Applications of Model-Free Deep Reinforcement Learning

Johanni Brea

4 April 2023

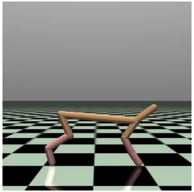
Artificial Neural Networks CS-456

EPFL

Deep Reinforcement Learning Applications

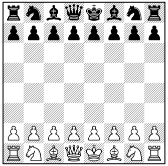
Video Games

Simulated Robotics

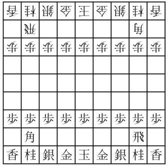


Board Games (next week)

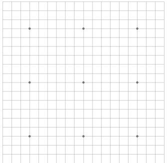
Chess



Shogi



Go



A Classification of Deep Reinforcement Learning Methods

- Deep RL Algorithms
  - Model-Free
    - Policy Optimization
      - Policy Gradient
      - A2C
      - PPO
      - TRPO
    - Q-Learning
      - DDPG
      - TD3
      - SAC
      - DQN
      - C51
      - HER
      - R2D2
      - Agent57
  - Model-Based
    - Learn Model
      - WorldModels
      - VaST
      - Dreamer
      - MuZero
    - Given Model
      - AlphaZero

inspired by `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`

Table of Contents

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

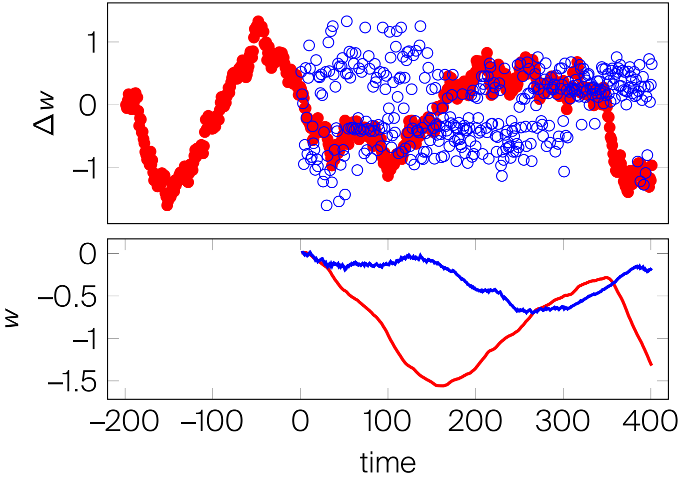
1. that we should not form mini-batches from sequentially acquired data in RL, but
2. use a **replay buffer** from which one can sample iid, or
3. run **multiple actors in parallel**.

Suggested reading: [Mnih et al., 2015] and [Mnih et al., 2016]

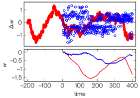
Correlation of Subsequent Observations in RL

- 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



- $w_0 = 0$
  $w_t = w_{t-1} + 0.02 \cdot \Delta w_t.$
- Temporally correlated weight updates can cause large weight fluctuations $\Rightarrow$ potentially unstable learning.
- Reshuffling updates helps to prevent this $\Rightarrow$ reshuffling stabilizes learning.



- $w_0 = 0$
  $w_t = w_{t-1} + 0.02 \cdot \Delta w_t$.
- Temporally correlated weight updates can cause large weight fluctuations $\Rightarrow$ potentially unstable learning.
- 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 Δw_t for $t < 150$ are negative. As a result, w_t moves to a strongly negative value within the first 150 updates and than up again. If we reshuffle the data – the blue points are obtained by sampling a Δw_s with $s \in \{t-200, t-199, \ldots, t\{$ – 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.

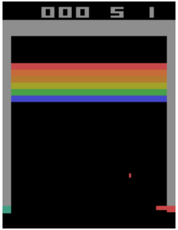
On-Policy Deep RL
e.g. A2C

- 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.

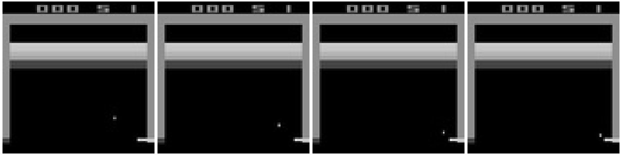
Deep Q-Network (DQN) for Atari Games

Input Encoding

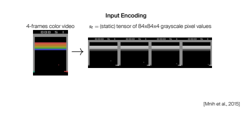
4-frames color video



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

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

[Mnih et al., 2015]

Deep Q-Network (DQN) for Atari Games

1: Initialize neural network Q_θ 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 θ with gradient of $\mathcal{L}(\theta) = \sum_j \left(r_j + \gamma \max_{a'} \hat{Q}(s_{j+1})_{a'} - Q_\theta(s_j)_{a_j}\right)^2$
8: Increment t and reset $\hat{Q} \leftarrow Q_\theta$ every C steps.
9: **until** some termination criterion is met.
10: **return** Q_θ

[Mnih et al., 2015]

Notes

1: For n_s dimensional input (e.g. 84x84x4 pixels) the neural network Q_θ has n_s input neurons and n_a (linear) output neurons.

