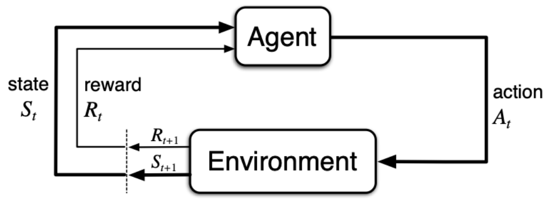
Policy Gradient Methods for Deep Reinforcement Learning

Hongshan Li(hongshal@amazon.com)

August 12, 2021

RL is big and rich, its history traces back to Pavlov's study on the psychology of animal learning (conditional response) back in 1902. The expository article History of Reinforcement Learning (http://incompleteideas.net/book/first/ebook/node12.html) is a good resource to learn how RL evolved in the last 100 years.
Main reference for this note is the unbeatable classic (http://incompleteideas.net/book/the-book.html)
Policy Gradient Methods is a class of RL algorithms that directly learns a policy from interaction with the environment. In contrast, to this, one can also learn a state value estimate of the current state of the env and the candidate actions. This class of algorithms is called Q-learning.
Review about setups in RL The objective of RL is to find a policy π that gets the highest rewards from a *Markov Decision Process*



At time step t, the env is at state S_t, the agent takes action A_t, the env transition into S_{t+1} and the agent receives a reward of R_{t+1}.
The critical assumption of MPD is

- The transition probability $P(S_{t+1}, R_{t+1}|S_t, A_t)$ is independent from the history
- The agent's action does not alter the transition probability, i.e. $P(S_{t+1}, R_{t+1}|S_t, A_t)$ is the same whenver the agent sees S_t and takes action A_t

The transition probability $P(S_{t+1}, R_{t+1}|S_t, A_t)$ is call the *model* of the env. The objective of RL is to find a *optimal policy* (what action to take at S_t) that maximize the total reward
If $P(S_{t+1}, R_{t+1}|S_t, A_t) = 1$ for all S and A, then we say the env is deterministic, otherwise stochastic.

The total reward from time t is

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

The MDP terminates at T.
To prioritize short term return over long term return, we use a discount factor $\gamma \in [0, 1]$ and define

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^k R_{t+k+1} + \cdots + \gamma^{T-t-1} R_T$$

A few ways to think about the discount factor

- prioritize short term reward
- A mathematical convenience for infinite episodic task; we want G_t to be a finite number even if the MDP never stops.

  $$\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$$

  so

  $$G_t \leq \frac{1}{1-\gamma} * R_{\max}$$

- a way to factor in the probability that agent dies. The agent survives with prob γ and die with prob $1-\gamma$. i.e reward at time t is

  $$\gamma R_{t+1} + (1-\gamma) * 0$$

Let $\mathbb{S}$ be the state space, and $\mathbb{A}$ the action space. A *deterministic policy* is a mapping

$$\pi : \mathbb{S} \to \mathbb{A}$$

and a *stochastic* policy is a mapping

$$\pi : \mathbb{S} \to Dist(\mathbb{A})$$

where $Dist(\mathbb{A})$ denote the space of probability distributions over the action spaces.
Both deterministc env and determinsitc policy are special cases of their stochastic counterparts.(all probability concentrate at 1 point). So we only address the stochastic cases in the rest of the note.
Suppose the agent's current policy is π, then at state S_t π yields a probability distribution over the action space

$$\pi(\cdot|S_t)$$

from which the agent samples actions.

So with the model of the env, we can write down the expected total reward if we follow the policy π. Assume the initial state is s_0

$$\begin{aligned} v_\pi(s_0) &:= \mathbb{E}_{\tau\sim\pi}[G_0|S_0 = s_0] \\ &\mathbb{E}_{\tau\sim\pi}[R_1 + \gamma v_\pi(s_1)] \end{aligned}$$

s_1 denotes the next state of the env. τ denotes a trajectory sampled from the policy π.
We can use Monte Carlo method to estimate $v_\pi(s_0)$ by running the policy a couple of times and compute the average return G_0.
Policy Iteration How do we improve the policy? That's the entire subject of RL. In supervised learning, the objective is to find a parametric model p_θ that maximize the likelihood of the input dataset; in RL, the objective is to train a policy that gets the highest reward from the env.
A fundamental difference supervised learning and RL is that for SL data is completely determined before any training, whereas for RL data is dynamically generated through interaction with the environment. This means crappy policy could generate useless data by visiting the useless state space. The distribution of data for RL changes over time, whereas for SL the distribution is static.
This might sounds pretty hopeless, how do you overcome the garbage in and garbage out positive feedback loop? Just hang on.
Suppose we have a value function v_π for the current policy π, one naive way to improve π is to "look ahead one step of time". At t, we are at $S_t = s$, the policy results an action $\pi(s)$, we want whether it is a good idea to choose a different action $a \neq \pi(s)$? We can simply take a and follow π afterwards. This is defined as the state-action value

