"Policy Iteration" through evolutionary of Augmenting Topology

Yunian Pan
Department of Electrical Engineering
yp1170@nyu.edu

Abstract

This report presents the project which follows the instructions of the paper "Evolutionary Function Approximation for Reinforcement Learning" [1] to do some simulations of reinforcement task using the related algorithm.

In the context that we have been using deep learning that in most real-world reinforcement learning tasks, TD methods require a function approximator to represent the value function. However, using function approximators requires manually making crucial representational decisions, our goal is to develop the algorithm to automate the search for best function representations.

1 Introduction

TD methods have been conceived as well-established algorithms that can handle reinforcement learning tasks, when the problem scale is not large, the value function can be represented as a table of state-action pairs, but for large scale systems or for problems that has infinite dimension of state space or action space, it remains problematic to only use a "table", instead the function approximator which represents the mapping from state-action pairs to values via a more concise, parameterized function and uses supervised learning methods to set its parameters, plays an important role where we can omit the step of enumering over the state-action space.

However, using function approximators requires making crucial representational decisions (e.g. the number of hidden units and initial weights of a neural network. Poor design choices can result in estimates that diverge from the optimal value function and agents that perform poorly. Even for reinforcement learning algorithms with guaranteed convergence, achieving high performance in practice requires finding an appropriate representation for the function approximator.

Section 2 will present the background including Q-learning and deep Q-learning, genetic algorithms and NEAT framework and NEAT algorithm for reinforcement learning and some basic related experiments; Section 3 will have some discussion over the results obtained from the experiments.

2 Background

This section mainly contains two parts, one is for the development of Q-learning, Deep Q learning, and some variance of them; The another will discuss the NEAT machinery, framework and using NEAT to do reinforcement learning, specifically, we will start from the general framework of genetic algorithm and with a little bit discussion on how it works, then introduce a gene encoding scheme that NEAT relies on.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2.1 Q-learning

In the tabular case, the algorithm is defined by the following update rule, applied each time the agent transitions from state s to state s':

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \min_{a'} Q(s',a')) \tag{1}$$

where $\alpha \in [0,1]$ is a learning rate parameter, $\gamma \in [0,1]$ is a discount factor, and r is the immediate reward the agent receives upon taking action a.

This equation was initially derived from Bellman equation regarding Markov Decision Process, recall the original operator of value function:

$$(TJ)(i) := \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)[g(i,u,j) + \alpha J(j)] \tag{2}$$

define: $Q(s,a) = \sum_{j=1}^{n} p_{ij}(u)[g(i,u,j) + \alpha J(j)]$, we can omit the probability transition, and through Robins-Monro algorithm we solve the Q-learning problem and end up with the update rule 1.

2.2 Q-Learning with NN

Particularly when combined with neural network function approximators, algorithm 1 describes the Q-learning algorithm using a neural network to approximate the value function. The parameter λ the sample roll-out cost decay, which is for the convenience of doing $TD(\lambda)$ extension, which we usually set to 0 or 1, respectively representing the value iteration method and the policy evaluation method.

Algorithm 1: Deep Q-Learning

Input: S : set of all states; A : set of all actions; α : learning rate; γ : discount factor; λ : eligibility decay rate; ϵ_{td} : exploration rate; e : total number of episodes;
Initialize: $N \leftarrow$ INIT-NET(S,A) ;
for $i \leftarrow 1$ *to* e **do**
  $s, s' \leftarrow$ null, INIT-STATE(S);
  while *Terminal-state?(s)* **do**
    $Q[] \leftarrow$EVAL-NET(N, s');
    With-prob(ϵ_{td}) $a' \leftarrow$ RANDOM(A);
    else: $a' \leftarrow \arg\max Q[j]$;
    if $s \neq null$ **then**
      BACKPROP$(N, s, a, (r + \gamma \max_j Q[j])/\alpha, \gamma, \lambda)$;
    else
      $s, a \leftarrow s', a'$;
      $r, s' \leftarrow$ TAKE-ACTION(a')
    end
  end
end

2.3 NEAT

2.3.1 genetic algorithm

Genetic algorithm(GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms(EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.[3]

The flow chart 1 describes the standard evolution scheme: Inspired by the framework, one might think about apply it to finding the best structure of a neural network.

