Evolutionary Function Approximation for Reinforcement Learning

Yunian Pan

ECE department

May. 10th 2019

Evolutionary Function Approximation for Reinforcement Learning

Outline:

- Some Background
- Methods and Results
- Experiments
- Conclusion

In a word, DQN(Deep Q-Learning) + NEAT(Neuroevolution of augmenting topologies)

Algorithm 1: Q-Learning

Input: S : set of all states; A : set of all actions; σ : standard deviation of initial weights; c : output scale; α : learning rate; γ : discount factor; λ : eligibility decay rate; ϵ_{td} : exploration rate; e : total number of episodes;

Initialize: $N \leftarrow INIT - NET(S, A, \sigma)$;

for $i \leftarrow 1$ *to* e **do**

  $s, s' \leftarrow$ null, INIT-STATE(S);

  while *Terminal-state(s)* **do**

    $Q[] \leftarrow c \times EVAL - NET(N, s')$;

    With-prob(ϵ_{td}) $a' \leftarrow RANDOM(A)$;

    else: $a' \leftarrow \arg\max Q[j]$;

    if $s \neq null$ **then**

      $BACKPROP(N, s, a, (r + \gamma \max_j Q[j])/c, \alpha, \gamma, \lambda)$;

    else

      $s, a \leftarrow s', a'$;

      $r, s' \leftarrow TAKE - ACTION(a')$

    end

  end

end

Genetic Algorithm

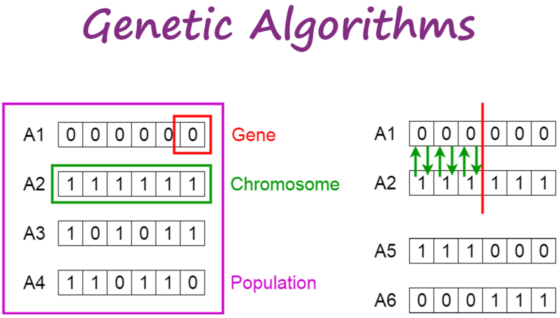


Figure 1: Illustration from Google image

Genetic Algorithm

Standard evolution framework:

(1) Initialize population

(2) Evolve from $1, \ldots, n_{generation}$:

  a) Select from population according to fitness
  b) Generate offspring through crossover and mutation
  c) Replace population with offspring.

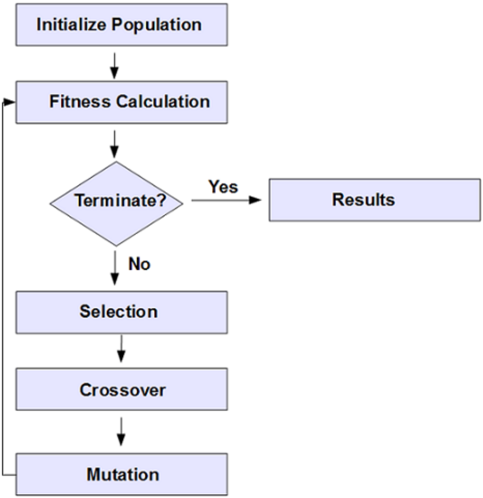


Figure 2: flow chart from Google image

Network encoding

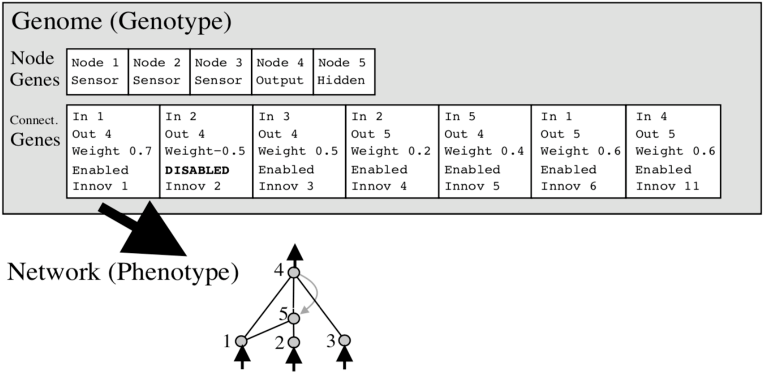


Figure 3: from Stanley, Kenneth O., and Risto Miikkulainen. (2002): 99-127.