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]$$

Recall

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi(s)]$$

If $q_\pi(s,a) - v_\pi(s) > 0$, then it is a good idea to assume action a at s.
This gives us a template for generalized policy iteration

Algorithm 1: Generalized Policy Iteration

```
1: Input: initial policy $\pi$
2: repeat
3:     evaluate the policy $\pi$ and get $v_\pi$
4:     for each $s \in \mathbb{S}$ do
5:         $oldAction \leftarrow \pi(s)$
6:         $\pi(s) \leftarrow \text{argmax}_a \mathbb{E}[r + \gamma v(s_1)$
7:     end for
8: until $\pi$ stops improving, i.e $oldAction = \pi(s) \forall s$
```

So this is what you do to overcome the garbage in and garbage out. You can get a good policy by iterating one-step a time.

1 Value Iteration with Dynamic Programming

How do we evaluate a policy π, i.e. How do we compute $v_\pi(s_0)$,i.e. line 3 of the GPI? Suppose the environment's model and the policy is completely known, then

$$\begin{aligned} v_\pi(s_0) &= \mathbb{E}_\pi[G_t|S_t = s_0] && (1)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_1)] && (2)\\ &= \sum_a \pi(a|s) \sum_{s_1,r_1} p(s_1, r_1|s,a)[r_1 + \gamma v_\pi(s_1)] && (3) \end{aligned}$$

If the MDP is finte with n states, then the above relations give s system of linear equations in variables $v_\pi(s_0), \cdots, v_\pi(s_{n-1})$. So solving finite MDP is nothing but solving a linear system.
For small MDP, you can just invoke some middle school math to find a exact solution to the MDP. Most interesting problems are big or even infinite, so Gaussian elimination does not really apply. We need a *approximate algorithm* to solve the MDP.
Value Iteration Suppose the agent is able to act infinitely many steps in the MDP (infinite episodic task). Then you can iteratively solve the MDP by letting the agent act for 0 step, 1 step, 2 steps, $\cdots$, n steps, $\cdots$. Let $v_k(s)$ denote the approximate solution for $v_\pi(s)$ if you let the agent to act k steps. Then

$$v_0(s) = 0, \forall s \in \mathbb{S}$$

and

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r'} p(s', r')[r' + \gamma v_k(s')]$$

i.e. you act $k + 1$ steps by acting 1 step then k steps (almost tataulogy).

Algorithm 2: Value Iteration

```
1: Input: a policy $\pi$ to be evaluated, a small residue tolerance $\epsilon > 0$ determinig
   the accuracy of the policy evaluation, $v(s) = 0 \forall s \in \mathbb{S}$
2: repeat
3:   $\Delta \leftarrow 0$
4:   for each $s \in \mathbb{S}$ do
5:     $v \leftarrow v(s)$
6:     $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s,a)[r + \gamma v(s']$
7:     $\Delta \leftarrow \max(\Delta, |v - v(s)|)$
8:   end for
9: until $\Delta \leq \epsilon$
```

2 Function Approximation and Policy Gradient Methods

The above generalized policy iteration works pretty well if the underlying MPD is finite (finitely states). One gets v_π by dynamic programing. But this becomes intractable if the state space is big. Suppose your state is given as a 2D image with dimension 512x512, then your state space is of size $(3x515x512)^{256} \sim 1.5x10^{1510}$.
Function approximation is the idea that addresses the curse of dimension problem when the state space gets big (or even infinite). The idea is if two states are "similiar" the agent should take the similar actions and the value of that states should be similar as well. So we can use parametrized functions v_θ and π_θ to represents the value and policy tables. The differentiabilities of v_θ and π_θ ensures that similar states have similar value and the agent acts similarly.
Now we are officially in the Deep Reinforcement Learning (DRL) world. **Policy gradient methods** concerns about the following question: How do we move the parameters of π_θ so that π_θ improves. "Move" in this context means updating θ according to the *policy gradient.*
In tabular method, we have defined the value of a policy. Analogous definition carries over

$$J(\theta) = v_{\pi_\theta}(s)$$

To improve $J(\theta)$, we do the usual calculus trick (gradient ascend)

$$\theta \leftarrow \theta + \alpha \nabla J(\theta)$$

This is basically the entirty of policy gradient methods.
So the next thing is how to compute $\nabla J(\theta)$

Theorem 1 (Policy Gradient Theorem) *Suppose the state space $\mathbb{S}$ and action space $\mathbb{A}$ are discrete*

