

Sample Efficient RL (Part 1)

Presenter: Jake Grigsby

University of Virginia

<https://qdata.github.io/deep2Read/>

202008

Sample Efficiency

While the usual goal of RL is to maximize the expected (discounted) return $J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{\tau,t} \right]$, **sample efficient** algorithms look to achieve some threshold level of performance while taking as few steps in the environment as possible

The problem with on-policy learning: policy gradients

Online policy gradient algorithms have two basic steps:

1 Policy Evaluation

- ▶ Use the current policy π_θ to collect trajectories of experience in the environment

2 Policy Improvement

- ▶ Use the experience collected to make some estimate of $\nabla_\theta J(\pi_\theta)$, and update θ with SGD

The problem with on-policy learning: policy gradients

The main problem is that the variance of the policy gradient $\nabla_{\theta} J(\pi_{\theta})$, is extremely high, requiring a large number of samples to estimate properly [8]

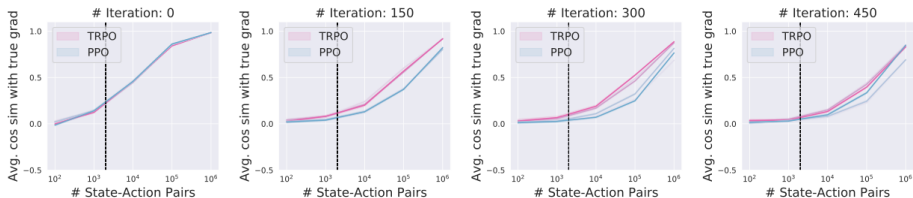


Figure: Convergence of the policy gradient in the Humanoid task

The problem with on-policy learning: policy gradients

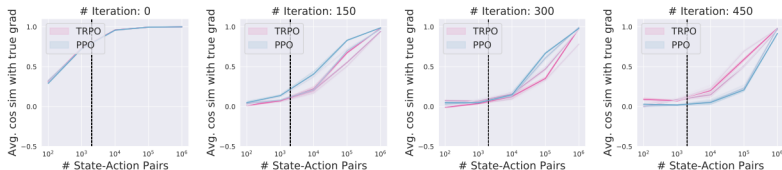


Figure: Convergence of the policy gradient in the Walker task

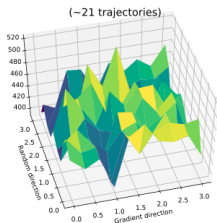


Figure: Convergence of the policy gradient in the Hopper task

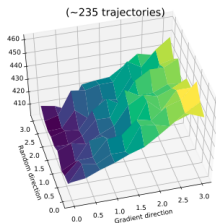
The problem with on-policy learning: policy gradients

Another way to visualize this: we need a lot of data for the gradient step to correlate with an actual increase in return [8]:

2,000 state-action pairs



20,000 state-action pairs



100,000 state-action pairs

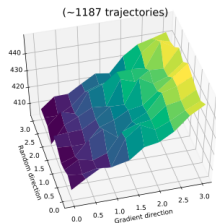


Figure: Reward landscape for TRPO on Humanoid task

The problem with on-policy learning: policy gradients

In summary, on-policy policy gradients:

- Need thousands of interactions to really find out how to improve at each step
- Have to throw away that experience after each update

The perfect storm for bad sample efficiency

We can improve by finding a way to reuse transitions several times

- off-policy

Experience Replay and Q-Learning

Q-learning fits the action-value function of the *optimal* policy

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} \left[\max_a Q^*(s', a) \right]$$

This means we can use any policy for experience collection, and we can replay experience as many times as we want.

Experience Replay - at first used mostly to stabilize training - is a key part of sample efficient algorithms.

Very Brief Q-learning Review

Just for the sake of standardizing some notation:

Replay Q-Learning

For **M** iterations:

For **K** env steps:

Use π_b to collect (s, a, r, s', d)

$\mathcal{D} = \mathcal{D} \cup \{(s, a, r, s', d)\}$

if $|\mathcal{D}| > \mathbf{C}$, remove oldest transition

For **G** updates:

sample **B** samples $\sim \mathcal{D}$

compute TD targets \hat{y} using (r, s', d)

