Reinforcement Learning Project

Applying Reinforcement Learning for Atari Games using NEAT Approach

Haibo Jin
ID: 014343698

University of Helsinki
Department of Computer Science
Master student

1 Topic Introduction

The project aims to build an AI agent for playing Atari games. The agent is based on Neuro-Evolution of Augmenting Topologies (NEAT). NEAT is a neuro-evolution algorithm, evolving topologies of Artificial Neural Networks (ANNs) along with its weights. Different from the algorithms that are based on value function space, neuro-evolution is a reinforcement learning approach that searches policy space by using evolutionary algorithms. With slight modification, NEAT can be applied for developing a general video games playing agent in Atari games, which is able to search in a larger space but with more general inputs.

2 Environment

The Atari 2600 is a video game console which is considered as a successful entertainment device. It consists of 418 original games and supports a second player in many of them. Furthermore, the Atari 2600 has a simple interface for building learning agents. The state of the game can be directly described by its 2D grahpics (a native resolution of 160×210, which makes it complex enough to simulate the real-world). The action space consists of 18 different possible actions (the combination of eight directions of movement and a single button), of which the size is just moderate. Even though the Atari 2600 is no longer popular as a video game, its diversity and extensibility make it adequate for the research of learning agent. The Arcade Learning Environment (ALE) is a free and open-source emulator of Atari 2600 games, which is based on a simple object-oriented framework. The ALE allows researchers to build AI agents for Atari 2600 games without focusing on the details of simulation.

With several interfaces to choose, we use the internal interface of ALE in the project, which are directly compiled inside ALE. We use C++ as the programming language of the project,being consistent with the source code.

3 Implementation

A new agent called NEAT_agent is built in the source code of ALE, under a folder named Agents. The agent mainly controls the action choice of the game. Thus, it mainly serves the evaluation part of the whole process. The process of training can be divided into three stages: evaluation, speciation and reproduction. The code for speciation and reproduction is basically in a folder named NEAT. The workflow of the process is presented in Figure 1.

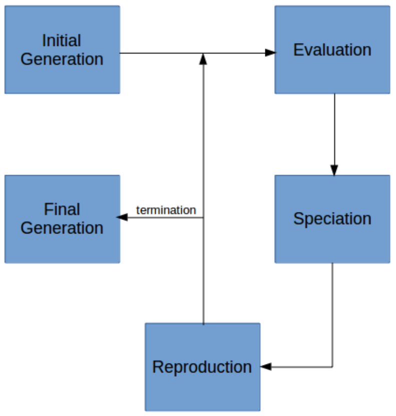

Figure 1: The agent starts from an initial generation. Then it enters into a loop which consists of three stages, namely evaluation, speciation and reproduction. When doing evaluation, every organism of the generation is evaluated by giving a fitness score. Then the generation will be split into several species based on a compatible testing rule. So the organisms with similar structures will be in the same species. For each species, it should calculate the amount of offspring to generate, then use the reproduction rules to get new organisms for the next loop.

After training, the speciation and reproduction part will be closed. Only the evaluation part will be reserved for choosing actions, using the best trained neural network.

3.0.1 Evaluation

For one organism, the evaluation is based on one episode. When one episode ends, the total score of the game represents the fitness of the corresponding organism.

In order to shorten the time of training, the feature extraction algorithm used on the screen shot is quite simple. The extraction is used on the lower half of the image, which is divided into 10×10 grids. The image is firstly transformed to a gray scale image so that every pixel can be represented as an integer (0 to 255). For each small grid, if its average pixel value is larger than the global average pixel value, it will be denoted as 1. Otherwise, it becomes 0. By doing this to all the 100 grids, the image can be transformed to a binary vector with length 100.

Since the length of input, a binary vector, is 100 and the total number of possible actions is 18, the neural network of each organism contains 100 input nodes, 1 bias node and 18 output nodes. The number of hidden nodes of the organisms are different, depending on the mutation of each network, which will be explained in section 4.1.3.

Figure 2 shows the process of one iteration about how an organism is evaluated.