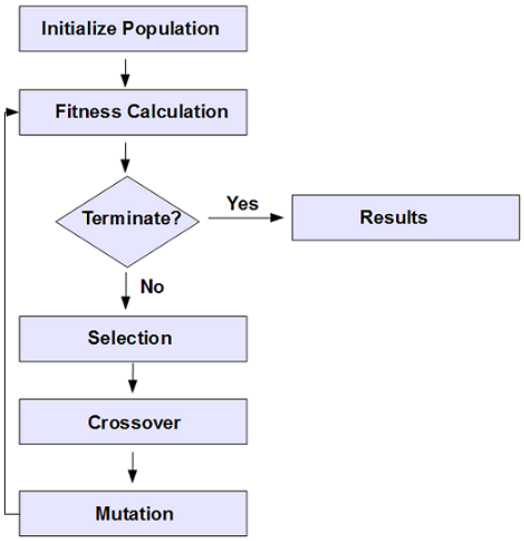

Figure 1: from Google Image

2.3.2 Genetic encoding

In order to develop the neural evolution algorithm, we need a standard method to encode the neural networks as genotypes so that we can do crossover and mutation operation based on the genotypes provided. NEAT encodes the neural networks in following aspects:

Genomes in NEAT are linear representations of network connectivity. The figure 2 illustrates a genotype to phenotype mapping example.

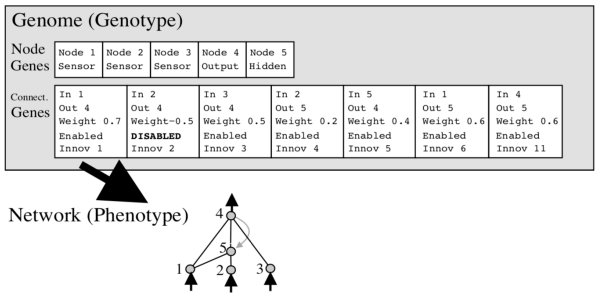

Figure 2: genotype and phenotype from [2]

The figure 3 illustrates Matching up genomes for different network topologies using innovation numbers.

Figure 4 illustrates the two types of structural mutation in NEAT.

2.3.3 NEAT algorithm

Without the aid of TD methods, namely Q-learning, NEAT can tacle reinforcement learning in that NEAT does not attempt to learn a value function but tries to find good policies directly by training action selectors, which map states to the action the agent should take in that state, as the same as we do in policy evaluation reinforcement learning, which uses the global optimizing techniques that directly search the space of potential policies.

Algorithm 2 contains a high-level description of the NEAT algorithm applied to an episodic reinforcement learning problem. It differs from the original algorithm in that for every step of choosing

