Investigating and Combining Approaches for Data-Efficient Model-based RL

Jakub Bednarz JB406103@STUDENTS.MIMUW.EDU.PL
University of Warsaw
ul. Banacha 2
02-097 Warszawa

1 Introduction

Reinforcement Learning (RL), a research field dedicated to developing intelligent autonomous agents, capable of learning to do tasks in various environments, has achieved remarkable progress in recent times. Through the use of deep learning and artificial neural networks, algorithms playing video games at a super-human level, controlling robots without any prior knowledge, or (...) have been developed.

However, most of the current state-of-the-art techniques have a significant drawback: they require excessive amounts of data (which, in this setting, we regard as the total time spent interacting with the environment) in order to achieve such performance. For example, standard benchmark on a collection of Atari video games assumes a total "budget" of 200 million game frames, equivalent to a human playing for $\approx$ 40 days without rest. This shortcoming greatly limits real-world deployment of such methods, as without a simulator of the target environment learning the proper behavior is impossible.

In order to remedy this problem, many approaches designed with *sample efficiency*, being the performance obtained with a given (small) number of interactions with the environment, in mind have been proposed, such as:

- Trying to improve sample efficiency of existing algorithms, without any major architectural changes, by carefully increasing the number of optimization steps per single environment step.
- Learning a model of the environment, with which one can either simulate environment interactions for the model-free method, or use the model in a more sophisticated way, e.g. through a planning algorithm.
- Better architectures of the neural networks comprising the agents.

All these have resulted in marked improvements in terms of sample efficiency. One may observe, however, that such methods have been developed independent of each other, and in many cases are orthogonal, meaning it is possible to envision a single RL agent combining many such strategies and modifications, hopefully capable of reaching even higher performance than standalone algorithms.

With this in mind, the goal of this thesis is to answer following questions:

1. *What are the existing methods for improving data efficiency of RL agents?*

2. *How to approach the design of a single algorithm combining these methods?*

3. *What are the synergies and interferences between these modifications?* And, relatedly, *what (if any) improvement in terms of performance can we expect from such an agent?*

The rest of this thesis is structured as follows: (...)

2 Background

2.1 Reinforcement Learning

Reinforcement learning is a field of artificial intelligence and machine learning concerned with constructing intelligent agents capable of learning what actions to take in an environment so as to maximize a scalar reward signal (hence *reinforcement* learning).

Formally, the environment can be represented by a Markov decision chain (MDP) in the form of a tuple $\langle \mathcal{S}, \mathcal{A}, p \rangle$ consisting of the state space $\mathcal{S}$, action space $\mathcal{A}$ and transition probability $p(s_{t+1}, r_t \mid s_t, a_t)$, where $r_t \in \mathbb{R}$ denotes the reward granted at that step. Additionally, in the case of partially observable environments, we have an observation probability $p(o_t \mid s_t)$. The agent, starting from an initial observation o_1, acts according to a probability distribution $\pi(a_t \mid o_{\leq t})$ conditioned on previous observations until it reaches a *terminal* state s_T. The goal of reinforcement learning and the RL agent is to maximize the expected sum of rewards obtained along the trajectory $\mathbb{E}\left\{\sum_{t=1}^{T-1} r_t\right\}$.

2.2 Model-based RL

Whereas a model-free RL agent seeks only to learn the policy π, a model-based agent seeks also to model the transition probability p via p_θ, or, more generally, to be able to simulate future observations $\widetilde{o_{t+1}}, \ldots, \widetilde{o_{t+H}}$ given past observations $o_{\leq T}$ and actions to perform $a_t, \ldots, a_{t+H-1}$.

3 Prior work

3.1 Sample-efficient RL

3.2 Integrated architectures

The main idea of this paper is inspired and motivated by Rainbow architecture and associated paper. In it, the authors took a base off-policy model-free RL algorithm, DQN, applied an array of modifications developed by other authors (specifically: double Q-learning, dueling networks, prioritized sampling, noisy networks, distributional RL and multi-step TD targets), adapted to fit together in a unified architecture, and achieved substantial improvements in terms of final performance of the agent.

Our goal is to develop a framework for sample-efficient RL, in a similar fashion to Rainbow - taking a base architecture, an array of approaches developed to improve sample efficiency, and combine them together into a single architecture. To our knowledge, this has not yet been investigated.

4 Method

4.1 Base architecture

We use Dreamer architecture as a starting point in our inquiry.

The agent consists of two modules:

- A world model. Specifically, the authors use observation encoder for compressing the input into a latent vector, and Recurrent State Space Model (RSSM) for modeling stochastic latent dynamics.
- An RL algorithm. The implementation is, in essence, entropy-regularized Advantage Actor-Critic (A2C) using multi-step predictions for better value estimates.

These are optimized using 1.

Algorithm 1 Dreamer training recipe

1: Gather P env samples with an agent acting randomly, and put them into a replay buffer
2: **while** Total number of env steps $< N$ **do**
3: Fetch a batch of B sequences $(o_1, a_1, o_2, r_1, \ldots, o_L, r_{L-1})$ of length L each from the replay buffer.
4: Optimize the world model using batch of real data. From it, we recover a batch of B sequences $(s_1, \ldots, s_L)$ of latent states, each representing the history of observations $o_{\leq t}$. Thus, we get BL states total.
5: Starting from these states, perform a rollout in imagination, using the RL agent to pick actions and getting next states with the world model. In the end, we get a batch of BL sequences $(s_1, a_1, \widetilde{s_2}, \widetilde{r_1}, \ldots, \widetilde{s_H}, \widetilde{r_{H-1}})$
6: Optimize the RL agent using the batch of dream data.
7: Gather E env steps, and store them in replay buffer
8: **end while**

References