

# Applications of Model-Free Deep Reinforcement Learning

Johanni Brea

16 April 2024

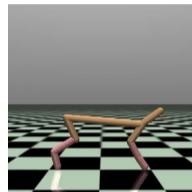
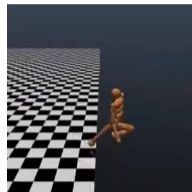
Artificial Neural Networks CS-456

# Deep Reinforcement Learning Applications

## Video Games

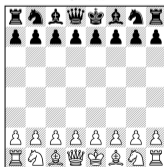


## Simulated Robotics



## Board Games (next week)

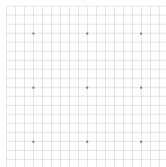
Chess



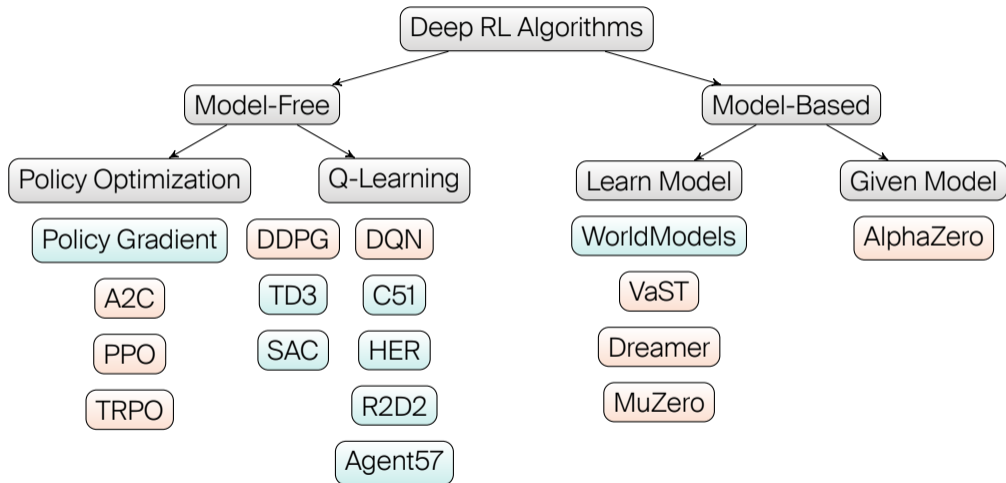
Shogi



Go



# A Classification of Deep Reinforcement Learning Methods



inspired by [https://spinningup.openai.com/en/latest/spinningup/rl\\_intro2.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html)

# Table of Contents

## 1. Mini-Batches in On- and Off-Policy Deep Reinforcement Learning

Temporally Correlated Weight Updates Can Cause Instabilities

Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)

Pros and Cons of On- and Off-Policy Deep RL

## 2. Deep Reinforcement Learning for Continuous Control.

Deep Deterministic Policy Gradient (DDPG)

Proximal Policy Optimization

Comparison of Algorithms in Simulated Robotics

# Mini-Batches in On- and Off-Policy Deep RL

Usually we train deep neural networks with independent and identically distributed (iid) mini-batches of training data.

## In this section you will learn

1. that we should not form mini-batches from sequentially acquired data in RL, but
2. use a **replay buffer** from which one can sample iid, or
3. run **multiple actors in parallel**.

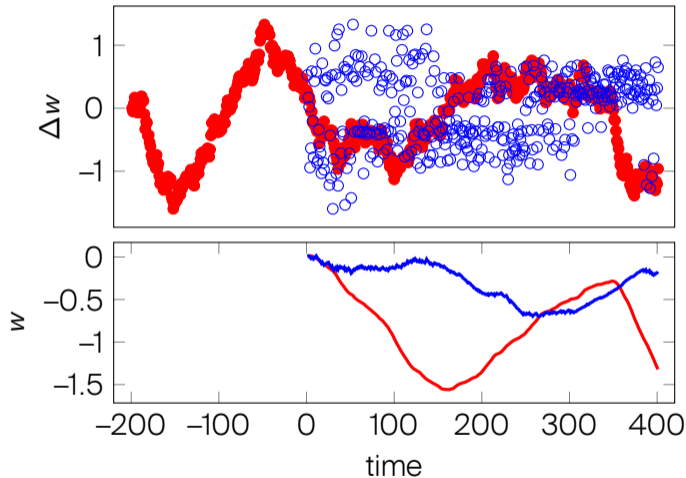
Suggested reading: [Mnih et al., 2015] and [Mnih et al., 2016]  
See <https://gymnasium.farama.org> for many environments.

# Correlation of Subsequent Observations in RL



- ▶ Subsequent images are highly correlated.
- ▶ Images at the end of the episode may look quite differently from those at the beginning of the episode.
- ▶ In image classification we shuffle the training data to have approximately independent and identically distributed mini-batches.

# Temporally Correlated Weight Updates Can Cause Instabilities

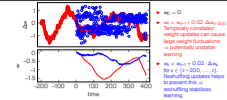


▶  $w_0 = 0$ .

▶  $w_t = w_{t-1} + 0.02 \cdot \Delta w_{t-200}$   
Temporally correlated weight updates can cause large weight fluctuations  $\Rightarrow$  potentially unstable learning.

▶  $w_t = w_{t-1} + 0.02 \cdot \Delta w_s$   
for  $s \in \{t-200, \dots, t\}$ .  
Reshuffling updates helps to prevent this  $\Rightarrow$  reshuffling stabilizes learning.

## Notes



Let us look at a simple example that illustrates why correlated samples can be problematic. You can think of  $w_t$  as a single weight in a deep neural network that is updated with gradient descent. The weight updates in this example are temporally correlated, e.g. most  $\Delta w_t$  for  $t < -50$  are negative. As a result,  $w_t$  moves to a strongly negative value within the first 150 updates and then up again. If we reshuffle the data – the blue points are obtained by sampling a  $\Delta w_s$  with  $s \in \{t - 200, t - 199, \dots, t\}$  – gradient descent never moves  $w_t$  to strongly negative values (blue curve in lower plot). The strong fluctuations of the weights during training may not be a big problem in supervised learning. But in deep reinforcement learning the policy depends on the weights of the neural network and therefore the samples that are obtained from interactions with the environment also depend on the weights of the neural network. In this example, the policy at time 150 or 400 for the reshuffled samples may be very different from the policy obtained with correlated samples and further data obtained with these two policies may differ.



# Proposed Solutions for Deep RL

On-policy methods, like policy gradient or SARSA, attempt to improve the policy that is used to make decisions, whereas off-policy methods, like Q-Learning, improve a policy different from that used to generate the data.