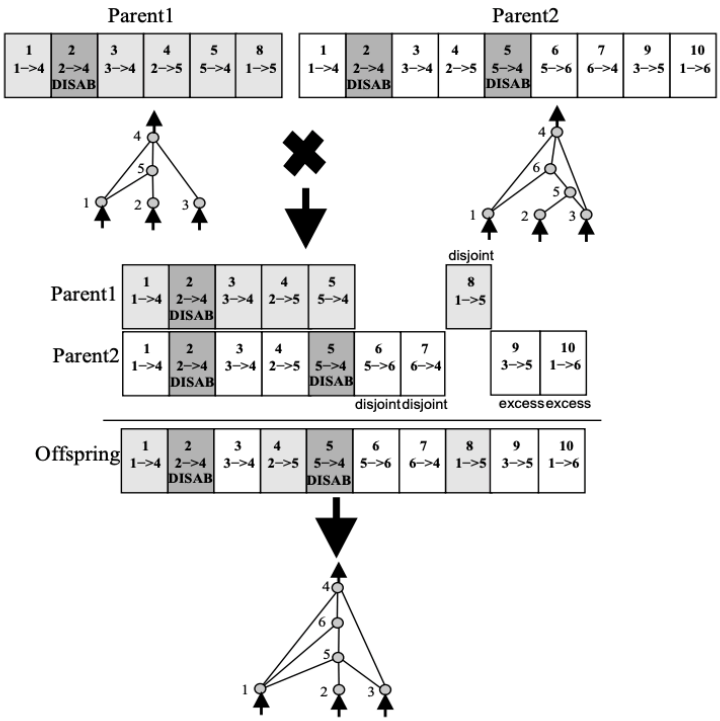

Figure 3: crossover from [2]

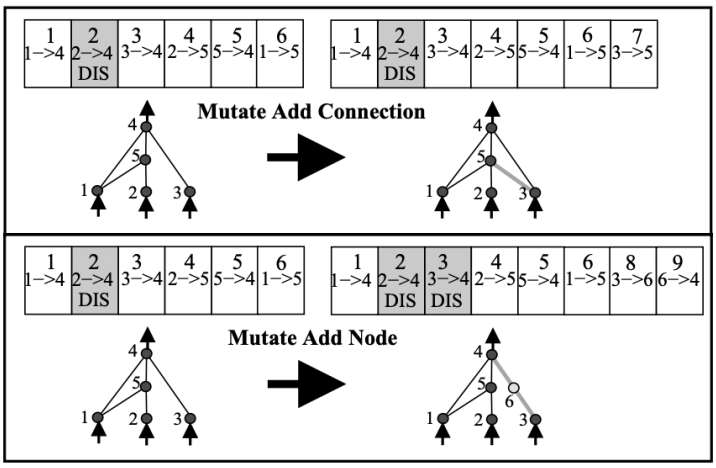

Figure 4: mutation from [2]

network for policy evaluation, the network is not chosen from directly the fixed order of population vector, but randomly selected. This change does not significantly alter NEATs behavior but facilitates to excel online by balancing exploration and exploitation within and across generations.

Algorithm 2: NEAT(S, A, p, m_n, m_l, g, e)

Input: S:set of all states, A:set of all actions, p:population size , m_n: rate of adding node, m_l: rate of adding link, g: generations, e: episodes;
Initialize: $P[] \leftarrow$ Init-Populations(S, A, p) ;
for $i \leftarrow 1$ *to* g **do**
  for $j \leftarrow 1$ *to* e **do**
    $N, s, s' \leftarrow$ Random(P), null, INIT-STATE(S);
    while *Terminal-state?(s)* **do**
      $Q[] \leftarrow$EVAL-NET(N, s');
      $a' \leftarrow \arg\max Q[j]$; $s, a \leftarrow s', a'$; $r, s' \leftarrow$ TAKE-ACTION(a');
      $N.fitness \leftarrow N.fitness + r$
    end
    $N.episodes \leftarrow N.episodes + 1$
  end
  $P' \leftarrow$ new array of size p;
  for $j \leftarrow 1$ *to* p **do**
    $P'[] \leftarrow$ Breed-Net($P[]$);
    with-probability m_n: ADD-Node-Mutation($P'[j]$);
    with-probability m_l: ADD-link-Mutation($P'[j]$)
  end
  $P[] \leftarrow P'[]$
end

NEAT speciates the population, in order to avoid that adding new structure to a network might initially reduces its fitness. so that individuals compete primarily within their own niches rather than with the population at large.

2.4 Comparisons

Several comparisons regarding the comparisons between different version of algorithms are listed in this section:

3 Simulations and comparison

There are some of the results demonstrated with an emphasis on NEAT algorithm.