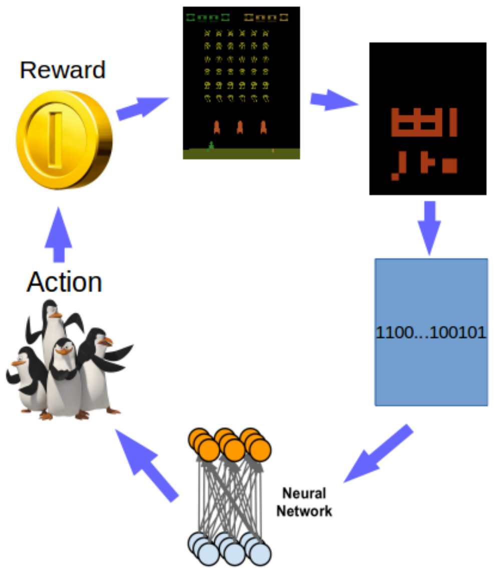

Figure 2: In each iteration, the screen shot of the current game state is firstly converted to a 10×10 binary grids. The grids are then transformed to a vector, as the input of the neural network. When the value of input is propagated to the output nodes, one of the 18 actions will be activated and executed. Then the laser cannon may die, or get a point, or nothing happens, all of which reflect the reward of the selected action.

3.0.2 Speciation

When doing speciation, for every species of the current generation (there is only one species initially), the best organism of a species is set to be the representative of the species. Then the rest others need to do a compatible testing with each representative one by one. Whenever a organism is compatible with a representative, the organism is added to the corresponding species. If none of the representatives is compatible, a new species will be built for the organism. Finally, all the organisms are divided into groups and the structures of networks are similar within the same group. The formula for compatible testing is listed as following:

$$\delta = c_1 D + c_2 W \tag{1}$$

where δ is the compatible distance of two organisms, D represents the number of mismatch genes and W represents the average weight differences of matching genes. c_1 and c_2 are coefficients that used to balance the weights of D and W.

In order to avoid any species taking over the entire population, explicit fitness sharing is used to solve such a problem. That is, the fitness of each organism is adjusted based on the species it belongs to. The adjusted fitness of organism O_i is calculated as follows:

$$f_i^{'} = \frac{f_i}{F_i} \tag{2}$$

where $f_i^{'}$ is adjusted fitness, f_i is original fitness and F_i is the sum of fitness of all the organisms that in the same species of O_i. With explicit fitness sharing, a species cannot become too big even many of its organisms have good performance. Assume N is the population of the whole generation, and F is the sum of the adjusted fitness of all the organisms, then the amount of offspring that a species needs to generate is calculated as follows:

$$N_i = N \times \frac{F_i}{F} \tag{3}$$

3.0.3 Reproduction

Reproduction means to generate offspring based on the current organisms, through some genetic operators, such as crossover and mutation. In the experiment, the reproduction is conducted within a species. For a certain species S_i, firstly, a proportion of S_i with worst performance will be eliminated. Then, the best ones will be cloned and directly added to the offspring. Additionally, some offspring come from crossover and some from mutation. Consequently, offspring with number of N_i will be generated.

For crossover, a problem is how to decide the structure of a offspring generated from two neural networks with different structures. Through historical markings, NEAT can perform crossover in a reasonable way by tracking genes. A gene, also known as the connection between two neurons, is identified by a unique innovation number. Whenever there is a new gene appears (through the mutation operator), the innovation number will be increased and assigned to that gene. Thus, same genes will have same innovation numbers, which makes it easy to do crossover between two networks with different topology structures. Figure 3 shows two networks with different topologies. Figure 4 presents the details of how the two networks doing crossover as well as the structure of their offspring.

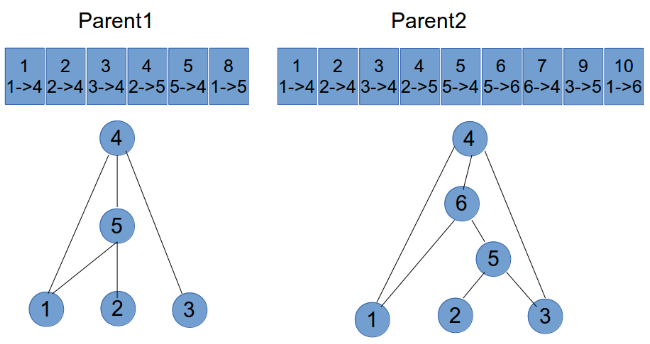

Figure 3: The two networks for crossover. The markings over the networks can be seen as a part of the genes, recording the topology of the network, namely the connections of nodes and their innovation numbers. As showed in the figure, same connections have the same innovation numbers.

