Sample Efficient RL (Part 2)

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

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 slides from two weeks agp)
   - 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. [4]
  - There is lots of work on model-based planning
- We are going to focus mostly on 'Dyna'-style algorithms,
  - where *a model is used to generate additional training samples for an otherwise model-free algorithm*

The general MBRL framework. When to use a model?

- [4] discuss how replay-based "models" create an upper bound for performance
  - Need to add planning to get a true advantage
- However, models that can predict new transitions before we put them in the replay can increase sample efficiency...

Algorithm 1 Model-based reinforcement learning

Input: state sample procedure d
Input: model m
Input: policy π
Input: predictions v
Input: environment $\mathcal{E}$
Get initial state $s \leftarrow \mathcal{E}$
for iteration $\in \{1, 2, \dots, K\}$ **do**
    for interaction $\in \{1, 2, \dots, M\}$ **do**
        Generate action: $a \leftarrow \pi(s)$
        Generate reward, next state: $r, s' \leftarrow \mathcal{E}(a)$
        $m, d \leftarrow \textsc{UpdateModel}(s, a, r, s')$
        $\pi, v \leftarrow \textsc{UpdateAgent}(s, a, r, s')$
        Update current state: $s \leftarrow s'$
    end for
    for planning step $\in \{1, 2, \dots, P\}$ **do**
        Generate state, action $\tilde{s}, \tilde{a} \leftarrow d$
        Generate reward, next state: $\tilde{r}, \tilde{s}' \leftarrow m(\tilde{s}, \tilde{a})$
        $\pi, v \leftarrow \textsc{UpdateAgent}(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')$
    end for
end for

Fitting the Model

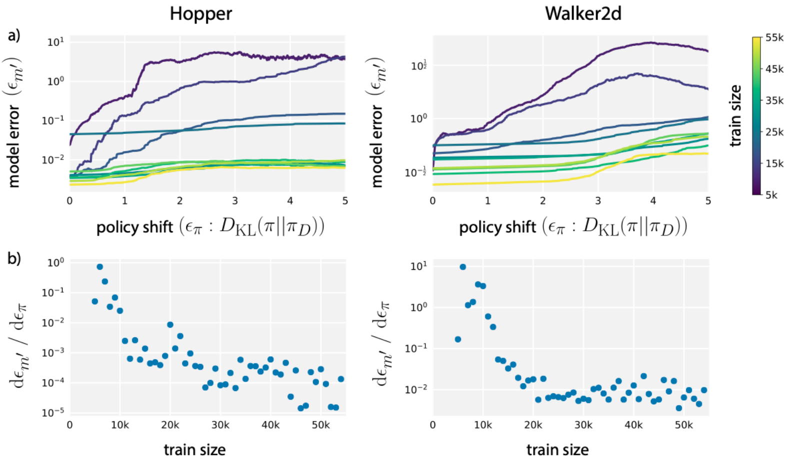
Training the model is a relatively standard supervised learning problem

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 $f_\theta(s, a) = (s', r, d)$

Model Performance Under a Shifting Policy



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? [5]

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

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

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 [5] 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

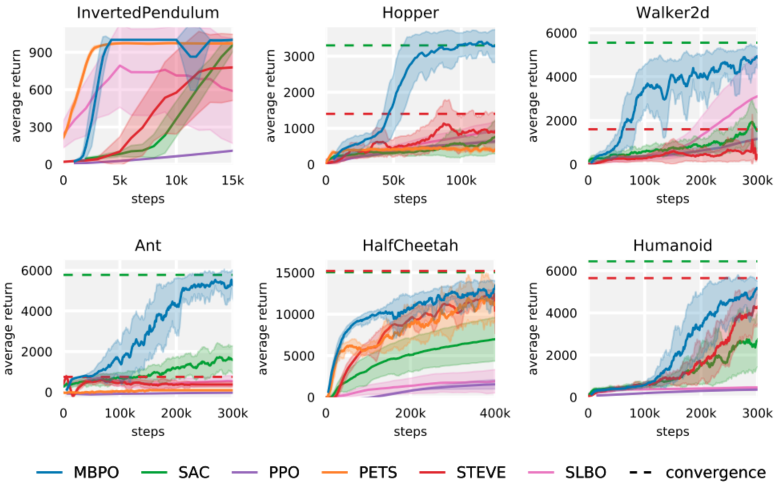
Model Based Policy Optimization

One popular (recent) implementation of this idea: `MBPO`

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

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

MBPO Results



ME-TRPO

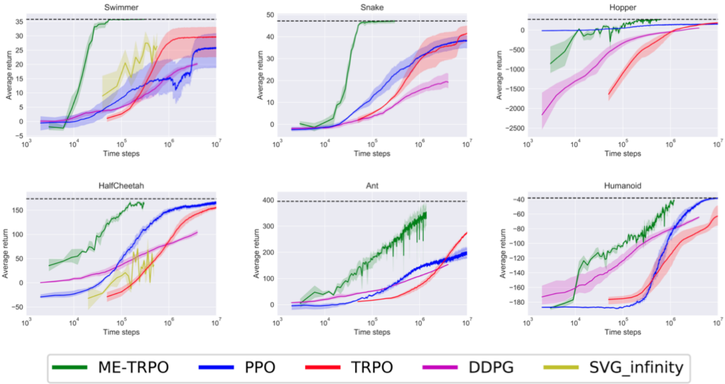
These ideas can also be extended to Policy Gradient methods, usually by getting rid of the branched rollouts and instead estimating the env's initial state distribution. [3]

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

1: Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, ..., \hat{f}_{\phi_K}$.
2: Initialize an empty dataset $\mathcal{D}$.
3: **repeat**
4: Collect samples from the real system f using π_θ and add them to D.
5: Train all models using $\mathcal{D}$.
6: **repeat** ▷ Optimize π_θ using all m
7: Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^{K}$ using π_θ.
8: Update the policy using TRPO on the fictitious samples.
9: Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, ..., K$.
10: **until** the performances stop improving.
11: **until** the policy performs well in real environment f.

