Deep Reinforcement Learning Actor-Critic and Trust region-based

Andres Becker

Technische Universität München
Fakultät für Mathematik

January 27, 2021

Abstract

In recent years, significant progress has been made in solving challenging problems across several fields using Reinforcement Learning (RL). Projects like Deep Mind's AlphaGo Zero [3] have accomplish outstanding results without supervised learning on human knowledge, and showed us the amazing results that RL can achieve. In this small resume, we will focus in a group of RL methods called Actor-Critic *and the* trust region-based *technique aimed to improve their stability.*

I. What is Reinforcement Learning?

We refer as Machine learning (ML) to the group of algorithms that improve (learn) automatically through experience. Among this algorithms, we could say that there are three main classes (which depend on the kind of experience we provide):

- **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.

The best way to summarize RL is "learning by interaction"; An agent interacts with and environment, which in return provides the agent with feedback in form of a reward (see figure 1). Ultimately, the goal of RL is that the agent learns a good (optimal) strategy from experimental trials, and a relative simple received feedback, so that the cumulative rewards is maximized. In reality, the scenario could be a bot playing a game to achieve high scores, or a robot trying to complete physical tasks with physical items.

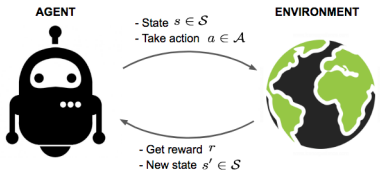

Figure 1: *Interaction between an Agent and the environment. Image source: [6].*

II. Key Concepts of RL

As we can see in figure 1, the agent interacts with the environment sequentially. Therefore, on time step $t \in \{1, ..., T\}$, the agent observe the state of the environment $s_t \in \mathcal{S}$, acts in consequence executing the action $a_t \in \mathcal{A}$, receive its reward $r_t \in \mathcal{R}$ from the environment and observes the new state $s_{t+1} \in \mathcal{S}$. The tuple (s_t, a_t, r_t, s_{t+1}) is known as *transition* step. This interactions continues until the agent: *i*) Completes the task successfully; *ii*) Fails; *iii*) Reach a time limit. This chain of transition steps from the beginning ($t = 1$) until the end ($t = T$) is call *Episode*. We are interested in environments for which the model that defines them is unknown (*Model-free RL*), and has to be learned explicitly as part of the algorithm (otherwise the optimal solution can be found by means of *Dynamec Programming* (DP)).
This interaction is an example of a Markov Decision Process (MDP)[4] $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R >$, where:

- $\mathcal{S}$ is the set of states that holds the Markov property[1].

[1]The conditional probability distribution of future states (con-

- $\mathcal{A}$ is the set of actions.
- P is the transition probability function: $P(s_{t+1}, r_{t+1}|s_t, a_t)$.
- R is the reward function: $R(s_t, a_t) := \mathbb{E}[R_{t+1}|s_t, a_t] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s_t, a_t)$.

However, note that in fact we are not interested in maximizing the reward on each time step but the *Return* instead, which is the future cumulative reward[2]:

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

where $\gamma \in [0, 1]$ is the discount factor meant to encourage the agent to finish the task as soon as possible, since in practice time is a valuable resource.

Recall that the ultimate goal of RL is to find an optimal strategy, such that it maximize the Return. Then, how to choose actions that help us to achieve this? The answer is the *Policy* π. The policy is the functions that defines the agent's behavior and maps a state s to an action a. This function can be either deterministic or stochastic:

- **Deterministic**: $\pi : \mathcal{S} \to \mathcal{A}$; i.e. $\pi(s) = a$.
- **Stochastic**: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$; i.e. $\pi(a|s) = P_\pi(A = a|S = s)$.

Now that we have (more or less) a clear idea of what RL seeks (maximize the return G) and how to do it (by finding an optimal policy π), we need to define a way to quantify how good a state s is (in terms of the return G) in a given environment (when we follow a policy π), and how good an action a is (in terms of the return G) when the environment is in state s (and when we follow a policy π). This is done through the *State-value* and *Action-value* functions respectively:

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

III. Policy Gradient methods

There are several ways to approximate an optimal strategy. For example, in *action-value* methods we learn the values of actions and then select actions based on their estimated action values (e.g. *Q-learning*). Nevertheless, this means that their policies would not even exist without the action-value estimates [4].

