ISSN (e): 2250 – 3005 || Volume, 07 || Issue, 01|| January – 2017 ||
International Journal of Computational Engineering Research (IJCER)

Advanced Approach to Play a Atari Game

Ms. M. Nandhini,
Master of Computer Application,

ABSTRACT
Addresses the challenge of learning to play many different video games with little domain-specific knowledge. It introduces a neuroevolution approach to general Atari 2600 game playing. Four neuroevolution algorithms were paired with three different state representations and evaluated on a set of 61 Atari games. The neuroevolution agents represent different points along the spectrum of algorithmic sophistication - including weight evolution on topologically fixed neural networks (conventional neuroevolution),covariance matrix adaptation evolution strategy (CMA–ES), neuroevolution of augmenting topologies (NEAT), and indirect network encoding (HyperNEAT).These results suggest adaptation evolution strategy (CMA–ES), neuroevolution of augmenting topologies (that neuroevolution is a promising approach to general video game playing (GVGP).

Index Terms: *Atari, HyperNEAT, Neuroevolution, object level state representations, raw-pixel representation, noise screen state representation, visual processing, atari-hyperneat interface.*

I. INTRODUCTION

Challenge for AI is to develop agents that can learn to perform many different tasks. 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 such as Pitfall, shooting games such as Space Invaders and Centipede, rudimentary 3-D games such as Battle zone, and arcade classics such as Frogger and Pac-Man.

Figure 1.1 Atari Device

The algorithm work builds upon HyperNEAT–GGP, a HyperNEAT-based general Atari game playing agent 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. Atari was considered wildly successful as an entertainment device.

- First, there are 418 original Atari games.
- Second, there exists a high-quality Atari emulator, Arcade Learning Environment (ALE), which is designed specifically to accommodate learning agents.
- Third, the Atari state and action interface is simple enough for learning agents.
- Fourth, the games have complex enough dynamics to be interesting, yet at the same time, they are relatively short and have a limited 2-D world with relatively few agents.

The state of an Atari game can be described relatively simply by its 2-D graphics. 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.

Neuroevolution Algorithm:
Neuroevolution is a form of machine learning that uses evolutionary algorithms to train artificial neural networks. It is the most commonly applied in artificial life, computer games and evolutionary robotics.
Neuroevolution can be applied more widely than supervised learning algorithms, which requires a syllabus of correct input and output pair. HyperNEAT-GGP, an agent which uses an evolutionary algorithm called Hypercube-based Neuroevolution of Augmenting Topologies (HyperNEAT).Unlike most other approaches, HyperNEAT is capable of exploiting geometric regularities present in the 2D game screen in order to evolve highly effective game playing policies.
The object detection and game playing machinery used by HyperNEAT-GGP must be generally applicable enough to handle multiple games rather than specialized towards a single game such as Pac-Man. What simple algorithm does is:

a) Take certain neural network architecture, feed forward, or even recurrent, and make N number (say 100) of these neural nets each randomised with different weights.
b) Try to accomplish the task at hand (balance the pendulum), for each of the 100 networks, and then score how well each performs the task (assign the score of say the average angle squared during the life of the task, where zero degree is upright)
c) Sort the networks by scores achieved, and keep the top 20 networks, and throw the rest away. For each network, dump all the weights and biases into a one-dimensional array of floats, in a consistent orderly way, and call that the chromosome of the network.
d) Generate 80 new chromosomes by mixing and matching the top 20 network's chromosomes by performing simple random crossover and mutation. The theory is that the 'good stuff' of how stuff should be done should be embedded in the chromosomes of the winners, and by combining the weights of the winners to generate new chromosomes, the hope is the 'offsprings' will also be good, or better than the parents. From these 80 new chromosomes, generate 80 new networks.
e) Repeat tasks (2) -> (4) until the desired score of the best network is somewhat satisfactory.

II. STATE REPRESENTATIONS:

I. 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.
Objects may be automatically identified at runtime by checking if any of the on the current screen matches any of the saved object images. 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.

