A Regression Approach and Stochastic traps

A well known vanilla uncertainty-based exploration method consists in predicting future targets z_{t+1} from a history-representation b_t of past observations-actions and future open-loop actions a_t via regression. This method is referred as a one-step prediction error method at the latent-level if z_{t+1} is a function of the observation or at the observation-level if $z_{t+1} = o_{t+1}$. The representation b_t can be learned via a representation learning method or simply be, in a toy scenario, $b_t = (o_t, a_{t-1})$. In any case, using a simple regression technique to compute the intrinsic rewards will lead towards trivial behaviors if the underlying dynamics $z_{t+1} \sim p(.|b_t, a_t)$ is stochastic. Indeed, let g_θ be a parameterized predictor that is trained to predict z_{t+1} with inputs (b_t, a_t) with the following regression loss:

$$\mathcal{L}_{\texttt{Reg}}(\theta, z_{t+1}) = \mathbb{E}\left[\|z_{t+1} - g_\theta(b_t, a_t)\|_2^2\right].$$

Then, if g_θ is expressive enough, we have:

$$\arg\min_\theta \mathcal{L}_{\texttt{Reg}}(\theta, z_{t+1}) = \mathbb{E}[z_{t+1}|(b_t, a_t)],$$
$$\min_\theta \mathcal{L}_{\texttt{Reg}}(\theta, z_{t+1}) = \mathbb{E}\left[\|z_{t+1} - \mathbb{E}[z_{t+1}|(b_t, a_t)]\|_2^2\right] = \texttt{Var}[z_{t+1}|(b_t, a_t)].$$

As the intrinsic rewards are derived from the prediction loss, we see that when the loss is minimized, we introduce the variance of the future-target distribution in the intrinsic rewards. This added variance is problematic because it encourages the agent to seek for states with uncertain future outcomes and stay there. Those states are known as stochastic traps.

B General `BYOL-Explore` and `BLaDE` Architecture

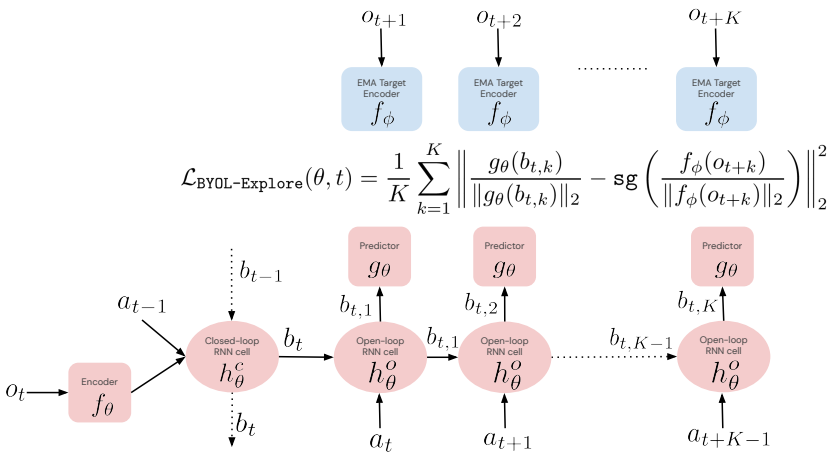

Figure 10: `BYOL-Explore`'s Neural Architecture.

The online network is composed of:

- Encoder: $f_\theta : \mathcal{O} \to \mathbb{R}^N$
- Close-loop RNN cell: $h_\theta^c : \mathbb{R}^M \times \mathbb{R}^N \times \mathcal{A} \to \mathbb{R}^M$
- Open-loop RNN cell: $h_\theta^o : \mathbb{R}^M \times \mathcal{A} \to \mathbb{R}^M$
- In the case of `BYOL-Explore`, the predictor is: $g_\theta : \mathbb{R}^M \to \mathbb{R}^N$
- In the case of `BLaDE` the predictor is: $g_\theta : \mathbb{R}^{M+N+L} \to \mathbb{R}^N$

The target network is composed of:

