Deep Reinforcement Learning in Mario

Kartik Singhal 150050025
Nilesh Gupta 150050059
Umesh Kumar 150050052
Arshdeep Singh 150050106

Abstract

Using AI and RL algorithms to play computer games has been widely discussed and investigated, because valuable observations can be made on the RL play pattern vs. that of a human player, and such observations provide knowledge on how to improve the algorithms. Genetic algorithms are known to perform well on non convex optimisation problems where there is no prior model known beforehand. NeuroEvolution of Augmenting Topologies (NEAT) is a method of neuroevolution, which outperforms the best fixed-topology method on challenging benchmark reinforcement learning task. In this report we explore applications of NEAT learning strategy on mario playing agent having no prior knowledge of the environment and with a raw view of world (i.e. no handcoded features).

1 Introduction

Video and computer games often use artificial intelligence (AI) code to give life to non-playable characters (NPCs) in the game. For example, enemy NPCs are programmed to attack your in-game character while ally NPCs are programmed to help. In our experiments, we used artificial intelligence for an entirely different purpose: to evolve an agent to excel at a certain task. Inspired by Seth Bling's "MarI/O Machine Learning for Video Games" video[6], we decided to conduct experiments on Mario.

In Super Mario Bros. (SMB), a popular two-dimensional platform game, the human player controls a character named Mario and tries to clear each level of the game by exploring and finding the finish flag. Along the way, the player can collect coins, kill enemies, be killed off by enemies, and more. Most levels are predictable so that the player can expect exactly what will happen, such as when and where an enemy will appear or where the coins are. Players often memorize key features in each level to optimize their score. SMB presents many challenges to reinforcement learning methods, including a large state space, unknown dynamics of obstructions and achieving a good score requires sophisticated and varied strategies.

NEAT is an approach where neural network evolves artificially over several generations. It is found to be better than fixed topologies neural networks in many reinforcement learning tasks. The fitness function acts as a feedback to the algorithm and it tries to find the best neural network which can maximize the fitness function.

Salient Features of NEAT[7]:

- **Genetic Encoding** : NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross over during mating.
- **Tracking Genes through Historical Markings** : Whenever a new gene appears (through structural mutation), a global innovation number is incremented and assigned to that gene. The innovation numbers thus represent a chronology of the appearance of every gene in the system.
- **Protecting Innovation through Speciation** : Speciating the population allows organisms to compete primarily within their own niches instead of with the population at large. This

way, topological innovations are protected in a new niche where they have time to optimize their structure through competition within the niche.

- **Minimizing Dimensionality through Incremental Growth from Minimal Structure** : NEAT biases the search towards minimal-dimensional spaces by starting out with a uniform population of networks with zero hidden nodes (i.e., all inputs connect directly to outputs). New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In other words, the structural elaborations that occur in NEAT are always justified.

We chose NEAT algorithm because it often arrives at effective networks more quickly than other contemporary neuro-evolutionary techniques and reinforcement learning methods.[8]

The main advantage of using NEAT is that the topologies are minimized and grown incrementally during evolution resulting in higher learning speed than other neural evolution algorithms. The increase in speed is attributed due to the following reasons:

- It allows disparate topologies to crossover in meaningful ways. It is possible because of genetic encodings used in NEAT.
- It protects structural innovation through speciation.
- It minimizes the dimensionality of search space through incremental growth from minimal structure.

1.1 NEAT overview

In NEAT, a population of individual genomes is maintained. Each genome contains two sets of genes that describe how to build an artificial neural network:

- Node genes, each of which specifies a single neuron.
- Connection genes, each of which specifies a single connection between neurons.

To evolve a solution to a problem, the user must provide a fitness function which computes a single real number indicating the quality of an individual genome: better ability to solve the problem means a higher score. The algorithm progresses through a user-specified number of generations, with each generation being produced by reproduction (either sexual or asexual) and mutation of the most fit individuals of the previous generation.

The reproduction and mutation operations may add nodes and/or connections to genomes, so as the algorithm proceeds genomes (and the neural networks they produce) may become more and more complex. When the preset number of generations is reached, or when at least one individual (for a fitness criterion function of max; others are configurable) exceeds the user-specified fitness threshold, the algorithm terminates.