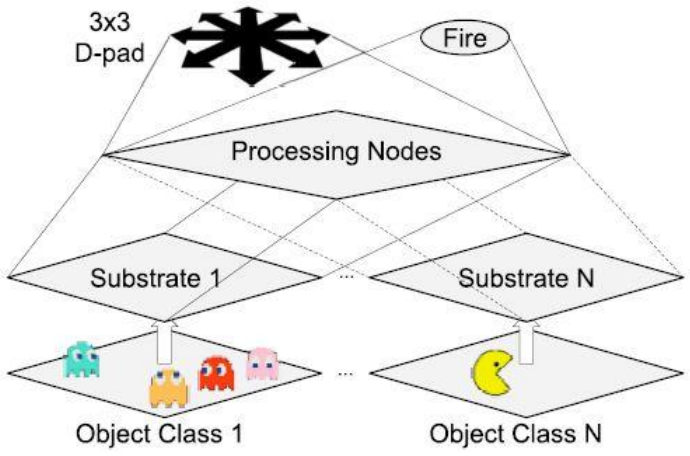

Figure 2.1 Object Level State Representations

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. 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 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 possible.

III. RAW-PIXEL REPRESENTATION

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 down sampled 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 colorpalete.
The raw-pixel representation (shown in Figure 3.1) 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).

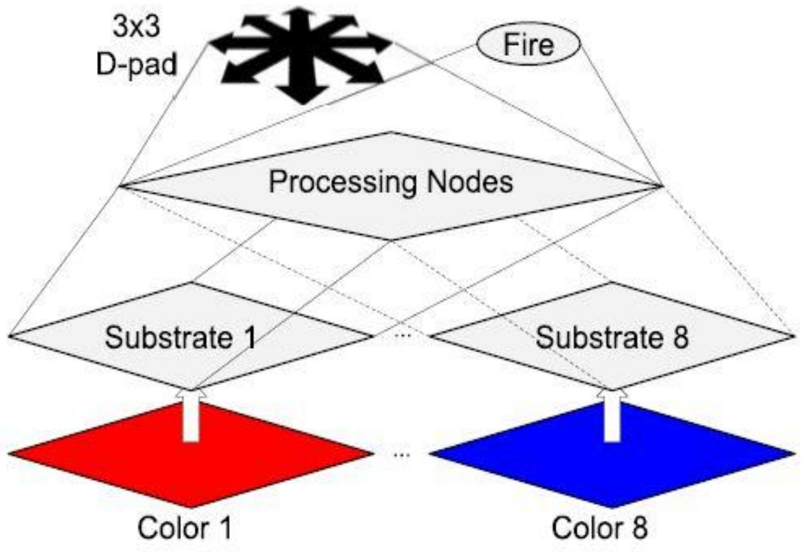

Figure 3.1 Raw Pixel State Representation

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.

IV. NOISE-SCREEN REPRESENTATION

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.
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. 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.

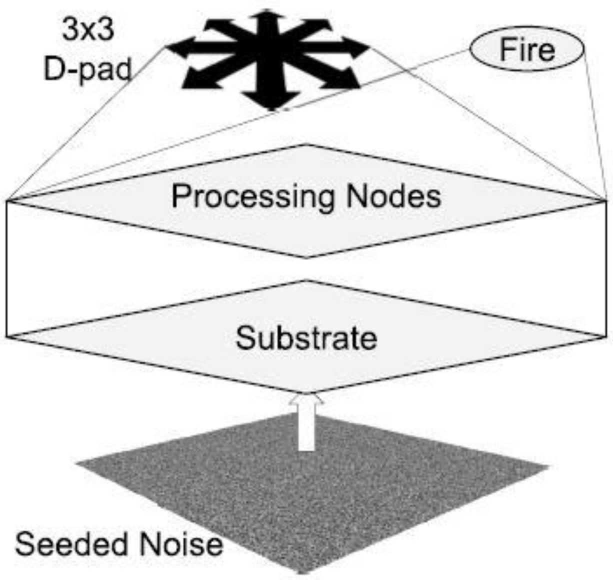

Figure 4.1 Noise Screen State Representation

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 is shown in figure 4.1.

I. Visual processing:
Machine Learning Problem, the question is how to encode the state space is of great importance. HyperNEAT-GGP uses an overhead object representation of the current game screen. Atari provides only the raw pixels of the screen as input a visual processing stack identifies objects and game entities without a priori knowledge of a specific game.
A graphical depiction of this stack identifies, it is likely possible to learn from the raw screen's pixels, object detection requires little work and reduces the complexity of the learning task by elimination pixels not directly relevant to relevant to playing the game.

