Exploration in Deep RL

UBC in Deep RL

How to integrate UBC bonus in an MDP?
Instead of usual $r(s,a)$, provide modified $r*(s,a) = r(s,a) + B(N(s))$ where $B(N(s))$ decreases as $N(s)$ increases
Extrinsic reward: r_t^e
Intrinsic reward: r_t^i
Intrinsic reward bonus(hyperparameter): β
Total reward:

$$r_t = r_t^e + \beta r_t^i$$

Count-based Exploration

Introduce a metric that measures how novel a state appears to the agent. Let $N_n(s)$ be the empirical count of visits to a state s in the sequence $s_{1:n}$
In high-dimensional and continuous state spaces, such counting is useless as we will never be able to see all states.
Some states are more similar than others

Density Models for counting

1. Fit a density model $p(s:\theta)$ to approximate the frequency of visits
2. Derive a pseudo count from the model:
   True density at time T: $p(s) = \frac{N(s)}{n}$
   True density at time T+1: $p'(s) = \frac{N(s)+1}{n+1}$
   How to get the density model to satisfy the 2 conditions above?
   STEPS:
3. Fit model $p_\theta(s)$ to all states D seen so far
4. Take a step i and observe s_i
5. Fit new model $p_{\theta'}(s)$ to $D \cup s_i$
6. Use $p_\theta(s_i)$ and $p'_\theta(s_i)$ to estimate $\bar{N}(s)$
7. Set $r_i^+ = r_i + B(\bar{N}(s))$ where B(N(s)) is the bonus?
   How to get $\bar{N}(s)$? Use the equations and solve for 2 unknowns

$$\bar{N}(s_i) = \bar{n} p_\theta(s_i)$$

$$\bar{n} = \frac{1 - p_{\theta'}(s_i)}{p_{\theta'}(s_i) - p_\theta(s_i)} * p_\theta(s_i)$$

   Bonus options:

- Upper Confidence Bounds: $\mathcal{B}(\hat{N}(s)) = \sqrt{\frac{2 \log t}{\hat{N}(s)}}$
- MBIE-EB[1,2]: $\mathcal{B}(\hat{N}(s)) = \sqrt{\frac{1}{\hat{N}(s)}}$
- BEB[3]: $\mathcal{B}(\hat{N}(s)) = \frac{1}{1+\hat{N}(s)}$

Density model options: we only need rough densities(no need for accuracy) nor sampling.

Counting with hashing
What if we still count states, but in a different space?
Idea: compress s into a k-bit code via $\phi(s)$, then count $N(\phi(s))$
Shorter codes = more harsh collisions. Similar states get similar hashes?
Locality-Sensitive Hashing
Hashing scheme that preserves the distancing information between data points

Larger k's lead to higher granularity and fewer collisions.
Better method is to learn **a compression** via autoencoders (better handles high-dimensional states such as images). A special dense layer uses k sigmoid functions in the latent space to generate a binary activation map

Prediction-based Exploration

Forward-dynamics prediction model
Derive a reward bonus based on the prediction error of the dynamics model
IAC: split the state space into regions and train a separate dynamics model on the data from this region. With each new action, the prediction error of the dynamics model is calculated using MSE and put in a sliding window to associated to that region

Deep Predictive Models: train a dynamics model on the latent space instead. Additionally, normalize the latent prediction error by the maximum error so far. Do the same normalization for extrinsic reward as well. The autoencoder can be trained using images collected by a random policy or together with the changing policy.

ICM: instead of an autoencoder, train state space encoding $\phi(s_t)$ with a proxy self-supervised **inverse dynamics** model. This low-dimensional feature space would have no incentive to learn the spurious or non-agent related environment changes:

$$g : (\phi(s_t), \phi(s_{t+1})) \rightarrow a_t$$

Agent is given current and previous state and it should predict the action. Now, given this learned feature space, train a forward dynamics model that predicts the feature representation of the next state, given the feature representation of current state and action.