2: $\hat{Q}$ is the target network.

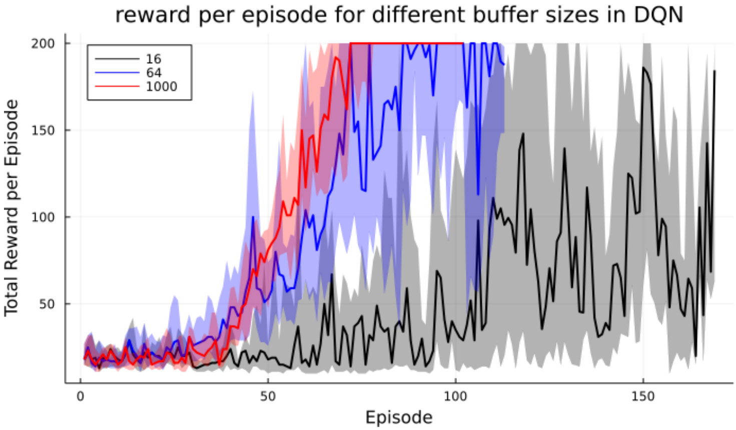
4: For ϵ-greedy policy, state s_t, action a_t is $\arg\max_a Q_\theta(s_t)_a$ with probability $1-\epsilon$ or a randomly chosen action with probability ϵ. Typically, ϵ is decreased over the course of learning, e.g. from 1 to 0.1 over the first million transitions (**exploration annealing**).

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

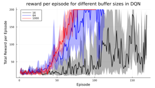


learning to balance a cart pole with 20 different random seeds; lines = median of the reward per episode; shaded area = range between 10th and 90th percentile.

https://lcnwww.epfl.ch/cs456/dqn_a2c.html

Notes

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.

https://lcnwww.epfl.ch/cs456/dqn_a2c.html

- 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.

Prioritized Replay

Instead of uniformly sampling from the replay buffer, sample transition with probability

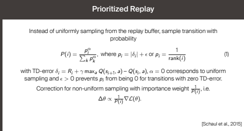
$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \text{ where } p_i = |\delta_i| + \epsilon \text{ or } p_i = \frac{1}{\mathrm{rank}(i)} \tag{1}$$

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 p_i from being 0 for transitions with zero TD-error.

Correction for non-uniform sampling with importance weight $\frac{1}{P(i)}$, i.e.

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

[Schaul et al., 2015]


Prioritized Replay

Instead of uniformly sampling from the replay buffer, sample transition with probability

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \text{ where } p_i = |\delta_i| + \epsilon \text{ or } p_i = \frac{1}{\mathrm{rank}(i)} \quad (1)$$

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 p_i from being 0 for transitions with zero TD-error.

Correction for non-uniform sampling with importance weight $\frac{1}{P(i)}$, i.e. $\Delta\theta \propto \frac{1}{P(i)} \nabla \mathcal{L}(\theta)$.

[Schaul et al., 2015]

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.

Importance sampling
$E[f(x)] = \int p(x) f(x) dx \approx \frac{1}{K} \sum_{i=1}^K f(x_i)$ if $x_i \sim p(x)$.
$E[f(x)] = \int p(x) f(x) dx = \int q(x) \frac{p(x)}{q(x)} f(x) dx \approx \frac{1}{K} \sum_{i=1}^K \frac{p(x)}{q(x)} f(x)$. The factor $\frac{p(x)}{q(x)}$ is called importance weight.
For sampling from the replay buffer the correct distribution $p(x)$ is the uniform distribution. If we sample instead from $P(i)$, we need to correct by $1/P(i)$.

In [Schaul et al., 2015], they used the “importance weight” $w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta / C$ where $C = \max_k w_k$, where β 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 $\beta \to 1$ over the course of training.