## Off-Policy Deep RL

e.g. DQN

- ▶ Put many (e.g. 1M) experiences (observation, action, reward) of a single agent into **replay buffer** (a first-in-first-out memory buffer).
- ▶ Randomly sample from the replay buffer to obtain mini-batches for training.

## On-Policy Deep RL

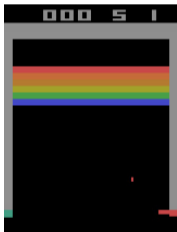
e.g. A2C

- ▶ Run **multiple agents** (e.g. 16) and environment simulations with different random seeds in parallel (ideally, every agents sees a different observation at any moment in time)
- ▶ Obtain mini-batches from the observations, actions and rewards of the multiple actors.

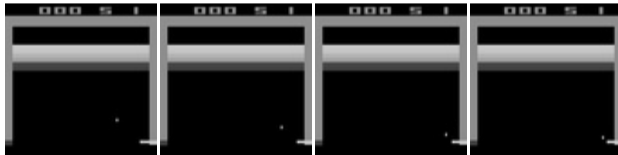
# Deep Q-Network (DQN) for Atari Games

## Input Encoding

4-frames color video

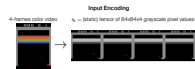


$s_t =$  (static) tensor of 84x84x4 grayscale pixel values



[Mnih et al., 2015]

## Notes



For Atari games 4 subsequent frames are taken as input to a convolutional neural network. The “color channel” of a convolutional layer is used in a creative way here: instead of representing RGB values it is used to represent the input at different time points. This allows the network to extract e.g. the direction of motion of a moving object.

Giving just a stack of raw gray-scale images as input is in the spirit of end-to-end learning: there is no sophisticated feature engineering involved (beyond the implicit inductive bias that color information is irrelevant and that all relevant state information is contained in 4 subsequent frames). There were, however, also attempts to engineer features and use shallow (i.e. standard tabular RL) for Atari games, see e.g. [Liang et al., 2015].

# Deep Q-Network (DQN) for Atari Games

# Deep Q-Network (DQN) for Atari Games

- 1: Initialize neural network  $Q_\theta$  and empty replay buffer  $R$ .
- 2: Set target  $\hat{Q} \leftarrow Q_\theta$ , counter  $t \leftarrow 0$ , observe  $s_0$ .
- 3: **repeat**
- 4:     Take action  $a_t$  and observe reward  $r_t$  and next state  $s_{t+1}$
- 5:     Store  $(s_t, a_t, r_t, s_{t+1})$  in  $R$
- 6:     Sample random minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $R$
- 7:     Update  $\theta$  with gradient of  $\mathcal{L}(\theta) = \sum_j (r_j + \gamma \max_{a'} \hat{Q}(s_{j+1})_{a'} - Q_\theta(s_j)_{a_j})^2$
- 8:     Increment  $t$  and reset  $\hat{Q} \leftarrow Q_\theta$  every  $C$  steps.
- 9: **until** some termination criterion is met.
- 10: **return**  $Q_\theta$

[Mnih et al., 2015]

## Notes

```

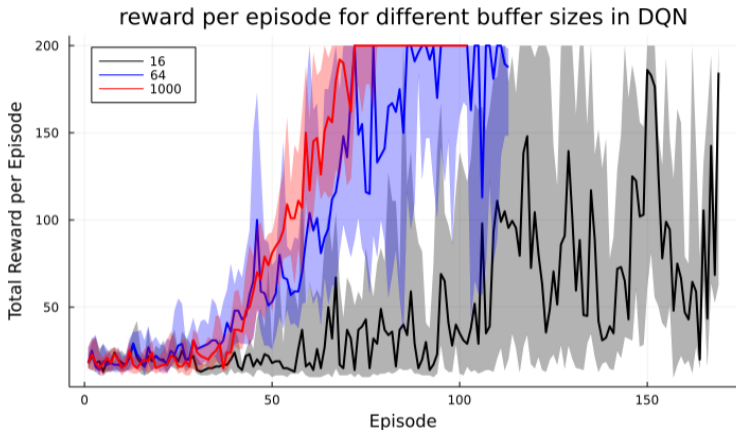
1: Initialize neural network  $Q_\theta$  and empty replay buffer  $R$ .
2: Set target  $\hat{Q} \leftarrow Q_\theta$ , counter  $t \leftarrow 0$ , observe  $s_0$ .
3: repeat
4:   Take action  $a_t$  and observe reward  $r_t$  and next state  $s_{t+1}$ .
5:   State  $(s_t, a_t, r_t, s_{t+1})$  in  $R$ .
6:   Sample random minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $R$ .
7:   Update  $\theta$  with gradient of  $L(\theta) = \sum_j (\gamma + \max_a Q(s_{j+1}, a) - Q_\theta(s_j, a))^2$ .
8:   Increment  $t$  and meet  $\hat{Q} \leftarrow Q_\theta$  every  $C$  steps.
9: until some termination criterion is met.
10: return  $Q_\theta$ 

```

[Mnih et al., 2015]

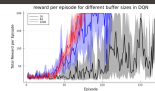
- 1: For  $n_s$  dimensional input (e.g. 84x84x4 pixels) the neural network  $Q_\theta$  has  $n_s$  input neurons and  $n_a$  (linear) output neurons.
- 2:  $\hat{Q}$  is the target network.
- 4: For  $\epsilon$ -greedy policy, state  $s_t$ , action  $a_t$  is  $\arg \max_a Q_\theta(s_t)_a$  with probability  $1 - \epsilon$  or a randomly chosen action with probability  $\epsilon$ . Typically,  $\epsilon$  is decreased over the course of learning, e.g. from 1 to 0.1 over the first million transitions (**exploration annealing**).
- 6: A minibatch can contain e.g. 32 transitions sampled randomly from  $R$ . Minibatches can be sampled uniformly from the replay buffer [Mnih et al., 2015] or use **prioritized replay** that favours transitions with a large TD-error [Schaul et al., 2015].
- 7: The sum runs over the indices of the sampled minibatch. To avoid large updates of the gradient (and thereby stabilize training) one can use gradient clipping, or losses like the Huber loss,  $L(x) = \frac{1}{2}x^2$  if  $|x| < 1$  and  $L(x) = |x| - \frac{1}{2}$  otherwise.
- 8: The update frequency can be quite low, e.g.  $C = 10'000$ .

# DQN on CartPole



learning to balance a cart pole with 20 different random seeds; lines = median of the reward per episode; shaded area = range between 10th and 90th percentile.

# Notes



Learning to balance a cart-pole with 20 different random seeds. Blue = median of the reward per episode. Shaded area = range between 10th and 90th percentile.