Crossover

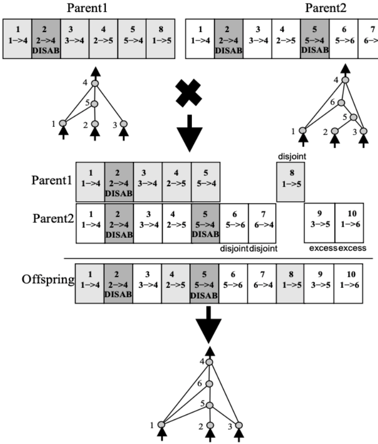


Figure 4: from Stanley, Kenneth O., and Risto Miikkulainen. (2002): 99-127.

mutation

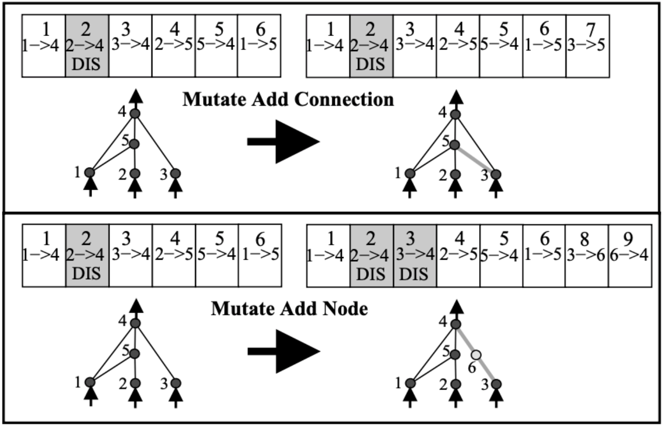


Figure 5: from Stanley, Kenneth O., and Risto Miikkulainen. (2002): 99-127.

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 ← 1 to g do
    for j ← 1 to e do
        N, s, s' ← Random(P), null, INIT-STATE(S);
        while Terminal-state?(s) do
            Q[] ←EVAL-NET(N, s');
            a' ← arg max Q[j];    s, a ← s', a';    r, s' ← TAKE-ACTION(a');
            N.fitness ← N.fitness + r
        end
        N.episodes ← N.episodes + 1
    end
    P' ← new array of size p;
    for j ← 1 to p do
        P'[] ← Breed-Net(P[]);
        with-probability m_n: ADD-Node-Mutation(P'[j]);
        with-probability m_l: ADD-link-Mutation(P'[j])
    end
    P[] ← P'[]
end
```

Algorithm 3: NEAT+Q(S, A, p, m_n, m_l, g, e)

Input: S, A, p, m_n, m_l, g, e, α, λ, γ, ϵ

Initialize: $P[] \leftarrow$ Init-Populations(S, A, p) ;

