HelmholtzZentrum münchen
Deutsches Forschungszentrum für Gesundheit und Umwelt

Technische Universität München

Deep Reinforcement Learning

Actor-Critic & Trust region-based Policy Optimization

Andres Becker
27.01.2021
Technische Universität München
Deep Learning Seminar
Supervisor: Dr. Hananeh Aliee

Road Map

01 What is Reinforcement Learning?

02 Key Concepts of Reinforcement Learning

03 Policy Gradient Methods

04 Actor-Critic

05 Trust region-based policy optimization

01

What is RL?

RL is "learning by doing"

Supervised Learning: The experience is given in the form of input and output examples, and the goal is to learn a general rule that maps inputs to outputs.

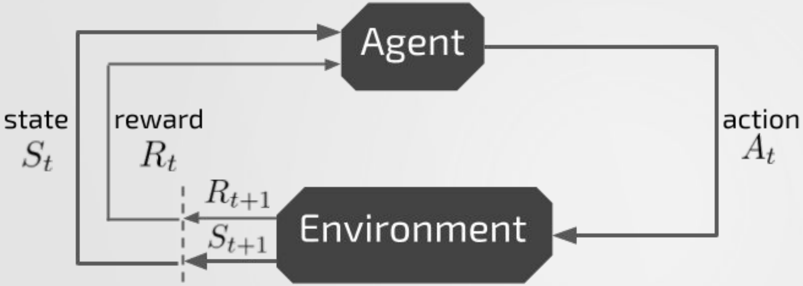
Unsupervised Learning: The experience is given in the form of data (no outputs provided) and the goal is to discover hidden patterns in data.

Reinforcement Learning: No experience (data) is given, instead a dynamic **Environment** is provided and an **Agent** must learn how to interact with it in order to achieve a goal.

What is an Agent and an Environment?

Agent:

- At every time step t, it execute one of many actions into the Environment: $a_t \in \mathcal{A}$



Environment:

- Is where the Agent acts (or interacts)
- At every time step t, the environment is in one of many states: $s_t \in \mathcal{S}$
- It is responsible for providing feedback (**reward**) to the agent: $r_t \in \mathcal{R}$
    - Positive reward = Good action taken by Agent
    - Negative reward = Bad action taken by Agent

LunarLander-v2 GYM Environment

The **Environment** is basically a section of the moon in 2D.

- The landing path is always at (0,0)
- The state of the environment is a vector $s \in \mathbb{R}^8$ where:
    - s[0], s[1]: (x, y) coordinates of the Lunar Lander
    - s[2], s[3]: x and y velocity components of the Lunar Lander
    - s[4]: angle of the Lunar Lander
    - s[5]: angular speed of the Lunar Lander
    - s[6], s[7]: status of the first and second leg of the Lunar Lander (1 if land contact, 0 ow)
- The **Reward** is a composition of the position, velocity, angel and legs status of the Lunar Lander. Also, it consider how many times the engines were activated.

The **Agent** is the Lunar Lander.

- The agent can take one of the following actions:
    - a_0: do nothing
    - a_1: Fire right orientation engine
    - a_2: Fire main engine (Thruster)
    - a_3: Fire right orientation engine

02

Key Concepts of RL

Elements of RL

- **Set of states:** $\mathcal{S}$
  - Holds the Markov property: $P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1) = P(s_{t+1}|s_t)$
- **Set of actions:** $\mathcal{A}$
- **Transition probability function:** $P(s_{t+1}, r_{t+1}|s_t, a_t)$
- **Reward function:** $R(s_t, a_t) := \mathbb{E}[R_{t+1}|s_t, a_t] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s_t, a_t)$

This 4 elements $< \mathcal{S}, \mathcal{A}, P, R >$ defines a **Markov Decision Process**

- **Transition step:** (s_t, a_t, r_t, s_{t+1})
- **Episode:** $\{s_0, a_0, r_t, s_2, a_2, \ldots, r_{T-1}, s_T\}$

The Return

$$G_t := \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

- **Discount factor:** $\gamma \in [0, 1]$
- **Reward:** r_t

How to choose actions that maximize the Return?

The Policy π

The **policy** is the functions that defines the agent's behavior and maps states with actions. It can be either deterministic or stochastic:

- **Deterministic:**

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

$$\pi(s) = a$$

- **Stochastic:**

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$$

$$\pi(a|s) = P_\pi(A = a|S = s)$$

Value Functions

If we follow a policy π, how to quantify how good a state ***s*** or an action ***a*** is in terms of the return ***G***?