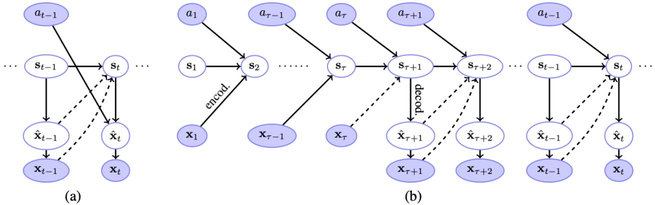
Increases risk of compounding errors

ME-TRPO Results



Modeling Complex Environments

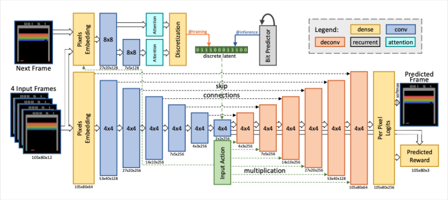
- Image-based POMDPS require more complicated models
  - [1] study how to use large Convolutional RNN architectures to predict the next frame directly.



- A similar model was used in [6], with great sample efficiency results

SimPLe

Use a sophisticated model architecture as part of a straightforward RL procedure:



Essentially `ME-PPO`, with short rollouts to reduce compounding error. No branching rollouts.

Algorithm 1: Pseudocode for SimPLe

Initialize policy $\boldsymbol{\pi}$
Initialize model parameters $\boldsymbol{\theta}$ of env'
Initialize empty set $\mathbf{D}$
while not done **do**
 ▷ collect observations from real env.
 $\mathbf{D} \leftarrow \mathbf{D} \cup \mathrm{COLLECT}(env, \pi)$
 ▷ update model using collected data.
 $\boldsymbol{\theta} \leftarrow \mathrm{TRAIN\_SUPERVISED}(env', \mathbf{D})$
 ▷ update policy using world model.
 $\boldsymbol{\pi} \leftarrow \mathrm{TRAIN\_RL}(\boldsymbol{\pi}, env')$
end while

SimPLe Results

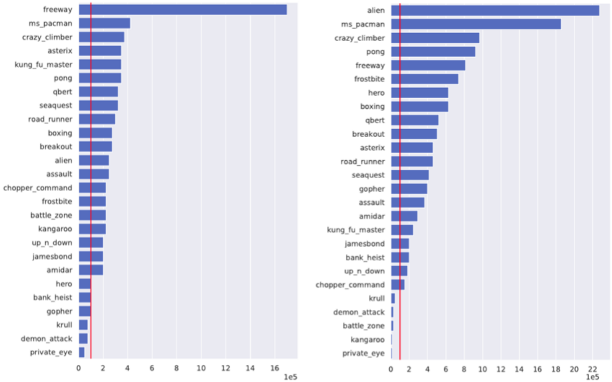
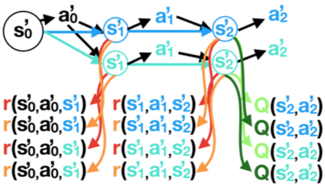


Figure 3: Comparison with Rainbow and PPO. Each bar illustrates the number of interactions with environment required by Rainbow (left) or PPO (right) to achieve the same score as our method (SimPLe). The red line indicates the 100K interactions threshold which is used by the our method.

Interestingly, this is not the SOTA on Atari 100k, after research on Experience Replay I talked about last week led to much faster model-free learning [4] (`Data-Efficient Rainbow`)

Other Ways to Use the Model

Models can be used to generate rollouts for target generation (STEVE) [2]



$$\mathcal{T}_2^{\mathrm{MVE}} = \begin{Bmatrix} r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_0,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \\ r + \gamma r(s'_0,a'_0,s'_1) + \gamma^2 r(s'_1,a'_1,s'_2) + \gamma^3 Q(s'_2,a'_2) \end{Bmatrix}$$

Key idea: Use an ensemble of models and multiple lookahead paths to reduce variance of the targets

This can be combined with standard Dyna ideas, although it's had mixed results so far...

References I

- Silvia Chiappa et al. *Recurrent Environment Simulators*. 2017. arXiv: 1704.02254 [cs.AI].
- Jacob Buckman et al. *Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion*. 2018. arXiv: 1807.01675 [cs.LG].
- Thanard Kurutach et al. *Model-Ensemble Trust-Region Policy Optimization*. 2018. arXiv: 1802.10592 [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].
- Lukasz Kaiser et al. *Model-Based Reinforcement Learning for Atari*. 2019. arXiv: 1903.00374 [cs.LG].