Combining Learning and Evolution in OpenNERO

Jacob Robertson and **Yun Wu**
Department of Computer Science
University of Texas at Austin

Abstract

AAAI

1 Introduction

Evolutionary algorithms have presented themselves as an interesting and useful method for the training of neural networks. Typically speaking the evolved genomes in some way represent the final behavior of agents being evaluated for fitness. Should we choose to take the evolutionary analog a step further, we may consider agents that learn during their "lifetime" as part of an evolutionary process. In computational models, non-biological models such as Lamarckian evolution, in which the expressed behavior of an agent is written back to its genome before reproduction.

There are multiple methods for combining learning and evolution in this manner, including supervised learning using labels for some task assumed to be positively related to evolutionary goals (Nolfi, Parisi, and Elman 1994), or using samples from human interaction to learn visibly intelligent behavior (Bryant and Miikkulainen 2007). Evolution can be combined with reinforcement learning by evolving networks for use as Q-value approximators (Whiteson and Stone 2006a). Additionally, social learning schemes may be applied in multi-agent scenarios (Tansey, Feasley, and Miikkulainen 2012)

OpenNERO[1] 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 are trained by a human player who is able to set positive or negative reward values for certain behaviors, such as approaching a flag, staying together as a group, firing upon enemy agents, and avoiding fire from enemy agents. This setting is continuous in its state space (an array of continuous-valued sensors), its action space (an array of actuators, in this case speed and rotation), and time. It can be used for multi-agent algorithms, including both cooperation within a team and competition between teams. Furthermore, it supports teams of agents trained through evolution through rtNEAT, an extension of the NeuroEvolution of Augmenting Topologies algorithm in which agents has individualized lifetimes as opposed to there being a discrete set of generations, as well as Q-learning for reinforcement learning (Stanley et al. 2006).

There appears to be some benefit to learning during evolution, though the results vary by domain and the mechanisms for benefit are not clearly understood. Furthermore, existing research into methods of combining learning and evolutions tend to examine relatively simple domains. For instance, NEAT+Q (Whiteson and Stone 2006a) is only suitable for discrete action spaces.

This paper examines NEAT+Q, adapts it for use in continuous action spaces, and evaluates its performance in the OpenNERO environment. We show that the introduction of learning can be advantageous over simple neuro-evolution, and is particularly well-suited for more difficult tasks in which local optima may be a problem.

2 Background

Let us review NEAT+Q, the basis of our implementation, as well as existing work in adapating Q-learning for continuous action spaces.

NEAT+Q

Temporal difference methods are well established both theoretically and empirically within reinforcement learning. In particular, Q-learning has enjoyed a significant amount of success, particularly when used with neural network function approximators. NEAT+Q takes advantage of NEAT to select the networks used for function approximation. NEAT selects network topologies and weights that are well suited to learn via backpropagation towards Q-value learning rules. The effectiveness of network Q-value approximators is senstive to the topology of the underlying network, which in this case is selected by NEAT to be useful without the need to be tuned by an experimenter.

The NEAT+Q algorithm may be viewed from this Q-learning perspective, or from an evolutionary perspective, in which the algorithm acts as a modification of the action selection mechanism used by evolved agents. Each agent in

[1]https://code.google.com/p/opennero/

order to select an action, activates its current sensors through its network. It then selects a random action with some probability ϵ, or the action corresponding to the output with the highest activation value otherwise. It is then able to use received reward, as well as the previously seen sensors and selected action to produce a new observed Q-value for the previously taken action. This creates an error signal on the output corresponding to that action that may be backpropagated. Note that there is an implicit assumption that each output from the network corresponds to an action from a discrete set of possible actions, and that when the network represents a Q-value approximator, selecting an action with the highest activation is equivalent to selecting an action with the highest Q-value.

Q-learning with continuous actions

Gaskett et al. describe a system for Q-learning in continuous action spaces with neural network function approximators. Their method makes uses of least-squares interpolator on top of the underlying network. Instead of the output of the network being a single output per action, the output of the network is a collection of "wires". Each wire represents a group of outputs: a continuous-valued action vector as well as a Q-value for that action vector. The set of wires produced by activating the network provide a set of samples for the wire-fitter to interpolate the Q-value function of the action space for a given state. Action selection may occur by choosing the action vector associated with the highest Q-value. Afterwards, the wire-fitter provides a new Q-value function based on observed rewards, and the output of the network can be backpropagated towards that new function. The new targets to be trained towards are themselves determined by gradient descent, incurring additional cost per iteration of backpropagation.