- **State-value** function: $V_\pi(s) := \mathbb{E}_\pi[G_t | s_t = s]$

- **Action-value** function: $Q_\pi(s, a) := \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$

03

Policy Gradient Methods

What is Policy Gradient?

- Recall that the goal of RL is to find an optimal behavior strategy for the agent (i.e. an strategy that maximizes a performance measure ***J***).
- **Policy Gradient method** model and optimize the policy directly by means of parameterized and differentiable functions.
- Artificial Neural Networks are an excellent option!

$$\pi(a|s,\boldsymbol{\theta}) = Pr(A_t = a|S_t = s; \boldsymbol{\theta}_t = \boldsymbol{\theta})$$

- Then, using gradient ascent we can optimize the policy parameters $\boldsymbol{\theta}$ while maximizing the performance measure $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} + \alpha\widehat{\nabla J(\boldsymbol{\theta})}$$

where $\mathbb{E}[\widehat{\nabla J(\boldsymbol{\theta})}] = \nabla J(\boldsymbol{\theta})$

Which performance measure $J(\boldsymbol{\theta})$ to use?

Recall that we seek to maximize the future cumulative reward at every time step, therefore:

$$\begin{aligned} J(\boldsymbol{\theta}) &:= \mathbb{E}_\pi[G_t | s_t = s] \\ &= V_{\pi_{\boldsymbol{\theta}}}(s_t) \end{aligned}$$

and therefore

$$\nabla J(\boldsymbol{\theta}) = \nabla V_{\pi_{\boldsymbol{\theta}}}(s_t)$$

But there is a problem…
We need to write the right hand side in terms of the policy derivative

...however, the **Policy Gradient Theorem** comes to our aid!

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla V_{\pi_{\boldsymbol{\theta}}}(s_t) \\
&= \nabla \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t;\boldsymbol{\theta}) Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \\
&\propto \sum_{s_t \in \mathcal{S}} \mu(s_t) \sum_{a_t \in \mathcal{A}} Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \nabla \pi(a_t|s_t;\boldsymbol{\theta}) \\
&= \sum_{s_t \in \mathcal{S}} \mu(s_t) \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t;\boldsymbol{\theta}) Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \frac{\nabla \pi(a_t|s_t;\boldsymbol{\theta})}{\pi(a_t|s_t;\boldsymbol{\theta})} \\
&= \sum_{s_t \in \mathcal{S}} \mu(s_t) \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t;\boldsymbol{\theta}) Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \nabla \log \pi(a_t|s_t;\boldsymbol{\theta}) \\
&= \mathbb{E}_{s \sim \mu_\pi, a \sim \pi_\theta}[Q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \nabla \log \pi(A_t|S_t;\boldsymbol{\theta})] \\
&= \mathbb{E}_{s \sim \mu_\pi, a \sim \pi_\theta}[G_t \nabla \log \pi(A_t|S_t;\boldsymbol{\theta})]
\end{aligned}
$$

Now the **update rule** ends as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \log \pi(A_t | S_t; \boldsymbol{\theta})$$

Generalizing the gradient of the performance measure

We can go even further and add a baseline function $b(s)$ to reduce variance, but more of it in a while...

$$\nabla J(\boldsymbol{\theta}) \propto \sum_{s_t \in \mathcal{S}} \mu(s_t) \sum_{a_t \in \mathcal{A}} \left(Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t) - b(s_t)\right) \nabla \pi(a_t | s_t; \boldsymbol{\theta})$$

This baseline can be any function, even a random variable, as long as it does not depend on the action ***a***:

$$\begin{aligned}\sum_{a_t \in \mathcal{A}} b(s_t) \nabla \pi(a_t | s_t; \boldsymbol{\theta}) &= b(s_t) \sum_{a_t \in \mathcal{A}} \nabla \pi(a_t | s_t; \boldsymbol{\theta}) \\ &= b(s_t) \nabla 1 \\ &= 0\end{aligned}$$

And finally the **update rule** for the policy parameters ends ends as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - b(S_t))\nabla \log \pi(A_t|S_t;\boldsymbol{\theta})$$

04

Actor-Critic Methods

What is the Actor and the Critic?