One difficulty in this setup is with the implementation of crossover - how does one do a crossover between two networks of differing structure? NEAT handles this by keeping track of the origins of the nodes, with an identifying number (new, higher numbers are generated for each additional node). Those derived from a common ancestor (that are homologous) are matched up for crossover, and connections are matched if the nodes they connect have common ancestry. (There are variations in exactly how this is done depending on the implementation of NEAT)

Another potential difficulty is that a structural mutation - as opposed to mutations in, for instance, the weights of the connections - such as the addition of a node or connection can, while being promising for the future, be disruptive in the short-term (until it has been fine-tuned by less-disruptive mutations). How NEAT deals with this is by dividing genomes into species, which have a close genomic distance due to similarity, then having competition most intense within species, not between species (fitness sharing). How is genomic distance measured? It uses a combination of the number of non-homologous nodes and connections with measures of how much homologous nodes and connections have diverged since their common origin. (Non-homologous nodes and connections are termed disjoint or excess, depending on whether the numbers are from the same range or beyond that range; like most NEAT implementations, this one makes no distinction between the two.)

2 Previous Approaches

In the recent past a lot of reinforcement learning techniques have been used for training various version of Mario Bros. including genetic algorithms, neural networks, Q-learning, et cetera. Liao, et. al. used Q-Learning which achieves tractable computation [2]. Baldominos, et. al. used genetic algorithm with lot of heuristics to train an agent [1]. Mora, et. al. used evolutionary methods to improve behavioural models (Finite State Machines, FSMs) [3]. Interestingly, Tom, et. al. used a simple approach of lexicographic orderings and time travel to train the agent, though it works only for first level of Super Mario Bros [9].

Additionaly, neural networks have been trained to enhance UX in the past. Pedersen et al. trained neural networks to predict which level a player would prefer in a clone of Super Mario Bros [5]. Shaker et al. later used trained perceptron models to evolve personalised levels for this Super Mario Bros game, through simply searching the parameter space for levels that maximise particular aspects of player experience [4].

3 Experimental Setup

3.1 Simulator

We use OpenAI to provide a way of letting the computer do the work of playing Super Mario Bros on the NES using FCEUX emulator. An NES file is a game ROM created from an NES (Nintendo Entertainment System) video game. It contains the same data as the original NES cartridge and can be opened and played on a Mac or PC using an NES emulator. The simulator creates an environment which acts as an interface between the game running on the NES and the agent that we wrote. The environment allows you to play the original Super Mario Bros in two flavors. First one is Regular in which the word view is simply an 256x224x3 RGB array representation of the current world, whereas the second called Tiles provides 16x13 array representation of the screen. It provides an encoding of the current world and corresponding to an action returns state of the game, coins collected, distance travelled, lives and time left and an aggregate score.

link : openai (`https://openai.com/about/`)

3.2 World Encoding

To demonstrate the true power of NEAT we decided to not use any hand coded features for the encoding, rather the agent receives a raw grid representation of the world. The pictorial representation of the grid can be seen in top left corner of Figure 1.

The grid is of dimension 13x16 and each entry is from 0, 1, 2, 3, empty space is encoded with 0. bricks are encoded with 1, enemies are encoded with 2 and agent is encoded with 3. This grid is flattened to a list and passed to the neural network to get corresponding action.

The action set for the agent is (Up, Left, Down, Right, A, B) which is encoded as a list of 6 booleans. For e.g. action = [0, 0, 0, 1, 1, 0] would activate right (4th element), and A (5th element). The output of the neural network is this list only.

3.3 NEAT parameters

The number of input and output nodes to the network are 208 (16x13) and 6 (corresponding to each key in the action) respectively. We further threshold each of the 6 outputs of the network at 0 to binarize the action before feeding it to the simulator. Following are relevant NEAT parameters :

- As the outputs were symmetrically thresholded about origin, we used tanh as activations for nodes in neural network.
- To keep the number of species and number of genomes in each species reasonable, we keep the compatibility threshold at around the mean genomic distance. Genomic distance is an approximate measure of the difference between genomes used in dividing the population into species according to compatibility threshold. We tried sum and max for aggregation options at each node.

Figure 1: World representation in SMB

- In order to avoid extinction of all species we have kept the species elitism to 2, i.e. we preserve 2 species with highest maximum fitness even after they become stagnated
- We have kept the mutation rate across all components around 0.5% in order to give some randomness to a progeny but not too much to thwart the evolution process
- We start our experiments with a population of 300 and it is initialized with random values from a gaussian distribution

The most important aspect of genetic algorithm is the fitness function for it's genomes. We tried several combinations of distance, score, coins collected, lives and time left. But in our experiments, found the distance to be most promising fitness function. Also, we truncate episodes when mario becomes stale for more than a threshold time for achieving speedup.