3 Approach

Our implementation makes use of ideas from wire-fitted Q-learning to adapt NEAT+Q to the continuous action space used by rtNEAT in OpenNERO. For brevity we will refer to our implementation as rtNEAT+Q. It is worth noting with wire-fitting that the new observation being used to update the Q-value function will always be just a new Q-value for a previously output action vector. For that action vector alone, it is simpler to just train it towards a newly observed Q-value than to use the full approach described by the wire-fitting algorithm. This is possible because it is unnecessary to compute a new target action vector to train towards. Additionally, we would like to avoid incurring the cost of a complete gradient descrent process at each action selection of our implementation.

The OpenNERO agent network's output is a continuous-valued action vector. NEAT+Q assumes a discrete action space in which each network output represents a single action. We instead use the format of the ouputs used by the wire-fitting algorithm, in which the network outputs a number of candidate action vectors and their associated Q-values. Actions are selected from these outputs in an epsilon-greedy fashion, and error is only propagated on the Q-value corresponding to the previously selected candidate. In this way we take advantage of the fact the we will always obtain a new Q-value for the previously selected action vector, but we do not apply any learning to the other candidates regardless of their similarity to the previously selected action vector. This avoids the cost of gradient descent to determine the full set of learning targets used in wire-fitting. We rely on NEAT to optimize the set of candidate actions over the course of the evolutionary process. From this approach we were able to extend the existing rtNEAT agents in OpenNERO to learn during their lifetimes and evolve in a Lamarckian fashion.

4 Experiments and Results

In this paper, we evaluate the rtNEAT+Q algorithm in OpenNERO. As shown in Figure 1, in our experiments we focus on the task of approaching a flag in different scenarios.

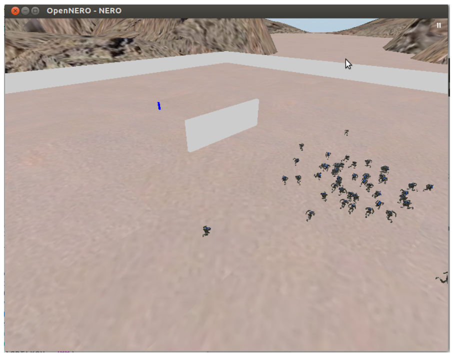


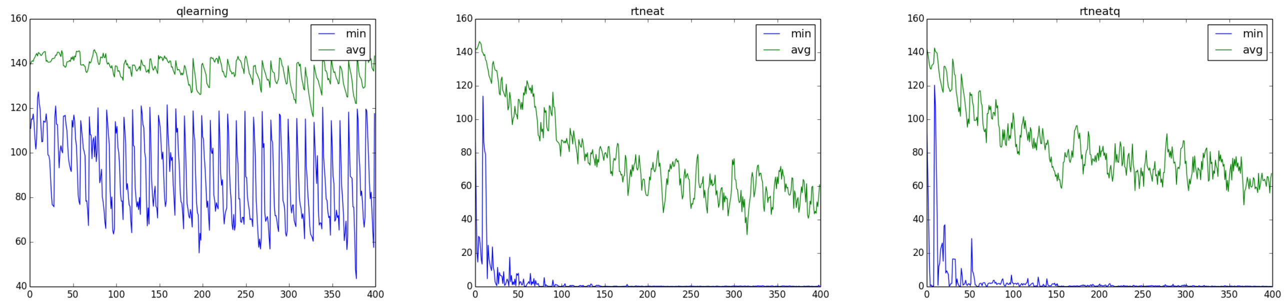
Figure 1: OpenNERO scenario

Approaching a flag

The first task is to run towards a flag directly. The distance between the spawn spot of agents and the flag is 141.4. Figure 2 illustrates the result. The green line is the average distance among all the agents and the blue line shows the change of the minimum distance. From Figure 2a we can find that the result of Q learning in continuous domain is quite unstable. rtNEAT, however, performs much better. The most fast agent gets to the flag quickly (around 25 ticks). The average distance converges at around 200 ticks when most agents has learnt to approach a flag. The combination of rtNEAT and Q learning has similar performance with rtNEAT. The average distance converges quickly but is longer than the rtNEAT agents.