$Q(s, a) \leftarrow Q(s, a) + \alpha [y - Q(s, a)]$

buffer capacity = **C**

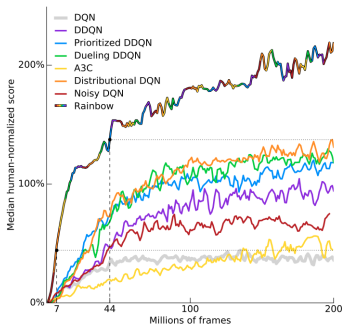
replay ratio = $\frac{\mathbf{G}}{\mathbf{K}}$

age of oldest policy = $\frac{\mathbf{C}}{\mathbf{K}}$

Benchmarks

Following the original DQN paper [2], there was a flood of new methods to improve performance and sample efficiency:

- 1 Double DQN [1]
 - ▶ Reduce overestimation in update rule
- 2 Dueling DQN [4]
 - ▶ Model the value and advantage instead of action-value
- 3 Prioritized Experience Replay [3]
 - ▶ Sample from buffer according to TD error
- 4 C51 [5]
 - ▶ Learn the distribution of returns instead of the expectation
- 5 Noisy Networks [6]
 - ▶ Control exploration with learned parameters



Ideas for Sample Efficiency

- 1 Making the most of existing samples
 - ▶ New network architectures
 - ▶ Hindsight/counterfactual credit assignment
 - ▶ Learning better representations of the environment
 - ▶ **Avoid overfitting to limited experience**

How does replay impact learning?

There are two main parameters to investigate [11]:

- 1 age of the oldest policy
 - ▶ how long ago (in terms of policy updates) was the oldest experience in the buffer collected?
- 2 buffer capacity

		Replay Capacity				
		100,000	316,228	1,000,000	3,162,278	10,000,000
Oldest Policy	25,000,000	250.000	79.057	25.000	7.906	2.500
	2,500,000	25.000	7.906	2.500	0.791	0.250
	250,000	2.500	0.791	0.250	0.079	0.025
	25,000	0.250	0.079	0.025	0.008	0.003

Figure: Performance of Rainbow on subset of Atari games with different replay ratios. Green indicates improvement over baseline.

How does replay impact learning?

Two main conclusions:

- 1 Decreasing the age of the oldest policy (making learning closer to on-policy) improves performance
 - ▶ Learning focuses on more recent, higher performance data.
 - ▶ Hard exploration games are a notable exception, where it seems more important to cover a large portion of the state space.
- 2 Increasing the capacity of the replay buffer improves performance
 - ▶ Better state-space coverage

Note that both of these suggest a lower replay ratio ($\frac{G}{K}$)

$$\begin{array}{c} \uparrow \downarrow \\ \downarrow \uparrow \end{array} \frac{\text{Age of Oldest Policy}}{\text{Buffer Capacity}} = \begin{array}{c} \uparrow \downarrow \\ \downarrow \uparrow \end{array} \frac{\text{Gradient Updates}}{\text{Env Steps}} = \begin{array}{c} \downarrow \uparrow \\ \uparrow \downarrow \end{array} \text{Replay Ratio}$$

How does replay impact learning?

Low replay ratios:

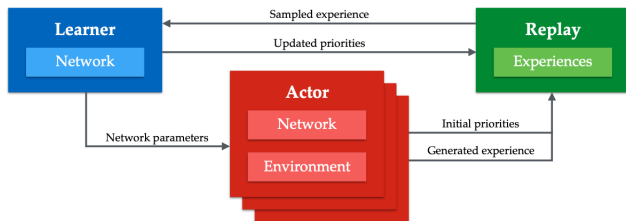
- 1 Reduce the risk of overfitting
- 2 Increase the diversity of experience in the buffer
- 3 Reduce sample efficiency

Some hope for improvement: taking lessons from distributed algorithms

Distributed Reinforcement Learning

Distributed RL algorithms:

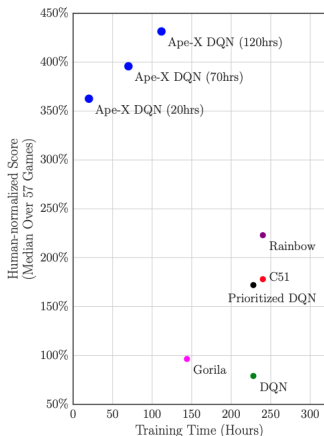
- Sync experience across a large number of actors and update parameters from a central server.
- Or sync gradients across a large number of learners.



They are intended to maximize data throughput and wall-clock training speed, with no concern for sample efficiency.