- Changing only the size of the replay buffer has a strong impact on learning in DQN.
- In the CartPole task a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright.
- With a replay buffer length of 16 we have basically online learning. We observe strong fluctuations in the performance, probably because of the correlated samples problem. With a much lower learning rate these fluctuations may become less problematic, but learning would be slower.
- With a replay buffer length of 1000 all 20 random seeds that were tested resulted in fast and reliable learning of the task.



# Importance Sampling

# Prioritized Replay

Instead of uniformly sampling (i.e.  $p_i = 1/|R|$ ) from the replay buffer, sample transition with probability

$$q_i = \frac{\tilde{q}_i^\alpha}{\sum_k \tilde{q}_k^\alpha}, \text{ where } \tilde{q}_i = |\delta_i| + \epsilon \quad (1)$$

with TD-error  $\delta_i = R_i + \gamma \max_a Q(s_{i+1}, a) - Q(s_i, a)$ .

$\alpha = 0$  corresponds to uniform sampling and  $\epsilon > 0$  prevents  $q_i$  from being 0 for transitions with zero TD-error.

Correction for non-uniform sampling with importance weight  $\frac{p_i}{q_i} = \frac{1}{|R|q_i}$ , i.e.

$$\Delta\theta \propto \frac{1}{q_i} \nabla \mathcal{L}(\theta).$$

[Schaul et al., 2015]

Instead of uniform sampling (i.e.  $p_i = 1/|R|$ ) from the replay buffer, sample transition with probability

$$p_i = \frac{q_i^\beta}{\sum_k q_k^\beta} \text{ where } q_i = |\delta_i| + \epsilon$$

with TD-error  $\delta_i = R_i + \gamma \max_{a'} Q(s_{i+1}, a') - Q(s_i, a)$   
 $\epsilon = 0$  corresponds to uniform sampling and  $\epsilon > 0$  prevents  $q_i$  from being 0 for transitions with zero TD-error

Correction for non-uniform sampling with importance weight  $w_i = \frac{1}{p_i C}$ , i.e.

$$\Delta J \approx \frac{1}{C} \sum_k \nabla_{\theta} J(\theta)$$

## Notes

Sampling uniformly from the replay buffer may not be efficient, if most Q-values do not change much. For example, if rewards are sparse, it would be more efficient to sample and update always the transitions with a large TD-error, i.e. first the states that lead directly to reward, than those that lead to reward in 1 step, than those that lead to reward in 2 steps, etc.

For sampling from the replay buffer the correct distribution  $p_i = 1/|R|$  is the uniform distribution. If we sample instead from  $q_i$  we need to correct by  $1/(|R|q_i)$ .

Side-note: in [Schaul et al., 2015], they used the “importance weight”  $w_i = \left(\frac{1}{|R|q_i}\right)^\beta / C$  where  $C = \max_k w_k$ ,  $\beta$  controls the importances correction. For  $\beta < 1$  the estimate of the gradient is biased (but the variance may be smaller). The importance correction is usually annealed towards  $\beta \rightarrow 1$  over the course of training.

# Directly Parametrizing Policy and Value

# Advantage Actor-Critic (A2C)

- 1: Initialize neural networks  $\pi_\theta$  and  $V_\phi$ .
- 2: Set counter  $t \leftarrow 0$ , observe  $s_0$ .
- 3: **repeat**
- 4:     **for all** workers  $k = 1, \dots, K$  **do**
- 5:         Take action  $a_t^{(k)}$  and observe reward  $r_t^{(k)}$  and next state  $s_{t+1}^{(k)}$
- 6:         Compute  $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$  and advantage  $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$
- 7:     **end for**
- 8:     Update  $\theta$  with gradient of  $\sum_k A_t^{(k)} \log \pi_\theta(a_t^{(k)}; s_t^{(k)})$
- 9:     Update  $\phi$  with gradient of  $\sum_k (R_t^{(k)} - V_\phi(s_t^{(k)}))^2$ .
- 10:     Increment  $t$ .
- 11: **until** some termination criterion is met.
- 12: **return**  $\pi_\theta$  and  $V_\phi$

## Notes

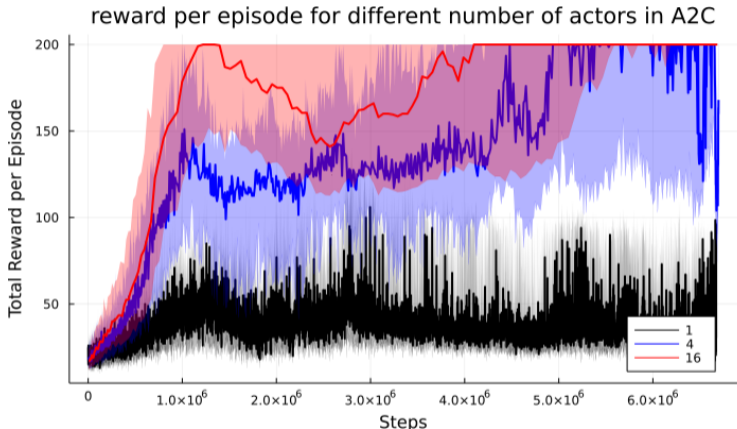
```

1: Initialize neural networks  $\pi_\theta$  and  $V_\phi$ .
2: Set counter  $t \leftarrow 0$ , observe  $s_0$ .
3: repeat
4:   for all workers  $k = 1, \dots, K$  do
5:     Take action  $A_t^{(k)}$  and observe reward  $r_t^{(k)}$  and next state  $s_{t+1}^{(k)}$ 
6:     Compute  $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$  and advantage  $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$ 
7:   end for
8:   Update  $\theta$  with gradient of  $\sum_k A_t^{(k)} \log \pi_\theta(s_t^{(k)} | s_t^{(k)})$ 
9:   Update  $\phi$  with gradient of  $\sum_k (R_t^{(k)} - V_\phi(s_t^{(k)}))^2$ 
10:  Increment  $t$ .
11: until some termination criterion is met.
12: return  $\pi_\theta$  and  $V_\phi$ 

```

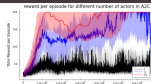
- 1: For  $n_s$ -dimensional input and  $n_a$  action, the policy network  $\pi_\theta$  has  $n_s$  inputs and as output a softmax layer with  $n_a$  units. The value network  $V_\phi$  has  $n_s$  inputs and one linear output.
  - 4: Each worker runs its own and independent copy of the environment, e.g. each one runs an emulator of an Atari video game. The different workers can e.g. run on different threads of a CPU.
  - 5-6: The pseudo-code shows a version with a one-step advantage. Alternatively, each worker can run  $n$  actions and observe the rewards and next states and compute an  $n$ -step advantage
 
$$A_t^{(k)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^{i+1} V_\phi(s_{t+n}^{(k)}).$$
- A3C stands for Asynchronous Advantage Actor-Critic where each worker computes the gradient for  $\theta$  and  $\phi$  and the parameters are updated asynchronously. However, the asynchronous updates do not seem to be crucial; the synchronous version (A2C) performs equally well.
  - A3C/A2C can be turned into an off-policy method with a replay buffer (see e.g. ACER <https://arxiv.org/abs/1611.01224>)

# A2C on CartPole



learning to balance a cart pole with 20 different random seeds; lines = median of the reward per step;  
shaded area = range between 10th and 90th percentile.