Advantage Actor-Critic (A2C)

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

1: Initialize neural networks π_θ and V_ϕ.
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 θ with gradient of $\sum_k A_t^{(k)} \log \pi_\theta(a_t^{(k)}; s_t^{(k)})$
9:   Update ϕ with gradient of $\sum_k \left(R_t^{(k)} - V_\phi(s_t^{(k)})\right)^2$.
10:   Increment t.
11: **until** some termination criterion is met.
12: **return** π_θ and V_ϕ

Notes

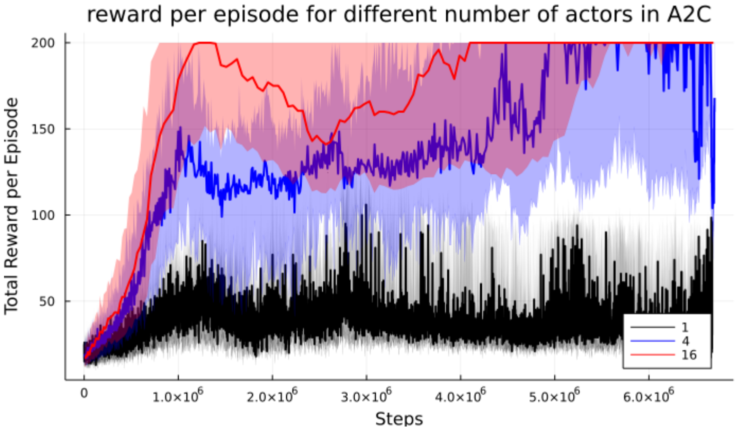
1: For n_s-dimensional input and n_a action, the policy network π_θ has n_s inputs and as output a softmax layer with n_a units. The value network V_ϕ has n_s inputs and one linear output.

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 θ and ϕ 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`)

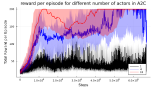
A2C on CartPole



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.

https://lcnwww.epfl.ch/cs456/dqn_a2c.html

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.

https://lcnwww.epfl.ch/cs456/dqn_a2c.html

Notes

- 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.

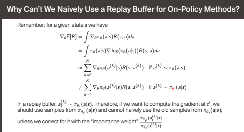
Why Can't We Naively Use a Replay Buffer for On-Policy Methods?

Remember: for a given state s we have

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

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" $\dfrac{\pi_{\theta_{t'}}(a_t^{(k)}|s)}{\pi_{\theta_t}(a_t^{(k)}|s)}$.



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

Remember: for a given state s we have

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

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

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 chonges 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.

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

Off-Policy Deep RL	On-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 sample complexity because old experiences cannot be used to update a policy that has already changed.
– higher memory complexity need to store many experiences in the replay buffer.	**+** 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?

- ☐ 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.
- ☐ We could use multiple actors instead of a replay memory with Q-Learning.
- ☐ 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.

Table of Contents

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, 2002, Schulman et al., 2015, Schulman et al., 2017, Lillicrap et al., 2015]

Deep Deterministic Policy Gradient (DDPG) [1]

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 θ 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.

[1] The name is confusing: DDPG is more closely related to DQN than to PG!

[Lillicrap et al., 2015]

Deep Deterministic Policy Gradient (DDPG)

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

[Lillicrap et al., 2015]

Deep Deterministic Policy Gradient (DDPG)

1: Initialize neural networks Q_θ, π_ψ 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 θ with gradient of $\sum_j \left(r_j + \hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1})) - Q_\theta(s_j, a_j)\right)^2$
8: Update ψ 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_θ

[Lillicrap et al., 2015]

Notes

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), 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 π_ψ takes states as input and is supposed to return the greedy action.

4: Here a_t is a n_a-dimensional vector and ϵ 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 ψ 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 θ of a neural network change according to

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

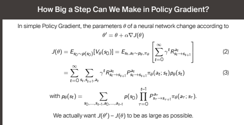
$$J(\theta) = E_{s_0 \sim p(s_0)}[V_\theta(s_0)] = E_{s_t, a_t \sim p_\theta, \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R^{a_t}_{s_t \to s_{t+1}}\right] \tag{2}$$

$$= \sum_{t=0}^{\infty} \sum_{s_t, s_{t+1}, a_t} \gamma^t R^{a_t}_{s_t \to s_{t+1}} P^{a_t}_{s_t \to s_{t+1}} \pi_\theta(a_t; s_t) p_\theta(s_t) \tag{3}$$

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

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

Notes



How Big a Step Can We Make in Policy Gradient?

