A Neuroevolution Approach to General Atari Game Playing

Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone
Department of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
{mhauskn,joel,risto,pstone}@cs.utexas.edu

December 16, 2013

Abstract

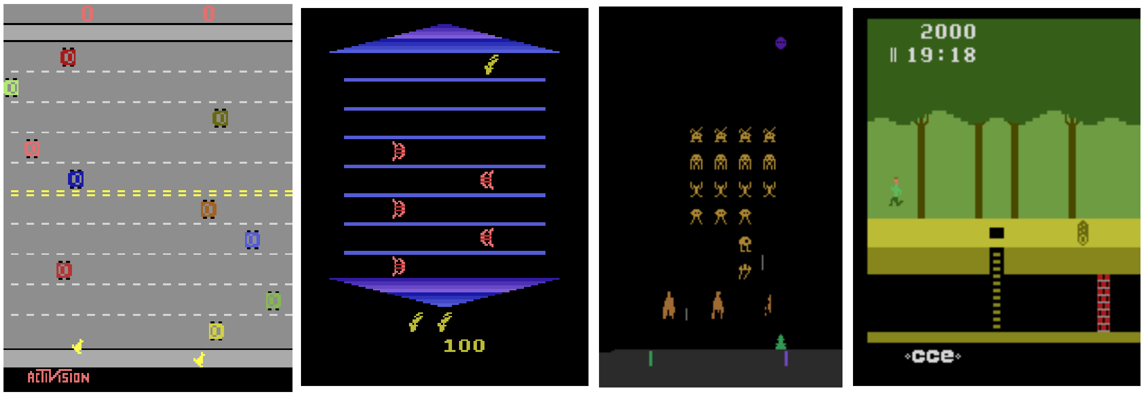
This article addresses the challenge of learning to play many different video games with little domain-specific knowledge. Specifically, it introduces a neuro-evolution approach to general Atari 2600 game playing. Four neuro-evolution algorithms were paired with three different state representations and evaluated on a set of 61 Atari games. The neuro-evolution agents represent different points along the spectrum of algorithmic sophistication - including weight evolution on topologically fixed neural networks (Conventional Neuro-evolution), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), evolution of network topology and weights (NEAT), and indirect network encoding (HyperNEAT). State representations include an object representation of the game screen, the raw pixels of the game screen, and seeded noise (a comparative baseline). Results indicate that direct-encoding methods work best on compact state representations while indirect-encoding methods (i.e. HyperNEAT) allow scaling to higher-dimensional representations (i.e. the raw game screen). Previous approaches based on temporal-difference learning had trouble dealing with the large state spaces and sparse reward gradients often found in Atari games. Neuro-evolution ameliorates these problems and evolved policies achieve state-of-the-art results, even surpassing human high scores on three games. These results suggest that neuro-evolution is a promising approach to general video game playing.

1 Introduction

A major challenge for AI is to develop agents that can learn to perform many different tasks. To this end, this article describes a class of agents capable of learning to play a large number of Atari 2600 games with little domain-specific knowledge.

Atari 2600 games represent a middle ground between classic board games and newer, graphically intensive video games. The Atari 2600 includes many different games spanning a number of genres, including board games such as chess and checkers, action-adventure games like Pitfall, shooting games like Space Invaders and Centipede, rudimentary 3-D games like Battlezone, and arcade classics such as Frogger and Pac-Man. A few example games are shown in Figure 1. From the perspective of an AI practitioner, Atari games are similar to traditional board games in that an AI agent may benefit from cunning planning and a solid understanding of the game's dynamics. They are also similar to modern video games in that they provide opportunities for an agent to process and learn from the action taking place on the game screen. Despite the variability of game dynamics, all Atari games share a standard interface designed for humans to use. Game state is conveyed to the player through a 2D game screen, and in response, the player controls game elements by manipulating a joystick and pressing a single button. The diversity of games, applicability of fundamental AI techniques, and standardized interface make the Atari 2600 an enticing platform for the development of a general video game playing agent.

In this article four neuro-evolution algorithms using three different state representations are applied to 61 Atari games. This work builds upon HyperNEAT-GGP, a HyperNEAT-based general Atari game



(a) Freeway (b) Asterix (c) Space Invaders (d) Pitfall

Figure 1: General Video Game Playing in Atari 2600: just a few of the hundreds of games available for the Atari 2600. Several factors make the Atari an ideal research platform: a large variety of games with interesting dynamics, the availability of a high quality emulator designed for learning agents, standard interface and controls across the games, and a simple 2D representation.

playing agent [18] demonstrated to learn on two different Atari games: Freeway and Asterix. HyperNEAT-GGP required game-specific knowledge in order to initialize the associated neural network and to select actions. These restrictions are removed in this article, extending HyperNEAT-GGP to many more Atari games. Additionally, this approach is evaluated in the context of a wider class of neuro-evolutionary learning algorithms with the intent of investigating algorithmic performance as a factor of state representation and algorithmic sophistication. Specifically the following questions are addressed: (1) How does performance change as a factor of algorithm sophistication? (2) How does neuro-evolution performance scale as a function of state representation? (3) How do the evolutionary algorithms presented in this article compare to previous benchmarks including temporal-difference learning algorithms, planning algorithms, and human experts?

The article is organized as follows: Section 2 outlines the merits of the Atari 2600 as a research platform, and Section 3 reviews related work in this area. Next, Section 4 covers the four different neuro-evolution algorithms, while Section 5 describes the state representations used by these algorithms. Finally, Sections 6-8 describe the specifics of interfacing these algorithms with the Atari domain and running experiments, and Sections 9 and 10 present results and their analysis, in particular answering the questions above.

2 Atari for Research

The Atari 2600 video game console was released in October 1977. It was the first console to create game cartridges that decoupled game code from console hardware (previous devices all contained dedicated hardware with games already built in). Selling over 30 million consoles [12], Atari was considered wildly successful as an entertainment device. Today, while Atari is no longer at the forefront of entertainment, the console has good research potential for four main reasons.

First, there are 418 original Atari games with more being continually developed by an active homebrew community. Many games support a second player, opening the possibility of multiagent learning. The large number of games creates an ideal testbed for general game playing agents by allowing a learning agent to be quickly deployed in a variety of domains.

Second, there exists a high quality Atari emulator, ALE (Arcade Learning Environment[1]), which is designed specifically to accommodate learning agents. Furthermore, since the Atari 2600 CPU runs at 1.19 megahertz, ALE, on modern processors, is many times faster - running at high speeds of up to 2000 frames

[1] http://www.arcadelearningenvironment.org/

per second, expediting the evaluation of agents and algorithms and thus making sophisticated learning simulations practical.

Third, the Atari state and action interface is simple enough for learning agents, but complex enough to control many different games. The state of an Atari game can be described relatively simply by its 2D graphics (containing between 8 or 256 colors depending on the color mode and a native resolution of 160×210), elementary sound effects, and 128 bytes of console RAM. The discrete action space for Atari consists of eight directions of movement for the joystick (up, down, left, right, up-left, up-right, down-left, down-right) as well as a single button. This button can be pressed alone or simultaneously with any of the joystick movements. Including *NO-OP* (no action), this setup yields a total of 18 possible actions.

Fourth, the games have complex enough dynamics to be interesting yet at the same time are relatively short and have a limited 2D world with relatively few agents.

One inconvenient aspect of the Atari 2600 is that it is necessary to acquire ROM files for the different games in order to play them. In practice this is not a major problem because Atari 2600 enthusiast websites such as `http://www.atarimania.com/` offer links to ROM packages.