## Notes



Learning to balance a cart pole with 20 different random seeds. Lines = median of the reward per step. Shaded area = range between 5th and 95th percentile.

- Changing the number of workers has a strong impact on learning in A2C.
- With only 1 worker policy gradient does not succeed at all, although we used 32 times as many interactions with the environment as in DQN. Maybe a smaller learning rate would help in this setting.
- With 16 actors all 20 random seeds that were tested resulted in learning the task. Gradient clipping was used to prevent large parameter updates towards the end of training.



# Pros and Cons of On- and Off-Policy Deep RL

## Off-Policy Deep RL

---

- + lower sample complexity  
i.e. fewer interactions with the environment are needed, because experiences in the replay buffer can be used multiple times
- higher memory complexity  
need to store many experiences in the replay buffer.

## On-Policy Deep RL

---

- higher sample complexity  
because old experiences cannot be used to update a policy that has already changed.
- + lower memory complexity  
only the current observations, actions and rewards of the parallel agents are kept to update the policy.

# Quiz

Which statement is correct?

- In A2C, if all parallel workers  $K$  start together in the first step of the episode and every episode has the same length, we do not get the desired effect of iid minibatches.
- We could use multiple actors instead of a replay memory with Q-Learning.
- If we use the SARSA loss  $r_j + \hat{Q}(s_{j+1}, a_{j+1}) - Q_\theta(s_j, a_j)$  in the DQN algorithm, we just need to include also the next action  $a_{j+1}$  in the replay buffer and everything will work.

# Table of Contents

## 1. Mini-Batches in On- and Off-Policy Deep Reinforcement Learning

Temporally Correlated Weight Updates Can Cause Instabilities

Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)

Pros and Cons of On- and Off-Policy Deep RL

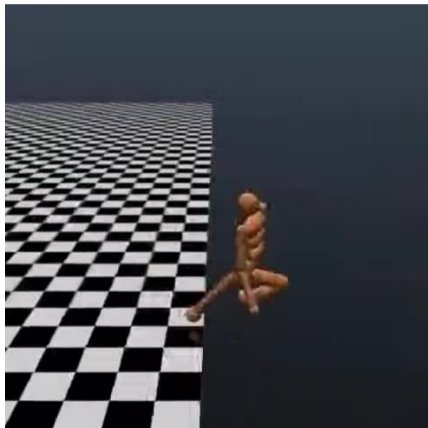
## 2. Deep Reinforcement Learning for Continuous Control.

Deep Deterministic Policy Gradient (DDPG)

Proximal Policy Optimization

Comparison of Algorithms in Simulated Robotics

# Deep Reinforcement Learning for Continuous Control



- ▶ High-dimensional continuous action spaces (e.g. forces and torques). and observation spaces (e.g. positions, angles and velocities).
- ▶ Standard policy gradient could be applied, but it is difficult to find hyper-parameter settings such that learning is neither unstable nor very slow.
- ▶ Standard DQN cannot be applied, because it is designed for discrete actions.

## In this section you will learn about

1. proximal policy optimization (PPO) methods that improve standard policy gradient methods and
2. an adaptation of DQN to continuous action spaces (DDPG).

Suggested reading:

[Kakade and Langford, 2002, Schulman et al., 2015, Schulman et al., 2017, Lillicrap et al., 2015]

# Deep Deterministic Policy Gradient (DDPG)<sup>1</sup>

In DQN for discrete actions the Q-values for  $N_a$  actions are given as the activity of  $N_a$  output neurons of a neural network with parameters  $\theta$  and input given by the state  $s$ .

For continuous actions there are infinitely many values; obviously we do not want  $N_a = \infty$ .

Proposed solution: use a policy network  $\pi_\psi(s)$  that maps deterministically states  $s$  to continuous actions  $a$ .

---

<sup>1</sup>The name is confusing: DDPG is more closely related to DQN than to PG!

[Lillicrap et al., 2015]

# Deep Deterministic Policy Gradient (DDPG): Networks

# Deep Deterministic Policy Gradient (DDPG)

- 1: Initialize neural networks  $Q_\theta$ ,  $\pi_\psi$  and empty replay buffer  $R$ .
- 2: Set target  $\hat{Q} \leftarrow Q_\theta$ ,  $\hat{\pi} \leftarrow \pi_\psi$ , counter  $t \leftarrow 0$ , observe  $s_0$ .
- 3: **repeat**
- 4:     Take action  $a_t = \pi_\psi(s_t) + \epsilon$  and observe reward  $r_t$  and next state  $s_{t+1}$
- 5:     Store  $(s_t, a_t, r_t, s_{t+1})$  in  $R$
- 6:     Sample random minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $R$
- 7:     Update  $\theta$  with gradient of  $\sum_j (r_j + \hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1})) - Q_\theta(s_j, a_j))^2$
- 8:     Update  $\psi$  with gradient of  $\sum_j Q_\theta(s_j, \pi_\psi(s_j))$ .
- 9:     Increment  $t$  and reset  $\hat{Q} \leftarrow Q_\theta$ ,  $\hat{\pi} \leftarrow \pi_\psi$  every  $C$  steps.
- 10: **until** some termination criterion is met.
- 11: **return**  $Q_\theta$

[Lillicrap et al., 2015]

## Notes

```

1: Initialize neural networks  $Q_\theta$ ,  $\pi_\psi$  and empty replay buffer  $R$ .
2: Set target  $\hat{Q} \leftarrow Q_\theta$ ,  $\hat{\pi} \leftarrow \pi_\psi$ , counter  $t \leftarrow 0$ , observe  $s_0$ .
3: repeat
4:   Take action  $a_t = \pi_\psi(s_t) + \epsilon$  and observe reward  $r_t$  and next state  $s_{t+1}$ .
5:   Store  $(s_t, a_t, r_t, s_{t+1})$  in  $R$ .
6:   Sample random minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $R$ .
7:   Update  $Q$  with gradient of  $\sum_j (r_j + Q(s_{j+1}, \hat{\pi}(s_{j+1})) - Q_\theta(s_j, a_j))^2$ .
8:   Update  $\pi$  with gradient of  $\sum_j Q_\theta(s_j, \pi_\psi(s_j))$ .
9:   Increment  $t$  and next  $\hat{Q} \leftarrow Q_\theta$ ,  $\hat{\pi} \leftarrow \pi_\psi$  every  $C$  steps.
10: until some termination criteria is met.
11: return  $Q_\theta$ 