```
for i ← 1 to g do
    for j ← 1 to e do
        N, s, s′ ← Random(P), null, INIT-STATE(S);
        while Terminal-state?(s) do
            Q[] ←EVAL-NET(N, s′);
            With-prob(ε_td) a′ ← RANDOM(A);
            else: a′ ← arg max Q[j];
            if s ≠ null then
                BACKPROP(N, s, a, (r + γ max_j Q[j])/c, α, γ, λ)
            end
            s, a ← s′, a′;
            r, s′ ← TAKE-ACTION(a′);
            N.fitness ← N.fitness + r
        end
        N.episodes ← N.episodes + 1
    end
    Crossover and mutation
end
```

Boltzman Selection

Exploration probability:

$$\Pr(\cdot|s) = \frac{e^{Q(s,\cdot)/\tau}}{\sum_{a\in A} e^{Q(s,a)/\tau}}$$

Can we use it in network choosing? Yes.

$$\Pr(\cdot) = \frac{e^{S(\cdot)/\tau}}{\sum_{q\in P} e^{S(q)/\tau}}$$

Algorithm 4: Boltzman Selection(P, τ)

Input: P : population, τ: softmax temperature

if $\exists N \in P \mid N.episodes = 0$ **then**

  return N

else

  $total \leftarrow \sum_{N \in P} e^{N.average/\tau}$;

  for $N \in P$ **do**

    with-prob $\left(\frac{e^{N.average/\tau}}{total}\right)$ return N else

    $total \leftarrow total - e^{N.average/\tau}$

  end

end

Some Comparison

- Online v.s. Offline
- Darwinian v.s. Lamarckian
- Annealing v.s. Without Annealing

Experiments from the paper

- mountain car

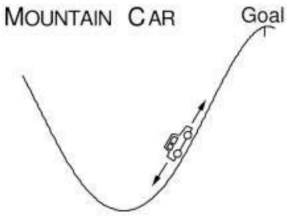


Figure 6: from Sutton and Barto (1998)

- Server job scheduling

Results

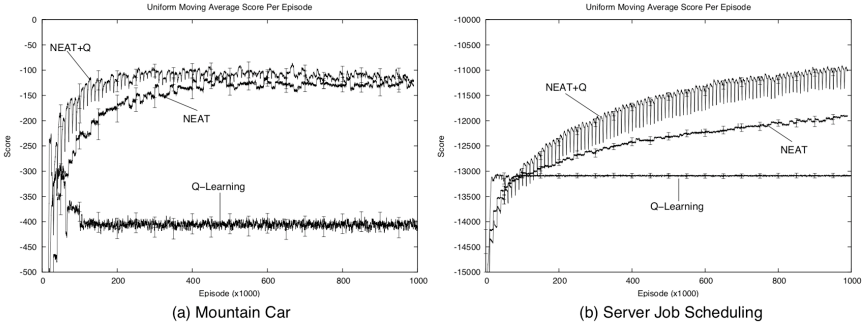
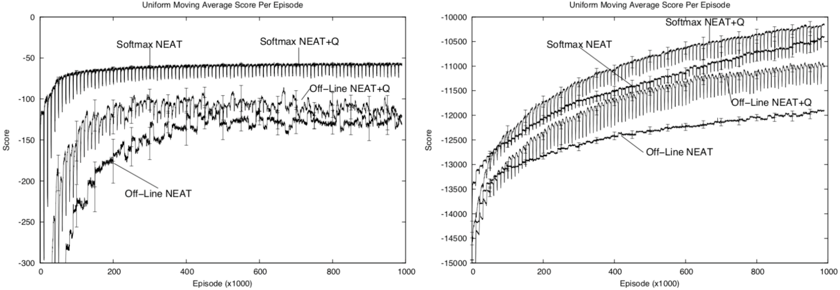


Figure 7: from Whiteson, Shimon, and Peter Stone. (2006): 877-917. Figure 6

Results



(a) Mountain Car (b) Server Job Scheduling

Figure 8: from Whiteson, Shimon, and Peter Stone. (2006): 877-917. Figure 7

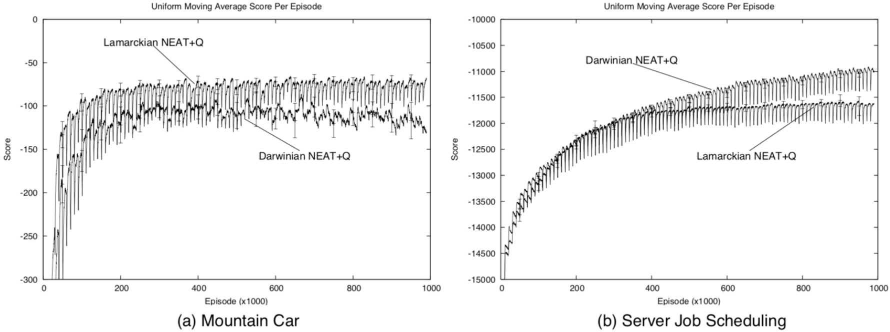
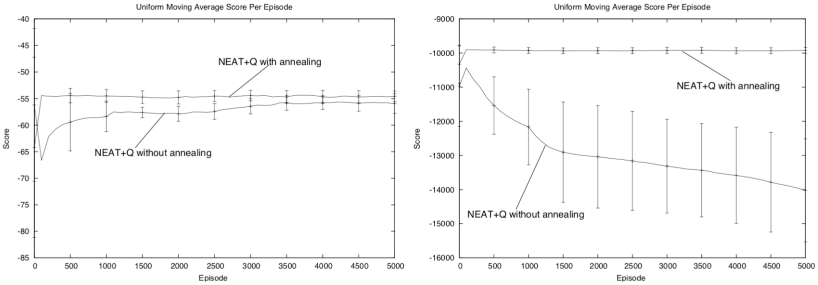


Figure 9: from Whiteson, Shimon, and Peter Stone. (2006): 877-917. Figure 10



(a) Mountain Car

(b) Server Job Scheduling

Figure 10: from Whiteson, Shimon, and Peter Stone. (2006): 877-917. Figure 11

Experiments

Let’s try it!

(a) Cartpole Balancing using NEAT and Q
(b) test on Atari game Pacman on both NEAT and Q

Conclusion

- NEAT outperform Q-learning in episodes, Generally, (NEAT+Q can perform better!)
- NEAT explore the function representation automatically.
- Some other Methods(DDQN and Duel QN, PG) can be combined with NEAT
- Non-stationary environment is challenging.

Thank You!