Learning Reinforcement Learning by Learning REINFORCE

TINGWU WANG,

MACHINE LEARNING GROUP,

UNIVERSITY OF TORONTO

Contents

A Sketch of REINFORCE Algorithm

1. Today's focus: Policy Gradient [1] and REINFORCE [2] algorithm.
   1. REINFORCE algorithm is an algorithm that is {

      discrete domain + continuous domain,

      policy-based,

      on-policy + off-policy,

      model-free,

      ~~shown up in last year's final~~

      }.

   No need to understand the colored part.

2. By the end of this course, you should be able to:
   1. Write down the algorithm box for REINFORCE algorithm.
   2. Calculate the objective function at each time step.
   3. Calculate the correct gradient for each parameter (small model).
   4. (Maybe) Have a rough idea of how solve a new RL problem.

Contents

Objective Function

1. Objective function for all policy-based algorithms
    1. In episodic environments we can use the start value:

    $$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

    2. In continuing environments we can use the average value:

    $$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

    3. Or the average reward per time-step

    $$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \mathcal{R}_s^a$$

    4. After all, training RL agents is just optimizing the objective function.
        1. All the optimization algorithms you learnt could be applied.
            1. Zero-order (gradient free)
            2. First-order (taking the gradient)
            3. Second-order (using the hessian …)

Policy Gradient

1. How do we optimize the objective function?
   1. Zero-order: Gradient-Free methods:
      1. Evolution algorithm [11]
      2. Grid-search (of course, and local-minima-proof if Lipschitz constraints met)

Algorithm 1 Evolution Strategies

```
Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for t = 0, 1, 2, ... do
    Sample ε_1, ... ε_n ~ N(0, I)
    Compute returns F_i = F(θ_t + σε_i) for i = 1, ..., n
    Set θ_{t+1} ← θ_t + α (1/(nσ)) Σ_{i=1}^{n} F_i ε_i
end for
```

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

   2. First-order: Estimate the Gradient:
      1. Finite Difference Estimation
         1. Estimate kth partial derivative of objective function by perturbing small amount in kth dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

      2. Policy Gradient Theorem
         1. **If we have differentiable policy function**

Policy Gradient Theorem

1. Policy Gradient in analytical form!
   1. Intuitively, consider a simple class of one-step MDPs. (black-board example, $R_{s,a}$ is r for short in the following equations.)

$$
\begin{aligned}
J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a} \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]
\end{aligned}
$$

      1. Why not $\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \pi_\theta(s,a) r\right]$?
         The expectation is on top of the sampled actions and states.
   2. Luckily, we have similar results on all MDPs (skipping proof).

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^{\pi_\theta}(s,a)\right]$$

REINFORCE

1. REINFORCE algorithm:
   1. If use the actual return value as an unbiased sample for Q(s, a)
      1. **v_t is the G_t** in the course slides!

$$Q^{\pi_\theta}(s_t, a_t) = v_t$$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

Toy Example of Rock-Paper-Scissors

1. Question:

Question 7. [20 MARKS]

We would like to use REINFORCE to train an agent that plays Rock Paper Scissors against the computer. The game is played as follows: both the agent and the computer pick an action from the set `{0, 1, 2}`. The reward is $+1$ if the tuple of `(agent, computer)` actions is one of `(0, 1)`, `(1, 2)`, or `(2, 0)`. The reward is -1 if the tuple of `(agent, computer)` actions is one of `(1, 0)`, `(2, 1)`, or `(0, 2)`. The reward is 0 otherwise. (For simplicity, we substitute the integers 0, 1, 2 for Rock, Paper, and Scissors from the familiar game.)

The computer is using an unknown strategy. For a computer action c_{t-1}, taken at time $t-1$, the policy function that defines the probability of agent action a_t is

$$\pi(a_t = a_i | c_{t-1}) = \frac{e^{p_{a_i, c_{t-1}}}}{\sum_{j=0,1,2} e^{p_{a_j, c_{t-1}}}}$$

1. Question: Write pseudocode to learn the parameters using REINFORCE.
2. Reward: +1 for wining, -1 for losing, 0 for draw.
3. Our policy: softmax policy, based on what computer did in the last timestep.
4. Parameters: 9 of them.
5. Game length: **T** (we assume)
6. discount factor = 1.