In summary, the large number of games, availability of high quality emulators, and the standardized interface make the Atari 2600 a great stepping stone from board games and declarative games (like those used in the GGP competition - see Section 3) to more modern video games.

3 Related Work

The Atari domain was first used as a research platform [11], then modified to interface with learning agents [3, 33], and released as the Atari Learning Environment (ALE). ALE has been subsequently used as a domain for reinforcement learning and search agents [2, 3] and to investigate aspects of prediction and self-detection [4]. The work presented in this article differs from previous Atari research in its application of neuro-evolution algorithms rather than temporal difference (TD) algorithms to this domain.

Broadly, this work is motivated by the idea of general-purpose intelligence [21, 25], applied to the domain of video game playing [37]. Research in this area has focused on learning to play single games using perceptual imagery [52], human demonstration [9], discovery of objective functions [48], and use of raw visual inputs [30, 34, 35, 49]. The work in this article differs because of its emphasis on **general** video game playing.

In a more traditional board game setting, the General Game Playing (GGP) [16] competition has been run since 2005. In this competition, agents are given a declarative description of an arbitrary game (including a complete description of the game dynamics) and without prior knowledge must formulate strategies to play this game on the fly. Thus, unlike specialized game players, general game playing agents cannot rely on algorithms designed in advance for specific games. Recently there has been a push to bring general game playing into the realm of video games [27], creating the area of General Video Game Playing (GVGP) [28, 36]. The GGP competition differs from the scenario considered in this article in four ways: (1) While learning techniques can be used in principle, there is little opportunity to do so in the competition. (2) While the GGP competition provides a description of game dynamics, algorithms studied in this article must learn dynamics exclusively by playing the game. (3) While the GGP competition is based on declarative representations, this article focuses on visual representations (as suggested by other recent work in GGP [1, 22]) (4) While GGP features board-like games, this article takes a step towards playing video games.

Neuro-evolution algorithms [13, 32] have been successful on a variety of tasks including quadruped locomotion [8, 50], evolution of morphology [6], design of robot morphology and control [29], checkers [14], Unreal Tournament [38], and game playing [5, 47]. Of the algorithms used in this article, HyperNEAT has been previously applied to game of Robocup Keepaway Soccer [45, 51], checkers [15], multi-agent predator prey [10], and quadruped locomotion [7]. NEAT has been applied to evolving agents for the NERO video game [39] and the game of Go [44]. CMA-ES has been applied to the task of walk-optimization in a robot soccer simulator [31]. None of these algorithms has been previously applied to the task of general video game playing.

4 Algorithms

The four different neuro-evolution algorithms applied to the Atari domain are simple weight evolution over a topologically static neural network (CNE), Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), evolution of weights and topology (NEAT [43]), and indirect encoding of network weights (HyperNEAT [15]). All algorithms encode Artificial Neural Networks (ANNs) which are represented by weights and connectivity (also called topology). The first two algorithms search over just the weights of the ANN while the latter two have the capacity to also modify the topology. Specifics of ANN topology are discussed in Section 6.

4.1 Conventional Neuro-evolution (CNE)

In the simplest of the algorithms, Conventional Neuro-evolution (CNE), the topology of the ANN is fixed and only the weights of the network are evolved. Crossover and mutation, typical of genetic algorithms, are used to generate offspring. As such, CNE represents a baseline of performance. In addition, speciation (similar to that in NEAT) is used to maintain diversity in the population. Thus the only difference between CNE and NEAT is that NEAT also evolves network topology. CNE is able to fine tune each weight independently; thus given enough time it is capable of finding high performing policies if they exist within the policy space [8].

4.2 CMA-ES

The second method to be evaluated in the experiments consists of a more sophisticated way to evolve fixed topology networks. Instead of crossover and mutation, the weights of the network are evolved through Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [17], a policy search algorithm that successively generates and evaluates sets of candidates sampled from a multivariate Gaussian distribution. Once CMA-ES generates a set of candidate policies, the fitness of each policy is evaluated with respect to the game score it accrues. When all candidates in the group have been evaluated, the mean of the multivariate Gaussian distribution is recalculated as a weighted average of the candidates with the highest fitness. The covariance matrix is also updated to bias the generation of the next set of candidates towards directions of previously successful search steps. The topology of the networks evolved by CMA-ES is identical to those of CNE.

4.3 NEAT

Neuro-Evolution of Augmenting Topologies (NEAT) is an evolutionary computation method that evolves the topology and weights of an Artificial Neural Network (ANN) [43]. As such it represents an advance from methods that evolve only the weights of topologically fixed networks. Allowing NEAT to control the topology of the ANN gives it the power to scale the complexity of the network to match that of the problem – increasing the number of nodes for more complex problems and reducing it for less complex ones, in addition to simply adjusting connection weights. Additionally, NEAT attempts to balance the fitness of evolved solutions and their diversity. It is based on applying three key techniques: tracking genes with history markers to allow crossover among topologies, applying speciation to preserve innovations, and developing topologies incrementally from simple initial structures (i.e. complexification) [42].

4.4 HyperNEAT

HyperNEAT [15, 40, 41] evolves an *indirect encoding* called a Compositional Pattern Producing Network (CPPN). The CPPN is used to define the weights of an ANN that produces a solution for the problem. This encoding differs from *direct encodings* (e.g. CNE, CMA-ES, and NEAT) that evolve the ANN itself. As shown in Figure 2, the ANN substrate has an input layer consisting of nodes spanning an (X_1, Y_1) plane. The weights between this input layer and the next layer in the network (X_2, Y_2) are determined by the CPPN. This CPPN is said to be geometrically aware because each network weight it outputs is a function of the (x_1, y_1), (x_2, y_2) coordinates of the nodes in the corresponding layers of the ANN. By ranging the inputs of the CPPN over all pairs of nodes in the first and second layers of the ANN, the weights in the network are fully determined. This geometric awareness allows the CPPN to encode weights that are functions of domain geometry, resulting in an ANN that implicitly contains knowledge about geometric relationships in the domain. In comparison to standard NEAT, HyperNEAT's encoding potentially allows it to take advantage of geometric regularities present in Atari 2600 games. In addition, its indirect encoding allows it to express very large neural networks while evolving only small, compact CPPNs.

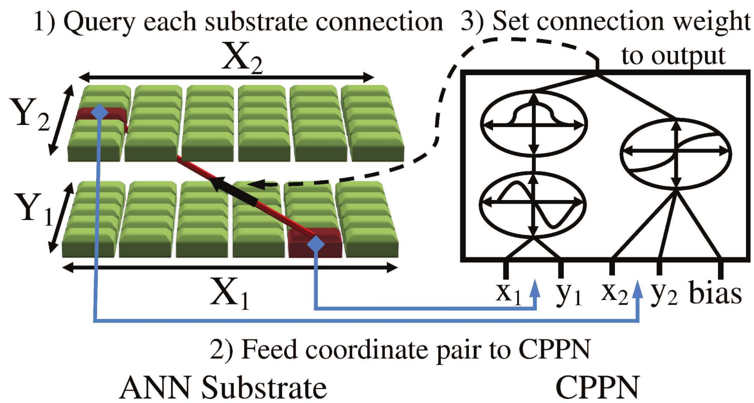


Figure 2: The HyperNEAT method of neuroevolution: NEAT evolves the weights and topology of a CPPN (right). This CPPN is subsequently used to determine all of the weights between substrate nodes in the ANN (left). Finally, the ANN is used to compute the solution to the desired problem. CPPNs are said to be geometrically aware because when they compute the weights of the associated ANN, they are given as input the x, y location of both the input and output node in the ANN. Such awareness could prove useful in Atari 2600 games. Also, the CPPN encoding allows HyperNEAT to represent very large networks compactly which makes scaling to large representations possible. (Figure replicated with permission from [51].)