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)$$

where $\mu(s)$ is the visition probability *(how often the agent visits state s*
If $\mathbb{S}$ and $\mathbb{A}$ are continuous

$$\nabla J(\theta) \propto \int_{\mathbb{S}} \mu(s) \int_{\mathbb{A}} \nabla \pi(a|s) q_\pi(s,a) ds da$$

More details about $\mu(s)$ before moving on the proof. Consider the visitation sequencce from state s to x following the policy π after k steps

$$s \xrightarrow{a \sim \pi(\cdot|s)} s_1 \xrightarrow{a \sim \pi(\cdot|s_1)} s_2 \cdots$$

(Markov Chain) denote the probability of transitioning from state s to x in k steps as $\rho^\pi(s \to x, k)$
Assume for simplicity that we only deal with finite episodic tasks, i.e. only allow the agent to take n steps of actions for $n \leq \infty$.

- $k = 0, \rho^{\pi}(s \to s, k = 0) = 1$
- $\rho^{\pi}(s \to s_1, k = 1) = \sum_a \pi(a|s) P(s_1|s, a)$
- $\rho(s \to x, k + 1) = \sum_{s_1} \rho(s \to s_1, k)\rho(s_1 \to x, 1)$
- $\eta(x) := \sum_k^n \rho(s_0 \to x, k)$
- Define $\mu(x) := \frac{\eta(x)}{\sum_x \eta(x)}$ if $\mathbb{S}$ is discrete and $\mu(x) := \frac{\eta(x)}{\int_s \eta(x)dx}$ if $\mathbb{S}$ is continuous. So $\mu(x)$ becomes a probability mass function (or probability distribution). It can be understood as a measure of how frequent the agent lands at state x

With the understanding of $\mu(x)$ as the visitation distribution (relative importance of the state x), how we do interpret the policy gradient?
Policy gradient is the weighted average of gradient of $\pi(a|s)$ by the value of action a and relative importance of the state s

Proof 1 (policy gradient theorem) *For notational convenience, write $v(s)$ for $v_\pi(s)$ and $\nabla v(s)$ to mean $\nabla_\theta v_\pi(s)$, i.e. gradient of $v(s)$ with respect to the parameter θ of π.*

$$\begin{aligned}\nabla v(s) &= \nabla[\sum_a \pi(a|s) q_\pi(s, a)] \\ &= \sum_a [\nabla \pi(a|s) q_\pi(s, a)] + \sum_a \pi(a|s) \nabla q_\pi(s, a) \\ q_\pi(s, a) &= \sum_{s_1, r} p(s_1, r|s, a)(r + v(s_1))\end{aligned}$$

so

$$\begin{aligned}\nabla v(s) &= \nabla[\sum_a \pi(a|s) q_\pi(s, a)] \\ &= \sum_a [\nabla \pi(a|s) q_\pi(s, a)] + \sum_a \pi(a|s) \sum_{s_1} p(s_1|s, a) \nabla v(s_1)\end{aligned}$$

Note the recursive relation between $\nabla v(s)$ and $\nabla v(s_1)$
Write $\phi(s) = \sum_a \nabla \pi(a|s) q(s, a)$ and $\rho(s \to s_1, 1) = \sum_a \pi(a|s) p(s_1|s, a)$
So

$$\begin{aligned}\nabla v(s) &= \phi(s) + \sum_{s_1} \rho(s \to s_1, 1) \nabla v(s_1) \\ &= \phi(s) + \sum_{s_1} \rho(s \to s_1, 1)\phi(s_1) + \sum_{s_2} \rho(s \to s_2, 2) \nabla v(s_2) \\ &= \phi(s) + \sum_{s_1} \rho(s \to s_1, 1)\phi(s_1) + \sum_{s_2} \rho(s \to s_2, 2)\phi(s_2) + \sum_{s_3} \rho(s \to s_3, 3) \nabla v(s_3) \\ &\vdots \\ &= \phi(x) + \sum_{s_1} \rho(s \to s_1, 1)\phi(s_1) + \cdots + \sum_{s_n} \rho(s \to s_n, n)\phi(s_n)\end{aligned}$$

The unrolling stops after n steps, because we only let the agent to act n steps. so $v(s_{n+1})$ *becomes a constant function and it has no derivative.*

$$\begin{aligned}
\nabla v(s) &= \sum_x \sum_{k=0}^{n} \rho(s \to x, k)\phi(x) \\
&= \sum_x \eta(x)\phi(x) \\
&= (\sum_x \eta(x)) \sum_x \frac{\eta(x)}{\sum_x \eta(x)} \phi(x) \\
&= (\sum_x \eta(x)) \sum_x \mu(x)\phi(x) \\
&\propto \sum_x \mu(x)\nabla\pi(a|s)q(s,a)
\end{aligned}$$