Approaching a moving flag

We also experimented on approaching a moving flag with different algorithms(Figure 3). Here, the location of the flag is updated randomly every 50 ticks. The minimum distance is 141 and the maximum is 283. For all three algorithm, the process of location update is the same. rtNEAT agents are able to run towards the flag within the first 50 ticks. And after every update, both minimum distance and average distance drop dramatically. The combination of rtNEAT and Q learning is able to learn the behavior within first three updates.



(a) Q Learning (b) rtNEAT (c) rtNEAT + Q Learning

Figure 2: Approaching a flag

And the average distance after 50 ticks is also longer than rtNEAT alone, which is consistent with static flag. The performance of Q learning is still unsatisfactory. The average distance does not change much after long time of training.

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. As shown in Figure 4, Q learning agents never get a chance to get close to the flag within 700 ticks. The minimum distance between agents and flag is about 170. However, from Figure 4b and Figure 4c, 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 to 180 while that of rtNEAT+Q to 150.

5 Discussion and Future Work

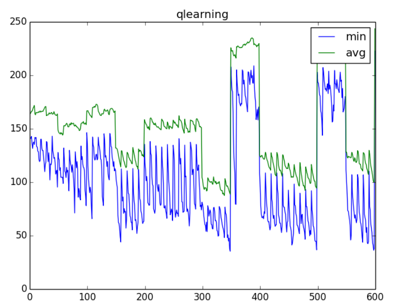
The results presented in Section 4, demonstrate the performance of combination of rtNEAT and Q learning. It shows that the performance is related to the complexity of the task. Intuitively, evolution can help develop ability to survive by coding into the genomes, like breathing and metabolism. Such mechanism is relatively simple. Also, it requires little calculation with changing environment. Therefore, such ability can be developed by evolution alone. On the other hand, scientific knowledge or etiquette are examples of complicated tasks for humans. A proper reaction to different situations in social life is intricate and requires large computation and memory search. In those cases, learning within a lifetime of a generation is more effective than relying on evolution itself. Similarly, in our experiments, approaching a flag is relatively simple. Thus evolution itself can build up an accurate neural network. However, it is not straightforward how to get to a flag with an obstacle in the way. Agents need to learn to go away from the flag to get around the wall. There are different paths and learning could also help us find the shortest one.

Here is the reason why rtNEAT+Q benefits complex tasks. rtNEAT+Q takes 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. Lamarckian system is used in our implementation, where the result of learning is kept to the next generation.

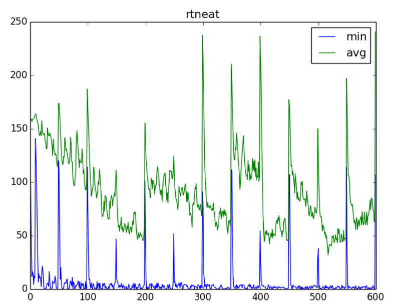
According to our result, rtNEAT+Q can successfully train neural network function approximators. However, rtNEAT+Q requires many more episodes to find good solutions than rtNEAT do in the same domain. Since each candidate network must be trained long enough to let Q-learning work, it has very high sample complexity. Moreover, for each episode, rtNEAT+Q also takes around 40% more time than rtNEAT. The extra time is used to back propagate neural networks according to TD updates. However, such process is independent of the policy the agent is following, one network can be updated while another is controlling the agent. Furthermore, a network can be updated based on data saved from previous sample episodes, regardless of what policy was used during those episodes. Consequently, it is not necessary to use different episodes to train each network. As shown in (Whiteson and Stone 2006b), we could save data from the episodes used by the previous generation, each network in the population can be pre-trained off-line

Also, Figure 2- 4 show that Q learning does not perform well in our experiments. It is always a challenge for reinforcement learning in continuous domain. The implementation there is simply discretization the orientation to lots of intervals. It will cause a huge search space. Thus it is difficult to find the optimal solution by TD update. Also, on OpenNERO platform, the lifetime of each agent is related to the fitness to current environment. If the speed of learning is too low, they will be reset and consequently it is difficult for Q learning agents to get to the flag.

