Paper summary: exploration by random network distillation

Marko Guberina 19970821-1256

March 17, 2022

1 Idea in few sentances

2 Explanation of the central concept

3 Methodology

4 Initial rambly notes

4.1 Abstract

Introduces exploration bonus for DRL which is easy to implement and adds minimal computational overhead to computation. The bonus is the error of a neural network which predicts features of observations given by a fixed randomly initialized neural network. The bonus is called random network distillation (RND). Also introduces method which combine intrinsic and extrinsic rewards. Achieve best results up until then on Montezuma's revenge.

4.2 Introduction

It's difficult to engineer dense rewards and sparse rewards are difficult to learn. To deal with that you need an to explicitely explore.

Vectorized environments (running the policy in parallel in the same environment) is great to solve challenging RL tasks. However exploration methods based on (pseudo)counts don't scale well to a large number of parallel environments. The method introduced in the paper only requires a single forward pass through a neural net per 1 experience batch.

Key observation: Neural networks tend to have significantly lower prediction errors on examples similar to those on which they have been trained.

Idea based on the observation Use prediction errors of networks trained on the agent's past experience to quantify the novelty of new experience.

Just using prediction loss when learning forward dynamics makes your agent stare at stochastic elements of the environment. Amending this by quantifying only the relative improvement of the prediction, rather the its absolute error, is hard to implement efficiently.

This method predicts the output of a fixed randomly initialized (deterministic) neural network on the current observation.

The method augments PPO so that it uses two value heads for the two reward streams.

4.3 Method

4.3.1 Random network distillation (RND)

You want to measure novely of states and use that to create "intrinsic reward" which is just added to the usual extrinsic reward. Other approaches do counts, but that's complicated and costly. The short version of this method is the following: instantiate a random network which tries to predict observations and keep it fixed. That's the *target* network and it sets (creates out of thin air) the prediction problem. It's inputs are observations and the output is some vector. Thus the target network provides an embedding $f : \mathcal{O} \to \mathbb{R}^k$. Then create another network and train it to predict the outputs of the first network. That network is the *predictor* $\hat{f} : \mathcal{O} \to \mathbb{R}^k$ and is trained on the MSE loss $||\hat{f}(x;\theta) - f(x)||^2$ with respect to the parameters θ_f. If the loss of the predictor is high, the state is novel and if it's not then the state is not novel. What useful properties does this have over counting? Of course, it's faster, but it also solves the problem of dealing with similar states for free. Namely, the target neural network will have a similar output for similar observations by default — that's just how neural networks are.

Let's look at specific causes of prediction error and see more clearly why this approach works:

1. *epistemic uncertainty* — amount of training data — higher error when there are few samples

2. *aleatoric uncertainty* — the prediction error is high because the target function is stochastic. Stochastic transitions are a source of error in forward dynamics models.

3. *model specification* — necessary information is missing, or the model is too limited to fit the complexity of the target function

4. *learning dynamics* — the optimizer fails to find a good predictor in the given model class

You really want only the first factor to contribute to the exploration bonus, but of course the prediction error will be the combination of all factors. There are costly ways to deal with 2 and 3, but RND ameliorates them by using a deterministic target network which is in the same class as the predictor network.

There's a connection between RND and uncertainty quantification introduced in another paper. Basically it's connected to regression in a Bayesian setting.

It's better not to give extrinsic reward at the end of episodes, but instead to do non-episodic returns. Also train prediction the values of extrinsic and intrinsic returns in the same network, but on different heads. Some reasonable but hand-wavy explanations for this. Also it's necessary to normalize (and clip) observations if you want this to work.

4.4 Other stuff

Look at the paper if you want to check out best discount rates, algorithm pseudocode etc.