A Comparison of performance between DQN and NEAT/Hyper-NEAT in Game Environment

Chunzi Lyu
School of Information Technology and Electrical Engineering
The University of Queensland, Qld., 4072, Australia

Abstract

Past experiments have been dedicated to applying either evolutionary algorithm (EA) [3] or deep Q learning network (DQN) [2] to solve a reinforcement learning problem, but only few researches were focusing on conducting an empirical comparison between the two [1]. It is a fact that DQN remains to be the most popular algorithm for an RL task, but few experiments were made to prove its strength against other alternatives and to explore the reason behind it. A comparative research not only holds a competition and observes a winner, but also allows deeper and more intuitive understanding of both of the competitor algorithms. In this project, both NEAT/HyperNEAT and DQN algorithms are applied to play games with different level of difficulties, training and testing data are collected for analysis and comparison with regards to training time, accuracy and network structure. There is also intra-algorithm comparison that studies a single algorithm's performance in a controlled variating environment. The whole purpose is to gain a comprehensive understanding of both algorithms and explores what contributes to their strengths and weaknesses. Some optimisation techniques are also recorded.

1 Introduction

DQN

Deep-**Q** **N**etwork is one of the most popular reinforcement learning algorithms. Reinforcement learning is a mathematical framework of behavioural psychology that learns the action being rewarded the most[7]. Neural network is to approximate the action-reward function when the state * action space is too large. By exploring the state space, the model constantly gets updated from the latest changes in estimating the action value from the old state to the new state. In this project, DQN implements the experience relay technique [8] that store the state, action, reward, new state tuple in a replay buffer, and selects a batch size of buffer content for each training.

NEAT

NEAT algorithm is one of the evolutionary algorithm (EA) that models the natural selection process in the computation domain. It has three distinct features differentiates from other EAs: 1. By applying historical markings, genes can crossover with each other directly without complicated analysis on the topology; 2. By calculating and comparing differences based on historical markings, it is possible to carry out speciation within populations. Speciation is necessary for protecting innovative solutions that may appear at an early stage with too low a fitness function. 3. The first population starts out with no hidden layers, and the structure gets to grow over populations. This design drastically improves the algorithm's efficiency by reducing the complexity of the search place. It is also made possible by the unique encoding scheme NEAT provides.

Hyper-NEAT

Hyper-NEAT is an evolved version of NEAT. It adopts a new encoding scheme that maps spatial patterns to connectivity patterns of the input, which makes reuse of structure possible. HyperNEAT consists of two chained neural networks, a fully connected neural network and a Compositional Pattern Producing Network (CPPN) [6]. CPPN, which is originally used to describe 2-d pattern, now outputs the connectivity patterns of the first ANN. The ANN is evaluated in terms of fitness, according to which CPPN's output will be adjusted. During the process, NEAT is used to evolve CPPN. Through experiments, HyperNEAT's scalability is confirmed, which implies that HyperNEAT can be used to solve realistic problems that involves high dimensional space search.

Game Environment Application

Game environment provides the perfect testbed for solving reinforcement learning problems [4]. One year before the breakthrough of Alpha-go, the DeepMind published [2] on Nature revealing the success of deep Q-learning in achieving human-level performance on 49 Atari games, and one year before that, a group of Ph.D students at the University of Texas also trained Atari game agents only with neuroevolutionary approach and achieved then state-of-the-art performance [3]. Looking further into the past studies, it is obvious that the history of testing algorithms in game environment is almost as old as the reinforcement study itself. Inspired by this observation, this project also tests algorithm performance in ALE environment and a self-created grid world game.

2 Methods

The algorithm will be tested in both Atari game environments and simple grid world patterns.