Specifically, HyperNEAT works in four stages:

1. The weights and topology of the CPPN are evolved. Internally a CPPN consists of functions such as Gaussians and sinusoids (chosen because they reflect the patterns and regularities observed in nature and therefore also in real-world domains) connected in a weighted topology (Figure 2).
2. The CPPN is used to determine the weights for every pair of *(input,output)* nodes in adjacent layers of the ANN.
3. With fully specified weights, the ANN is applied to the problem of interest. The performance of the ANN determines the fitness of the CPPN that generates it.
4. Based on fitness scores, the population of CPPNs is maintained, evaluated, and evolved via NEAT.

Having discussed the selected neuro-evolution algorithms, the next section describes the three state representations paired with these algorithms.

5 State Representations

This section presents the state representations used by the algorithms in Section 4. Three specific state representations are investigated: object representation, raw-pixel representation, and the noise-screen representation. These state representations convey information to the agent about the state of the game. This information is translated into activations of the input level of an ANN. The architecture of this ANN will be discussed in Section 6, but the translation of state information to input activations is a topic of this section.

5.1 Object Representation

The object representation relies on a set of manually identified object images categorized into object classes. These images were manually created by the authors playing each game, saving images of encountered objects, and categorizing them into classes. Subsequently, these objects may be automatically identified at runtime by checking if any of objects on the current screen matches any of the saved object images. In order to capture animated sprites, multiple images for each object may be saved. Object matching must be exact and objects that are partially occluded are not identified unless an image of the partial occlusion was saved.

One problem with the manual object detection approach is that for level-based games new objects arise at more advanced game levels. This behavior can be a problem when algorithm performance exceeds human performance, leading to the algorithm encountering objects for which there have been no saved images - rendering them invisible. This happens in several games including Asterix, Beam Rider, and Phoenix. In such cases the agent will play on, blind but confident.

Information about the objects on screen is conveyed to the input layer of an ANN via N substrates - one for each class of object (Figure 3a). At every game screen, the relevant on-screen entities in each object class are identified and transformed into substrate activations in the substrate corresponding to that object class and at the node corresponding to that object's location on the game screen. For example, substrate N may be assigned the object class of cars. Each car in the current game screen is given an activation in substrate N at the corresponding location (subject to downscaled resolution). Thus each activated substrate mirrors the locations of a single class of objects on the screen. Together the substrates represent a decomposition of on-screen entities.

The object representation is a clean and informative characterization of the visual information on the game screen. As such it represents the highest level and least general features that are used in this article. Correspondingly, algorithms using the object representation are the least general game players as they require relevant game objects to be identified before the start of the game. It may be possible to do this recognition automatically; this possibility is discussed in Section 11.

5.2 Raw-Pixel State Representation

In contrast to the object representation, the raw-pixel representation is a low-level representation of the information on the game screen. Quite simply it provides the downsampled pixel values as input to the agent. In order to limit the dimensionality of the screen, it was necessary to minimize the number of colors present. Fortunately, Atari has support for three different color modes: NTSC (containing 128 unique colors), PAL (104 colors), and SECAM (8 colors). Thus, the SECAM mode was used to minimize the color palette.

The raw-pixel representation (shown in Figure 3b) has eight input substrates - one to represent each color. Colors from the screen were transformed into substrate activations by activating a node in a substrate if the corresponding color was present in the screen at that geometric location.

Individual pixels of the game screen are low-level, uninformative features, yet together they encode a great deal of information (2^{2688} unique game screens can be represented). The raw-pixel representation is the most general of the representations used and mirrors what a human would use when playing an Atari game. Therefore this representation provides a true test of general game playing.

5.3 Noise-Screen Representation

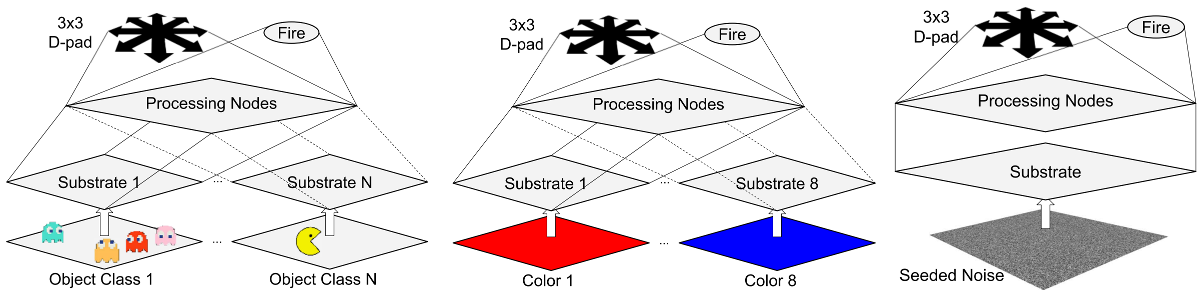
In order to investigate how much of the learning is based on memorization and how much on learning general concepts, noise-screens were used as the third category of input representation. Noise-screens consist of a single substrate with seeded random activations. While learning may seem hopeless at first glance, Atari games are deterministic, meaning that the start state for each game is the same and there is no stochasticity in the transition and reward functions.[2] Thus it is possible for HyperNEAT-GGP to evolve a policy that takes advantage of a deterministic sequence of random activations. In some sense the noise-screen representation

[2] The latest version of ALE adds an option to utilize a random starting state. Experimental work in this article pre-dates that release.

tests the memorization capability of an algorithm - how well can it learn to associate the correct actions with state inputs that are deterministic but entirely uncorrelated with the action taking place on-screen. Additionally, the noise-screen representation represents a sanity check to make sure the neuro-evolution algorithms using the object and pixel representations are doing more than just memorizing sequences of moves. The noise-screen representation uses a single input substrate as shown in Figure 3c.

Having described the algorithms and state representations, the next section describes the topology the Artificial Neural Networks and how they are interfaced with the Atari game screen.

6 Neural Network Architecture



(a) Object Architecture (b) Raw-Pixel Architecture (c) Noise-Screen Architecture

Figure 3: ANN topology for (a) the object-level state representation, (b) the raw-pixel state representation, and (c) the noise-screen state representation. In (a) the geometric locations of objects from multiple classes (ghosts and pac-man in this case) are provided as input activations to the Substrate layer of the ANN. In (b) the geometric locations of colors present in the screen are transformed into substrate activations. In (c) random noise is provided as substrate activation. In all cases, networks are feed-forward and input activations are propagated upwards through the processing level to the output layer, which consists of a 3×3 directional substrate (D-pad) and a fire button, mirroring the physical controls of the Atari 2600.

The ANN consists of three layers (Figure 3): a substrate layer in which information from the game screen (raw pixels, objects, or noise) is given as input to the network, a processing layer which adds a non-linear internal representation, and a non-linear output layer from which actions are read and conveyed to the Atari emulator. Both the input and output layers are fully connected to the processing layer. The substrate dimensionality of the input and processing layers is 8×10 in the case of the object representation and 16×21 for the pixel and noise representations.[3] The output layer consists of a 3×3 substrate mirroring the nine possible directions of the Atari 2600 joystick and a single node representing the Fire button. A sigmoid non-linearity was used in the processing and output units.

