the ball, it always rebounds back to the agent. In all three evolutionary runs, after discovering this policy (in as little as 37 generations), the population stagnates. This occurs because the policy is reinforced in each subsequent generation, due to the fitness of agents that use this policy being heavily skewed and vastly surpassing others when they encounter the right conditions.

To play Tennis, the agent and the opponent take it in turns to serve each game within the episode. When it is the agents turn to serve, the agent must execute the FIRE action to begin. However, agents that learn to do this typically perform very poorly initially, as they cannot track and hit the ball each time it is returned. Therefore, these agents lose the majority of games and therefore, all sets. As a result, the agent accumulates total reward close to the minimum (-24). Because of the frame cap on the length of episodes (put in place to prevent endless play in games without a definitive end), the population of agents quickly converges on the strategy of taking no actions to stop the Tennis match from progressing and receive a total reward of zero.

To address the issue of the agent refusing to play, we changed the reward given to Tennis agents if they reach the frame cap. Since the frame cap should never be reached when playing Tennis, if the agent is attempting to play the game, we set the reward that the agent receives to the minimum possible reward (-24). This prevents the agent from exploiting the loophole and leads to objectively worse, but not misleading results. We report the results for the original and modified settings as Tennis A and B, respectively.

6.3 The Effect of Representation Quality and Network Size

When considering performance and representation quality, except for Video Pinball, all of the agents that match or exceed human performance are provided *good* state representations. However, none of the other agents for games with *good* representations eclipse human performance. This suggests, unsurprisingly, that more than just representation quality influences performance.

One potential factor that might also influence the ability to evolve solutions is the combined size of the state representation and action space for each game. This can be considered a pseudo-measure for game difficulty, something that is hard to define precisely. Games with both large state representations and large action spaces require the agent to process more inputs and choose from many possible actions, making the task of choosing a good action more difficult.

Fig. 4 helps to illustrate and investigate the relationship between performance, representation quality and network size. From this, we can see that the highest performing agents were all in games that had relatively small combined input and output space sizes. Given this trend, there are several outliers, such as Pong and Tennis, with surprisingly low performance, when considering the small size of the input and output spaces, representation quality and the performance achieved with other methods.

[2] averaged over 30 samples of 100 episodes

Figure 4: The relationship between performance, representation quality and initial network (input + action space) size.

6.4 Evolved Architectures

Inspecting the architectures of high-performing agents reveals some surprising simplicity. None of the solutions to any games evolved many hidden nodes (the maximum number of hidden nodes was six for Asteroids). Even the networks for high-performing solutions were very simple. This may have been a consequence of the particular hyperparameter values chosen, but it shows that simplicity is not the sole explanation of poor performance. The best-performing agents for Bowling and Freeway epitomise this simplicity. These agents have no and only one hidden node and are shown in figures 5 and 6, respectively. These figures show the input and output nodes, all hidden nodes connected to an output, and all enabled connections.

As mentioned earlier, a consequence of our time-delayed recurrent neural network implementation is that not only self-loops and backward connections create memory, but also different path lengths. For example, despite the best performing Bowling agent having no hidden nodes, the player's x position ($player\_x$) at the current time step and the previous time step influences their decision dues to the connection added between the $player\_x$ node and the $pin\_existence\_8$ node. Similarly, for the best performing Freeway agent, the value of the DOWN output at previous time steps influences the likelihood of choosing NOOP at later time steps. These connections may have evolved to help the agent stop moving downwards and wait before continuing upwards to avoid a car. Another interesting feature about the top Freeway agent is that the positions of cars 3, 5 and 6 are disconnected from the outputs. Since the agent is still able to dodge these cars, we hypothesise that this information must be available from the other inputs. For example, Car 3 may always follow Car 4 at a certain distance.

7 DISCUSSION

The variation in performance, behaviour, and architecture of evolved policies for different games leads to several interesting points worthy of discussion. In this section, we elaborate on each of these points before discussing avenues for further research.