- EMA encoder: $f_\phi : \mathcal{O} \to \mathbb{R}^N$

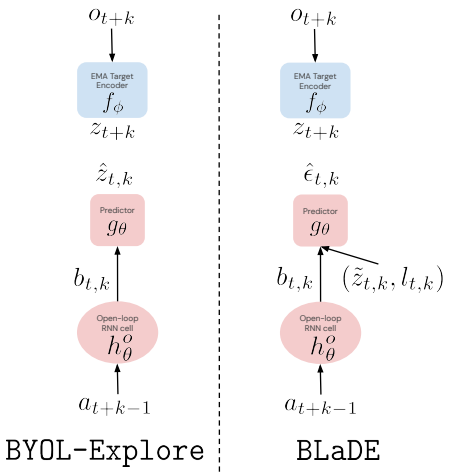

Figure 11: `BLaDE`'s and `BYOL-Explore`'s predictor architectures.

B.1 Detailed `BYOL-Explore` and `BLaDE` architecture for `Atari`

In `Atari`, the size of the observation-representation $N = 512$ and the size of the history-representation $M = 256$.

- Encoder: $f_\theta : \mathcal{O} \to \mathbb{R}^N$: The encoder is instantiated as a Deep ResNet [12] stack. The greyscale image observation is passed through a stack of 3 units, each comprised of a 3×3 convolutional layer, a 3×3 maxpool layer, and 2 residual blocks. The number of channels for the convolutional layer and the residual blocks are 16, 32, and 32 within each of the 3 units respectively. We use GroupNorm normalization [31] with one group at the end of each of the 3 units, and use ReLU activations everywhere. The output of the final residual block is flattened and projected using a single linear layer to an embedding of dimension 512.
- Close-loop RNN cell: $h^c_\theta : \mathbb{R}^M \times \mathbb{R}^N \times \mathcal{A} \to \mathbb{R}^M$ is a simple Gated Recurrent Unit (GRU) [5]. We provide the past-action to the close-loop RNN cell, embedded into a representation of size 32.
- Open-loop RNN cell: $h^o_\theta : \mathbb{R}^M \times \mathcal{A} \to \mathbb{R}^M$ is a simple Gated Recurrent Unit. We provide the past-action to the open-loop RNN cell, embedded into a representation of size 32.
- Policy head $\pi_\psi : \mathbb{R}^N \to \mathbb{R}^{|\mathcal{A}|}$, value head $v_\psi : \mathbb{R}^N \to \mathbb{R}$. The outputs of the policy head are passed through a softmax layer to form the probabilities for each action to be taken.
- The predictor for `BYOL-Explore` $g_\theta : \mathbb{R}^M \to \mathbb{R}^N$ is a simple Multi-Layer Perceptron (MLP) with three hidden layer of size $(512, 512, 512)$.
- The predictor for `BLaDE` $g_\theta : \mathbb{R}^{M+N+L} \to \mathbb{R}^N$ is a simple Multi-Layer Perceptron (MLP) with three hidden layer of size $(512, 512, 512)$.

B.2 Details of Reward Normalization Mechanism

We use a similar reward normalization scheme as in `RND` [4] and normalize the raw rewards $((\ell^j_t)^{T-2}_{t=0})^{B-1}_{j=0}$ by an EMA estimate of their standard deviation.

More precisely, we first set the EMA mean to $\overline{r} = 0$, the EMA mean of squares to $\overline{r^2} = 0$ and the counter to $c = 1$. Then, for the c-th batch of raw rewards $((\ell^j_t)^{T-2}_{t=0})^{B-1}_{j=0}$, we compute the batch mean $\overline{r_c}$ and the batch mean of squares $\overline{r^2_c}$:

$$\overline{r_c} = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} \ell^i_t, \qquad \overline{r^2_c} = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} (\ell^i_t)^2.$$