In the output layer, the 3×3 directional substrate contains the four primary directions of movement in the North, South, West, and East positions. Combinations of these directions (North-East, etc) are found at four corners of the substrate, and the center node corresponds to no joystick movement. Actions are read from the output layer by first selecting the node with the highest activation from the directional substrate, then pairing it with the activity of the fire button. If the activation of the fire button exceeds a threshold (0.5) then it is considered active, otherwise inactive. By pairing the joystick direction and the fire button, all eighteen actions can be created in a manner isomorphic to the physical Atari controls.

Non-HyperNEAT algorithms do not require geometric output layers and instead used an output layer consisting of a single node for each action available in the current game for a dimensionality of $1\times A$ where A is eighteen or less.[4] The action corresponding to the output node with the highest activation is selected

[3] These resolutions downsample the 160×210 native screen resolution by a factor and 20 and 10, respectively.

[4] Games vary in the number of actions that are used to play. In some games certain actions are deemed illegal because they will restart the game or otherwise change the desired MDP structure.

as the agent's action.

The ANNs execute as follows: At the beginning of each timestep, the current game screen is acquired. Next the on-screen objects, pixels, or noise are transformed into activations in their respective input substrates. These activations are propagated upwards in the ANN through the processing layer and to the output layer. Finally, an action is selected at the output layer and fed into the emulator. This cycle continues until the game has terminated. ANN weights are set at the beginning of each episode and do not change throughout the episode.

Although all of the neuro-evolution algorithms in Section 4 can evolve recurrent networks, experiments in this article consider only feed-forward architectures because most Atari games are MDPs given the state representations discussed in Section 5.

7 Experimental Setup

As mentioned in the introduction, the experiments in this article evaluate the performance of four different neuro-evolution algorithms paired with the three different state representations on a set of 61 Atari games. The 61 games were selected as a representative sample of different game genres including action-adventure, shooting, 3-D, and arcade classics. Additionally these games represent a super-set of the 55 games used by Bellemare et at. [3], allowing direct score comparisons on that subset. Each of the algorithms was paired with each of the different representations, unless the pairing was found to be computationally intractable.

A separate policy was evolved for each game. This policy consisted of 150 generations with a population size of 100 individuals per generation, a total of 15,000 episodes of play per game.[5] All episodes were terminated after 50,000 frames or about thirteen minutes of game time. Each member of the population earned a fitness score equal to the game score accrued from a single episode of play (e.g. until the game-over or the frame limit is reached). The reported score for each neuro-evolution agent corresponds to the champion fitness at generation 150. Appendix A lists other experimental parameters. Source code necessary to replicate experiments can be found online at `https://github.com/mhauskn/HyperNEAT`.

The performance of the neuro-evolution algorithms was compared to other previously published benchmarks [3] including SARSA(λ), planning, human play, and random play. SARSA(λ) refers to a class of agents using model-free learning with linear function approximation, replacing traces, and ϵ-greedy exploration. Bellemare et al. [3] paired SARSA(λ) with the following state representations: discretized NTSC (128 colored) screen, discretized SECAM (8-colored) screen, on-screen objects, RAM content of the game, and locally-sensitive hashes of the game screen. The SARSA column in Appendix B represents the maximum reported performance of any of the five SARSA(λ) agents on each game.

Planning is accomplished in the Atari environment by using the emulator as a generative model of future states. This approach reduces the problem of playing an Atari game to a search problem over future states. Bellemare et al. [3] investigated two planning algorithms: a breadth-first search and an Upper Confidence Bound for Trees (UCT [23]) based search. Due to the size of the action space, the breadth-first search could only search approximately one third of a second into the future given 100,000 search steps. While such search depth may be sufficient to avoid immediate death, it is unable to perform any sort of long-term planning. UCT uses a more sophisticated sampling strategy, applying the bandit algorithm UCB1 to guide sampling of actions and future states. Monte-Carlo rollouts are performed when this sampling phase reaches a leaf node in the game tree. This sampling strategy allows UCT to follow promising trajectories further into the future than breadth-first search. The planning column in Appendix B represents the maximum reported performance of either of the planning algorithms on each game.

Human expert high scores were collected from `http://www.jvgs.net/2600/top50.htm`. These scores represent the best reported human scores for a number of Atari 2600 games. Finally, the random play algorithm takes a random action at each timestep. These two scores represent upper and lower bounds for performance expected from neuro-evolution algorithms.

[5] For comparison, Bellemare et al. [3] reports that SARSA(λ) agents were trained for 5,000 episodes. Presumably, the performance of these methods had flatlined at each of these points, making it possible to compare final performance.

8 Score Normalization

Scoring metrics vary from game to game. In order to compare the performance of different algorithms across multiple games it is necessary to normalize the scores of each algorithm on each game. Previously, Bellemare et al. [3] proposed a normalization scheme in which an algorithm's score on a game was normalized into a $[0, 1]$ range given how well it performed in comparison to a baseline set of all algorithm scores $\mathbf{s}$ on the same game:

$$\text{normalized score} = \frac{\text{score} - \min(\mathbf{s})}{\max(\mathbf{s}) - \min(\mathbf{s})} \tag{1}$$

Instead this article uses a z-score normalization scheme in which an algorithm's score is normalized against a population of scores where μ and σ are the mean and standard deviation of the set of all algorithm scores on that game:

$$\text{z-score} = \frac{\text{score} - \mu}{\sigma}. \tag{2}$$

The z-score tells how many standard deviations a given score is above or below the mean score. The advantage of using z-score normalization is that it gives a better sense of how well or poorly each algorithm is performing with respect to the mean and standard deviation of the population. This approach makes it a more informative statistic than the normalized score. For example the best scoring algorithm may have a z-score of 3 (meaning that it was three standard deviations above the mean) while the normalized score would simply be 1.[6]

9 Results

This section examines experimental results in order to answer the questions posed in the introduction: (1) How does performance change as a factor of algorithm sophistication? (2) How does algorithm performance scale as a function of state representation? (3) How do the neuro-evolution algorithms compare to temporal difference learning algorithms, planning algorithms, and human experts based on result in the literature? In each case, the performance of the algorithms was compared using a one-way Analysis of Variance (ANOVA) statistical test followed by a Tukey's HSD significance test. The full set of games scores can be found in Appendix B.

9.1 Neuro-evolution Results

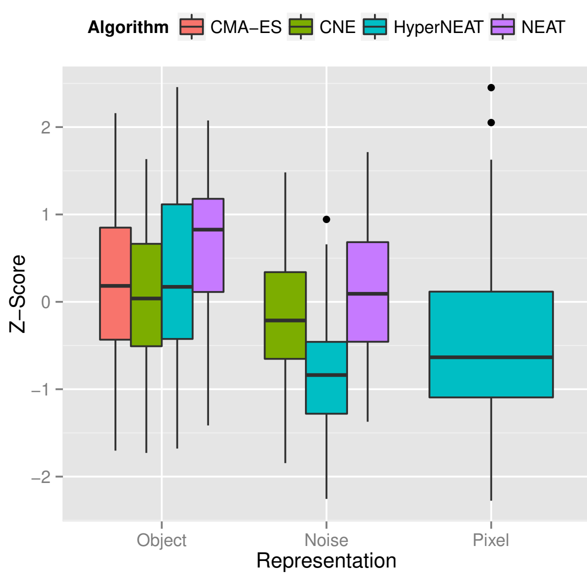
Figure 4 summarizes the results: With the object representations, NEAT statistically significantly outperformed CNE and CMA-ES and was statistically similar to HyperNEAT.[7] With the noise representations, NEAT significantly outperformed CNE and HyperNEAT.