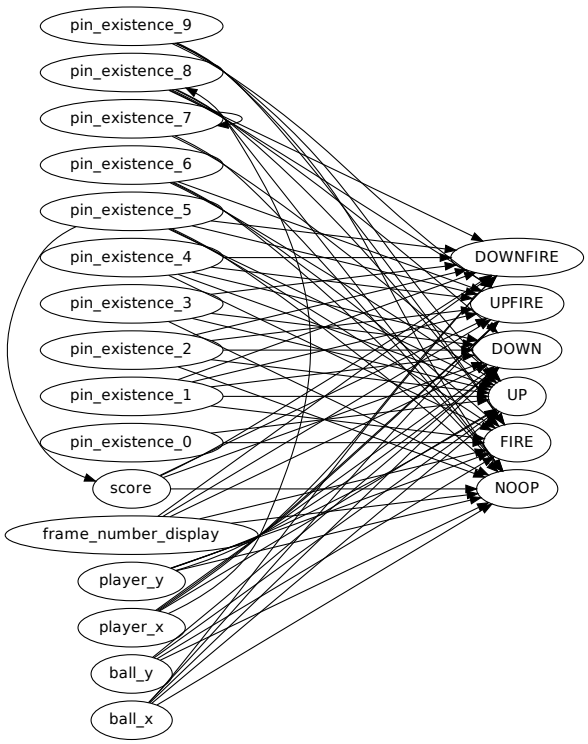

Figure 5: The evolved network architecture for the top performing Bowling agent.

7.1 Dealing with Partial Observability

Without incorporating information from multiple frames, many Atari games are partially observable Markov decision processes. Good examples of such games are Pong, Tennis and Breakout, as they require the agent to evolve memory to track the direction of movement and speed of a ball.

Despite requiring memory to learn good strategies, none of the solutions for Pong, Tennis or Breakout appears to have developed the structural innovations required to track the ball. This partially explains their poor performance. One potential reason for this is that the structures required for tracking the ball require many successful mutations and as such lie some distance from the initial conditions. If this is the case, devising speciation methods that better protect intermediate mutations that are initially detrimental to performance but are needed for more complex structures may be the key to unlocking better performance in games that require complex or long term memory.

7.2 Balancing Exploration and Exploitation

The evolved solutions for some games attest to the fact that evolutionary algorithms are not immune to becoming stuck in local optima. An excellent example of this is the loophole exploited by the evolved solutions for Pong. This local optimum is particularly problematic because it is discovered early on, when agents in the

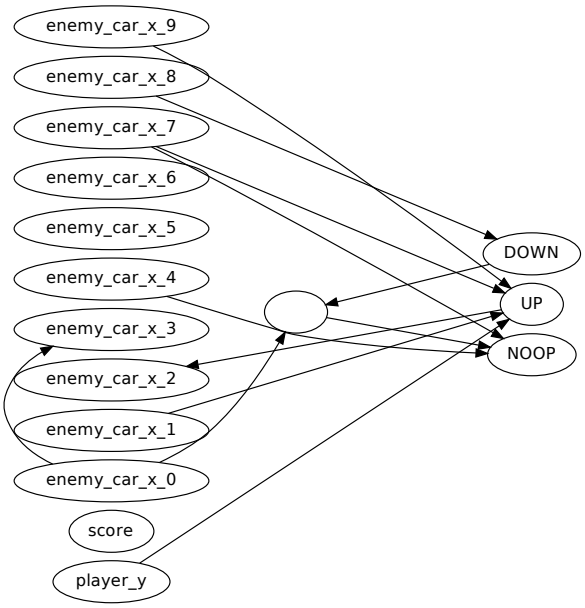

Figure 6: The evolved network architecture for the top performing Freeway agent.

process of developing what we know to be more promising strategies, such as tracking the ball or opponent, are still losing heavily to the computer and attaining negative fitnesses.

From the moment the loophole is discovered, the exploration of more promising behaviours, such as tracking the ball or opponent, slowly ceases. Despite speciation, in the short term, these strategies are not rewarded highly enough to be preserved; highlighting the need for a better balance between exploration and exploitation. It is possible that including elements of novelty search or intrinsic motivation may help to achieve this balance.

7.3 Surprising Simplicity of Solutions

Perhaps the most interesting of our results is that for some games there exist surprisingly simple solutions. Often, human-designed neural networks are highly over-parameterised, and our results begin to lift the veil on the actual complexity required to encode effective policies for some games. The size of our evolved solutions, particularly those that are high-performing, stand in stark contrast to the sizes of networks trained using other techniques. For example, even after the convolutional feature extraction layers, the commonly used DQN architecture has a layer of 512 hidden nodes before the output layer. Our results show that given a sufficient state representation, some games only require comparatively tiny policy networks. These findings suggest that the possibility of finding smaller overall networks by separating state representation and policy learning may indeed be possible and is worthy of further investigation.