We then update $\overline{r}$, $\overline{r^2}$ and c:

$$\overline{r} \leftarrow \alpha_r \overline{r} + (1-\alpha_r)\overline{r_c}, \qquad \overline{r^2} \leftarrow \alpha_r \overline{r^2} + (1-\alpha_r)\overline{r_c^2}, \qquad c \leftarrow c+1,$$

where $\alpha_r = 0.99$. We compute the adjusted EMA mean μ_r, the adjusted EMA mean of squares μ_{r^2}:

$$\mu_r = \frac{\overline{r}}{1-\alpha_r^c}, \qquad \mu_{r^2} = \frac{\overline{r^2}}{1-\alpha_r^c}.$$

Finally the EMA estimation of the standard deviation is $\sigma_r = \sqrt{\max(\mu_{r^2} - \mu_r^2, 0) + \epsilon}$, where $\varepsilon = 10^{-8}$ is a small numerical regularization. The normalized rewards are $r_{i,t}^j = \ell_t^j / \sigma_r$.

C Baselines

Random Network Distillation (`RND`) [4] is a simple exploration method that consists in training an encoder such that its outputs fit the outputs of *another* fixed and randomly initialized encoder and using the training loss as an intrinsic reward to be optimized by an RL algorithm. More precisely, let $N \in \mathbb{N}^*$ be the embedding size and let us note $f_\theta : \mathcal{O} \to \mathbb{R}^N$ the encoder, also called predictor network, with trainable weights θ and $f_\phi : \mathcal{O} \to \mathbb{R}^N$ the fixed and randomly initialized encoder, also called target network, with fixed weights ϕ. In addition, let us suppose that we have a batch of trajectories $\left((o_t^j, a_t^j)_{t=0}^{T-1}\right)_{j=0}^{B-1}$ collected by our RL agent, then the loss $\mathcal{L}_{\texttt{RND}}(\theta)$ to minimize w.r.t. the online network parameters is defined as:

$$\mathcal{L}_{\texttt{RND}}(\theta, j, t) = \|f_\theta(o_t^j) - \texttt{sg}(f_\phi(o_t^j))\|_2^2, \quad \mathcal{L}_{\texttt{RND}}(\theta) = \frac{1}{BT}\sum_{j=0}^{B-1}\sum_{t=0}^{T-1}\mathcal{L}_{\texttt{RND}}(\theta, j, t),$$

and the unnormalized reward associated to the transition $(o_t^j, a_t^j, o_{t+1}^j)$ is defined as $\ell_t^j = \mathcal{L}_{\texttt{RND}}(\theta, j, t+1)$ where $0 \leq t \leq T-2$. To obtained the final intrinsic rewards, we just normalize them to be as close as possible to the original `RND` implementation:$r_{i,t}^j = \frac{\ell_t^i}{\sigma_r}$.

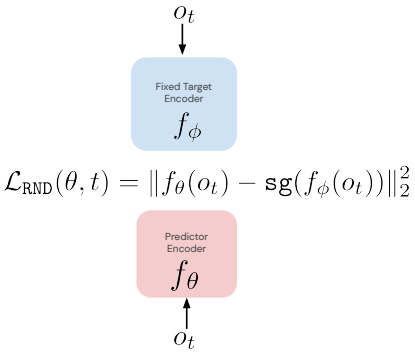

Figure 12: `RND`'s Neural Architecture.

Intrinsic Curiosity Module (`ICM`) [24] is a one-step prediction error method at the latent level. It consists in training an encoder $f_\theta : \mathcal{O} \to \mathbb{R}^N$ that outputs a representation that is robust to uncontrollable aspects of the environment and then use this representation as inputs of a one-step prediction error model $g_\phi : \mathbb{R}^N \times \mathcal{A} \to \mathbb{R}^N$ which error is used as an intrinsic reward to be optimized by an RL algorithm. To build a representation robust to uncontrollable dynamics, the idea used in `ICM` is to train an inverse dynamics model $p_\theta : \mathbb{R}^N \times \mathcal{R}^N \to \mathcal{A}$ that predicts the distribution of actions that led to the transition between two consecutive representations $f_\theta(o_t), f_\theta(o_{t+1})$. More precisely,