In simple Policy Gradient, the parameters θ of a neural network change according to

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

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

$$= \sum_{t=0}^{\infty} \sum_{s_t, s_{t+1}, a_t} \gamma^t R^{a_t}_{s_t \to s_{t+1}} P^{a_t}_{s_t \to s_{t+1}} \pi_\theta(a_t; s_t) p_\theta(s_t) \quad (3)$$

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

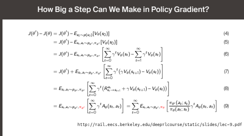
We actually want $J(\theta') - J(\theta)$ to be as large as possible.

- The objective function $J(\theta)$ is the expected future discounted return given parameters θ 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 $\displaystyle\sum_{s_0, s_1, \ldots, a_0, a_1, \ldots} p(s_0) \prod_\tau P^{a_\tau}_{s_\tau \to s_{\tau+1}} \pi_\theta(a_\tau; s_\tau)$ (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 $R^{a_t}_{s_t \to s_{t+1}}$ does not depend on the future and therefore all sums over future states and actions ($s_{t+2}, s_{t+3}, \ldots, a_{t+1}, a_{t+2}, \ldots$) evaluate to 1, because of the normalization of the probabilities.
- The optimal learning rate α 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?

$$
\begin{aligned}
J(\theta') - J(\theta) &= J(\theta') - E_{s_0 \sim p(s_0)}[V_\theta(s_0)] && (4)\\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}[V_\theta(s_0)] && (5)\\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t V_\theta(s_t) - \sum_{t=1}^{\infty} \gamma^t V_\theta(s_t)\right] && (6)\\
&= J(\theta') + E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(\gamma V_\theta(s_{t+1}) - V_\theta(s_t)\right)\right] && (7)\\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(R^{a_t}_{s_t \to s_{t+1}} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)\right)\right] && (8)\\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t)\right] = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_\theta}\left[\frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)} \gamma^t A_\theta(s_t, a_t)\right] && (9)
\end{aligned}
$$

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



How Big a Step Can We Make in Policy Gradient?

$$
\begin{aligned}
J(\theta') - J(\theta) &= J(\theta') - E_{s_0 \sim p(s_0)}[V_\theta(s_0)] && (4)\\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}[V_\theta(s_0)] && (5)\\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t V_\theta(s_t) - \sum_{t=1}^{\infty} \gamma^t V_\theta(s_t)\right] && (6)\\
&= J(\theta') + E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t (\gamma V_\theta(s_{t+1}) - V_\theta(s_t))\right] && (7)\\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t (R^{a_t}_{s_t \to s_{t+1}} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t))\right] && (8)\\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t)\right] = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta}}\left[\frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)} \gamma^t A_\theta(s_t, a_t)\right] && (9)
\end{aligned}
$$

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

Notes

(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 θ (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 π_θ while correcting with the importance weight $\frac{\pi_{\theta'}(a_t;s_t)}{\pi_\theta(a_t;s_t)}$.

Proximal Policy Optimization: Idea

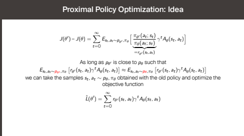
$$J(\theta') - J(\theta) = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_\theta} \left[\underbrace{\frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)}}_{= r_{\theta'}(s_t, a_t)} \gamma^t A_\theta(s_t, a_t) \right]$$

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

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

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

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 θ'. But we can sample from p_θ and if $p_{\theta'}$ is sufficiently close to p_θ, we have approximately $\hat{L}(\theta') \approx J(\theta') - J(\theta)$.

Instead of searching for an optimal learning rate α, 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_θ, 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 $\mathrm{KL}[\pi_\theta \| \pi_{\theta'}] \leq \delta$.

Clipped Surrogate Objectives (PPO-CLIP)

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

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

Trust-Region Policy Optimization (TRPO)

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

Clipped Surrogate Objectives (PPO-CLIP)

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

Notes

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

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

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

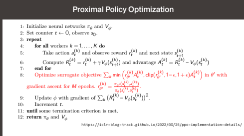
Its derivative is 0 when $x < l$ or $x > u$. See exercise 2 for details.