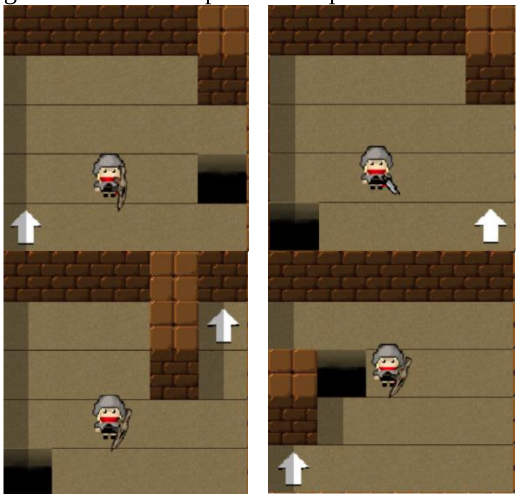
Atari

Two of the Atari games, Breakout and Pong, are chose to apply the algorithm's performance on. This is mainly to test training efficiency on raw pixel data. The deep Q neural network is equipped with three convolutional layers and two fully connected layers. Detailed implementation follows [2]. It takes in the last four frames of screen pixels and is batch trained with experience replay. NEAT algorithm implements the same epoch number of generation and the same train step number of population, for the convenience of comparison.

Grid World

Atari games are good for inter-algorithm comparison, however, since it's a commercialized game with fixed source code, it's impossible to twig the environment parameters for comparative observation. To enrich the compared items, this project implemented its own version of a grid world game and designed 27 maps that either differs on map size or map pattern/ difficulty. The 27 maps are to disintegrate the whole learning/evolving process into 27 types of small tasks, and by observing how individual solutions are formed, this project can evaluate the capacity of both algorithms at solving a RL problem.

On these maps, Hyper-NEAT and DQN is tested. Agent is instantiated at random starting location other than wall, goal and pit location. The model that is trained by algorithms should move the agent towards goal with the least possible steps.



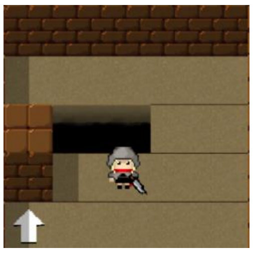


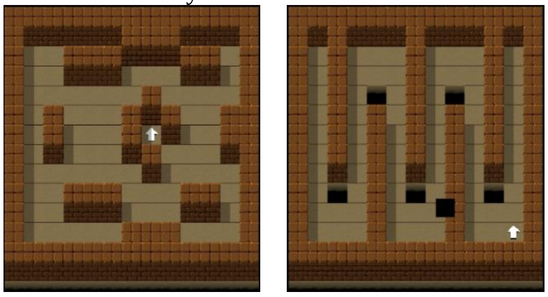
Fig 1-5. Maps of size 4x4 to 6x6 are drew with the same patterns #1 to #5 (from left to right, top to bottom)

White arrow is goal, stepping on goal will return +10 reward and terminate the game. The agent has four available actions: go up, down, left and right. Moving into wall returns -1 reward and the agent remains at the old position. Each move yields -1 reward to encourage goal reaching with the least number of steps. Stepping on pit will receive -10 reward and episode termination.

Map Design

For maps of size 4x4 to 6x6 (Figure 1-5), each training/ evolution process is set to stop after test accuracy reaches 100%. Test episode number is maximised at 100, much larger than the total grid number on map, which means the model should navigate towards goal from every location on map. For pattern #1 and #2, nothing is blocking the way from any location to the goal, which makes it the easiest maps that looks for a linear model. For pattern #3, about 2/3 of the locations need to make one turn to reach goal, and some of the trips may end up in the pit. For pattern #4 and #5, if randomised at location (0,0), the agent should make more than 3 turns to reach the goal. To finish within 10 steps, it takes 5 turns on pattern #5 to win. The same layout is repeated on size 5x5 and 6x6 to test if map size has any impact on model training.