Distributed Reinforcement Learning

Interestingly, they also perform significantly better than single-actor agents [7]



What's going on here?

Distributed Reinforcement Learning

As a byproduct of maximizing data throughput, distributed algorithms:

- 1 Reduce the age of the oldest policy in the replay buffer
- 2 Typically increase buffer capacity

→ They (significantly) decrease the replay ratio

Does that explain the performance increase?

- Early experiments suggest no...

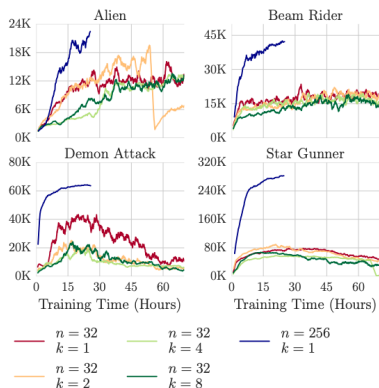


Figure 6: Testing whether improved performance is caused by recency alone: n denotes the number of actors, k the number of times each transition is replicated in the replay. The data in the run with $n = 32$, $k = 8$ is therefore as recent as the data in the run with $n = 256$, $k = 1$, but performance is not as good.

Distributed Reinforcement Learning

What else may be going on?

Collecting this much parallel experience also:

- Increases the diversity of experience going into the buffer (in terms of state space coverage)
 - ▶ Exploration is independent
- Decouples the agent from a particular timestep in a trajectory
 - ▶ Each env is likely on a different step count

Is there a way to take these ideas and apply them to single (or few) actor agents that are sample efficient?

Ideas for Sample Efficiency

- 1 Making the most of existing samples
 - ▶ New network architectures
 - ▶ Hindsight/counterfactual credit assignment
 - ▶ Learning better representations of the environment
 - ▶ Avoid overfitting to limited experience
- 2 Making more data
 - ▶ Data Augmentation (see last week)
 - ▶ Creating new transitions by modeling the environment
 - ★ **Model-based Reinforcement Learning**

Model-based Reinforcement Learning

- Any method that attempts to learn a model of the transition function of the env is considered model-based. This is a very wide range of methods.
 - ▶ Even Experience Replay can be viewed as an accurate non-parametric model of the env that we improve by adding new transitions. [9]
 - ▶ There is lots of work on model-based planning

Model-based Reinforcement Learning

- Any method that attempts to learn a model of the transition function of the env is considered model-based. This is a very wide range of methods.
 - ▶ Even Experience Replay can be viewed as an accurate non-parametric model of the env that we improve by adding new transitions. [9]
 - ▶ There is lots of work on model-based planning
- We are going to focus on 'Dyna'-style algorithms, where *a model is used to generate additional training samples for an otherwise model-free algorithm*

Fitting the Model

Training the model is a relatively standard supervised learning problem

Given buffer \mathcal{D} of experience

For batch $\{(s, a, r, s', d)\} \sim \mathcal{D}$:

Train with maximum-likelihood on predictions of (s', r, d) , given (s, a)

There is lots of room to experiment here with advancements in time-series modeling, especially in POMDPs...

The result is a model $f_{\theta}(s, a) = (s', r, d)$

Dyna Algorithms

Environment \rightarrow Model \rightarrow Model-free Agent

Dyna Q-Learning

For **M** iterations:

For **N** real env steps:

Use π_b to collect (s, a, r, s', d)

$\mathcal{D}_{env} = \mathcal{D}_{env} \cup \{(s, a, r, s', d)\}$

if $|\mathcal{D}_{env}| > \mathbf{C}_1$, remove oldest transition

Fit model f_θ using \mathcal{D}_{env}

For **K** modeled env steps:

Use π_b to collect (s, a, r, s', d)

$\mathcal{D}_{model} = \mathcal{D}_{model} \cup \{(s, a, r, s', d)\}$

if $|\mathcal{D}_{model}| > \mathbf{C}_2$, remove oldest transition

For **G** updates:

sample **B** samples $\sim \mathcal{D}_{model}$

compute TD targets \hat{y} using (r, s', d)

$Q(s, a) \leftarrow Q(s, a) + \alpha [y - Q(s, a)]$

Performance Bounds

How does performance in the modeled env correspond to the real one? [10]

Given real env MDP \mathcal{M} , modeled MDP $\hat{\mathcal{M}}$, transition distributions TV bound ϵ_m , policy divergence upper bound ϵ_π :

$$J_{\mathcal{M}}(\pi) \geq J_{\hat{\mathcal{M}}}(\pi) - \left[\frac{2\gamma r_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4r_{\max}\epsilon_\pi}{(1-\gamma)} \right]$$

This means that if it's possible to improve performance by at least as much as the right-most term, we can expect to improve in the actual env.