3.1 Mountain car

Starting from the simplest case, the first environment we tested is the Mountain car, just as the paper has presented, As 5 illustrates, the car which is stuck in the valley doesn't have enough power to

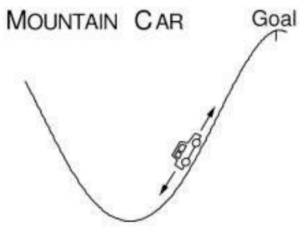

Figure 5: Mountain car from Sutton(1998)

drive to the goal, instead it has to drive to the left hill in order to build some inertia and reach the goal leveraging the speed it obtains.

3.1.1 Q-learning

As a counterpart, when manually determine the neural network using tensorflow with 811 total parameters, the mountain car task remains problematic after 1000 episodes of training, the episode reward keeps at -200 in the current used OpenAI gym environment, to achieve better results the exploration rate annealing was implemented. the test results will be covered in the video, the learning curve will not be provided here since it's a straight horizontal line.

3.1.2 NEAT

Table 1 shows some basic configuration parameters that was used for NEAT(still some others not listed).

Table 1: parameter setting

state	action	node bias	compa-threshold	stagnation	sp-elitism
2	discrete(3)	[-30,30]	3.0	15	4
pop size	mutation rate	Node-add/delete	Connect-add/delete	activation	elitism
150	0.1	0.9/0.2	0.4/0.1	relu	2

As 6 shows, we are able to find the best individuals from the population within the 60 generations, at the very beginning the performance is poor, but once there are some "good" mutations happening, the next generation will improve by inheriting these nice topologies. The final winner's structure is

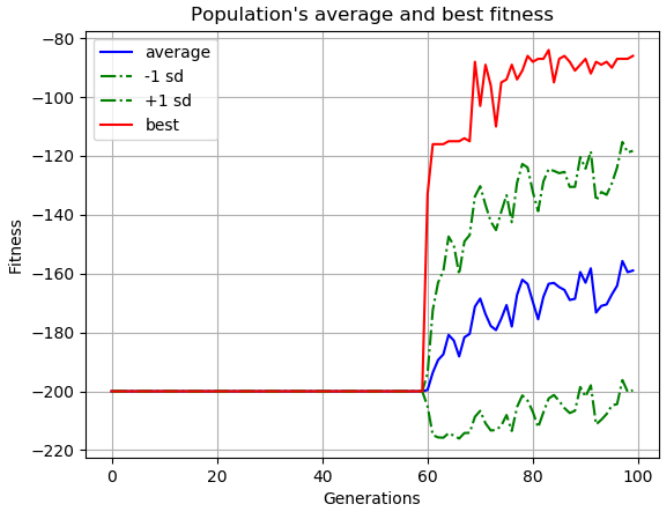

Figure 6: Learning curve of mountain car

as shown in 7,

8 demonstrates the speciation evolution, genomes that with genome distance less than compatibility threshold 3 would be considered one species.

3.2 Atari game: AirRaid

An environment with larger complexity called AirRaid has also been tested using the both Q-learning and NEAT algorithm.

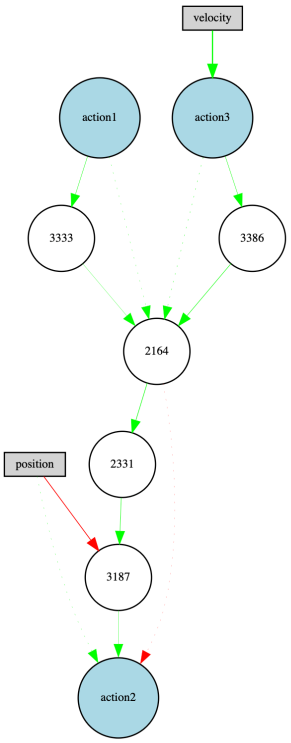

Figure 7: Final conectivity of mountain car, the network is allowed to become recurrent as we add nodes and connections

