BLaDE: Robust Exploration via Diffusion Models

Zhaohan Daniel Guo[*]
DeepMind
danielguo@deepmind.com

Shantanu Thakoor[*]
DeepMind
thakoor@deepmind.com

Bilal Piot[*]
DeepMind
piot@deepmind.com

Mohammad Gheshlaghi Azar[*]
DeepMind
mazar@deepmind.com

Abstract

We present Bootstrap your own Latents with Diffusion models for Exploration (BLaDE), a general approach for curiosity-driven exploration in complex, partially-observable and stochastic environments. BLaDE is a natural extension of Bootstrap Your Own Latents for Exploration (BYOL-Explore) which is a multi-step prediction-error method at the latent level that learns a world representation, the world dynamics, and provides an intrinsic-reward all-together by optimizing a single prediction loss with no additional auxiliary objective. Contrary to BYOL-Explore that predicts future latents from past latents and future open-loop actions, BLaDE predicts, via a diffusion model, future latents from past observations, future open-loop actions and a noisy version of future latents. Leaking information about future latents allows to control the variance of the distribution of future latents which makes the method agnostic to stochastic traps. Our experiments on different noisy versions of Montezuma's Revenge show that BLaDE handles stochasticity better than Random Network Distillation, Intrinsic Curiosity Module and BYOL-Explore without degrading the performance of BYOL-Explore in the non-noisy and fairly deterministic Montezuma's Revenge.

1 Introduction

Real-world environments (i) are complex (with high-dimensional and noisy observations) and (ii) have stochastic dynamics. These properties can cause vanilla uncertainty-based and count-based exploration methods to fail and generate trivial behavior. In this paper, we focus on two important types of stochasticity that are omnipresent in the real world and relate to the properties described earlier. First, (i) stochasticity at the observation-level due to sensor noise and complexity of the environment that is independent of the underlying dynamics—for example, observations with white noise [24]. Second, (ii) stochasticity at the action-level due to imperfect actuators or to inherently stochastic dynamics—for example, sticky actions [17].

Stochasticity of type (i) gives rise to what is popularly known as *the TV noise problem* [24]. This type of stochasticity is inherently difficult for vanilla count-based methods because noise at the observation level may dramatically increase the number of possible observations. This could render the exploration method ineffective if there is no representation learning method that helps filtering the noise from the observation. Stochasticty of type (ii) causes the phenomenon commonly known as *stochastic traps* [25]. For instance, methods based on future targets regression (minimizing the

[*]Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

squared-norm error between future predictions and future targets) to generate intrinsic rewards are easily stuck in states where the actions have controllable but high-variance future outcomes such as throwing a dice or playing roulette. Indeed, those methods will converge towards predicting the expectation of future targets and their intrinsic rewards towards the variance of the future-target distribution (see App. A for a detailed explanation). This incompressible variance term in future-target predictions is often referred as aleatoric uncertainty.

Theoretically, one well-known, sound and general method to deal with stochasticities of type (i) and (ii) is to use ensemble methods [25, 28, 27, 19, 23, 22]. However, practically, those methods are hard to train and scale. Indeed, they require careful initialisation and data-masking of the different networks of the ensemble to avoid all the networks to converge towards the same values to quickly [22]. Besides, by nature, they do require several networks to form the ensemble which may not scale for bigger architectures. In this paper, we chose another direction to deal with stochasticities of type (i) and (ii) that is scalable. Our idea is very simple and consists in modifying a state-of the art prediction-error method at the latent level called `BYOL-Explore` [11]. Specifically, instead of using a regression approach to predict future (latent) targets, our approach, called Bootstrap you own Latents with Diffusion models for Exploration (`BLaDE`), uses a generative approach via a diffusion model where noisy future targets are provided as inputs in addition to a representation of past observations-actions and future open-loop actions. First, this allows to fully model the distribution whereas a regression approach only model the expectation which is problematic to avoid stochastic traps. Second, by providing bits of information of the future targets to the diffusion model, it becomes possible to predict future targets and considerably reduce the variance of future targets (aleatoric uncertainty). This allows to generate intrinsic rewards that are agnostic to stochasticity. In our experiments, we show that `BLaDE` can perform well under stochasticities of type (i) and (ii) whereas other state-of-the art exploration methods such as Random Network Distillation (`RND`) [4] fails under stochaticity of type (i) and `BYOL-Explore` [11] fails under stochaticity of type (ii). In addition, `BLaDE` retains the performance of `BYOL-Explore` on environments with deterministic dynamics.