Performance Bounds

- *branched rollouts* are trajectories that start by rolling out a policy in the real env, and then switch to using model-based transitions for k steps
 - ▶ If we started from the initial state dist, inaccurate models would be useless at distant regions of the state space, due to compounding errors
 - ▶ See [10] for a modified performance bound using this idea
- To prevent against model exploitation, we train an ensemble of models, and switch between them when generating trajectories
 - ▶ This makes it difficult for the agent to reliably exploit inaccuracies in a particular model

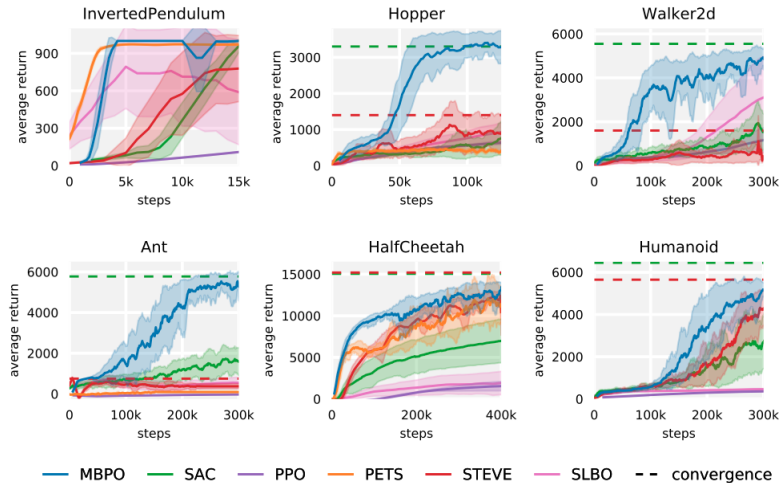
Model Based Policy Optimization

One popular (recent) implementation of this idea: MBPO

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

- 1: Initialize policy π_ϕ , predictive model p_θ , environment dataset \mathcal{D}_{env} , model dataset $\mathcal{D}_{\text{model}}$
 - 2: **for** N epochs **do**
 - 3: Train model p_θ on \mathcal{D}_{env} via maximum likelihood
 - 4: **for** E steps **do**
 - 5: Take action in environment according to π_ϕ ; add to \mathcal{D}_{env}
 - 6: **for** M model rollouts **do**
 - 7: Sample s_t uniformly from \mathcal{D}_{env}
 - 8: Perform k -step model rollout starting from s_t using policy π_ϕ ; add to $\mathcal{D}_{\text{model}}$
 - 9: **for** G gradient updates **do**
 - 10: Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$
-

MBPO Results









More Model-Based Ideas






Next week:

- Modeling high dimensional observation spaces
 - ▶ Video prediction
 - ▶ Modeling in a compressed space (using AEs)
- Model-based online methods
- Offline RL
 - ▶ Extreme case where \mathcal{D}_{env} is fixed
- Hindsight Credit Assignment

References I

-  Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].
-  Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
-  Tom Schaul et al. *Prioritized Experience Replay*. 2015. arXiv: 1511.05952 [cs.LG].
-  Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2015. arXiv: 1511.06581 [cs.LG].
-  Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG].
-  Meire Fortunato et al. *Noisy Networks for Exploration*. 2017. arXiv: 1706.10295 [cs.LG].

References II

-  Dan Horgan et al. “Distributed prioritized experience replay”. In: *arXiv preprint arXiv:1803.00933* (2018).
-  Andrew Ilyas et al. *A Closer Look at Deep Policy Gradients*. 2018. arXiv: 1811.02553 [cs.LG].
-  Hado P van Hasselt, Matteo Hessel, and John Aslanides. “When to use parametric models in reinforcement learning?” In: *Advances in Neural Information Processing Systems*. 2019, pp. 14322–14333.
-  Michael Janner et al. *When to Trust Your Model: Model-Based Policy Optimization*. 2019. arXiv: 1906.08253 [cs.LG].
-  William Fedus et al. *Revisiting Fundamentals of Experience Replay*. 2020. arXiv: 2007.06700 [cs.LG].