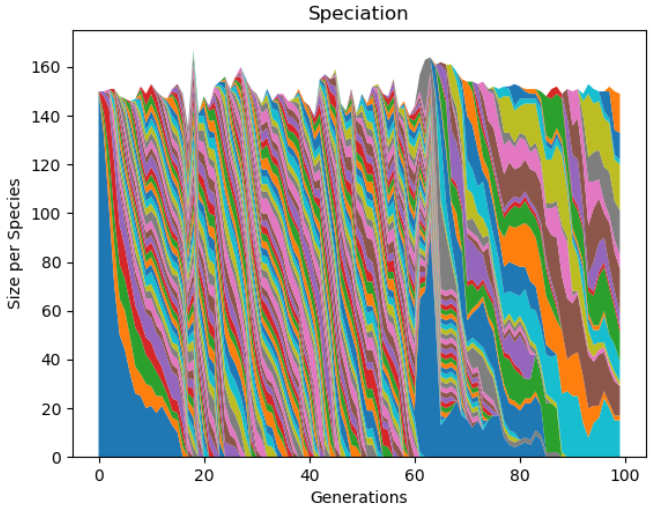

Figure 8: Species evolution

3.2.1 Q-learning

Manually determined neural network has 297,478 parameters, during the exploration steps, the model might have achieved greater score higher than 1000 but overall the learning curve has been stuck in somewhere. Using the same algorithm as before in the mountain car task, still it has a bad performance.

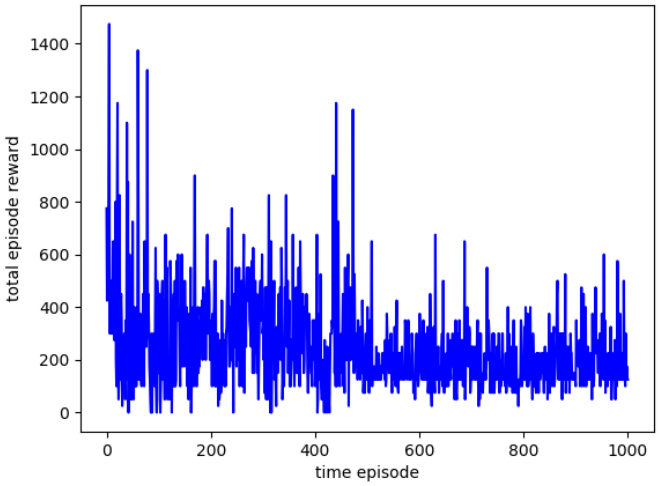

Figure 9: Learning curve of AirRaid Q-learning

3.2.2 NEAT

Table 2 shows the basic configuration of neat algorithm:

Table 2: parameter setting

state	action	node bias	compa-threshold	stagnation	sp-elitism
128	discrete(6)	[-30,30]	3.0	15	4
pop size	mutation rate	Node-add/delete	Connect-add/delete	activation	elitism
150	0.1	0.9/0.2	0.4/0.1	relu	2

After 100 generations of training, each generation contains variant episodes within which my action in the environment will be finished, learning curve is just shown in Figure 10.

Limited by the scale of graph, the connectivity will not shown here. As we continue breeding the networks, it will exceed the fitness threshold and stop training as we have defined in the configuration file, as shown in 12.

4 Conclusion

Having evidences listed before, we can summarize that :

- NEAT outperform Q-learning in episodes, Generally, NEAT+Q can even perform better as shown in the mountain car and server job scheduling domain of the original paper;
- NEAT explore the function representation automatically, similar to exploring a better policy, the episodes of ficticious play is equivalent to what we do during policy evaluation, only different in that the policy is hidden beneath the represented function of neural network, and we have multiple choice when it comes to policy update.