```

- 1: If the dimensionality of the state is  $n_s$  and the dimensionality of the action is  $n_a$  (e.g.  $n_a$  different joints of a robot; note: in contrast to the finite actions setting where we used  $n_a$  for the number of actions, it denotes the dimensionality of the action space here), the Q-network takes as input a  $n_s + n_a$ -dimensional vector and outputs a scalar number: the Q-value for this state and action. The policy network  $\pi_\psi$  takes states as input and is supposed to return the greedy action.
- 4: Here  $a_t$  is a  $n_a$ -dimensional vector and  $\epsilon$  is a random vector of  $n_a$  dimensions. This random vector (sampled e.g. independently at each time step from a multivariate Gaussian or, for temporally correlated exploration, from an Ornstein-Uhlenbeck process) drives exploration around the greedy action  $\pi_\psi(s_t)$ .
- 7: Note that the  $\max$  operation is not needed here, because  $\hat{\pi}(s_{j+1}) \approx \arg \max_a \hat{Q}(s_{j+1}, a)$  (see line 8).
- 8: The gradient ascent procedure in this line moves  $\psi$  such that  $\pi_\psi(s_j)$  moves closer to  $\arg \max_a Q_\theta(s_{j+1}, a)$ .



# How Big a Step Can We Make in Policy Gradient?

In simple Policy Gradient, the parameters  $\theta$  of a neural network change according to

$$\theta' = \theta + \alpha \nabla J(\theta)$$

$$J(\theta) = E_{s_0 \sim p(s_0)} [V_\theta(s_0)] = E_{s_t, a_t \sim p_\theta, \pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t R_{s_t \rightarrow s_{t+1}}^{a_t} \right] \quad (2)$$

$$= \sum_{t=0}^{\infty} \sum_{s_t, s_{t+1}, a_t} \gamma^t R_{s_t \rightarrow s_{t+1}}^{a_t} P_{s_t \rightarrow s_{t+1}}^{a_t} \pi_\theta(a_t; s_t) p_\theta(s_t) \quad (3)$$

$$\text{with } p_\theta(s_t) = \sum_{s_0, \dots, s_{t-1}, a_0, \dots, a_{t-1}} p(s_0) \prod_{\tau=0}^{t-1} P_{s_\tau \rightarrow s_{\tau+1}}^{a_\tau} \pi_\theta(a_\tau; s_\tau).$$

We actually want  $J(\theta') - J(\theta)$  to be as large as possible.

[Kakade and Langford, 2002, Schulman et al., 2015, Schulman et al., 2017]

## Notes

In simple Policy Gradient, the parameters  $\theta$  of a neural network change according to

$$\theta' := \theta + \alpha \nabla J(\theta)$$

$$J(\theta) = \mathbb{E}_{s_0, a_0 \sim p_0} [V(s_0)] = \mathbb{E}_{s_0, a_0 \sim p_0} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1}^{a_t} \right] \quad (2)$$

$$= \sum_{t=0}^{\infty} \sum_{s_t, a_t} \gamma^t R_{t+1}^{a_t} P_{t+1}^{a_t}(s_{t+1}; a_t, s_t) p_t(s_t) \quad (3)$$

$$\text{with } p_t(s_t) = \sum_{s_0, a_0, \dots, a_{t-1}} p(s_0) \prod_{\tau=0}^{t-1} P_{\tau+1}^{a_\tau}(s_{\tau+1}; a_\tau, s_\tau).$$

We actually want  $J(\theta') - J(\theta)$  to be as large as possible.

(Jaakkola and Langford, 2002; Schuman et al., 2005; Schuman et al., 2007)

- The objective function  $J(\theta)$  is the expected future discounted return given parameters  $\theta$  and distribution over initial states  $p(s_0)$ .
- The notation  $E_{s_t, a_t \sim p_\theta, \pi_\theta}$  is a short-hand for  $\sum_{s_0, s_1, \dots, a_0, a_1, \dots} p(s_0) \prod_{\tau} P_{s_\tau \rightarrow s_{\tau+1}}^{a_\tau} \pi_\theta(a_\tau; s_\tau)$  (replace sums by integrals for continuous spaces).
- To obtain the second line we interchange the sum signs, define  $p_\theta(s_t)$  (the policy-dependent probability of reaching state  $s_t$ ) and note that the reward  $R_{s_t \rightarrow s_{t+1}}^{a_t}$  does not depend on the future and therefore all sums over future states and actions ( $s_{t+2}, s_{t+3}, \dots, a_{t+1}, a_{t+2}, \dots$ ) evaluate to 1, because of the normalization of the probabilities.
- The optimal learning rate  $\alpha$  would be the one that maximizes  $J(\theta') - J(\theta) = J(\theta + \alpha \nabla J(\theta)) - J(\theta)$ .

# How Big a Step Can We Make in Policy Gradient?

$$J(\theta') - J(\theta) = J(\theta') - E_{s_0 \sim p(s_0)}[V_\theta(s_0)] \quad (4)$$

$$= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}[V_\theta(s_0)] \quad (5)$$

$$= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[ \sum_{t=0}^{\infty} \gamma^t V_\theta(s_t) - \sum_{t=1}^{\infty} \gamma^t V_\theta(s_t) \right] \quad (6)$$

$$= J(\theta') + E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[ \sum_{t=0}^{\infty} \gamma^t (\gamma V_\theta(s_{t+1}) - V_\theta(s_t)) \right] \quad (7)$$

$$= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[ \sum_{t=0}^{\infty} \gamma^t (R_{s_t \rightarrow s_{t+1}}^{a_t} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)) \right] \quad (8)$$

$$= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[ \sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t) \right] = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[ \frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)} \gamma^t A_\theta(s_t, a_t) \right] \quad (9)$$