The first main result is that NEAT had the highest mean z-score across the object and noise representations. Surprisingly, HyperNEAT did not significantly outperform the other algorithms despite the additional complexity inherent in its indirect encodings and its ability to incorporate geometric information. This result implies that many Atari games can be well-learned using a compact and well-tuned network such as the ones NEAT is capable of creating as long as the representations are low-dimensional.

The second main results is that HyperNEAT was the **only** neuro-evolution algorithm capable of learning to play based on the visual, raw-pixel representation. It was capable of doing so because of its indirect encoding: each individual in the population was encoded by a small CPPN rather than a massive ANN. (ANNs using the raw-pixel state representation had over 900000 weights.) While these ANNs could still be

[6]One possible disadvantage of using z-scores is that they are not bounded and are therefore more difficult to compare and combine in future experiments. Also, since z-scores are population based, it is possible to inflate/deflate z-score by adding duplicate low/high scoring agents. This would not affect normalized scores.

[7]References to statistical significance imply p-values of less than 0.05 unless otherwise noted.



Algorithm	Object	Noise	Pixel
HyperNEAT	.34	-.78	-.46
NEAT	.67	.09	-
CNE	.08	-.12	-
CMA-ES	.21	-	-

Figure 4: Performance of Neuro-evolution algorithms with different representations: **Above**: Mean Z-Scores for Neuro-evolution algorithms. Algorithms missing from this table were too computationally expensive to run at the desired substrate resolution. **Left**: the box-plot of these scores as a factor of state representation. NEAT had the highest mean z-scores across the object and noise representations, however HyperNEAT was the only algorithm capable of scaling to the pixel representation. Comparisons were performed on the full set of 61 games followed by z-score normalization.

run in a feed-forward manner, the crossover operation became computationally intractable for any of the direct encoding algorithms.

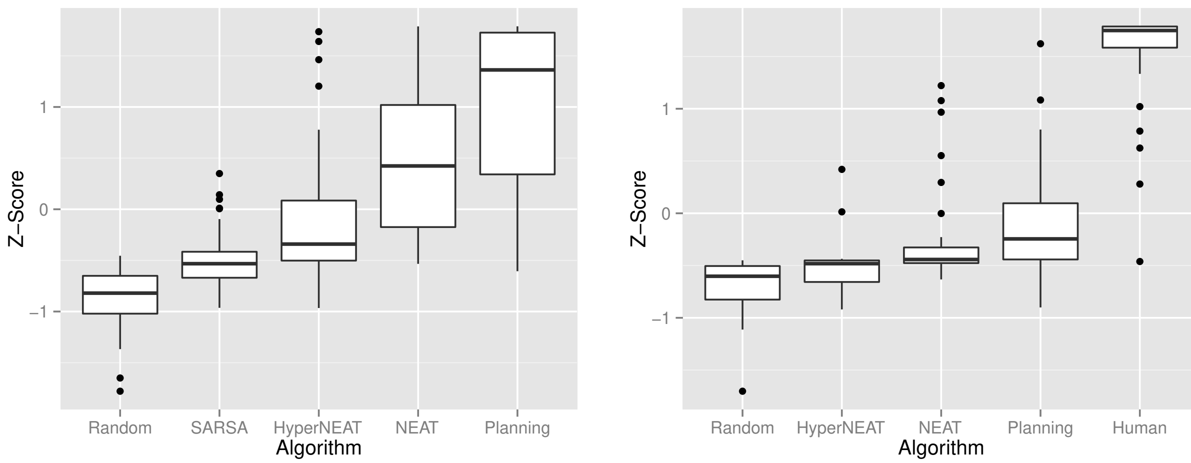
In all cases, algorithms performed worse as the representations became more general (and less informative). The object representation for both HyperNEAT and NEAT significantly outperformed the noise representation. CNE shows the same trend but without statistical significance. The pixel representations for HyperNEAT tend to perform better than noise-screens, but the difference was not significant.

The noise-screen encoding is challenging for all the algorithms. However, the resulting policies still show performance comparable to the SARSA(λ) agents and far better than random play. This level of performance shows the ability of evolutionary-algorithms to discover a policy that represents a simple mapping from noise-screens to game actions. At the same time, the performance gap between the noise-screen representation and the object representation shows that it is possible to discover general concepts for these games, and such concepts are superior to simply memorizing a deterministic sequence of actions. The fact that HyperNEAT with raw-pixel representations tends to perform better than with noise-screens suggests that it is beginning to discover and use such concepts as well.

9.2 Results With Other Methods

Figure 5a compares neuro-evolution against previously published results for the SARSA(λ) and planning agents on the 55 game subset that both shared. In this comparison, HyperNEAT with the pixel representation is included because it is the most general algorithm and NEAT with the object representation is included because it is the highest performing. All comparisons in this plot are statistically significant: planning (with access to future states) performs the best followed by the learning methods NEAT and HyperNEAT, SARSA(λ), all significantly better than random play.

Similarly, in Figure 5b, the neuro-evolutionary, planning, and random algorithms were compared against human high scores on the set of 33 games for which human high scores were known. Human experts significantly outperformed all algorithms on average. Interestingly, neuro-evolution was able to beat the best human score in several games, as described in Section 10.



(a) Neuro, RL, Planning

(b) Human High Scores

Figure 5: **(a)**: Comparison of the Neuro-evolution algorithms against the SARSA(λ) and planning algorithms on the set of 55 games reported by Bellemare et al. [3]. Of the neuro-evolution algorithms, the most general (HyperNEAT-Pixel) and highest performing (NEAT-Object) are included. **(b)**: On average, humans perform significantly better, although neuro-evolution was able to exceed human high scores on three games and discover infinite scoring opportunities on three others. Comparison and z-score normalization done on the 33 games for which there were human high scores. Videos of evolved policies can be viewed at `http://www.cs.utexas.edu/users/ai-lab/?atari`.

10 Discussion

The performance of the neuro-evolution algorithms described in this article represents the state-of-the-art for the problem of learning to play Atari games. There is still a long way to go before neuro-evolutionary performance reaches the level of planners (which have access to future game states) or human experts. However, there are striking successes as well as insightful failures in these results as will be reviewed in this section.

10.1 Successes and Failures of Neuro-evolution Algorithms

Neuro-evolution algorithms exceeded human expert performance on the games Bowling, Kung Fu Master, and Video Pinball. CNE evolved a bowler capable of scoring 252 points in comparison to 237, the top human score, 99,800 points on Kung Fu Master in comparison to the human best of 65,130, and 407,876 points in Video Pinball, far outperforming the human score of 56,851. Counter-intuitively, CNE, the least sophisticated of all of the neuro-evolution algorithms achieved these high scores even though its average game scores were some of the worst overall. Future analysis of actual games played is necessary to uncover why. A possible explanation is that for these three games, unexpectedly good strategies exist as extensions of simple-minded strategies.

Evolved policies are superior to planning algorithms in many games, including Amidar, Assault, Asteroids, Berzerk, Bowling, Carnival, Elevator Action, Freeway, Frostbite, James Bond, Journey Escape, Kangaroo, Krull, Kung Fu Master, Montezuma's Revenge, Private Eye, Riverraid, Star Gunner, Venture, and Video Pinball. In general, evolved policies can be superior to planned ones if they are capable of exploiting a sequence of actions longer than the planning horizon. For example, in Krull, evolved policies exploited a repeatable, multi-screen sequence of play.