For map of size 7x7 to 9x9, much complex patterns are drawn (Figure 6-9) – a concentric circle map, a stripe map and a vortex map. Due to time limit, models are trained within 50 epochs * 100 steps per epoch, or 50 generations * 100 population size, hence they are not expected to win 100% in all of them. However, 50 epochs are enough to observe the overall trend of development, and it's enough to complete the comparison. These designs are to observe how the map patterns influence the training process, what algorithm is good at which, and why is that. The same layout is also drawn on all three sizes.



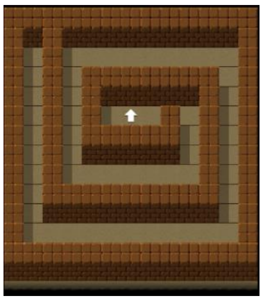


Fig 6-9. Map size 7x7 to 9x9 Pattern #1 to 3 (from left to right, top to bottom)

Comparison Framework

8 out of 27 of the maps are trained with multiple tries to examine the algorithm's resistance to randomness or stability. Performance in each map is compared regarding to efficiency ($\frac{test\ accuracy}{time}$), stability (how each epoch progresses) and network structure (number of layers, nodes and connections). Accuracy is defined by the number of wins. 10 wins out of 10 tests is deemed as 100% test accuracy. Efficiency is to evaluate accuracy with regards to time. In addition, results are also analysed vertically to examine each algorithm's reaction to different map sizes and map difficulties. Comparison framework is listed as follow:

- Inter-algorithm comparison
  - Accuracy / time
  - Stability
  - Network structure
- Intra-algorithm comparison
  - Different map size and difficulty
    - Accuracy / time
    - Network structure

3 Results and Analysis

NEAT Fails at Performing in Atari Environment

After 200 epochs of 250,000 training steps, DQ returns proper however not stable performance f game Breakout, and an excellently stable one f Pong. The reward/epoch plotting is similar to that o [2]. For NEAT on the other hand, fails at evolving a solution that wins the game. No correlation can be observed between evolution and score improvement because there has been none.

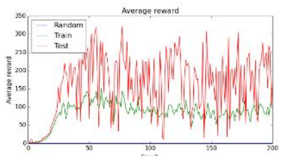


Fig 10. Breakout

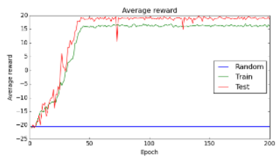


Fig 11. Pong

The method described in [3] requires a decent amount of pre-processing of object representation that converts a 210 x 160 pixel of screen into a 21 x 16 grid of objects, with each grid labelled a class number. This pre-processing greatly simplifies the environment into a grid world game. In [3], NEAT is used to evolve the substrate nodes' geometric relation, which as illustrated in the introduction, is the advanced version of NEAT, Hyper-NEAT. Hence for the grid world environment, the candidate algorithms will be DQN and Hyper-NEAT.

DQN Wins at Efficiency

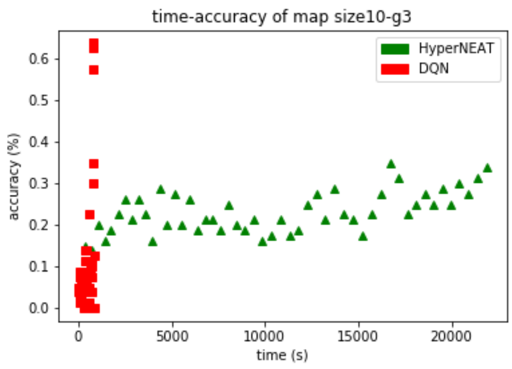


Fig 12. DQN in tall and thin, shape, Hyper-NEAT in wide and short shape.

For 21 out of 27 maps, DQN and Hyper-NEAT exhibits a similar accuracy/time distribution as shown in figure 12 -- DQN being thin and tall, Hyper-NEAT wide and short, which suggests that DQN is way more efficient at solving the task, with shorter time usage than Hyper-NEAT at a factor of 0.01, and is able to render 2 times more accurate results.