Ii. Atari-HyperNEAT interface
After extracting object classes as well as the location of the self from the raw game screen, this information needs to be sent to HyperNEAT. HyperNEAT evolves a CPPN that encodes an ANN. HyperNEAT access to a fully connected 2-layer ANN whose weights have been specified by the CPPN. At a high level, information from the game screen needs to be translated to activations of nodes in the input layer of the ANN. Then, after the network has been run in the standard feed-forward fashion, the activation of nodes on the output layer must be interpreted in order to select an action.

Iii. 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.
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 s on the same game:

$$\text{Normalized Score} = \frac{\text{Score} - \min(s)}{\max(s) - \min(s)}$$

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

$$\text{z-score} = \frac{\text{Score} - \mu}{\sigma}$$

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.

IV. HyperNEAT:

HyperNEAT evolves an indirect encoding called a Compositional Pattern Producing Network (CPPN). The CPPN is used to de ne 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, 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 input 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.

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.

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.

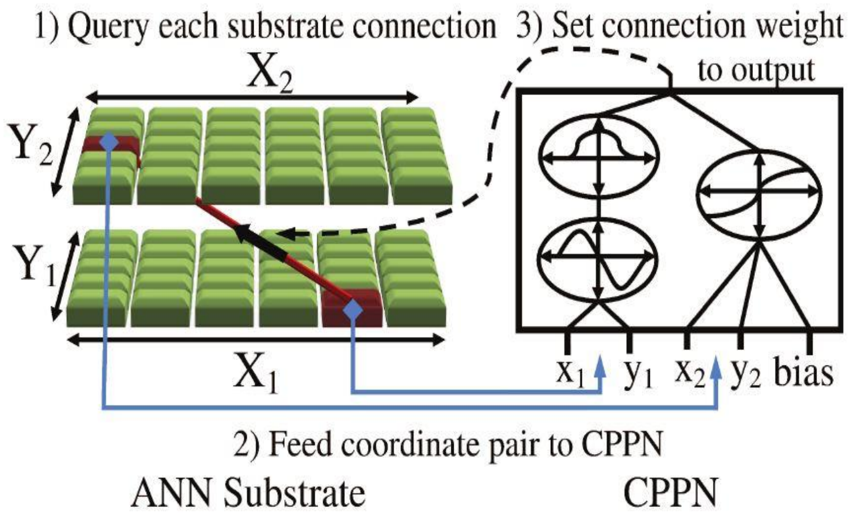
1) Query each substrate connection
3) Set connection weight to output
X2
Y2
Y1
X1
x1 y1 x2 y2 bias
2) Feed coordinate pair to CPPN
ANN Substrate
CPPN

Figure 4.2 The HyperNEAT Method Of Neuroevolution

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. Specifically, HyperNEAT works in four stages:

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.

The CPPN is used to determine the weights for every pair of *(input, output)* nodes in adjacent layers of the ANN.

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. Based on fitness scores, the population of CPPNs is maintained, evaluated, and evolved via NEAT.

V. Advantages of HyperNEAT

HyperNEAT handles,

- Problems with a large number of inputs.
- Problems with a large number of outputs.
- Problems with variable resolution.
- Problems with geometric relationships among inputs and/or outputs.
- Multi-agent learning problem.
- Neural plasticity

V. CONCLUSION

HyperNEAT-GGP, into a fully general Atari 2600 video game player. 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. 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.

Comparing the different neuroevolution 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.

REFERENCES

[1]. Mathhew Hausknecht, Joel Lehman,Risto Miikkulainen, Peter Stone. A Neuroevolution Approach tp General Atari Game Playing.

[2]. Barbu, S. Narayanaswamy, and J. M. Siskind. Learning physically-instantiated game play through visual observation.

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

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

[5]. S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, editors, *Articial and Computational Intelligence in Games*, volume 6 of *DagstuhlFollow-Ups*SchlossDagstuhl Leibniz-ZentrumfuerInformatik, Dagstuhl, Germany,2013.

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

BIOGRAPHIES

Ms. M.Nandhini, completed MASTER OF COMPUTER APPLICATION. She is a talented, dedicated and hard working student.