

Notes for: Exploration by Random Network Distillation

M. Nef. *

July 4, 2020

1 Preliminaries

1.1 What's the problem?

Exploration is challenging. This is especially true when there is sparse rewards (since there's not much guidance for the agent). ϵ -greedy methods are suuuper sketchy when you think about them, since you explore states at random. Not only that, but the states you visit during exploration are going to be relatively close to the ones you'd visit anyway (so it would be hard to say if the exploration was actually valuable).

Example situation: Motezuma's Revenge, this is an incredibly complicated game with very sparse rewards, it can even be challenging for some people. The agent needs to learn how to manoeuvre around the environment well (since it's platformer game). Items you find later in the game can open up multiple options, etc, etc...

We need a principled way to explore. There should be some method to the agent's (exploratory) madness.

Extra little note: (didn't know where to slip this in). If we do come up with a principled way of performing exploration it might be nice to allow for off policy data to be incorporated (so we look at off policy data and understand we don't really need to explore these states as much).

1.2 Solution(s)

1. Forward Dynamics approaches
2. Inverse Dynamics approaches
3. Random Network Distillation

Before RND came forward dynamics... The RND procedure is pretty straightforward but it'll be nice to know what came before.

Suppose we want our agent to explore, it's a pretty intuitive idea to say "well we should somehow reward the agent for exploration to incentivise this behaviour". Then we ask, what does it mean to explore (from the agent's perspective)? An interpretation may be that exploration is the behaviour of taking actions or going into states which you're unfamiliar with. If our agent could confidently and correctly predict what the environment looks like after it takes an action then intuitively it probably doesn't gain much from exploring that state/action.

If we trained a model to predict what the next observation would be based on the current state and action, the error of this model could be used as a reward for the policy network. When this model makes poor predictions, it's likely due to the fact that the agent hasn't been exposed to these states very much.

This is the principle behind exploration via forward dynamics.

*paper: <https://arxiv.org/abs/1810.12894>

Inverse dynamics I'm not familiar with using inverse dynamics for exploration. But the inverse dynamics problem is about predicting what (forces) caused a particular "body" to move/change. Regardless, it's not necessary to understanding RNDs.

Why do we need RND then? As said before, in the forward dynamics scheme, if the agent can't predict what the next state will look like it will be rewarded. Therefore, if it encounters some source of randomness in an environment (e.g. a coin flip or static TV) which the forward dynamics model can not predict, then the agent will be heavily rewarded for watching the static tv or coin toss (even if it's experienced it hundreds of times before). If only there was a way to measure

2 Random Distillation Networks

In the RDN architecture we have a function which performs a complicated, non-trivial transformation on an observation to produce some weird transformed result. This function is a randomly initialised neural network.

$$f : \mathcal{O} \rightarrow \mathbb{R}^k \quad (1)$$

Using a similar principle to before, we know that an agent has already "explored" a particular state if it's had lots of experience in that state (or similar states - otherwise we end up at the static TV problem). Using the assumption that a neural network will learn to correctly transform inputs to outputs (given enough samples), if we train a neural network to *predict* the output of f given the input, \mathcal{O} we would expect that the prediction error will decrease as we gain similar observation samples. We will therefore use the prediction error as an intrinsic reward for exploration.

The "predictor" function:

$$\hat{f} : \mathcal{O} \rightarrow \mathbb{R}^k \quad (2)$$

As we train \hat{f} we distil the randomly initialised network into the trained one.

Why does this solve the static TV problem? Because similar inputs are transformed into similar outputs (even though the transformation process is very complicated).

Couldn't the predictor perfectly match \hat{f} given enough samples? Yep, that just doesn't happen empirically (according to their tests).

And that's basically it... It's a very compact idea.

3 Implementation

3.1 Combining intrinsic + extrinsic rewards

Since we want to balance exploration and exploitation we need a way to balance our intrinsic (exploratory) rewards and our extrinsic (exploitatory?) rewards.

A unique insight is that intrinsic rewards shouldn't be episodic and instead should persist even after a terminal state, this is because the whole concept of exploration is about discovering different states overall (rather than just within a single trajectory). The consequence of having non-episodic intrinsic rewards is that the agent may be incentivised to:

- suicide itself meaning it can go and check out something super novel near the start of the environment.
- A negative consequence of using episodic intrinsic rewards is that since the return is 0 at the end of the episode the agent could become overly risk averse (because it thinks that episode termination is the end of the world) the truth is that the cost of the episode ending is the wasted time to get back to that point.

3.2 Normalisation

Since the random network can't change, and therefore can't adapt to strangely scaled data, all the data entering it should be normalised. We just store a running mean and running standard deviation of the observation vector and then normalise using these. Consequently we need to collect a few samples initially (just to get a good mean + std). The same normalised data that's fed into f goes into \hat{f} .

the end :)