<http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-9.pdf>

## Notes

$$\begin{aligned}
 A^{\theta'} - A^{\theta} &= A^{\theta'} - \mathbb{E}_{\pi_{\theta}}[V_{\theta}(s_t)] \\
 &= A^{\theta'} - \mathbb{E}_{\pi_{\theta}}[\sum_{s'} \gamma V_{\theta}(s')] \\
 &= A^{\theta'} - \mathbb{E}_{\pi_{\theta}}[\sum_{s'} \gamma V_{\theta}(s') - \sum_{s'} \gamma V_{\theta}(s')] \\
 &= A^{\theta'} + \mathbb{E}_{\pi_{\theta}}[\sum_{s'} \gamma (V_{\theta}(s_t) - V_{\theta}(s'))] \\
 &= \mathbb{E}_{\pi_{\theta}}[\sum_{s'} \gamma (R_{t+1} + \gamma V_{\theta}(s_{t+1}) - V_{\theta}(s_t))] \\
 &= \mathbb{E}_{\pi_{\theta}}[\sum_{s'} \gamma A_{\theta}(s_t)] = \sum_{s'} \mathbb{E}_{\pi_{\theta}}[\gamma A_{\theta}(s_t)]
 \end{aligned}$$

<http://rail.eecs.berkeley.edu/deep-learning/cs234c11/lec4/lec4-4>

- (3) We plug in the definition for the second term.
- (4) We take the expectation also over all future state action pairs (see previous slide for the definition of this notation). These expectations do not change the expression, because  $V_{\theta}(s_0)$  does not depend on future state-action pairs. (e.g.
- $$E_{X,Y}[X] = \sum_{X,Y} P(X,Y)X = \sum_{X,Y} P(X)P(Y|X)X = \sum_X P(X)X \sum_Y P(Y|X) = E_X[X].$$
- (5) We write  $V_{\theta}(s_0)$  as to difference of two infinite sums. Note that the second sum runs from  $t = 1$ , whereas the first one runs from  $t = 0$ .
- (6) We write the second sum as  $\sum_{t=1}^{\infty} \gamma^t V_{\theta}(s_t) = \sum_{t=0}^{\infty} \gamma \cdot \gamma^t V_{\theta}(s_{t+1})$  and swap the order of the two sums in the square bracket (note the change of sign in front of the second term).
- (7) We plug in the definition of the first term and get the advantage for  $\theta$  (note that the expectation is taken over  $p_{\theta'}, \pi_{\theta'}$ ).
- (8) We swap the sum with the expectation and take the expectation with respect to  $\pi_{\theta}$  while correcting with the importance weight  $\frac{\pi_{\theta'}(a_t; s_t)}{\pi_{\theta}(a_t; s_t)}$ .

# Proximal Policy Optimization: Idea

$$J(\theta') - J(\theta) = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta}} \left[ \underbrace{\frac{\pi_{\theta'}(a_t; s_t)}{\pi_{\theta}(a_t; s_t)}}_{= r_{\theta'}(s_t, a_t)} \gamma^t A_{\theta}(s_t, a_t) \right]$$

As long as  $p_{\theta'}$  is close to  $p_{\theta}$  such that

$$E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta}} [r_{\theta'}(s_t, a_t) \gamma^t A_{\theta}(s_t, a_t)] \approx E_{s_t, a_t \sim p_{\theta}, \pi_{\theta}} [r_{\theta'}(s_t, a_t) \gamma^t A_{\theta}(s_t, a_t)]$$

we can take the samples  $s_t, a_t \sim p_{\theta}, \pi_{\theta}$  obtained with the old policy and optimize the objective function

$$\hat{L}(\theta') = \sum_{t=0}^{\infty} r_{\theta'}(s_t, a_t) \gamma^t A_{\theta}(s_t, a_t)$$

## Notes

$$J(\theta') - J(\theta) = \sum_{s_t} \mathbb{E}_{\pi_{\theta'}(\cdot|s_t)} \left[ \frac{v_{\theta'}(s_t)}{v_{\theta}(s_t)} \right] A_{\theta'}(s_t, A_t)$$

As long as  $\theta'$  is close to  $\theta$ , such that  $\mathbb{E}_{\pi_{\theta'}(\cdot|s_t)} [v_{\theta'}(s_t)] \approx \mathbb{E}_{\pi_{\theta}(\cdot|s_t)} [v_{\theta}(s_t)]$ , we can take the samples  $s_t \sim \pi_{\theta}$  obtained with the old policy and optimize the objective function:

$$\hat{L}(\theta') = \sum_{s_t} \psi(s_t, \pi) A_{\theta'}(s_t, \pi)$$

There is still  $p_{\theta'}$  (the probability of reaching state  $s_t$  under policy  $\pi_{\theta'}$ ) in  $J(\theta') - J(\theta)$ . We cannot easily sample from this probability as long as we do not have  $\theta'$ . But we can sample from  $p_{\theta}$  and if  $p_{\theta'}$  is sufficiently close to  $p_{\theta}$ , we have approximately  $\hat{L}(\theta') \approx J(\theta') - J(\theta)$ .

Instead of searching for an optimal learning rate  $\alpha$ , the idea is now to optimize  $\hat{L}(\theta')$  for a few steps (with gradient ascent, ADAM, RMSProp, or similar) while making sure that  $p_{\theta'}$  does not move too far away from  $p_{\theta}$ , before taking further actions in the environment.

# Proximal Policy Optimization: Losses

$$\hat{L}(\theta') = \sum_{t=0}^{\infty} r_{\theta'}(s_t, a_t) \gamma^t A_{\theta}(s_t, a_t)$$

## Trust-Region Policy Optimization (TRPO)

Maximize  $\hat{L}(\theta')$  subject to  $\text{KL}[\pi_{\theta} \parallel \pi_{\theta'}] \leq \delta$ .

## Clipped Surrogate Objectives (PPO-CLIP)

Maximize  $\hat{L}^{\text{CLIP}}(\theta') = \sum_{t=0}^{\infty} \min(r_{\theta'} \gamma^t A_{\theta}, \text{clip}(r_{\theta'}, 1 - \epsilon, 1 + \epsilon) \gamma^t A_{\theta})$ .

## Notes

$$J(\theta') = \sum_{s \sim \pi_{\theta'}} v(s, A_s) / A_s(s, A_s)$$

Trust-Region Policy Optimization (TRPO)  
 Maximize  $J(\theta')$  subject to  $KL[\pi_{\theta'}] \leq \delta$ .

Clipped Surrogate Objective (PPO-CLIP)  
 Maximize  $J^{CLIP}(\theta') = \sum_{s \sim \pi_{\theta'}} \min(v(s, A_s), \text{clip}(v(s, A_s), 1 - \epsilon, 1 + \epsilon) / A_s)$

One way to keep  $p_{\theta'}$  close to  $p_{\theta}$  is to make sure the policy  $\pi_{\theta'}$  does not move far away from  $\pi_{\theta}$  by explicitly constraining the KL divergence from original policy to new policy to be smaller than  $\delta$ .