DQN Shows Stable Performance in Different Tries of the Same Training

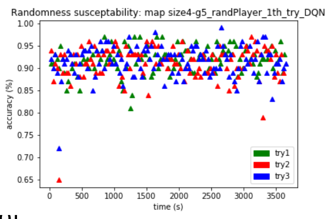


Fig 13. DQN

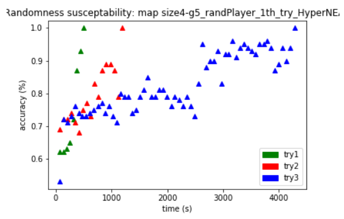


Fig 14. Hyper-NEAT

For map size4 of pattern #1 to #5, the shortest and the longest try of DQN finishing the 100% accuracy training are within the range of 70 seconds on average, while that of Hyper-NEAT are within 2233 seconds on average. Hyper-NEAT is significantly more unstable, demonstrates lower resistance to randomness. To support this point, figure 13 and 14 shows the three tries of DQN's training performance on map size4 #5 as a mixed of points, with no apparent differences in finishing time and accuracy, on the other hand, Hyper-NEAT's finishing time ranges from 500s to 4300s.

Hyper-NEAT Wins at Constructing Simpler and diversified Network Structure

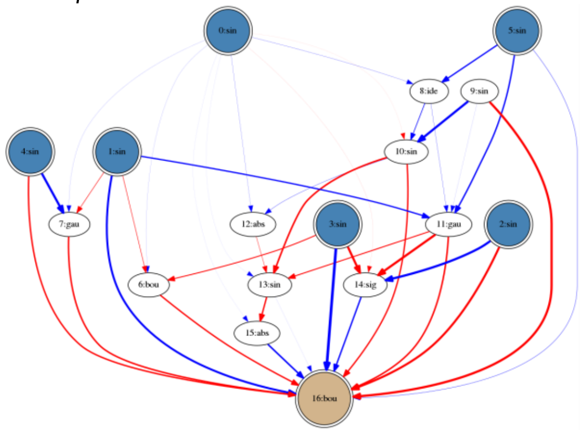


Fig 15. Network Structure Evolved with Hyper-NEAT

Fig 15 shows the most complicated network structure evolved in map size9 of pattern #1 with 6 nodes and 10 CCPN nodes, which means that all the other net structures are way simpler than DQN network. DQN network is made up with 2 layers of fully connected hidden layers with 164 nodes and 150 nodes each. With an incremental evolution of topology and weight, evolutionary algorithm is better at generating small and redundancy free network structure. It is also able to deliver diversified solution topologies of the same task.

Hyper-NEAT Wins at Solving Certain Map Patterns

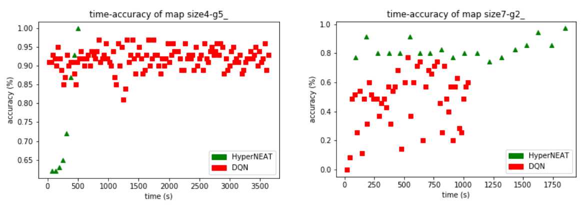
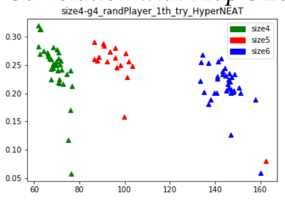


Fig 16-17. Hyper-NEAT Outperforms DQN

For small map of pattern #5 and the large map of pattern #2, Hyper-NEAT's performance in terms of accuracy and/or time is highlighted by DQN's incompetence. DQN is restrained by the extra pit location placed on the side of the first pit. Since DQN training is carried out by exploring as much as possible the locations on map, this extra pit greatly limited its exploration speed and even stopped it from reaching the only location that returns positive rewards, hence undermined the whole training accuracy. However, the emergence of solution for Hyper-NEAT is in backward order – it evolves the solution first and then test the solution to get a fitness value. This part of its strength is highlighted in map of pattern #5 and #2. For large map of pattern #2, it is composed of long stripe pattern of walls, which takes at most 55 steps for DQN to reach the grid of positive reward on map size 10x10. This means that DQN must make 55 right decisions before finding the right exit location, which is of 7.7e-34 possibility. Again, since Hyper-NEAT is not restrained by object arrangement on map, it is able to evolve a 100% workable solution eventually.