Toy Example of Rock-Paper-Scissors

1. Basic ideas:
   1. Initialization
      1. Good initialization will boost the training

         Of course we could use uniform policy.
   2. At each iteration
      1. Generate the training data D of length T
      2. Train the policy using the data D
      3. Usually, the more iterations you use, the better performance you have.

function REINFORCE

    Initialise θ arbitrarily

    for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

        for $t = 1$ to $T - 1$ **do**

            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

        end for

    end for

    return θ

end function

Toy Example of Rock-Paper-Scissors

1. Generate the trajectories (length T)
    1. For t = 1 to T (record all the data):
        1. Calculate the softmax probability based on c_{t-1}.

           How to calculate a softmax probability?
        2. Randomly sample a_t from the softmax probability.
        3. Interact with the environment and get feed-back reward r_t & observation c_t (computer's action).

function REINFORCE

  Initialise θ arbitrarily

  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

    for $t = 1$ to $T - 1$ **do**

      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

    end for

  end for

  return θ

end function

Toy Example of Rock-Paper-Scissors

1. **Calculate the total returned reward v_t or G_t**
   1. v_t or G_t = sum(r_t to r_T)
   2. Example:
      1. v_0 or $G_0 = r_0 + r_1 + r_2 + r_3 + r_4 + \ldots\ r_{T\text{-}1} + r_T$
      2. v_1 or $G_1 = r_1 + r_2 + r_3 + r_4 + \ldots\ r_{T\text{-}1} + r_T$
      3. v_2 or $G_2 = r_2 + r_3 + r_4 + \ldots\ r_{T\text{-}1} + r_T$
      4. …
      5. v_T or $G_T = r_T$

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

Toy Example of Rock-Paper-Scissors

1. For t = 1 to T - 1 (every collected game sample), do
   1. Calculate the $\nabla_\theta \log \pi_\theta(s_t, a_t)$ for each parameter based on a_t, v_t, c_{t-1}

$$\frac{\partial \log(\pi(a_t, c_{t-1}))}{\partial p_{a_k, c_j}} = \begin{cases} 0, & \text{if } c_{t-1} \neq c_j \\ \mathcal{I}[a_t = a_k] - \pi(a_k, c_{t-1}), & \text{if } c_{t-1} = c_j \end{cases}$$

      1. How to get this results? (see blackboard)
   2. Update the parameters using gradient descent.

function REINFORCE
    $\boxed{\text{Initialise } \theta \text{ arbitrarily}}$
    for each episode $\boxed{\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta}$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\boxed{\theta \leftarrow \theta + \alpha \boxed{\nabla_\theta \log \pi_\theta(s_t, a_t)} \boxed{v_t}}$
        end for
    end for
    return θ
end function

Toy Example of Rock-Paper-Scissors

1. Putting everything together:
   1. Initialization
   2. for each iteration
      1. Generate the training data D of length T
         1. for t = 1 to T-1
            1. Calculate the action probabilty based on current parameters
            2. Sampled the actions a_t
            3. Record the data (a_t, r_t, c_t)
      2. Train the policy using the data D:
         1. Calculate the returns G_t (or call it v_t)
         2. for t = 1 to T-1
            1. Calculate the gradients.
            2. Do one step of gradient descent.
   3. Return the trained model