Proximal Policy Optimization

Initialize neural networks π_θ and V_ϕ.
Set counter $t \leftarrow 0$, observe s_0.
repeat
    for all workers $k = 1, \ldots, K$ **do**
        Take action $a_t^{(k)}$ and observe reward $r_t^{(k)}$ and next state $s_{t+1}^{(k)}$
        Compute $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$ and advantage $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$
    end for
    Optimize surrogate objective $\sum_k \min\left(r_{\theta'}^{(k)} A_t^{(k)}, \mathrm{clip}(r_{\theta'}^{(k)}, 1-\epsilon, 1+\epsilon) A_t^{(k)}\right)$ in θ' with gradient ascent for M epochs. $r_{\theta'}^{(k)} = \frac{\pi_{\theta'}(s_t^{(k)}, a_t^{(k)})}{\pi_\theta(s_t^{(k)}, a_t^{(k)})}$.
    Update ϕ with gradient of $\sum_k \left(R_t^{(k)} - V_\phi(s_t^{(k)})\right)^2$.
    Increment t.
until some termination criterion is met.
return π_θ and V_ϕ

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

Notes



Proximal Policy Optimization

1: Initialize neural networks π_θ and V_ϕ.
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 $\sum_k \min\left(r_{\theta'}^{(k)} A_t^{(k)}, \mathrm{clip}(r_{\theta'}^{(k)}, 1-\epsilon, 1+\epsilon) A_t^{(k)}\right)$ in θ' 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 ϕ with gradient of $\sum_k \left(R_t^{(k)} - V_\phi(s_t^{(k)})\right)^2$.
10: Increment t.
11: **until** some termination criterion is met.
12: **return** π_θ and V_ϕ

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

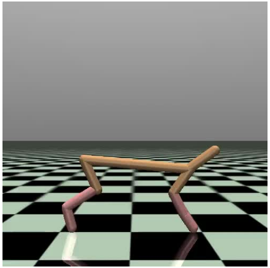
As in A2C, the workers send their experiences back to the learner before starting to optimize the surrogate objective in line 8. In contrast to A2C that does a single gradient ascent update step in line 8, PPO uses each observation multiple times to optimize the surrogate objective in line 8. Instead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

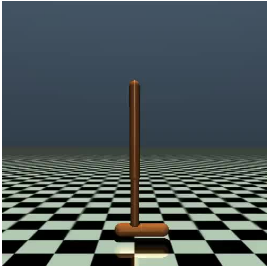
Quiz

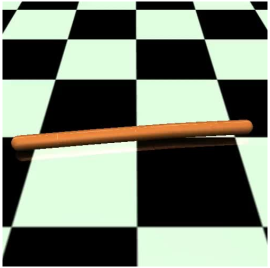
Which statement is correct?

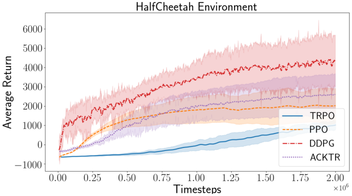
- ☐ With the update of policy gradient, $\theta' = \theta + \alpha \nabla J(\theta)$ and a fixed learning rate α, $J(\theta') - J(\theta)$ will always be positive.
- ☐ In proximal policy optimization methods we want to keep the ratio $r_{\theta'}(s_t, a_t) = \frac{\pi_{\theta'}(a_t; s_t)}{\pi_{\theta}(a_t; s_t)}$ close to one, such that the state visitation probabilities $p_\theta(s_t)$ and $p_{\theta'}(s_t)$ are roughly the same.
- ☐ A2C uses each minibatch once to update the policy, whereas proximal policy methods use each minibatch multiple times, usually.

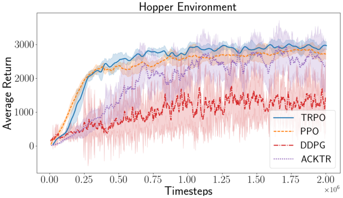
Comparison of Algorithms in Simulated Robotics

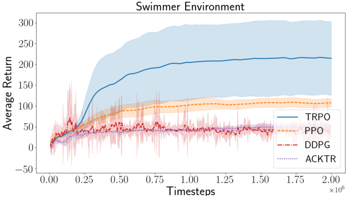












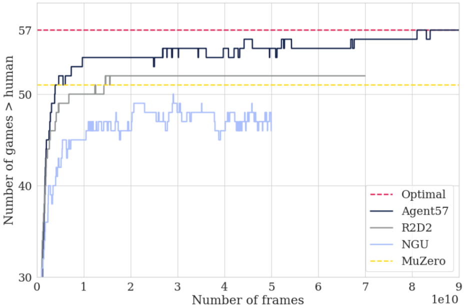
[Henderson et al., 2017]

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 π_ψ (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]).