Now we have an update rule for the policy parameters:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - b(S_t))\nabla \log \pi(A_t|S_t;\boldsymbol{\theta})$$

However, we actually do not know the value of the Return **G** at every time step… But fortunately:

$$G_t = R_{t+1} + \gamma G_{t+1} = R_{t+1} + V_\pi(S_{t+1})$$

But again, we do not know the state-value function $V_\pi(S_{t+1})$!
So, what if we approximate the state-value function in the same way we do with the policy, i.e. with a parametrized differentiable function $V(S;\boldsymbol{w})$?

And therefore:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1};\boldsymbol{w}) - b(S_t))\nabla \log \pi(A_t|S_t;\boldsymbol{\theta})$$

What is the Actor and the Critic?

Actor-critic methods consist of two models (which may optionally share parameters):

- **Critic** updates the parameters ***w*** of the state-value function $V(S;\boldsymbol{w})$ and measures (criticize) how good the action take by the policy was.
- **Actor** updates the parameters $\boldsymbol{\theta}$ of the policy function $\pi(a|s;\boldsymbol{\theta})$, in the direction suggested by the critic.

But what about the baseline?

- The **baseline** $b(s)$ leaves the expected value of the update unchanged, but it can significantly reduce the variance (and thus speed the learning).
- In MDPs the baseline should vary with the state.
- In some states all actions may have high values and we would need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions may have low values and a low baseline would be appropriate.
- One natural choice for the baseline is an estimate of the state-value function, $V(S_t; \boldsymbol{w})$. Hence, the (final) **update rule** for the policy parameters ends as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}; \boldsymbol{w}) - V(S_t; \boldsymbol{w}))\nabla \log \pi(A_t|S_t; \boldsymbol{\theta})$$

Update rule for state-value function parameters

- As it was done with the policy, we need to start by defining an objective function.
- Since $V : \mathcal{S} \to \mathbb{R}$ (regression problem), we can just use the **Squared Error** as a loss function:

