THE REACHER ENVIRONMENT SOLVED USING A DEEP DETERMINISTIC POLICY GRADIENT ALGORITHM

JULIAN WERGIELUK

ABSTRACT. This short note provides a concise description of the model architecture and learning algorithms of the agent developed in this project. We also report learning performance of the agent and provide a list of possible future model improvements.

1. DESCRIPTION OF THE LEARNING ALGORITHM

1.1. **Summary of the notation.** Let's setup up the stage for the show and define a probability space $(\Omega, \mathscr{F}, \mathbb{P})$. Consider a Markov Decision Process (MDP) with measurable state and action spaces $(\mathbb{S}, \mathscr{S})$ and $(\mathbb{A}, \mathscr{A})$. A policy π is a Markov kernel from $(\mathbb{S}, \mathscr{S})$ to $(\mathbb{A}, \mathscr{A})$, i.e. for each $s \in \mathbb{S}$, $\pi(. \mid s)$ is a probability distribution on $\mathscr{A}$. A trajectory

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots)$$

encodes a sequence of states, actions and resulting numerical rewards pertaining to a policy π. The elements of τ are random variables, such that S_0 has the *start-state distribution* ρ_0 (which does not depend on the policy π), $\pi(.|s)$ is the conditional distribution of A_i given $S_i = s$ for any $s \in \mathbb{S}$, and $\mathbb{P}(.|S_{i-1} = s, A_{i-1} = a)$ is the conditional distribution of S_i. The rewards R_i are random variables taking values in $\mathbb{R}$. The return or discounted future reward G_t at $i \in \mathbb{N}$ is defined as

$$G_i = \sum_{k=0}^{\infty} \gamma^k R_{i+k+1}, \tag{1}$$

where γ is a hyperparameter from the interval $(0, 1)$, used to guarantee the convergence of the series (1).

Since τ is a sequence of random variables, and therefore itself a random variable taking values in the trajectory space

$$\mathbb{T} = \mathbb{S} \times \mathbb{A} \times (\mathbb{S} \times \mathbb{A} \times \mathbb{R})^{\infty},$$

we can sample from τ to obtain sequences of the form

$$(s_0, a_0, r_1, s_1, a_1, r_2, \ldots) \in \mathbb{T}.$$

These samples of τ are called *episodes* or *rollouts*.

The central object of interest is the mechanics of the world encoded in the transition probability measure $\mathbb{P}(.\mid s,a)$. For $i>0$, it is the distribution of S_i under the conditions $S_{i-1}=s$ and $A_{i-1}=a$ for fixed $s\in\mathbb{S}$ and $a\in\mathbb{A}$. This distribution does not depend on i, and, more importantly, does not depend on the policy π.

To simplify the setup, we assume that the reward R_i at time i is a deterministic function of the relevant states and actions. Depending on the problem at hand these are the three common choices encountered in the literature:

$$\begin{aligned} R_i &= \phi_1(S_i),\\ R_i &= \phi_2(A_{i-1},S_i),\\ R_i &= \phi_3(S_{i-1},A_{i-1},S_i) \end{aligned}$$

ϕ_1 is probably the most frequently used form.

In reinforcement learning, we parametrize the policy π_θ with an $\mathbb{R}^d$ vector θ, $d\geq 1$. The goal is to find good values for θ which lead to high cumulative expected reward

$$J(\theta)=\mathbb{E}_\theta\left[G_0\right]=\mathbb{E}_\theta\left[\sum_{k=0}^{\infty}\gamma^k R_{k+1}\right].$$

The subscript θ next to the expectation operator indicates that the underlying probability measure $\mathbb{P}_\theta$ combines both $\mathbb{P}$ and π_θ.