However, in *Policy Gradient* we consider methods that instead learn a parameterized policy (e.g. an artificial neural network) that can select actions without consulting a value function[3]. Therefore, we write

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

to denote the probability that the action a is taken, given the parameters $\boldsymbol{\theta}$ and that the environment is in state s at time t. Then, we can approximate an optimal policy by means of gradient ascent while maximizing a performance measure $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{2}$$

where $\widehat{\nabla J(\boldsymbol{\theta})}$ is a stochastic unbiased estimator of $\nabla J(\boldsymbol{\theta})$ (i.e. $\mathbb{E}[\widehat{\nabla J(\boldsymbol{\theta})}] = \nabla J(\boldsymbol{\theta})$).

Then, how to define the performance measure $J(\boldsymbol{\theta})$? There are many ways to define J, which depend mainly on the method used to approximate the policy. However, since we are considering only episodic environments[4], then we seek to maximize the return in a given time step (G_t), and therefore we define it as the value of the state s_t , i.e. $J(\boldsymbol{\theta}) := V_{\pi_\theta}(s_t)$.

Since we are approximating an optimal policy, we need to express $\nabla J(\boldsymbol{\theta})$ in terms of $\pi(\boldsymbol{\theta})$. Therefore, after marginalize $V_{\pi_\theta}(s_t)$ over the action space $\mathcal{A}$ and applying Baye's rule we have:

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &= \nabla V_{\pi_\theta}(s_t) \\ &= \nabla \sum_{a_t\in\mathcal{A}} \pi(a_t|s_t; \boldsymbol{\theta}) Q_{\pi_\theta}(s_t, a_t)\end{aligned} \tag{3}$$

The problem is that the right hand side of equation 3 depends on both the action selections and the distribution of states in which those selections are made, and that both of these are affected by the policy parameter. Fortunately, with the aid of the *Policy Gradient Theorem* we can reformulate $\nabla J(\boldsymbol{\theta})$ to be proportional to $\nabla\pi(a_t|s_t; \boldsymbol{\theta})$[5]:

ditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it, i.e. $P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1) = P(s_{t+1}|s_t)$.

[2]This might sound counterintuitive. However, think of it as when you have to make less rewarding decisions in the short term (e.g. watching Netflix instead of studying) in order to achieve a greater long-term goal (e.g. approving an exam).

[3]A value function may still be used to learn the policy parameter, but is not required for action selection.

[4]The definition and explanation of the continuous case is out of the scope of this work.

[5]A complete proof to this result can be found in page 325 of [4].

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

where $\mathbb{E}_{s\sim\mu_\pi,a\sim\pi_\theta}$ is the expectation of both state and action distributions following the policy $\pi_{\boldsymbol{\theta}}$; A_t and S_t are random variables ($A_t \sim \mu_\pi$ and $S_t \sim \pi_\theta$); and $Q_{\pi_{\boldsymbol{\theta}}}(S_t,A_t)$ can be replaced by G_t since $\mathbb{E}[G_t|S_t,A_t] = Q_{\pi_{\boldsymbol{\theta}}}(S_t,A_t)$.

Therefore, our learning rule defined in 2 can be updated to:

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

However, the *Policy Gradient Theorem* can be generalized to include a comparison of the action-value function to an arbitrary baseline $b(s)$ [4], which can be any function, even a random variable, as long as it does not depend on the action a:

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

the last equation remains valid because the subtracted quantity is zero:

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

Finally, we can use this to derive a new update rule that includes a general baseline:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - b(S_t))\nabla\log\pi(A_t|S_t;\boldsymbol{\theta}) \qquad (4)$$

IV. Actor-Critic

Equation 4 give us a rule to update the policy parameters $\boldsymbol{\theta}$ on every time step. However, this rule is incomplete since we don't know the value of the return G_t. But, what if we express G_t in something that depends on the state, so we can learn it in parallel with the policy $\pi_{\boldsymbol{\theta}}$? Fortunately, $G_t \approx V_\pi(s_t)$ (since recall that $V_\pi(s) := \mathbb{E}_\pi[G_t|s_t = s]$).

However, if we just replace G_t by a parameterized approximation of the state-value function (e.g. $V(s;\boldsymbol{w})$), we would not be using the feedback (reward) given by the environment, and hence we would hardly learn something meaningful. Instead, we could use the identity $G_t = R_{t+1} + \gamma G_{t+1}$ and replace G_{t+1} with $V(S_{t+1};\boldsymbol{w})$, so we can actually use the observed reward R_{t+1} on every time step. Hence, the update rule ends as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1};\boldsymbol{w}) - b(S_t))\nabla\log\pi(A_t|S_t;\boldsymbol{\theta}) \quad (5)$$

Equation 5 shows way this method is call *Actor-Critic*. Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the parameters $\boldsymbol{w}$ of the state-value[6] function $V(s;\boldsymbol{w})$ and measures (criticize) how good the taken action is.
- **Actor** updates the parameters $\boldsymbol{\theta}$ of the policy $\pi(a|s;\boldsymbol{\theta})$ function, in the direction suggested by the critic.