$$E(\boldsymbol{w}) := (V_\pi(S) - V(S; \boldsymbol{w}))^2$$

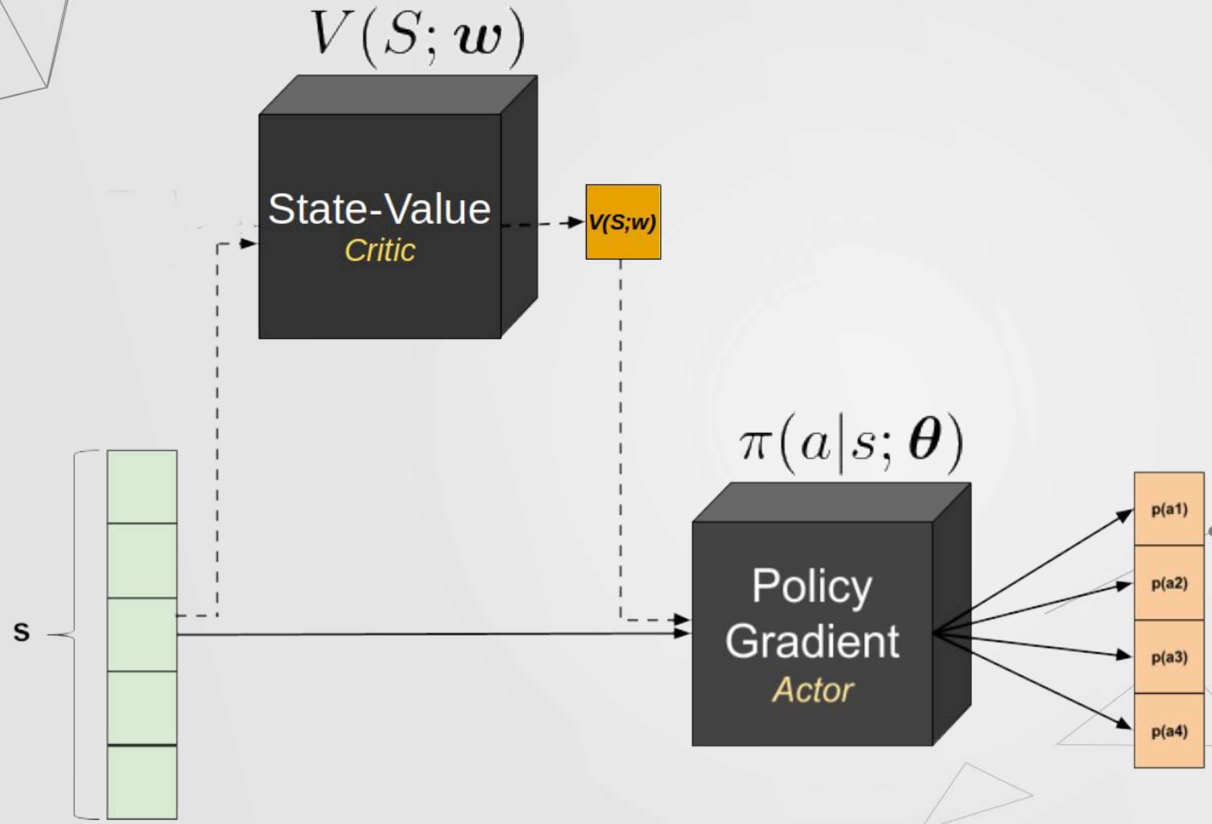
then

$$\begin{aligned}\nabla E(\boldsymbol{w}) &= \nabla (V_\pi(S_t) - V(S_t; \boldsymbol{w}))^2 \\ &= 2(V_\pi(S_t)) - V(S_t; \boldsymbol{w}))\nabla V(S_t; \boldsymbol{w})) \\ &\approx 2(R_{t+1} + \gamma V(S_{t+1}; \boldsymbol{w}) - V(S_t; \boldsymbol{w}))\nabla V(S_t; \boldsymbol{w}))\end{aligned}$$

and therefore

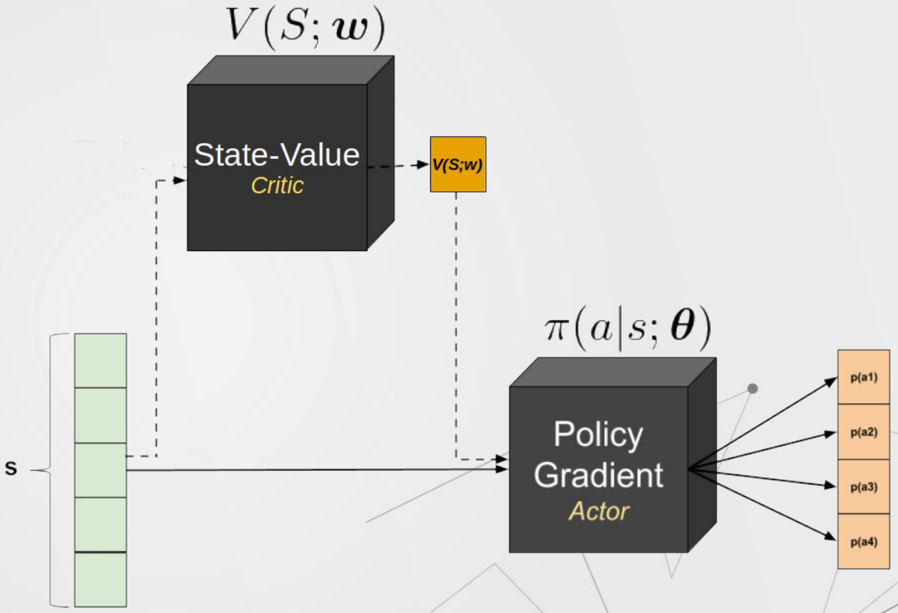
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha(R_{t+1} + \gamma V(S_{t+1}; \boldsymbol{w}) - V(S_t; \boldsymbol{w}))\nabla V(S_t; \boldsymbol{w})$$

Schematic view of an Actor-Critic architecture



Advantage Actor Critic

Input: differentiable policy and state-value function parameterization $\pi(a|s;\boldsymbol{\theta})$ and $V(s;\boldsymbol{w})$;
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\boldsymbol{w}} > 0$ and discount factor $\gamma \in [0,1]$;
Initialize policy and state-value parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\boldsymbol{w} \in \mathbb{R}^{d'}$;
for *Episode = 1, 2, ...* **do**
  while S_t *is not terminal (for each time step t)* **do**
    Sample action $A_t \sim \pi(\cdot|S_t;\boldsymbol{\theta})$;
    Take action A_t and observe S_{t+1}, R_{t+1};
    if S_{t+1} *is terminal* **then**
      $V(S_{t+1};\boldsymbol{w}) = 0$;
    end
    $\delta \leftarrow R_{t+1} + \gamma V(S_{t+1};\boldsymbol{w}) - V(S_t;\boldsymbol{w})$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha^{\boldsymbol{w}} \delta \nabla V(S_t;\boldsymbol{w})$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \nabla \log \pi(A_t|S_t;\boldsymbol{\theta})$;
    $S_t \leftarrow S_{t+1}$;
  end
end