The weak point of evolved policies is that in many cases their play differs greatly from intended game play. For example, in the high scoring Kung Fu Master policy evolved by CNE, rather than exploring the map and defeating enemies, the agent simply sits still and attacks enemies that come to it. Policies of this type are in some sense an expected result from a learning algorithm that exclusively optimizes the final game score. In addition to the Kung Fu Master policy described above, several other such exploitative strategies were learned. For example, on the game Beam Rider, HyperNEAT learned how to quickly move the joystick left and right to stay in-between lanes, which made the agent invulnerable to attacks. The caveat of this strategy is that the player must leave this invincible half-space in order to fire back at enemies, thus briefly exposing itself to their fire as well. It therefore did not perform as well as the best human score, but it nonetheless exploits the game environment in a way unintended by the designers.

Infinite score loops were found on the games Gopher, Elevator Action, and Krull. The score remained finite only because of the 50,000 frame cap on any episode. The score loop in Gopher, discovered by HyperNEAT, depends on quick reactions and would likely be very hard for a human to duplicate for any extended period of time. Similarly, the loop in Elevator Action requires a repeated sequence of timed jumps and ducks to dodge bullets and defeat enemies. The loop in Krull, discovered by HyperNEAT, seems more likely to be a design flaw as the agent is rewarded an extra life after completing a repeatable sequence of play. Most Atari games take the safer approach and reward extra lives in accordance with (exponentially) increasing score thresholds. The reader is encouraged to view the videos associated with these policies and others at `http://www.cs.utexas.edu/users/ai-lab/?atari`.

10.2 Successes and Failures of SARSA(λ) Learners

SARSA(λ) agents perform extraordinarily well on the racing game Enduro, but otherwise generally perform worse than their neuro- evolutionary counterparts. When applying temporal difference-based learners to the complex environments embodied by many Atari 2600 games there are three main hurdles to overcome: the enormously large state space, the frequently delayed effects of actions and rewards, and the initial phase with no reward gradient - which can be quite long when rewards are sparse. Whereas there are some temporal difference methods that partially address these problems, policy search methods such as neuro-evolution are better able to ameliorate them by evolving policies that are insensitive to the number of states, do not need to begin the game by exploring, and are evaluated only on accumulated reward at the end of the game (avoiding problems with delayed in-game rewards).

For example Freeway gives a positive reward only when the agent successfully crosses the Freeway. Starting at the bottom of the screen and moving upwards towards the goal, it is unlikely that an agent will select enough UP actions to cross the Freeway when using a random exploration policy. This problem is even further complicated by the agent being pushed downwards if it comes into contact with a car. The neuro-evolution algorithms in this article were able to avoid this problem by quickly evolving simple networks which execute only the UP action. These networks are then further refined in later generations to avoid cars.

10.3 Successes and Failures of Planning Algorithms

The planning algorithms outperform humans on games like Video Pinball and Wizard of Wor, both of which benefit from quick reactions that can be selected withing the planner's search horizon. Video Pinball primarily requires reflexes to repel the ball as it approaches the paddles while Wizard of Wor takes place in a pacman-esque maze in which the player shoots at on-coming monsters.

The games least favorable for the planning algorithms are ones that rely least on quick reflexes and instead require long-term planning beyond the planner's horizon. Examples of these games are action-adventure games such as Montezuma's Revenge and Pitfall in which players control an avatar that must explore multiple rooms on different screens and collect objects like keys to open doorways. In the case of Montezuma's Revenge, none of the algorithms come anywhere close to the human score, however the neuro-evolution algorithms perform far better than the planners. In general however, action-adventure games often feature very sparse reward gradients and are difficult for all algorithms.

10.4 Additional Experiments

Several variations of neuro-evolution experimental parameters were explored. Specifically, different substrate resolutions, generations of evolution, and substrate configurations were attempted. These investigations revealed that the neuro-evolution methods presented (and HyperNEAT in particular) were robust to changes in substrate design, resolution, and evolution time. This result is supported by previous work [51] which shows that HyperNEAT can scale to different substrate resolutions.

11 Future Work

Since most Atari 2600 games are MDPs, the networks evolved in this article were feed-forward, without recurrent connections. Such networks represent reactive controllers i.e. the same input to the network will produce the same output actions, regardless of context or history. Recurrent networks, on the other hand, have an internal state that can act as a type of memory. Such memory is likely to be useful for exploration games that are POMDPs such as Montezuma's Revenge, in which it would be helpful to remember where the player came from in order to decide where to go. Recurrent networks can readily be evolved with all the methods tested in this article and it should be interesting to do so in future work.

Another interesting direction is deep learning which has shown promise in the fields of image processing and object recognition [20, 24, 46]. It would be interesting to explore the space of features learned by the application of deep learning to the Atari game screens. The learned features could be used as an alternative form of state representation, removing the need for manual identification of on-screen entities.

Some games such as Pitfall are still too complicated and have too sparse reward signals for any of the algorithms to learn to play effectively. In Pitfall, the first possibility of collecting treasure takes place many screens away from the start of the game and only after the player has jumped over crocodiles and swung on vines. One potentially fruitful avenue would be to apply intrinsically motivated exploration algorithms such as novelty search [26] or targeted exploration [19] to these games so that the intrinsic reward signal could act as a stand-in until the first real reward is encountered.

Finally, it is possible to perform automatic object recognition in order to avoid having to identify objects manually for each game. The problem with automatic object recognition is that the substrate topology of the ANN depends on knowing the number of input substrate layers (and hence object classes). Networks whose topology could adapt as new object classes are discovered would possibly allow neuro-evolution algorithms to take advantage of automatic object recognition, and constitute a most interesting direction of future work.

12 Conclusion

This article extends prior work on HyperNEAT-GGP [18] into a fully general Atari 2600 video game player. In addition it contributes a detailed empirical analysis of a range of representations and algorithms in this domain and presents the best learning results reported to date. Significantly, evolved policies beat human high scores on three different Atari games - Bowling, Kung Fu Master, and Video Pinball, and discover opportunities for infinite scores in three others (Gopher, Elevator Action, and Krull).

Comparing the different neuro-evolution methods reveals that direct encoding algorithms such as NEAT outperform indirect encoding methods such as HyperNEAT on low-dimensional, pre-processed object and noise-screen representations. However, HyperNEAT was the only algorithm capable of learning based on the fully-general raw-pixel representation. Thus, the algorithmic sophistication of HyperNEAT's indirect encoding is justified by its ability to learn large networks, which are necessary for dealing with raw-pixel input as well as possible vision and image processing tasks more generally.

While no single Atari game, if studied in isolation and given extensive feature engineering, likely poses too great a challenge for modern AI techniques, the full collection of over 400 Atari games still presents a daunting task for a video game playing agent. The results presented in this article suggest that neuro-evolution is a promising approach towards that goal.

Acknowledgments

Thanks to Jeff Clune and Ken Stanley for useful discussions. Source code for the Arcade Learning Environment (ALE) emulator environment can be found at `http://www.arcadelearningenvironment.org/`. Source code for the algorithms and experiments presented in this article can be found at `https://github.com/mhauskn/HyperNEAT`.

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122, IIS-0915038, and DBI-0939454), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030).

References

[1] A. Barbu, S. Narayanaswamy, and J. M. Siskind. Learning physically-instantiated game play through visual observation. In *ICRA*, pages 1879–1886. IEEE, 2010.