Equation 5, give us a rule that we can finally implement and use. However, there are still one last detail, the definition of the baseline $b(s)$. In general, the baseline 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. Therefore, one natural choice for the baseline is an estimate of the state-value function, $V(S_t;\boldsymbol{w})$ [4]. Hence, the update rule ends as:

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

But, what about the update rule for the state-value function $V(S;\boldsymbol{w})$ parameters $\boldsymbol{w}$? As we did with the policy, we need to start by defining an objective function $E(\boldsymbol{w})$. Since $V : \mathcal{S} \to \mathbb{R}$ (regression problem), we can just use the *Squared Error* as a loss function (i.e. $E(\boldsymbol{w}) := (V_\pi(S) - V(S;\boldsymbol{w}))^2$), and then optimize the approximator parameters $\boldsymbol{w}$ while minimizing it. Therefore:

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

[6] Depending on the algorithm, the Critic could be instead the action-value $Q(a|s;\boldsymbol{w})$ function.

Therefore, the update rule for $\boldsymbol{w}$ is:

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

A pseudocode for an algorithm which approximate both the policy and the state-value function is shown in 1.

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

Algorithm 1: Advantage Actor Critic pseudocode [4].

The *Advantage Actor Critic* showed in algorithm 1 was named after the *Advantage* function $A(S_t, A_t) := R_{t+1} + \gamma V(S_{t+1}; \boldsymbol{w}) - V(S_t; \boldsymbol{w})$, which captures how better an action is compared to the others at a given state[7].

V. Trust region-based policy optimization

The Advantage Actor Critic pseudocode shown in algorithm 1 should in principle be able to approximate an optimal policy. However, in practice the training process is often unstable. One reason for this is the policy update. Recall that RL algorithms learn from data generated by the interaction between the environment and the agent (i.e. the policy). Therefore, a strong change in the policy means a strong change in the data distribution. Unlike supervised learning, where the learning and data are typically independent (the learning process can not mess the dataset), in RL the training and the data is strongly related. Therefore, 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.

One way to prevent instability in the training process is by adding regularization to the difference between subsequent policies. This can be achieved using the Kullback–Leibler divergence D_{KL} between the actions distribution of the previous policy $\pi_{\theta_{old}}$ and the new policy π_θ ([1] and [2]:

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

If the action space $\mathcal{A}$ is not continuous (i.e. discrete), then we can sum over all the actions instead of integrating.

To consider the difference between subsequent policies in our training, we need to transform our original optimization problem (maximize $J(\boldsymbol{\theta})$) into a constrained optimization problem:

$$\begin{aligned} \text{maximize} \quad & J(\boldsymbol{\theta}) \\ \text{s.t.} \quad & D_{KL}(\pi_{\theta_{old}}||\pi_\theta) \leq \eta \end{aligned} \tag{6}$$

where $\eta > 0$ is the regularization hyperparameter (usually small) which specifies how much do we penalize the difference between subsequent policies.

Finally, we can approximate the constrained optimization problem 6 just by maximizing [5]:

$$J(\boldsymbol{\theta}) - \eta D_{KL}(\pi_{\theta_{old}}||\pi_\theta) \tag{7}$$

[7] In some sense, instead of having the critic to learn the state values, we make it learn the Advantage values. In that way, the evaluation of an action is based not only on how good the action is, but also how much better it can be.

References

[1] Vincent François-Lavet et al. "An Introduction to Deep Reinforcement Learning". In: *CoRR* abs/1811.12560 (2018). arXiv: 1811.12560. URL: http://arxiv.org/abs/1811.12560.

[2] Hado Van Hasselt. *Advanced Deep Learning & Reinforcement Learning*
Lecture 12: Policy Gradients and Actor Critics. https://youtu.be/bRfUxQs6xIM. [Online; accessed 19-Dec-2020]. 2018.

[3] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (Oct. 2017), pp. 354–. URL: http://dx.doi.org/10.1038/nature24270.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[5] Julien Vitay. "Natural Gradients". In: *julien-vitay.net/deeprl* (2018). URL: https://julien-vitay.net/deeprl/NaturalGradient.html.

[6] Lilian Weng. *A (Long) Peek into Reinforcement Learning*. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html. [Online; accessed 23-Dec-2020]. 2018.