1.2. **Value functions.** The *state-value function* is defined for each state $s\in\mathbb{S}$ as

$$V_\theta(s)=\mathbb{E}_\theta\left[G_0|S_0=s\right]=\mathbb{E}_\theta\left[\sum_{k=0}^{\infty}\gamma^k R_{k+1}|S_0=s\right]. \tag{2}$$

The *action-value function* is defined for each state-action pair $(s,a)\in\mathbb{S}\times\mathbb{A}$ as

$$Q_\theta(s,a)=\mathbb{E}_\theta\left[G_0|S_0=s,A_0=a\right]=\mathbb{E}_\theta\left[\sum_{k=0}^{\infty}\gamma^k R_{k+1}|S_0=s,A_0=a\right]. \tag{3}$$

The *advantage function* is the difference

$$A_\theta(s,a)=V_\theta(s)-Q_\theta(s,a). \tag{4}$$

1.3. **Episodic and perpetual tasks.** Reinforcement learning tasks can be partitioned into two categories, depending on whether the task can be completed. A task is called *episodic* if there is an end of the task, i.e. some notion of “being done”. If there is no such notion, the task is called *perpetual* [1]. Typical examples of

[1]In the literature, people use the adjective “continuous”. But we already have “continuous spaces”, “continuous time”, “continuous function”, etc. The word “perpetual” is much better at conveying the property of running indefinitely long.

episodic tasks are games. Most board games like chess and go, and computer games fall into this category.

It is important to note, that many episodic tasks can also run forever. For example, in chess, we can design a policy that "runs in circles" and does not pursue the goal of winning the game. Letting such a policy play against itself will result in an endless game.

What distinguishes an episodic task from a perpetual one, is the existence of a non-empty set of end states Δ in $\mathbb{S}$, aptly called the *coffin space*, for which we require that $\mathbb{P}(S_{i+1} = S_i | S_i \in \Delta) = 1$ and $\mathbb{P}(R_i = 0 | S_i \in \Delta) = 1$. This ensures that all the states in Δ are absorbing and that no additional reward can be accumulated after the task has been completed.

1.4. Categorial and continuous action spaces.

The cardinality of the action space has a huge impact on the development of the theory of policy gradients as well as on the design and implementation of the policy optimization algorithms. Essentially, there are two cases we need to deal with. First, if the action space has finite number of elements, we call it *categorial*. Finite number of possible actions implies that the policy $\pi(.|s)$ evaluated at the state s, can be identified with a vector with $|\mathbb{A}|$ number of elements. Also, it is convenient to define the likelihood function f of the policy π as

$$f(a|s) = \pi(\{a\}\,|s),\ a \in \mathbb{A}.$$

$f(.|s)$ is the probability function of the probability measure $\pi(.|s)$.

The second case encountered in the literature is the *continuous* action space $\mathbb{A}$ which is a "nice" subset of $\mathbb{R}^{n_{\mathbb{A}}}$. Typical example of a continuous action space is a product of finite or infinite intervals in $\mathbb{R}^{n_{\mathbb{A}}}$. Let us assume that, for each state s, the measure $\pi(.|s)$ has a density $f(.|s)$, i.e. for an action set $\eta \in \mathscr{A}$ we have

$$\pi(\eta|s) = \int_{\eta} f(a|s)\,da.$$

As in the categorial case, we call $f(.|s)$ the likelihood function of the measure $\pi(.|s)$.

1.5. Deep deterministic policy gradients.

Deep deterministic policy gradient algorithm is an actor-critic type method with a deterministic actor policy π mapping states from $\mathbb{S}$ to actions in $\mathbb{A}$. During the training, the action-value function Q is optimized using a standard DQN algorithm as described in [3] employing a replay buffer and a separate target network.

The policy network is trained using a policy gradient derived from the action-value function approximation. Specifically, the parameter vector θ is given as $\theta = (\theta^Q, \theta^\pi)$, where θ^Q determines Q