7.4 Potential Avenues for Further Research

Our work opens up many potential avenues of future work, some of the most promising of which are summarised below.

7.4.1 Separated state representation and policy learning. Our work provides evidence that using neuroevolution to evolve policy networks from compact state representations is a promising approach for developing intelligent and compact agents. Furthermore, our results should encourage efforts to create general game-playing agents by separating state representation and policy learning. However, more work in this direction is required to validate the sample efficiency, robustness and performance of separated methods, especially compared to end-to-end, gradient-based deep reinforcement learning approaches.

7.4.2 Speciation by differences in policy. One of the issues exposed by our experiments is that evolutionary methods still suffer from balancing exploration and exploitation; particularly when there are large, flat local optima that require many successful mutations to escape. Though NEAT speciates the population to protect structural innovations and genetic diversity, one potential avenue for investigation is speciation by differences in policy and behaviour. This could protect different behaviours instead of the population gradually converging on one strategy or style of play, and help to develop solutions for games that require more significant memory.

8 CONCLUSIONS

In this paper, we evaluated the ability to use neuroevolution to evolve agents capable of playing Atari games using compact state representations provided by the Atari ARI. Although evolved policies only exceeded or were competitive with expert human performance in a handful of games - Video Pinball, Boxing, Bowling and Freeway, we discovered that surprisingly simple and small neural networks could play these games effectively. Furthermore, we also identify games that the Atari ARI may be insufficient for evaluating state representation learning methods, due to missing information that appears important for learning effective policies.

Overall, our results show that through the separation of state representation and policy learning, neuroevolution methods that optimise both weights and topology can find elegant solutions to complex reinforcement learning problems. In the future, we plan to extend our work by incorporating NEAT in a separated state representation and policy learning framework and evaluating the overall effectiveness of this approach.

REFERENCES

[1] Samuel Alvernaz and Julian Togelius. 2017. Autoencoder-augmented neuroevolution for visual doom playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. 1–8.

[2] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. 2019. Unsupervised state representation learning in atari. In *Advances in neural information processing systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8766–8779.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: an evaluation platform for general agents. *J. Artif. Int. Res.* 47, 1 (2013), 253–279.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] S. Geva and J. Sitte. 1993. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine* 13, 5 (Oct. 1993), 40–51.

[6] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. 2008. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research* 9, May (2008), 937–965.

[7] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation* 11, 1 (March 2003), 1–18.

[8] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. 2014. A Neuroevolution Approach to General Atari Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (Dec. 2014), 355–366.

[9] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber. 2009. Super mario evolution. In *2009 IEEE Symposium on Computational Intelligence and Games*. 156–161.

[10] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9, 6 (Dec. 2005), 653–668.

[11] Jan Koutník, Juergen Schmidhuber, and Faustino Gomez. 2014. Evolving Deep Unsupervised Convolutional Networks for Vision-based Reinforcement Learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, New York, NY, USA, 541–548.

[12] Alan McIntyre, Matt Kallada, Cesar G. Miguel, and Carolina Feher da Silva. 2017. neat-python. (2017). https://github.com/CodeReclaimers/neat-python

[13] V. Mnih, A.P. Badia, L. Mirza, A. Graves, T. Harley, T.P. Lillicrap, D. Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *Int. Conf. Mach. Learn., ICML*, Weinberger K.Q. and Balcan M.F. (Eds.), Vol. 4. International Machine Learning Society (IMLS), 2850–2869.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (Feb. 2015), 529.

[15] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2017. Ray: A distributed framework for emerging AI applications. (2017). arXiv: 1712.05889 [cs.DC].

[16] Andreas Precht Poulsen, Mark Thorhauge, Mikkel Hvilshj Funch, and Sebastian Risi. 2017. A Deep Learning / Neuroevolution Hybrid for Visual Control. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, New York, NY, USA, 93–94.

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

[18] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (June 2002), 99–127.

[19] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv e-prints* (2017).

[20] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[21] E. Todorov, T. Erez, and Y. Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033.

[22] Andrew James Turner and Julian Francis Miller. 2013. The Importance of Topology Evolution in NeuroEvolution: A Case Study Using Cartesian Genetic Programming of Artificial Neural Networks. In *Research and Development in Intelligent Systems XXX*, Max Bramer and Miltos Petridis (Eds.). Springer International Publishing, 213–226.