

Combining Learning and Evolution in OpenNERO

Jacob Robertson and Yun Wu

Department of Computer Science
University of Texas at Austin

Abstract

AAAI

Experiments and Results

In this paper, we evaluate our rtNEAT+Q algorithm in OpenNERO¹. OpenNERO is an open source software platform designed for research and education in Artificial Intelligence. The project is based on the Neuro-Evolving Robotic Operatives (NERO) game developed by graduate and undergraduate students at the Neural Networks Research Group and Department of Computer Science at the University of Texas at Austin. In the NERO game, The learning agents in NERO are simulated robots, and the goal is to train a team of these agents for combat. The agents begin the game with no skills and only the ability to learn. We tested the algorithm in different tasks and compare its performance to that of rtNEAT and Q-learning.

Approaching a flag

The first task is to train the agents running towards a flag. The distance between the spawn spot of agents and the flag is 141.4. Figure 1 illustrates the result. The green line is the average distance among all the agents and the blue line shows the minimum distance. From Figure 1a we can find that the result of Q learning in continuous domain is quite unstable. Also, rtNEAT, on the other hand, performs much better. Agents get to the flag pretty quick (around 25 ticks). The average distance converges at around 200 ticks when almost all the agents has learnt to approach a flag. The combination of rtNEAT and Q learning has similar performance with rtNEAT. It converges quickly but the average distance is longer than rtNEAT alone.

Approaching a moving flag

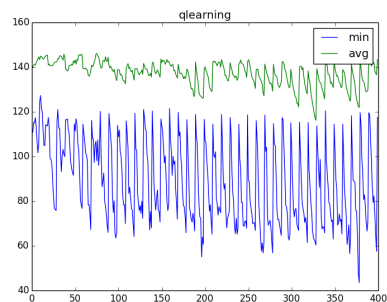
Approaching a flag with obstacles

However, when it comes to more complex tasks, the benefit of learning appears. In the third experiments, the distance between agents and the flag is 267. And there is a wall with width 100 between them. Agents need to learn to get around the wall to approach the flag. The result is in Figure 2. Within 700 ticks, Q learning agents never get a chance

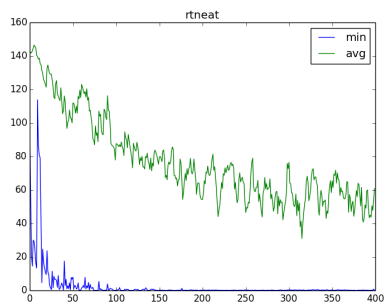
to get close to the flag. From Figure 2b and Figure 2c, it is clear that at first, agents learn to go directly towards flag until they are blocked by the wall. Then exploration then helps find the way around. Q learning then back propagate their network and learn the route. Finally, although rtNEAT+Q converges more slowly than rtNEAT, agents runs toward the flag more fast. The average distance of rtNEAT converges at 180 while that of rtNEAT+Q at 150.

Here is the reason why rtNEAT+Q benefits complex tasks. As shown in previous section, rtNEAT+Q take advantage of TD updates to tune parameters of neural network. However, Q learning needs time to learn. In simple task, structure of neural network is relatively simple. rtNEAT itself is able to evolve the optimal method quickly. However, for complicated tasks, it takes more time for evolution. In the mean time, Q learning is able to back propagate the network based on TD updates. Larmerkian system is used in our implementation, where the result of learning is kept to the next generation.

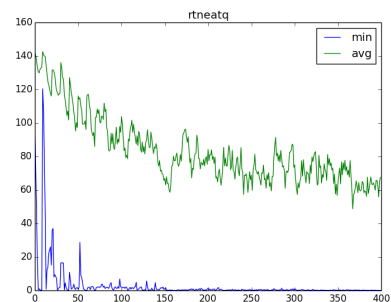
¹<https://code.google.com/p/opennero/>



(a) Q Learning

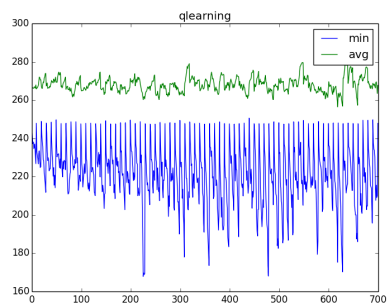


(b) rtNEAT

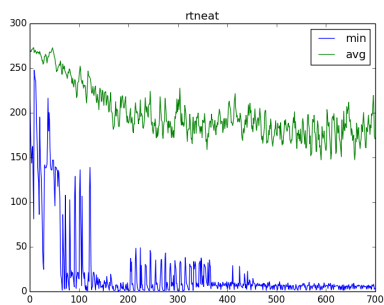


(c) rtNEAT + Q Learning

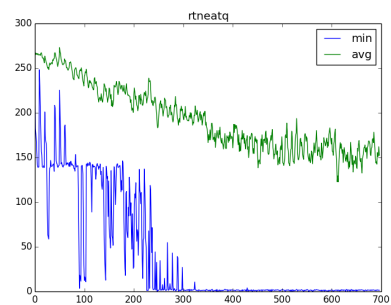
Figure 1: Approaching a flag



(a) Q Learning



(b) rtNEAT



(c) rtNEAT + Q Learning

Figure 2: Approaching a flag with obstacles