DQN Training Time Exibits No Positive Correlation with Map Sizes



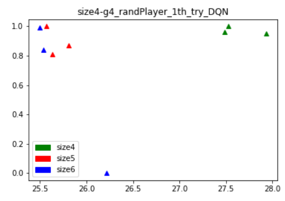


Fig 18. Hyper-NEAT *Fig 19. DQN*

As illustrated in the last section, DQN trains model along the way of its exploration, it only makes sense if map of larger size takes longer to find a solution, however results show that there is no such correlation exists between them, if anything, a negative correlation is observed. This may seem counter-intuitive at the first sight, but looking deeper, it makes perfect sense. Larger map allows more action available to take, meaning less possibility to fall in the pit and terminate the game. To support this point, figure 19 shows that DQN spends the least time to reach 100% accuracy on map of size 6x6, and the longest time on map of size 4x4. For Hyper-NEAT evolving time grows as map enlarges. This results further prove the hypothesis that DQN performance is more related to map pattern than map size.

4 Conclusion

One of the EA this project is concerned with, NEAT, is unable to perform a reinforcement learning task in a stochastic game environment. Its performance is similar to a random play. Hyper-NEAT is equipped with geometric awareness, evolves the function between objects and their locations, can successfully solve grid world game task, however with less ideal efficiency and stability. DQN presents extraordinary efficiency in most maps, but fails if the exploration space is embedded with too many traps that terminate the game, this may cause the agent to get stuck at a premature state and never be able to move toward the state of positive reward. The network topology returned by DQN is way more complicated than that of Hyper-NEAT. In summary, DQN is good at solving task of less tricky exploration space shortly and stably. However, where DQN fails, Hyper-NEAT can fill in the space. Hyper-NEAT ignores how far off the goal state is from where it currently is, hence can render promising performance and generate smaller and diversified network structure.

References

[1] M. Taylor, S. Whiteson and P. Stone, "Comparing evolutionary and temporal difference methods in a reinforcement learning domain", in *the 8th annual conference on Genetic and evolutionary computation*, Seattle, Washington, USA, 2006, pp. 1321-1328.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529-533, 2015

[3] M. Hausknecht, J. Lehman, R. Miikkulainen and P. Stone, "A Neuroevolution Approach to General Atari Game Playing", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355-366, 2014.

[4] M. Hausknecht, J. Lehman, R. Miikkulainen and P. Stone, "A Neuroevolution Approach to General Atari Game Playing", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355-366, 2014.

[5] J. Gauci and K. Stanley, "Autonomous Evolution of Topographic Regularities in Artificial Neural Networks", *Neural Computation*, vol. 22, no. 7, pp. 1860-1898, 2010.

[6] K. Stanley and R. Miikkulainen, "Efficient Evolution of Neural Network Topologies", in *the 4th Annual Conference on Genetic and Evolutionary Computation*, New York City, 2002, pp. 569-577.

[7] R. Sutton and A. Barto, *Reinforcement learning: An Introduction*, 1st ed. Cambridge, Mass. [u.a.]: MIT Press, 1998.

[8] T. Matiisen, "Demystifying Deep Reinforcement Learning", *nervana*, 2015. [Online]. Available: http://demystifying deep reinforcement learning. [Accessed: 13- Mar- 2017].

Biography

Chunzi Lyu, currently doing IT master at the University of Queensland. Has a passion for Machine Learning and is self-teaching Reinforcement Learning and Evolutionary Algorithms.