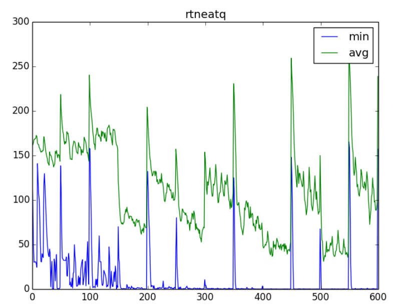
Another problem with Q-learning is the difficulty of exploration. Here, we use ϵ-greedy selection. Each time the



(a) Q Learning

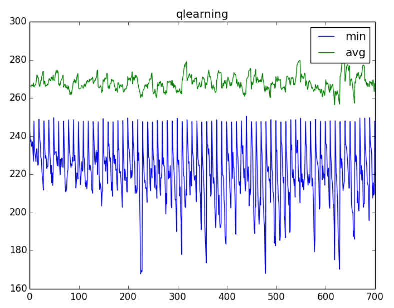


(b) rtNEAT

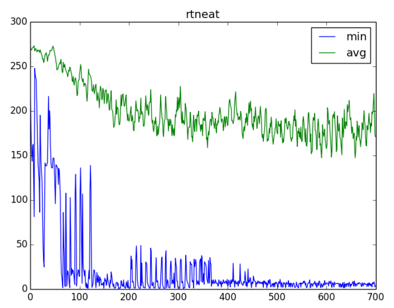


(c) rtNEAT + Q Learning

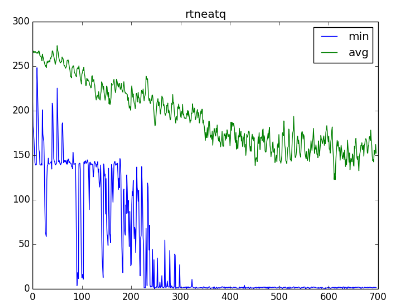
Figure 3: Approaching a moving flag



(a) Q Learning



(b) rtNEAT



(c) rtNEAT + Q Learning

Figure 4: Approaching a flag with obstacles

agents selects an action, it choose probabilistically between exploration and exploitation. With probability ϵ, it will explore by selecting randomly from the available actions. With probability 1-ϵ, it will exploit by selecting the greedy action. Such mechanism has been proven to provide a balance between exploration and exploitation in discrete domains. However, things are different in continuous domain like OpenNERO. If you pick a random action some small percentage of the time that will just be some little fluctuation in behavior since it will usually be immediately followed by an optimal action. It's like exploration consists of walking just one step off the beaten path instead of taking another path.

.

6 Conclusion

Reinforcement learning is an appealing and empirically successful approach to finding effective control policies in large probabilistic domains. There are three ways to solve the problem: evolution, TD update and their combination. Their performances depend on the properties of the task. According to our research and experiments, TD update is more suitable in discrete domains with limited states and actions. Evolution outperforms it in continuous domains and converges quickly in simple tasks. The combination of evolution and TD update can solve complex problems with better result than learning or evolution alone. It takes more time but it also evolves a function approximator during learning. In this project, rtNEAT+Q is implemented for combination of reinforcement learning in continuous domain and realtime evolution. It is tested in OpenNERO to train the agents to approach a flag. The experiments provide insight about the advantages of models related to task complexity.

Acknowledgements

Thanks to Professor Risto Miikkulainen for his kind discussion with us about the project. Thanks to Kim Houck for her help with understanding OpenNERO platform.

References

Bryant, B. D., and Miikkulainen, R. 2007. Acquiring visibly intelligent behavior with example-guided neuroevolution. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 801. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Nolfi, S.; Parisi, D.; and Elman, J. L. 1994. Learning and evolution in neural networks. *Adaptive Behavior* 3(1):5–28.

Stanley, K. O.; Bryant, B. D.; Karpov, I.; and Miikkulainen, R. 2006. Real-time evolution of neural networks in the nero video game. In *AAAI*, volume 6, 1671–1674.

Tansey, W.; Feasley, E.; and Miikkulainen, R. 2012. Accelerating evolution via egalitarian social learning. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, 919–926. ACM.

Whiteson, S., and Stone, P. 2006a. Evolutionary function approximation for reinforcement learning. *The Journal of Machine Learning Research* 7:877–917.

Whiteson, S., and Stone, P. 2006b. Sample-efficient evolutionary function approximation for reinforcement learning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 21, 518. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.