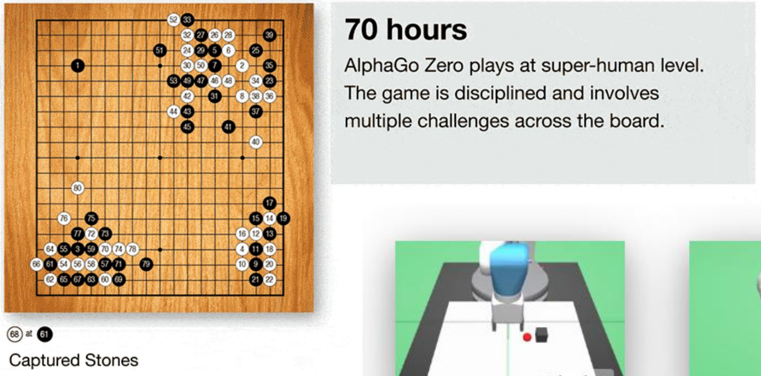
Contents

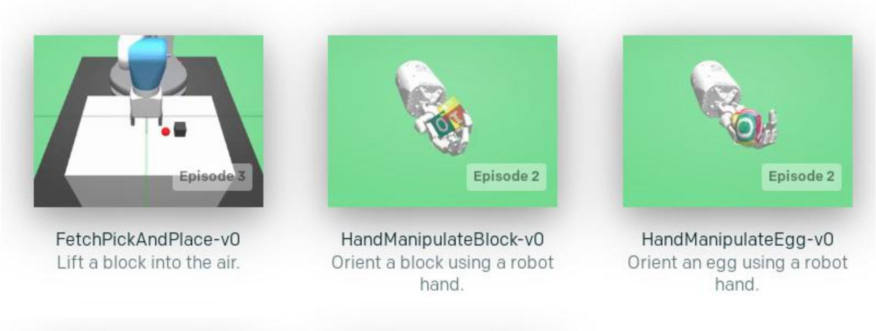
Other Method

1. Trust Region Methods:
   1. State-of-the-art on continuous domian
      1. PPO / TRPO
2. DDPG [12, 13]:
   1. Variants of Policy Gradient
   2. Could achieve state-of-the-art, high variance
   3. Recent Update: D4PG [14]
3. A2C / A3C:
   1. Using critic to reduce variance
   2. Not as good on continuous control as discrete control.

Discrete Domain vs. Continuous Domain

1. Action-Space
   1. Discrete action space [3, 4, 5, 6, 10].
      1. Only several actions are available (e.g. up, down, left, right).
   2. Continuous action space [7].
      1. Action is a value from a continous interval.





Policy Based vs. Value Based

1. Policy Gradient:
   1. Objective function:

      $$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a)\mathcal{R}_s^a$$

   2. Takeing the gradient (Policy Gradient Theorem)

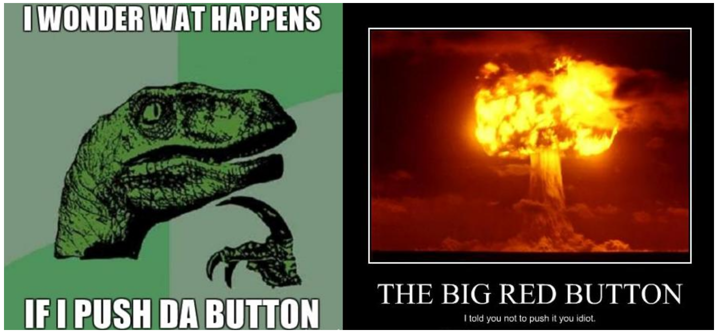
      $$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q_w(s,a)\right]$$

2. Value based methods are more interested in "Value"
   1. Estimate the expected reward for different actions given the initial states (table from Silver's slides [9]).

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
        $S \leftarrow S'$;
    until S is terminal

On-policy vs. Off-policy

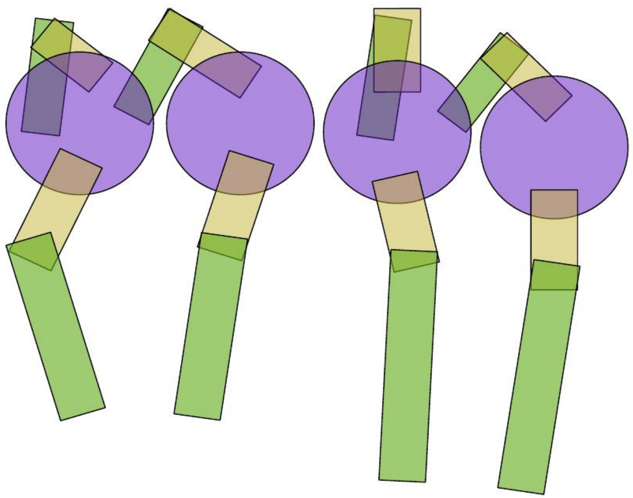
1. Behavior policy & target policy.
    1. Behavior policy is the policy used to generate training data.
        1. Could be generated by other agents (learning by watching)
        2. Could be that the agent just want to do something new to explore the world.
        3. Re-use generated data.