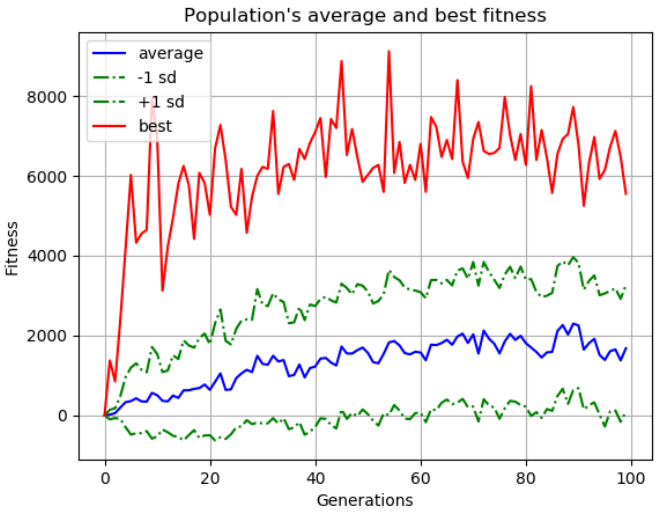

Figure 10: Learning curve of AirRaid

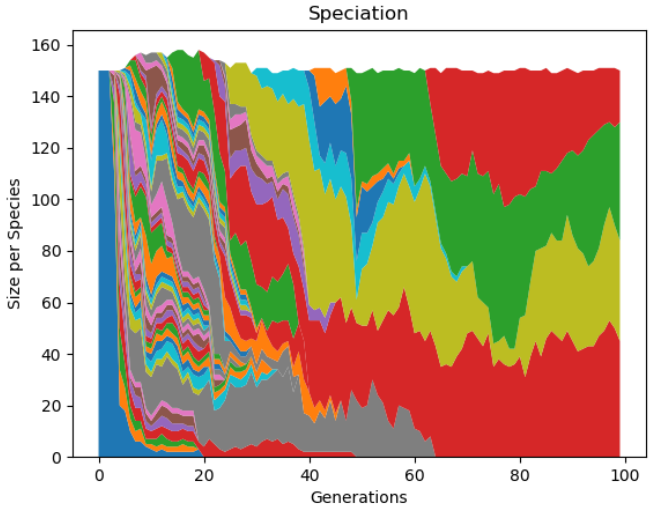

Figure 11: Evolution diagram of AirRaid

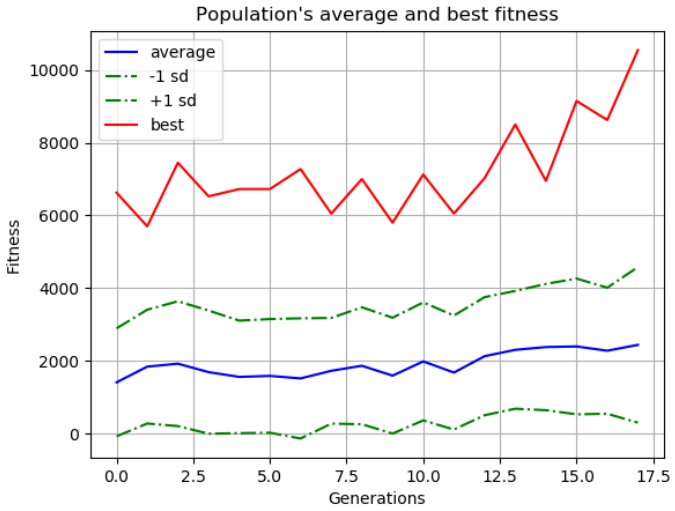

Figure 12: Learning curve of AirRaid

- It's safe to anticipate that some other Methods(DDQN and Duel QN) can be combined with NEAT, but in order to in corporate with those tools we have to add properties to the existing NEAT system so that it can perform backpropagation.
- although the training process might work in some deterministic systems, it is still challenging to tacle non-stationary environment as the action selector only take the current states into consideration, whether the algorithm is adaptive to the non-stationary environments or not remains to be verified.

References

[1] Whiteson, Shimon, and Peter Stone. "Evolutionary function approximation for reinforcement learning." Journal of Machine Learning Research 7.May (2006): 877-917.

[2] Stanley, Kenneth O., and Risto Miikkulainen. "Efficient reinforcement learning through evolving neural network topologies." Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc., 2002.

[3] Mitchell, Melanie. An introduction to genetic algorithms. MIT press, 1998.