4 Observation and Results

4.1 Experimentation

We tried many different approaches for the learning, our first approach was to give the agent a limited view of the world (i.e. instead of full grid, we passed a 7x7 grid) in first few generations. The motive behind this approach was to make the agent learn responses corresponding to immediate surroundings first. This approach did give some good initial results but it failed to give a candidate which could do very good in the long run.

We also tried a fitness function in which, in addition to distance reward we gave additional reward for total actions executed, in hope that this would ensure liveness of the agent. But in our experiments this approach was not able to produce very good candidates, this might be because this approach gave greater reward to an agent who is not dying but trying different actions at its place than an agent who dies quickly but progresses further in the game.

4.2 Results

After running upto 35 generations, our agent was able to successfully complete the level. We can see the performance comparison of 5 prominent species over the generations in Figure 2 and comparison of generations vs average fitness in Figure 3. The fitness gradually increases with generations and the species such as 1, 2 and 3 which show stagnation became extinct over time.

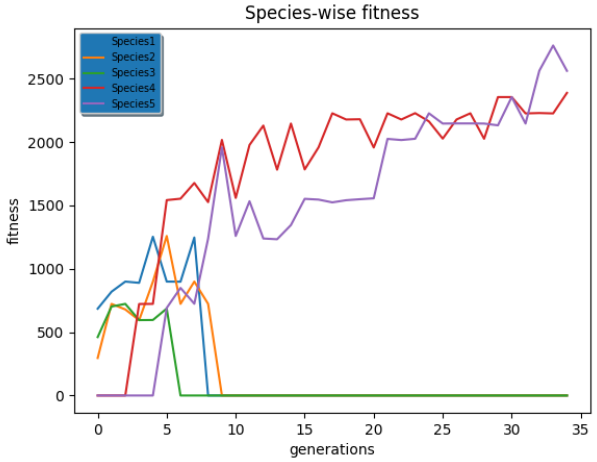

Figure 2: Generation vs Max Fitness over different species

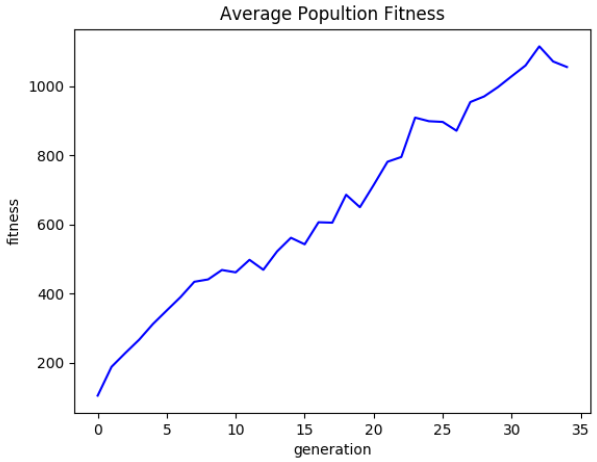

Figure 3: Generation vs Average Fitness

5 Conclusion

The agent learned to complete the level in about 35 generations and it takes around 3 hours to run a complete simulation, which speaks strongly for the ability of NEAT to perform well in complex games in less time. For future work, we plan to use CNN on successful scenarios to extract defining features from the grid and then use these features as input to NEAT neural network.

Following is the github link of our code : `https://github.com/dhruvumesh25/mario-neat`

References

[1] Alejandro Baldominos, Yago Saez, Gustavo Recio, and Javier Calle. Learning levels of mario ai using genetic algorithms. In *Proceedings of the 16th Conference of the Spanish Association for Artificial Intelligence on Advances in Artificial Intelligence - Volume 9422*, pages 267–277, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[2] Y. Liao and Kun Yi. Cs 229 final report reinforcement learning to play mario. 2012.

[3] Antonio Mora, Juan Merelo Guervós, Pablo García-Sánchez, Pedro Castillo, Maria Rodríguez-Domingo, and Rosa Hidalgo-Bermúdez. Creating autonomous agents for playing super mario bros game by means of evolutionary finite state machines. *Evolutionary Intelligence*, 6:1–14, 03 2014.

[4] Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling player experience in super mario bros. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy, 7-10 September, 2009*, pages 132–139, 2009.

[5] Christopher Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 3 2010.

[6] SethBling. *MarI/O - Machine Learning for Video Games*, 2015.

[7] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[8] Matthew Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods for reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–28, July 2006.

[9] Dr. Tom and Murphy Vii Ph. D. The first level of super mario bros. is easy with lexicographic orderings and time travel... after that it gets a little tricky., 2013.