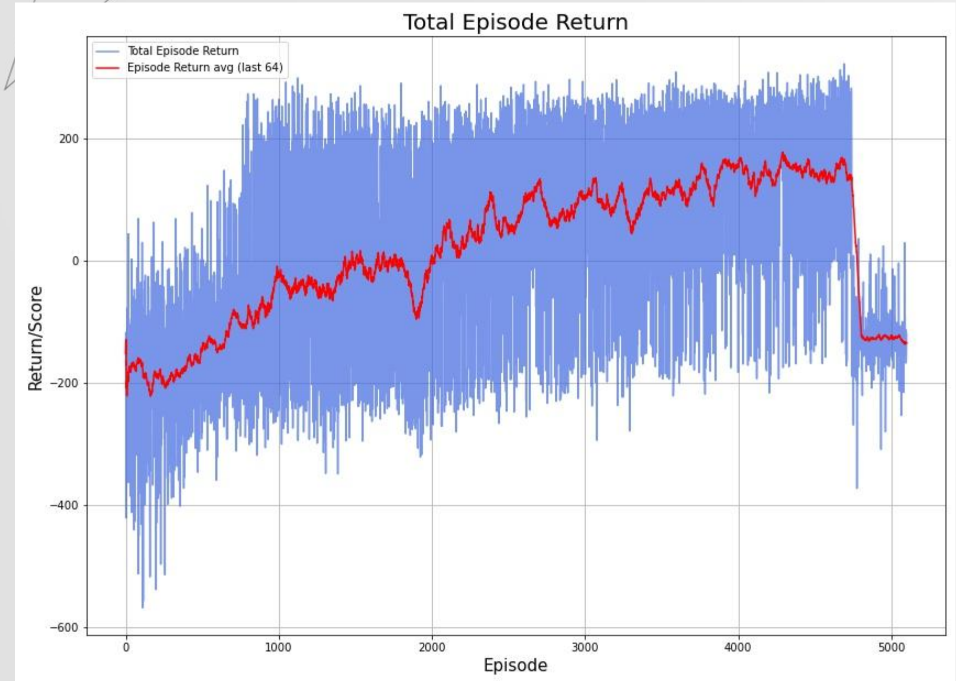


05

Trust region-based policy optimization

Drawbacks of Policy Gradient

Sometimes things don't go the way we expect...



- In practice the training process is often unstable.
- RL algorithms learn from data generated by the interaction between the environment and the agent. Thus, a strong change in the policy means a strong change in the data distribution.
- A too aggressive update in the policy could lead to a bad policy, which will result in poor data, and therefore in a poor training process.

Bad policy = Bad data

= Poor or meaningless learning

Solution: constrain Policy updates!

One way to prevent instability in the training process is by adding **regularization** to the difference between subsequent policies:

$$D_{KL}(\pi_{\theta_{old}}||\pi_{\theta}) = \mathbb{E}\left[\int \pi(A|S;\boldsymbol{\theta}_{old}) \log \frac{\pi(A|S;\boldsymbol{\theta})}{\pi(A|S;\boldsymbol{\theta}_{old})} \mathrm{d}a\right]$$

where $\eta > 0$ is the regularization hyperparameter.
Then, transform the original optimization problem (maximize $J(\boldsymbol{\theta})$) into a constrained optimization problem:

$$\begin{aligned} \text{maximize} \quad & J(\boldsymbol{\theta}) \\ \text{s.t.} \quad & D_{KL}(\pi_{\theta_{old}}||\pi_{\theta}) \leq \eta \end{aligned}$$

Finally, approximate the constrained optimization problem just by maximizing:

$$J(\boldsymbol{\theta}) - \eta D_{KL}(\pi_{\theta_{old}}||\pi_{\theta})$$

THANKS!

You can find an implementation of the vanilla Advantage Actor Critic algorithm on my git:

github.com/andresbecker/Deep_RL_Actor_Critic

Do you need more motivation?