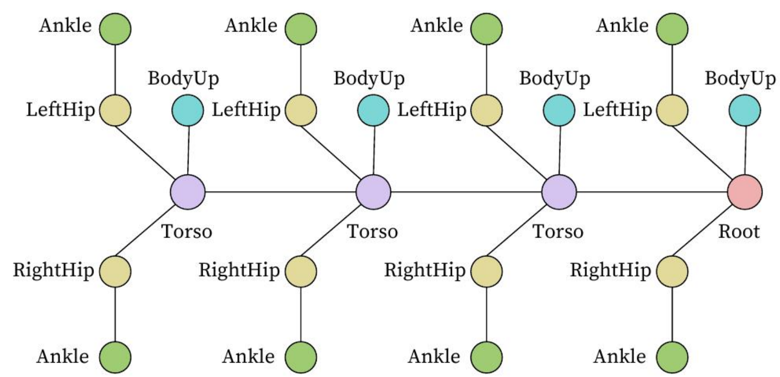


    2. Target policy is the policy the agent want to use if the agent is put into testing.
    3. Behavior policy == target policy: On-policy, otherwise Off-policy

NerveNet: Learning Stuctured Policy in RL

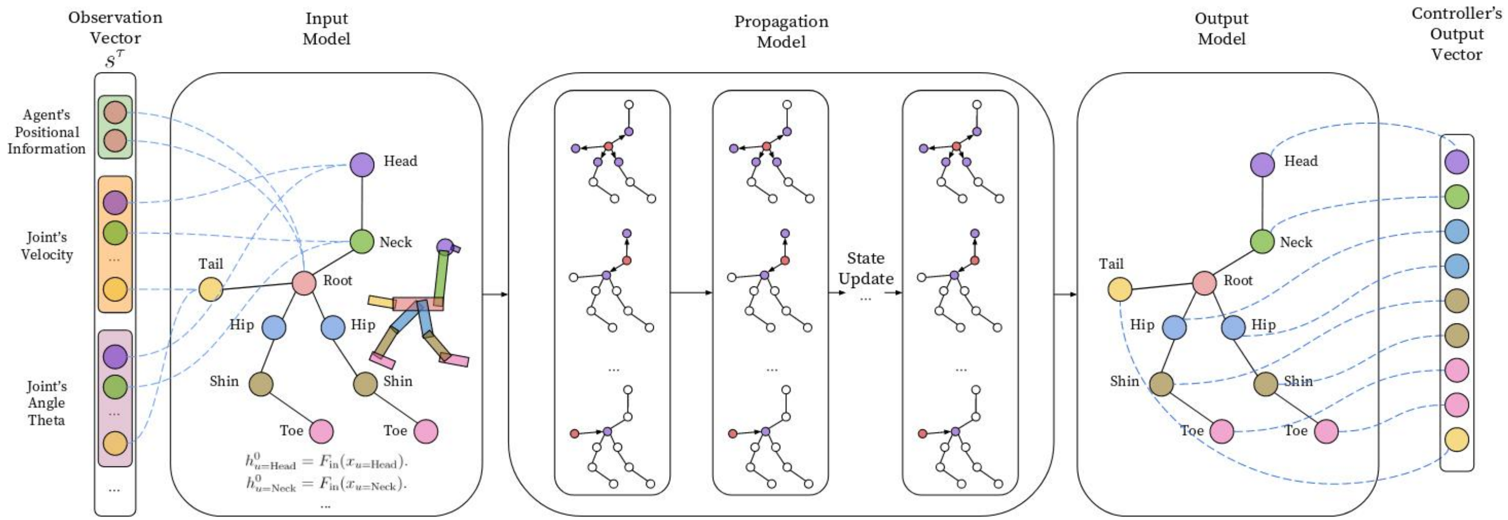
1. NerveNet ICLR'18:
   1. In traditional reinforcement learning, policies of agents are learned by MLPs which take the concatenation of all observations from the environment as input for predicting actions.
   2. We propose NerveNet to explicitly model the structure of an agent, which naturally takes the form of a graph.





NerveNet: Learning Stuctured Policy in RL

1. NerveNet:
   1. Using graph neural network to encode structure information.



Contents

Reference

[1] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems. 2000.

[2] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Reinforcement Learning. Springer, Boston, MA, 1992. 5-32.

[3] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[4] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

[5] Schulman, John, et al. "Trust region policy optimization." Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 2015.

[6] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[7] Todorov, Emanuel, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control." Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. IEEE, 2012.

[8] Tamar, Aviv, et al. "Value iteration networks." Advances in Neural Information Processing Systems. 2016.

[9] Silver, David, UCL Course on RL, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

[10] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.

[11] Salimans, Tim, et al. "Evolution strategies as a scalable alternative to reinforcement learning." arXiv preprint arXiv:1703.03864 (2017).

[12] Silver, David, et al. "Deterministic policy gradient algorithms." Proceedings of the 31st International Conference on Machine Learning (ICML-14). 2014.

[13] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[14] Tassa, Yuval, et al. "DeepMind Control Suite." arXiv preprint arXiv:1801.00690 (2018).