Another way is to clip the objective function such that the gradient becomes zero when  $r$  moves out of the interval  $[1 - \epsilon, 1 + \epsilon]$ . The clip function is defined as

$$\text{clip}(x, l, u) = \begin{cases} u & x > u \\ x & x \in [l, u] \\ l & x < l \end{cases}$$

Its derivative is 0 when  $x < l$  or  $x > u$ . See exercise 2 for details.

Analogy: PPO to policy gradient is a bit like Newton's trust region method to gradient descent; in both cases we approximate the loss function locally by a surrogate, and optimize this surrogate as long as we still trust the approximation.



# Proximal Policy Optimization

- 1: Initialize neural networks  $\pi_\theta$  and  $V_\phi$ .
- 2: Set counter  $t \leftarrow 0$ , observe  $s_0$ .
- 3: **repeat**
- 4:   **for all** workers  $k = 1, \dots, K$  **do**
- 5:     Take action  $a_t^{(k)}$  and observe reward  $r_t^{(k)}$  and next state  $s_{t+1}^{(k)}$
- 6:     Compute  $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$  and advantage  $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$
- 7:   **end for**
- 8:   Optimize surrogate objective  $\sum_k \min(r_{\theta'}^{(k)} A_t^{(k)}, \text{clip}(r_{\theta'}^{(k)}, 1 - \epsilon, 1 + \epsilon) A_t^{(k)})$  in  $\theta'$  with gradient ascent for  $M$  epochs.  $r_{\theta'}^{(k)} = \frac{\pi_{\theta'}(s_t^{(k)}, a_t^{(k)})}{\pi_\theta(s_t^{(k)}, a_t^{(k)})}$ .
- 9:   Update  $\phi$  with gradient of  $\sum_k (R_t^{(k)} - V_\phi(s_t^{(k)}))^2$ .
- 10:   Increment  $t$ .
- 11: **until** some termination criterion is met.
- 12: **return**  $\pi_\theta$  and  $V_\phi$

<https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>

## Notes

```

1. Initialize neural networks  $\pi_\theta$  and  $V_\phi$ .
2. Set counter  $k \leftarrow 0$ , observe  $s_0$ .
3. repeat
4.   for all workers  $i = 1, \dots, N$  do
5.     Take action  $a_i^{(k)}$  and observe reward  $r_i^{(k)}$  and next state  $s_{i+1}^{(k)}$ 
6.     Compute  $\delta_i^{(k)} = r_i^{(k)} + \gamma V_\phi(s_{i+1}^{(k)}) - V_\phi(s_i^{(k)})$  and advantage  $A_i^{(k)} = \delta_i^{(k)} - V_\phi(s_i^{(k)})$ 
7.   end for
8.   Optimize surrogate objective  $\sum_{i=1}^N \text{min}_{\theta'} \left( \frac{e^{\eta A_i^{(k)}} \pi_\theta(a_i^{(k)} | s_i^{(k)})}{\pi_\theta(a_i^{(k)} | s_i^{(k)})} (1 - \epsilon) + \epsilon V_\phi(s_i^{(k)}) \right)$  in  $\theta'$  with
   gradient ascent for  $M$  epochs:  $\theta^{(k+1)} = \underset{\theta}{\text{argmax}} \frac{\partial}{\partial \theta} \left( \sum_{i=1}^N \text{min}_{\theta'} \left( \frac{e^{\eta A_i^{(k)}} \pi_\theta(a_i^{(k)} | s_i^{(k)})}{\pi_\theta(a_i^{(k)} | s_i^{(k)})} (1 - \epsilon) + \epsilon V_\phi(s_i^{(k)}) \right) \right)$ 
9.   Update  $\phi$  with gradient of  $\sum_{i=1}^N (A_i^{(k)} - V_\phi(s_i^{(k)}))^2$ .
10.  Increment  $k$ .
11. until some termination criterion is met.
12. return  $\pi_\theta$  and  $V_\phi$ 

```

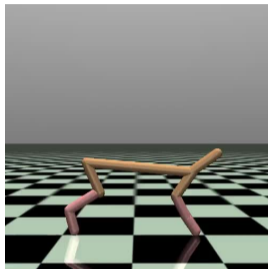
As in A2C, the workers send their experiences back to the learner before starting to optimize the surrogate objective in line 8. In contrast to A2C that does a single gradient ascent update step in line 8, PPO uses each observation multiple times to optimize the surrogate objective in line 8. Instead of the PPO-CLIP objective one can also use the TRPO objective.

# Quiz

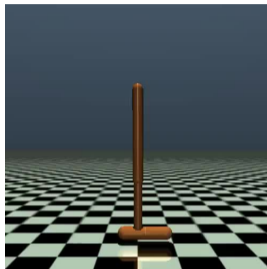
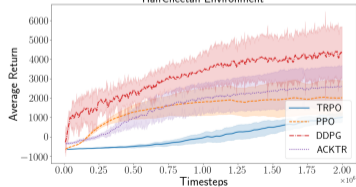
Which statement is correct?

- DDPG works only for continuous action spaces.
- PPO works only for continuous action spaces.
- In proximal policy optimization methods we want to keep the ratio  $r_{\theta'}(s_t, a_t) = \frac{\pi_{\theta'}(a_t; s_t)}{\pi_{\theta}(a_t; s_t)}$  close to one, such that the state visitation probabilities  $p_{\theta}(s_t)$  and  $p_{\theta'}(s_t)$  are roughly the same.
- With the update of policy gradient,  $\theta' = \theta + \alpha \nabla J(\theta)$  and a fixed learning rate  $\alpha$ ,  $J(\theta') - J(\theta)$  will always be positive.

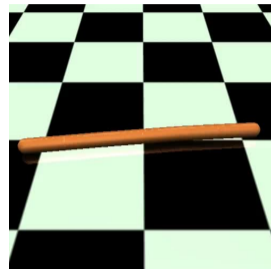
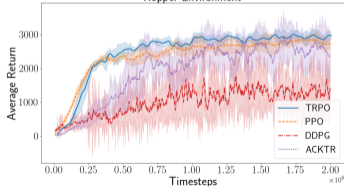
# Comparison of Algorithms in Simulated Robotics



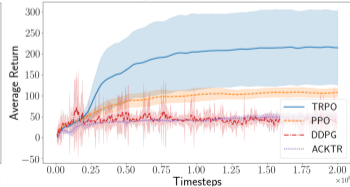
HalfCheetah Environment



Hopper Environment



Swimmer Environment



[Henderson et al., 2017]

Continuous Control (PPO & DDPG)

○○○○○○○○○○●○○

# Summary

- ▶ One can improve the stability and sample efficiency of policy gradient methods by maximizing in an inner loop a surrogate objective function, like the one of TRPO or PPO-CLIP.
- ▶ DQN can be adapted to domains with continuous actions by training an additional policy network  $\pi_{\psi}$  (DDPG).
- ▶ Which algorithm works best depends on the problem, usually.
- ▶ We did not discuss sufficient and efficient exploration, but it usually has a strong impact on the learning curve. A simple strategy for Policy Gradient methods is to add entropy regularization such that the policy does not become deterministic too quickly, but there are more advanced methods (see e.g. soft actor-critic SAC [Haarnoja et al., 2018]).