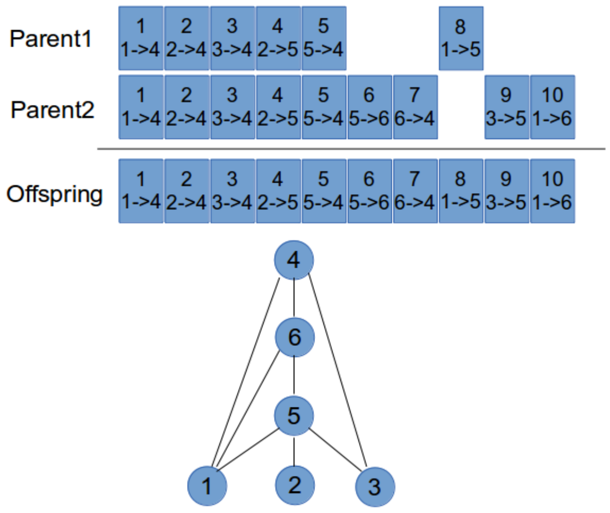

Figure 4: For the genes of the two parents, there must be some that can match up upon innovation numbers and some not. When doing crossover, for those match up genes, they will be randomly inherited by offspring from one of the parents. For those genes that do not match up, they will be inherited from one of the parents that owns better fitness score. In this case, the two parents have the same fitness, then both of their genes are inherited to the offspring.

As for mutation, there are three ways. One is to mutate the weights. For every weight of the network, there is p_w percent chance to mutate the weights, in which there is p_{w1} percent chance of uniformly perturb the weight and p_{w2}

percent chance of randomly assign a new value. Another way is adding a new connection to the network with p_c percent chance and adding a new hidden node with p_n percent probability.

4 Usage

To use the NEAT_agent to play space invaders, first is to open a terminal. Suppose the name of the parent folder is RL_Project, we need to change the current directory to RL_Project/ale_0.4.3/ale_0_4, then first type the command:

```
make
```

to compile the files and generate an executable file. Then type the follow command to execute space invaders with NEAT being the agent:

```
./ale -game_controller internal -player_agent NEAT_agent -display_screen true
space_invaders.bin
```

5 Experiment and Result

The goal of the experiment is to apply NEAT for building an AI agent of Atari games. The agent should be able to play several Atari games, which is known as general video games playing. However, due to the limitation of time and device, the experiment is simplified on some details and only tested on one game: space invaders. Space invaders is a shooting game that aims to defeat a wave of aliens with a laser cannon to get as many points as possible.

5.1 Parameter Settings

The population size and generation times are both set to 10, which is quite small for a neuro-evolution algorithm. This is for the purpose of saving training time, and it still works, as we can see in the next section. The coefficients of compatible testing is $c_1 = 1.0$ and $c_2 = 0.4$. The threshold of compatible testing, δ is 15. In reproduction, 20% of the organisms will be eliminated, then the best two are added to the next generation directly. 40% of the rest offspring are from crossover and the others come from mutation. In mutation, $p_w = 0.8$ for weights mutation, in which $p_{w1} = 0.9$ and $p_{w2} = 0.1$. The probability of adding a new link is $p_c = 0.15$ and $p_n = 0.1$ for adding a new node. The sigmoid function used in the neural network is $\varphi(x) = \frac{1}{1+e^{-4.9x}}$.

5.2 Result and Comparison

Based on five running times, the average score of one episode game of space invaders is 526, as listed in Table 1, in which the implementation of the experiment is denoted as pixel-based NEAT.

NEAT pixel	Random	SARSA object	NEAT object	Human
526	157	250	1481	42905

Table 1: The first item shows the average score of pixel-based NEAT. It also lists the scores of some other methods.

We can also see the comparison between pixel-based NEAT and some other approaches. It is worth mentioning that the implementation of the pixel-based NEAT is simplified and not well tuned. Under such conditions, the result of pixel-based NEAT is obviously better than Random, which means the agent of pixel-based NEAT has learned to choose suitable policies. SARSA (State-Action-Reward-State-Action) is a widely used reinforcement learning algorithm that has been used in a large number of problems. Compared to SARSA, pixel-based NEAT still performs a better score significantly, which means the pixel-based NEAT is effective for building an AI agent of video games. The fourth item in Table 1 gives the score of an object-based NEAT implemented by Hausknecht et al.. In addition to some implementation details and training times, the difference between pixel-based and object-based NEAT is the feature representations. The pixel-based NEAT uses a feature of down sampled screen pixels as representation while the object-based NEAT uses the detected objects as features. The performance of object-based NEAT significantly surpasses pixel-based NEAT, which implies that a more precise features can improve the performance of an agent. As we can see, the best recorded human score is 42905. Unsurprisingly, there is still a big gap between reinforcement learning algorithms and humans.