[2] M. Bellemare, J. Veness, and M. Bowling. Sketch-Based Linear Value Function Approximation. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2222–2230. 2012.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR*, abs/1207.4708, 2012.

[4] M. G. Bellemare, J. Veness, and M. Bowling. Investigating Contingency Awareness Using Atari 2600 Games. In J. Hoffmann and B. Selman, editors, *AAAI*. AAAI Press, 2012.

[5] N. Böhm, G. Kókai, and S. Mandl. Evolving a Heuristic Function for the Game of Tetris. In A. Abecker, S. Bickel, U. Brefeld, I. Drost, N. Henze, O. Herden, M. Minor, T. Scheffer, L. Stojanovic, and S. Weibelzahl, editors, *LWA*, pages 118–122. Humbold-Universität Berlin, 2004.

[6] J. Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239, Jan. 2011.

[7] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 2764–2771, Piscataway, NJ, USA, 2009. IEEE Press.

[8] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. On the Performance of Indirect Encoding Across the Continuum of Regularity. *Trans. Evol. Comp*, 15(3):346–367, June 2011.

[9] L. C. Cobo, P. Zang, C. L. I. Jr., and A. L. Thomaz. Automatic State Abstraction from Demonstration. In T. Walsh, editor, *IJCAI*, pages 1243–1248. IJCAI/AAAI, 2011.

[10] D. B. D'Ambrosio and K. O. Stanley. Generative encoding for multiagent learning. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 819–826, New York, NY, USA, 2008. ACM.

[11] C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of 25th International Conference on Machine Learning (ICML)*, pages 240–247, 2008.

[12] G. Edgers. Atari and the deep history of video games. `http://www.boston.com/bostonglobe/ideas/articles/2009/03/08/a_talk_with_nick_montfort/`.

[13] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, 1:47–62, 2008.

[14] D. B. Fogel. *Blondie24: Playing at the Edge of AI.* Morgan Kaufmann Publishers, Sept. 2001.

[15] J. Gauci and K. O. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI)*, 2008.

[16] M. Genesereth and N. Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[17] N. Hansen. The cma evolution strategy: A comparing review. In J. Lozano, P. Larraaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer Berlin Heidelberg, 2006.

[18] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone. HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player. In *Genetic and Evolutionary Computation Conference (GECCO) 2012*, July 2012.

[19] T. Hester and P. Stone. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, 2013.

[20] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[21] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, Berlin, 2005. 300 pages, http://www.hutter1.net/ai/uaibook.htm.

[22] L. Kaiser. Learning games from videos guided by descriptive complexity. In J. Hoffmann and B. Selman, editors, *AAAI*. AAAI Press, 2012.

[23] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning, Sept. 2006.

[24] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011.

[25] S. Legg and J. Veness. An approximation of the universal intelligence measure. *CoRR*, abs/1109.5951, 2011.

[26] J. Lehman and K. O. Stanley. Novelty Seach and the Problem with Objectives. In *Genetic Programming in Theory and Practice IX (GPTP 2011)*, chapter 3, pages 37–56. Springer, 2011.

[27] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson. General video game playing. 2013.

[28] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson. General Video Game Playing. In S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 77–83. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[29] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, (406):974–978, 31 Aug. 2000.

[30] S. M. Lucas. Ms Pac-Man Competition. *SIGEVOlution*, 2(4):37–38, 2007.

[31] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, July 2012.

[32] R. Miikkulainen. Neuroevolution. In *Encyclopedia of Machine Learning*. 2010.

[33] Y. Naddaf. Game-Independent AI Agents For Playing Atari 2600 Console Games. Master's thesis, University of Alberta, 2010.

[34] M. Parker and B. Bryant. Backpropagation without Human Supervision for Visual Control in Quake II. *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG'09)*, pages 287–293, 2009.

[35] M. Parker and B. D. Bryant. Visual control in quake II with a cyclic controller. In P. Hingston and L. Barone, editors, *CIG*, pages 151–158. IEEE, 2008.

[36] T. Schaul. A video game description language for model-based or interactive learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, Niagara Falls. IEEE Press.

[37] T. Schaul, J. Togelius, and J. Schmidhuber. Measuring intelligence through games. *CoRR*, abs/1109.1314, 2011.

[38] J. Schrum, I. V. Karpov, and R. Miikkulainen. *Humanlike Combat Behavior via Multiobjective Neuroevolution*, pages 119–150. Springer Berlin Heidelberg, 2012.

[39] K. O. Stanley. Evolving neural network agents in the NERO video game, 2005.

[40] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

[41] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[42] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Technical Report AI01-290, The University of Texas at Austin, Department of Computer Sciences, June 1 2001. Mon, 28 Apr 103 21:15:41 GMT.

[43] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[44] K. O. Stanley and R. Miikkulainen. Evolving a roving eye for go. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *GECCO (2)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1226–1238. Springer, 2004.

[45] P. Stone and R. S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

[46] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In L. Getoor and T. Scheffer, editors, *ICML*, pages 1017–1024. Omnipress, 2011.

[47] I. Szita and A. Lörincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, 2006.

[48] T. Murphy VII. The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel. In *The Association for Computational Heresy (SIGBOVIK) 2013*, April 2013.

[49] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super mario evolution. In P. L. Lanzi, editor, *CIG*, pages 156–161. IEEE, 2009.

[50] V. K. Valsalam, J. Hiller, R. MacCurdy, H. Lipson, and R. Miikkulainen. Constructing controllers for physical multilegged robots using the enso neuroevolution approach. *Evolutionary Intelligence*, 5(1):1–12, 2012.

[51] P. Verbancsics and K. O. Stanley. Evolving Static Representations for Task Transfer. *J. Mach. Learn. Res.*, 11:1737–1769, August 2010.

[52] S. Wintermute. Using Imagery to Simplify Perceptual Abstraction in Reinforcement Learning Agents. In M. Fox and D. Poole, editors, *AAAI*. AAAI Press, 2010.

Appendix A Experimental Parameters

ALE Params	Max Frames per Episode	50,000
	Action Set	Legal Actions
	Max Action Set Size	18
	Min Action Set Size	9
Evolution Params	Number of Generations	150
	Individuals per Generation	100
	Add Node Probability	.03
	Add Link Probability	.05
	Demolish Link Probability	0
	Mutate Link Weight Probability	.8
Substrate Dimensionality	Object-based experiments	8×10
	Noise-based experiments	16×21
	Pixel-based experiments	16×21
	CMA-ES based experiments	4×5
Object Recognition	Max Number Object Classes	8
	Min Number of Object Classes	1
Number Weights in ANN	Object-based experiments	7120-52640
	Noise-based experiments	115920-118944
	Pixel-based experiments	906129-909216
	CMA-ES based experiments	580-3560
Number of Hidden Units in ANN	Object-based experiments	80
	Noise-based experiments	336
	Pixel-based experiments	336
	CMA-ES based experiments	20
Nonlinearities	Artificial Neural Network	Sigmoid
	Compositional Pattern Producing Network	Sinusoid, Gaussian
Trials Per Result	HyperNEAT (all representations)	5
	NEAT (all representations)	5
	CNE Object Representation	5
	CNE Noise Representation	1
	CMA-ES Object Representation	5
	Random	30