# Why Can't We Naively Use a Replay Buffer for On-Policy Methods?

Remember: for a given state  $s$  we have

$$\begin{aligned}\nabla_{\theta} E[R] &= \int \nabla_{\theta} \pi_{\theta}(a|s) R(s, a) da \\ &= \int \pi_{\theta}(a|s) \nabla \log(\pi_{\theta}(a|s)) R(s, a) da \\ &\approx \sum_{k=1}^K \nabla_{\theta} \pi_{\theta}(a^{(k)}|s) R(s, a^{(k)}) \quad \text{if } a^{(k)} \sim \pi_{\theta}(a|s) \\ &\neq \sum_{k=1}^K \nabla_{\theta} \pi_{\theta}(a^{(k)}|s) R(s, a^{(k)}) \quad \text{if } a^{(k)} \sim \pi_{\theta'}(a|s)\end{aligned}$$

In a replay buffer,  $a_t^{(k)} \sim \pi_{\theta_t}(a|s)$ . Therefore, if we want to compute the gradient at  $t'$ , we should use samples from  $\pi_{\theta_{t'}}(a|s)$  and cannot naively use the old samples from  $\pi_{\theta_t}(a|s)$ ;

unless we correct for it with the “importance weight”  $\frac{\pi_{\theta_{t'}}(a_t^{(k)}|s)}{\pi_{\theta_t}(a_t^{(k)}|s)}$ .

## Notes

Remember: for a given state  $x$  we have

$$\begin{aligned} \nabla_x \pi(x) &= \int \nabla_x \pi(x, a) R(x, a) dx \\ &= \int \pi(x, a) \nabla_x \log(\pi(x, a)) R(x, a) dx \\ &= \sum_a \pi(x, a) \nabla_x \log(\pi(x, a)) R(x, a) \\ &= \sum_a \pi(x, a) \nabla_x \log(\pi(x, a)) R(x, a) \end{aligned}$$

In a replay buffer,  $\pi^{\text{old}} = \pi(x, a)$ . Therefore, if we want to compute the gradient at  $x'$ , we should use samples from  $\pi(x', a)$  and cannot naively use the old samples from  $\pi(x, a)$  unless we correct for it with the "importance weights"  $\frac{\pi(x', a)}{\pi(x, a)}$ .

1. line Replace the integral by a sum, if the action space is discrete.

2. line We used  $\nabla \pi = \pi \nabla \log \pi$ .

$\approx$  in 3. line Sample (Monte Carlo) estimate of the full expectation.

last line There is no equality, if the actions are sampled from another distribution. As the policy distribution changes during learning, the old actions in the replay buffer are not anymore samples from the same distribution. Hence, one cannot naively use the old samples.

- The correction with importance weights works in theory, but there is one such factor for each time step of an episode (we presented the problem for a single state). The product of all importance weights for entire episodes has usually high variance (this is a known issue for so-called importance sampling, where one samples from a 'wrong' distribution and corrects with an importance weight). Therefore, more sophisticated approaches are usually used in practice, see e.g. ACER [Wang et al., 2016].
- Although we presented the problem here with a policy gradient method, the same problem appears in all on-policy methods.

-  Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017).  
Hindsight Experience Replay.  
[arXiv e-prints](#), page arXiv:1707.01495.
-  Bellemare, M. G., Dabney, W., and Munos, R. (2017).  
A Distributional Perspective on Reinforcement Learning.  
[arXiv e-prints](#), page arXiv:1707.06887.
-  Corneil, D., Gerstner, W., and Brea, J. (2018).  
Efficient model-based deep reinforcement learning with variational state tabulation.  
In Dy, J. and Krause, A., editors, [Proceedings of the 35th International Conference on Machine Learning](#), volume 80 of [Proceedings of Machine Learning Research](#), pages 1057-1066, Stockholm, Sweden, Stockholm Sweden. PMLR.
-  Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017).  
Distributional Reinforcement Learning with Quantile Regression.  
[arXiv e-prints](#), page arXiv:1710.10044.
-  Fujimoto, S., van Hoof, H., and Meger, D. (2018).  
Addressing Function Approximation Error in Actor-Critic Methods.  
[arXiv e-prints](#), page arXiv:1802.09477.
-  Ha, D. and Schmidhuber, J. (2018).  
World Models.  
[ArXiv e-prints](#).
-  Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018).  
Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.



In Dy, J. and Krause, A., editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 1861–1870, Stockholm, Sweden, Stockholm Sweden. PMLR.



Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017).

Deep Reinforcement Learning that Matters.  
[arXiv e-prints](#), page arXiv:1709.06560.



Kakade, S. and Langford, J. (2002).

Approximately optimal approximate reinforcement learning.  
*ICML*, 2:267 – 274.



Liang, Y., Machado, M. C., Talvitie, E., and Bowling, M. (2015).

State of the Art Control of Atari Games Using Shallow Reinforcement Learning.  
[ArXiv e-prints](#).



Lillicrap, T. P., Hunt, J. J., Pritzel, A. e., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015).

Continuous control with deep reinforcement learning.  
[arXiv e-prints](#), page arXiv:1509.02971.









Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016).

Asynchronous methods for deep reinforcement learning.  
In Balcan, M. F. and Weinberger, K. Q., editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 1928–1937, New York, New York, USA. PMLR.



Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and et al. (2015).

Human-level control through deep reinforcement learning.  
*Nature*, 518(7540):529–533.

-  Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, Advances in Neural Information Processing Systems 30, pages 5690–5701. Curran Associates, Inc.
-  Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized Experience Replay. [ArXiv e-prints](#).
-  Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. [arXiv e-prints](#), page arXiv:1911.08265.
-  Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust Region Policy Optimization. [ArXiv e-prints](#).
-  Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. [arXiv e-prints](#), page arXiv:1707.06347.
-  Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and et al. (2016). Mastering the game of go with deep neural networks and tree search. [Nature](#), 529(7587):484–489.



Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., and et al. (2018).

A general reinforcement learning algorithm that masters chess, shogi, and go through self-play.

[Science](#), 362(6419):1140–1144.



Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016).

Sample Efficient Actor-Critic with Experience Replay.

[ArXiv e-prints](#).