Then, they provide the prediction error of the forward dynamics model to the agent as IM.
Overall, they train two networks: state space encoding forward dynamics model and inverse dynamics model
If given only intrinsic reward, train an A3C.
Their insight is transforming the original feature space (i.e .images) into a lower-dimensional representation, such that the new feature space represents **only information relevant to the action from the agent**. The feature space ignores environmental changes that aren't affected by the agent's actions. If this were Breakout for instance, one example would be the changes in environment that happen when the ball is in the back and hitting everything without the agent moving at all, though this might be hard to detect.
Limitation of Prediction Error: noisy TV problem

The intrinsic reward is defined as the error between an MLP forward model's next state prediction and the actual next state. As this method needs access to the true next state, the intrinsic reward can only **be computed retrospectively.**

VIME: VIME provides a better exploration strategy than the heuristics such as epsilon-greedy methods by ensuring that the agent selects actions that **maximize information gain** about the environment. They use a **Bayesian NN** so that they can get posterior estimates of θ
They use information theory and optimizing a variational lower bound to train their BNN. It's algebraically **a bit more tractable** than one would expect since they assume they can factorize θ into the product of univariate Gaussians (**Variational Inference**).

Self-supervised Exploration via Disagreement
Use ensemble disagreement as bonus. Intrinsic reward is differentiable. This approach doesn't get stuck in stochastic-dynamics scenarios because all the models in the ensemble converge to mean, eventually reducing the variance of the ensemble.

Random Network Distillation(RND): First model to achieve better than average human performance without using demonstration or simulation model(OpenAI).
The prediction error has several issues:

1. The agent is attracted to highly stochastic states, such as noisy TV.
2. Predicting raw pixels are hard
   To get around these, they use prediction based on a **fixed, randomly initialized network** so that the prediction problem is deterministic, not stochastic.
   Two neural networks are used:

1. A randomly initialized but **fixed** network to transform a state into a feature space $f(s_t)$
2. A network $\tilde{f}(s_t;\theta)$ that is trained to predict the same features as the fixed network
   In the end goal is to have $\bar{f}(s_t;\theta) = f(s_t)$ and the exploration bonus is the error between the two.
   Overall, the fixed network makes the prediction target deterministic. RND works well for hard exploration and non-episodic tasks especially if only intrinsic rewards are used.

Memory-based Exploration

Go-Explore

IM methods do a poor job at continuing to explore promising areas far away from the start state. This is called Detachment.

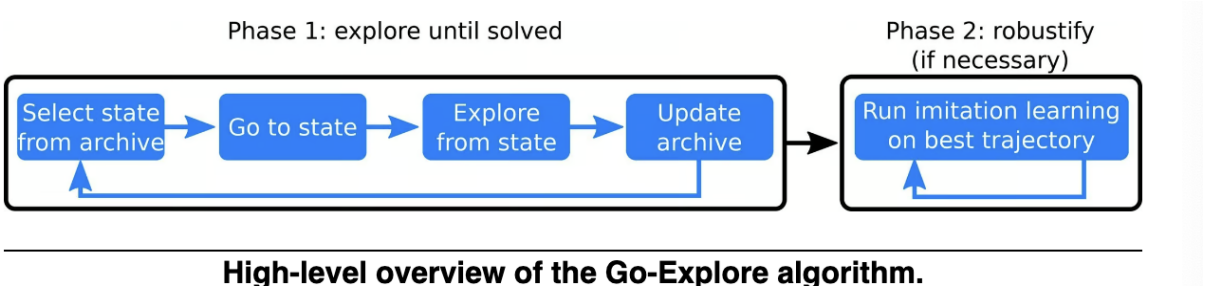
Detachment: catastrophic forgetting. IM algorithms forget about promising areas they have visited, meaning they do not return to them to see if they lead to new states. How to make the agent **return t**o previously discovered **promising states** for further exploration?

Solution: maintain a memory of previously visited novel states. No IM here. Just randomly explore from the previously visited state.

Derailment: The trajectories learned from this approach can be brittle to small action deviations. To make it robust, we can do self-imitation learning, where we take trajectories in a deterministic environment, and learn to reproduce them in randomized versions of the environment

Solve these two problems:

1. Explore until solved
2. Robustify



High-level overview of the Go-Explore algorithm.

Cell representation: downscaling game frame is enough!

Returning to cells: there are 3 options:

- in a *resettable* environment, simply reset the environment state to that of the cell.
- in a *deterministic* environment, replay the trajectory to the cell.
- in a *stochastic* environment, train a goal-conditioned policy that learns to reliably return to a cell.
  Interesting thing about Go-Explore is that, we can **first** solve the problem, and **then** deal with making the solution more robust later.

Check this blogpost by Uber about Go-Explore