Appendix B Scores

Game	H-NEAT Object	H-NEAT Pixel	H-NEAT Noise	NEAT Object	NEAT Noise	CNE Object	CNE Noise	CMA-ES Object	Random	SARSA	Planning	Human
air raid	4335	**7415**	2380	4945	3450	4180	3075	4615	548.3	–	–	–
alien	2246	1586	764	**4320**	1842	1988	1960	2730	116.7	939.2	7785	–
amidar	218.8	184.4	124	325.2	187.2	161.8	195	**467.4**	0.9	103.4	180.3	5342
assault	2396	912.6	764.4	**2717.2**	799	1276.8	693	2219.4	366.1	628	1512.2	–
asterix	2550	2340	1410	1490	1240	1310	1300	**2970**	315	987.3	290700	–
asteroids	220	1694	5316	4144	7204	4800	**6940**	3734	1394.3	907.3	4660.6	663000
atlantis	44200	61260	78640	**126260**	103120	96080	86000	81980	32063.3	62687	193858	372400
bank heist	**1308**	214	68	380	106	372	110	354	17.3	190.8	497.8	–
battle zone	37600	36200	48600	45000	41800	30600	20000	**51600**	2000	15819.7	70333.3	238000
beam rider	1443.2	1412.8	1360.8	1900	1437.6	1494.4	1380	**1736.8**	414.7	929.4	6624.6	23020
berzerk	1358	1394	1296	1202	1536	988	**1560**	1044	171.7	501.3	670	104450
bowling	**250.4**	135.8	101.4	231.6	170.6	223	147	168.6	23.8	43.9	43.9	237
boxing	91.6	16.4	33.8	**92.8**	60.8	45.2	57	71.8	-2.2	44	100	–
breakout	40.8	2.8	2.4	**43.6**	5	17.2	4	23.2	0.8	5.2	364.4	825
carnival	3532	2544	3048	4920	4338	**5514**	3880	3064	915.3	2323.9	5132	824580
centipede	**33326.6**	25275.2	20309.8	22469.6	24831.6	22866	21849	27293	2711.3	8803.8	125123	163278
chopper command	8120	3960	2680	4580	3360	5960	3500	**13360**	836.7	1581.5	34018.8	201600
crazy climber	12840	0	11680	25060	23560	**25320**	21400	19980	2506.7	23410.6	98172.2	–
defender	15080	14620	22830	16250	**36300**	16410	35250	12230	4370	–	–	550000
demon attack	3082	**3590**	2967	3464	2707	3377	3065	2834	354.7	520.5	28158.8	92420
double dunk	4	2	0	10.8	10.8	**12.4**	10	2.4	-16	-13.1	24	–
elevator action	21600	0	800	**32820**	31120	32460	30300	23320	4026.7	3220.6	18100	–
enduro	112.8	93.6	90.6	**133.8**	99.4	27.4	35	95.4	0	129.1	286.3	3159.2
fishing derby	**-37**	-49.8	-72.6	-43.8	-47.4	-46.6	-51	-45.4	-93.5	-89.5	37.8	–
freeway	29.6	29	28	**30.8**	30	26.4	29	32	0	19.1	22.5	34
frostbite	2226	2260	270	1452	338	362	320	**3786**	66.7	216.9	270.5	416560
gopher	6252	364	1260	6092	2600	2588	2440	**10428**	213.3	1288.3	20560	–
gravitar	1990	370	710	**2840**	2530	2390	2300	2010	170	387.7	2850	8000
hero	3638	5090	1858	3894	5729	5067	**5775**	3546	834.5	6458.8	12859.5	159900
ice hockey	9	**10.6**	7.4	3.8	3	2.2	2	3.8	-9.9	-9.5	39.4	–
jamesbond	**12730**	5660	2060	2380	6900	410	250	1770	18.3	202.8	330	–
journey escape	20760	17100	11460	**21840**	18600	19100	18300	14660	-20026.7	-8713.5	7683.3	–
kangaroo	4880	800	720	**12800**	1280	1160	1400	8840	53.3	1622.1	1990	29700
krull	23890.2	12601.4	10005.4	20337.8	6864.2	6101.8	5688	**25908.8**	1780.3	3371.5	5037	–
kung fu master	47820	7720	20580	87340	57820	**83080**	54900	78960	553.3	19544	48854.5	65130
montezuma revenge	0	0	0	**340**	60	**340**	100	**340**	3.3	10.7	10.7	70500
ms pacman	3830	3408	2378	**4902**	3916	4112	3980	3728	167.7	1691.8	22336	211480
name this game	**8346**	6742	4570	7084	5452	6396	5210	6820	1806	2500.1	15410	–
phoenix	5860	1762	4898	7832	6290	**9178**	7960	5472	1173.7	–	–	365240
pitfall	0	0	0	0	0	0	0	0	-383.3	–	–	109858
pong	4	-17.4	-19.6	**15.2**	-16	-10	-16	-7.4	-20.7	-19	21	–
pooyan	**2900**	1222	2566	2761	2782	2883	2670	2210	505.2	1225.3	17763.4	–
private eye	**15045.2**	10747.4	10692.6	1926.4	14412.8	9750.4	14385	880	-690.3	684.3	1947.3	–
qbert	810	695	795	1935	1275	**2165**	1200	1675	222.5	613.5	17343.4	119300
riverraid	**4736**	2616	2136	4718	3116	4120	3000	3760	1524.7	1904.3	4449	135830
road runner	**14420**	3220	2900	9600	7440	5140	7000	11320	43.3	67.7	38725	–
robotank	42.4	**43.8**	37.8	18	35.6	15	33	14.4	1.6	28.7	50.4	–
seaquest	**2508**	716	760	944	860	800	840	880	114	664.8	5132.4	205210
skiing	-9127.2	-7983.6	-8836	**-6984**	-9071.2	-8452.4	-9612	-7361.2	-18196.9	–	–	–
solaris	0	160	11020	14764	**16760**	14768	16120	11740	2048	–	–	49640
space invaders	**1481**	1251	809	**1481**	1138	1246	945	1428	157.2	250.1	2718	42905
star gunner	4160	2720	1260	9580	6740	**11300**	5900	9260	683.3	1069.5	1345	–
tennis	0.2	0	0	1.2	0.6	**1.4**	0	-1	-24	-0.1	2.8	–
time pilot	**15640**	7340	10020	14320	14600	13060	12800	10180	3660	3741.2	63854.5	–
tutankham	110	23.6	20.4	142.4	126	**163.4**	139	126	24.3	114.3	225.5	–
up n down	6818	**43734**	11632	10220	9050	6466	7790	8404	140.3	3532.7	74473.6	–
venture	**400**	0	0	340	200	160	200	0	0	66	66	–
video pinball	82646	0	72326.4	253986	**426194.4**	335109	419666	234699.8	14130	15046.8	254748	56851
wizard of wor	3760	3360	6660	**17700**	14260	10720	15900	7600	1050	1981.3	105500	47100
yars revenge	24405.4	24096.4	29921	32836.4	**48288.6**	21472.2	44085	27328	3252.2	–	–	965347
zaxxon	4680	3000	600	**6460**	3600	6260	3000	6420	0	3365.1	22610	44100
Times Best	14	5	0	17	4	10	3	8	–	–	–	–

Table 1: Scores for different learning algorithms and comparison methods. All episodes are capped at 50,000 frames. Planning, SARSA, and Random agents are averaged over 30 trials. Planning and SARSA scores obtained from [3]. Evolved agents show champion fitness from a single run. The skiing game in ALE gives a reward of -1 for each frame resulting in negative scores required to encourage an RL algorithms to quickly finish a slope. The human expert's score is listed in seconds required to complete the ski-course.