let us suppose that we have a batch of trajectories $\left((o_t^j, a_t^j)_{t=0}^{T-1}\right)_{j=0}^{B-1}$ collected by our RL agent, then the loss $\mathcal{L}_{\texttt{INV}}(\theta)$ to minimize in order to train our encoder and inverse dynamcis model is:

$$\mathcal{L}_{\texttt{INV}}(\theta, j, t) = -\ln\left(p_\theta(a_t^j | f_\theta(o_t^j), f_\theta(o_{t+1}^j))\right), \quad \mathcal{L}_{\texttt{INV}}(\theta) = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} \mathcal{L}_{\texttt{INV}}(\theta, j, t),$$

which is a simple cross-entropy loss. Simultaneously, ICM also trains the one step prediction error model by minimizing the following one-step prediction loss:

$$\mathcal{L}_{\texttt{ICM}}(\phi, j, t) = \|g_\phi(f_\theta(o_t^j), a_t^j) - \texttt{sg}(f_\theta(o_{t+1}^j))\|_2^2, \quad \mathcal{L}_{\texttt{ICM}}(\phi) = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} \mathcal{L}_{\texttt{ICM}}(\theta, j, t),$$

and the unnormalized reward associated to the transition $(o_t^j, a_t^j, o_{t+1}^j)$ is defined as $\ell_t^j = \mathcal{L}_{\texttt{ICM}}(\theta, j, t)$ where $0 \leq t \leq T-2$. To obtained the final intrinsic rewards, we just normalize them :$r_{i,t}^j = \frac{\ell_t^j}{\sigma_r}$.

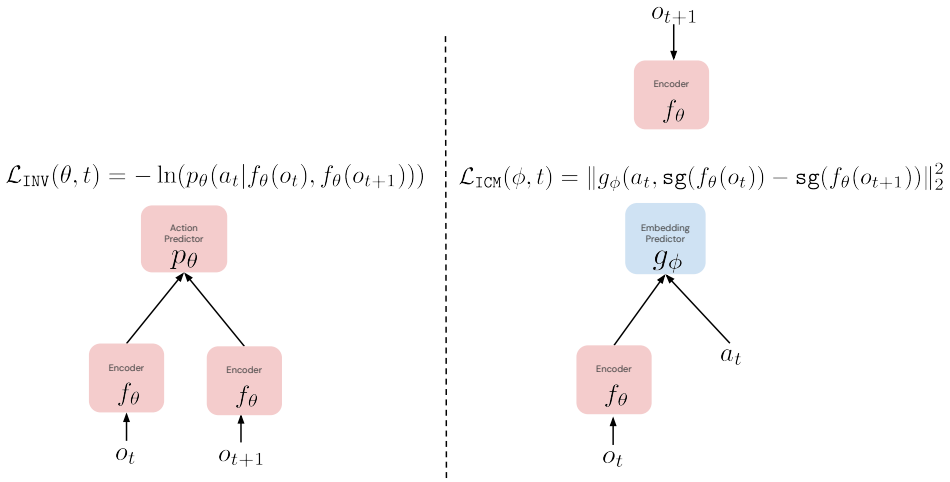

Figure 13: ICM's Neural Architecture.

D Hyperparameter Settings

After normalizing the rewards, we rescale them by $1 - \gamma$. We similarly use PopArt normalization on the output of the value network. We choose an horizon $K = 8$. We use a discount factor of $\gamma = 0.999$. To train the value function, we use VTrace [8] without offpolicy corrections to define TD targets for MSE loss with a loss weight of 0.5. We add an entropy loss with a loss weight of 0.001. The VMPO parameters η_{init} and α_{init} are initialized to 0.5. ϵ_η and ϵ_α are set to 0.01 and 0.005 respectively. We scale the BYOL loss by a factor of 5.0 when combining losses. The VMPO top-k parameter is set to 0.5. We use the Adam optimizer with learning rate 10^{-4} and $b_1 = 0.9$. The target network for VMPO is updated every 10 learner steps.

We use a batch size of 32 and a sequence length of 128; and a distributed learning setup using 4 TPUv2 for learning and 400 CPU actors for generating data via another inference server using 4 TPUv2 to evaluate the policy, similar to Agent57 [1].