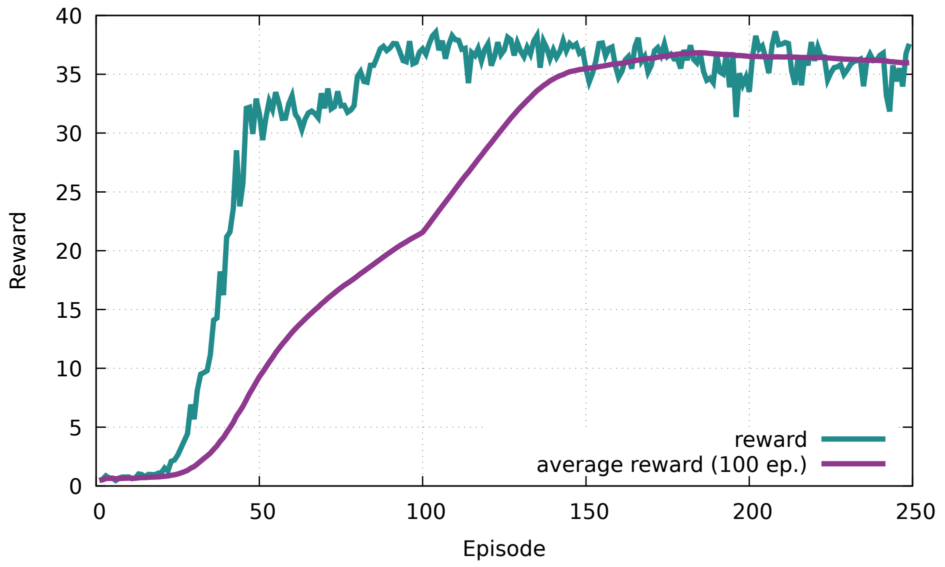

FIGURE 1. We train a DDPG agent for 250 episodes. The "reward" curve gives the average reward for all 20 arms at the end of each episode. The "average reward" curve is the rolling mean of the "reward" curve calculated over 100 episodes. (For episodes $1,\cdots,99$ we use an expanding window.)

and θ^π determines π. Let $J(\theta)$ be the cumulative expected reward. Then the policy gradient can be approximated by

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_a Q(S_t, \pi(A_t)) \nabla_{\theta^\pi} \pi(S_t)\right].$$

To improve the algorithm stability we use soft target updates for the parameters of both actor and critic networks, described in [3], and batch normalization.

2. TRAINING ANALYSIS

We train the DDPG agent for 250 episodes consisting of 1000 time steps each. The cumulative reward averaged over 100 episodes reaches the level of 30.23 after completing the episode 124.

3. ALGORITHM PARAMETRIZATION

The Actor network architecture. The actor is a feed-forward neural network mapping the state of the environment to a deterministic action. The network consists of three layers sandwiched with batch normalization. The final layer uses the tanh nonlinearity to produce an action vector in the cube $[-1,1]^4$.

```
Actor(
  (_pi_net): Sequential(
    (0): BatchNorm1d(33)
    (1): Linear(in_features=33, out_features=128)
    (2): BatchNorm1d(128)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=128)
    (5): BatchNorm1d(128)
    (6): ReLU()
    (7): Linear(in_features=128, out_features=4)
    (8): Tanh()
  )
)
```

The Critic network architecture. The critic is a feed-forward neural network approximating the state-action value function Q mapping state-action pairs to the expected cumulative reward. The critic network consists of two subnetworks: a one-layer network combined with batch normalization that maps the state to a high-dimensional state representation, and a two-layer network that maps the aforementioned representation of the state concatenated with the action vector to the approximation of the state-action value $Q(s,a)$.

```
Critic(
  (_q_state_net): Sequential(
    (0): BatchNorm1d(33)
    (1): Linear(in_features=33, out_features=128)
    (2): ReLU()
  )
  (_q_net): Sequential(
    (0): Linear(in_features=132, out_features=128)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=1)
  )
)
```

The list of hyperparameters used by the agent is listed in the Table 1.

4. Ideas for future work

- Investigation whether the OU process generating the exploratory behavior of the actor could be replaced with a simple Gaussian noise.
- Experimenting with bigger networks.
- Actor and critic could use a common network for state processing.
- Performance comparison with other continuous control policy gradient algorithms, like, e.g., PPO.

TABLE 1. List of hyperparameters and their values

Hyperparameter	Variable name	Value
Replay buffer size	`BUFFER_SIZE`	1e6
Batch size	`BATCH_SIZE`	256
Discount factor (γ)	`GAMMA`	0.99
Tau	`TAU`	1e-3
Actor learning rate	`LR_ACTOR`	0.001
Critic learning rate	`LR_CRITIC`	0.001
Initial value of the OU process scaling factor ε	`EPSILON`	1.0
OU mean reversion level		0.0
OU mean reversion speed		0.0
OU volatility (sigma)		0.05
Learning frequency		every 20 time steps

REFERENCES

[1] Erhan Çinlar. *Probability and Stochastics*. 1st ed. Springer Verlag, 2011, p. 572. ISBN: 0387878580.

[2] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: `1509.02971 [cs.LG]`.

[3] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836.

Email address: `julian.wergieluk@risklab.com`