Table of Contents

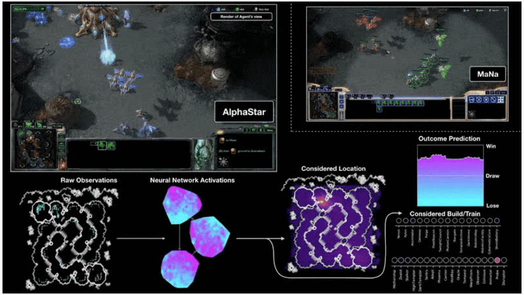
Agent 57



- Super-human on 57 Atari games.
- Prioritized replay for efficient learning from the replay buffer.
- LSTM to deal with partial observability; similar to Recurrent Replay Distributed DQN (R2D2).
- Intrinsic motivation for efficient exploration; similar to Never Give Up (NGU). $Q(s, a) = Q^r(s, a) + \beta Q^i(s, a)$, where Q^r is trained with standard reward and the intrinsic part Q^i is given a novelty signal as reward (think of 1/(visitation count)).

[Badia et al., 2020]

AlphaStar



- Grandmaster level in StarCraft II.
- Supervised pre-training to imitate human players.
- Transformer & LSTM.
- V-trace (version of actor-critic with off-policy corrections)
- distributed training (multiple TPUs)

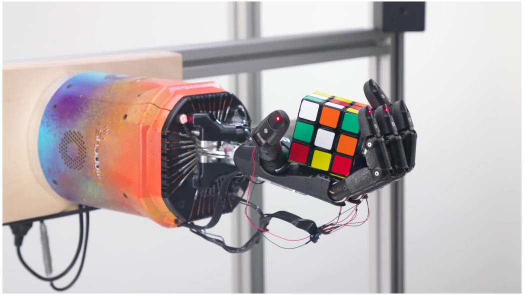
[Vinyals et al., 2019]

OpenAI Five

- Defeated world champion in Dota 2
- LSTM with shared weights for all 5 “heros”
- PPO
- distributed training (thousands of GPUs for many months)

[OpenAI et al., 2019a]

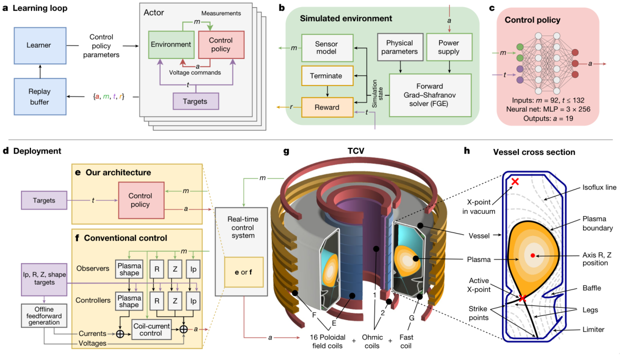
Solving Rubik's Cube with a Robot Hand



- Solves Rubik's cube with a robot hand
- Vision model based on ResNet50 & LSTM controller
- PPO
- Automated Domain Randomization: training on simulator with a distribution of environments (slightly different physics).

[OpenAI et al., 2019b]

Magnetic control of tokamak plasmas



- Controls nuclear fusion plasma
- Maximum a posteriori policy optimization (MPO); an actor-critic algorithm.
- Training on simulator.

[Degrave et al., 2022]

References I

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.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020).
Agent57: Outperforming the Atari human benchmark.
In III, H. D. and Singh, A., editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 507–517. PMLR.

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, Stockholmsmässan, 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.

References II

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., and Riedmiller, M. (2022).
Magnetic control of tokamak plasmas through deep reinforcement learning.
Nature, 602(7897):414–419.

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, Stockholmsmässan, Stockholm Sweden. PMLR.

References III

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. M. (2002).
A natural policy gradient.
In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, Advances in Neural Information Processing Systems 14, pages 1531–1538. MIT Press.

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.

References IV

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.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019a).
Dota 2 with Large Scale Deep Reinforcement Learning.
arXiv e-prints, page arXiv:1912.06680.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019b).
Solving Rubik's Cube with a Robot Hand.
arXiv e-prints, page arXiv:1910.07113.

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.

References V

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.

References VI

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.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019).
Grandmaster level in starcraft ii using multi-agent reinforcement learning.
Nature, 575(7782):350–354.

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.