2 Method

2.1 Background and Notation

We consider a discrete-time interaction process [18, 15, 16, 7] between an agent and its environment where, at each time step $t \in \mathbb{N}$, the agent receives an observation $o_t \in \mathcal{O}$ and generates an action $a_t \in \mathcal{A}$. We consider an environment with stochastic dynamics $p : \mathcal{H} \times \mathcal{A} \to \Delta_{\mathcal{O}}$[2] that maps a history of past observations-actions and a current action to a probability distribution over future observations. More precisely, the space of past observations-actions is $\mathcal{H} = \bigcup_{t\in\mathbb{N}} \mathcal{H}_t$ where $\mathcal{H}_0 = \mathcal{O}$ and $\forall t \in \mathbb{N}^*, \mathcal{H}_{t+1} = \mathcal{H}_t \times \mathcal{A} \times \mathcal{O}$. We consider policies $\pi : \mathcal{H} \to \Delta_{\mathcal{A}}$ that maps a history of past observations-actions to a probability distribution over actions. Finally, an extrinsic reward function $r_e : \mathcal{H} \times \mathcal{A} \to \mathbb{R}$ maps a history of past observations-actions to a real number.

2.2 Reminder of `BYOL-Explore`

`BYOL-Explore` world model is a multi-step predictive world model operating at the latent level. It is inspired by the self-supervised learning method `BYOL` [10] in computer vision and adapted to interactive environments (see Section 2.1). Similar to `BYOL`, `BYOL-Explore` model trains an online network using targets generated by an exponential moving average (EMA) target network. However, `BYOL` obtains its targets by applying different augmentations to the same observation as the online representation, whereas `BYOL-Explore` model gets its targets from future observations processed by an EMA of the online network, with no hand-crafted augmentation. Also `BYOL-Explore` model, uses a recurrent neural network (RNN) [14, 6] to build the agent state, i.e., the state of RNN, from the history of observations, whereas the original `BYOL` only uses a feed-forward network for encoding the observations. In the remainder of this section, we will explain: (i) how the online network builds future predictions, (ii) how targets for our predictions are obtained through a target network, (iii) the loss used to train the online network, and (iv) how we compute the uncertainties of the world model.

[2] We write $\Delta_{\mathcal{Y}}$ the set of probability distributions over a set $\mathcal{Y}$.

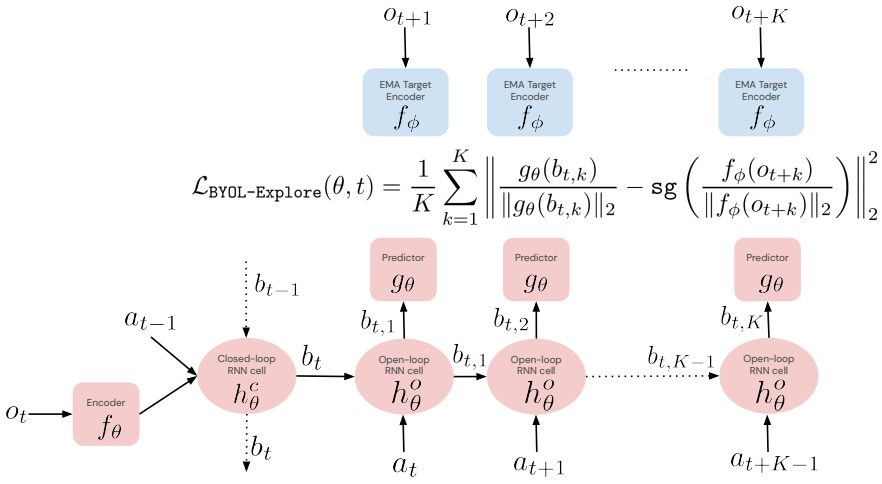


Figure 1: BYOL-Explore's Neural Architecture (see main text for details).

(i) Future Predictions. The online network is composed of an encoder f_θ that transforms an observation o_t into an observation-representation $f_\theta(o_t) \in \mathbb{R}^N$, where $N \in \mathbb{N}^*$ is the embedding size. The observation-representation $f_\theta(o_t)$ is then fed alongside the previous action a_{t-1} to a RNN cell h_θ^c that is referred as the close-loop RNN cell. It computes a representation $b_t \in \mathbb{R}^M$ of the history $h_t \in \mathcal{H}_t$ seen so far as $b_t = h_\theta^c(b_{t-1}, a_{t-1}, f_\theta(o_t))$, where $M \in \mathbb{N}^*$ is the size of the history-representation. Then, the history-representation b_t is used to initialize an open-loop RNN cell h_θ^o that outputs open-loop representations $(b_{t,k} \in \mathbb{R}^M)_{k=1}^{K-1}$ as $b_{t,k} = h_\theta^o(b_{t,k-1}, a_{t+k-1})$ where $b_{t,0} = b_t$ and K is the open-loop horizon. The role of the open-loop RNN cell is to *simulate* future history-representations while observing only the future actions. Finally, the open-loop representation $b_{t,k}$ is fed to a predictor g_θ to output the open-loop prediction $\hat{z}_{t,k} = g_\theta(b_{t,k}) \in \mathbb{R}^N$ at time $t+k$ that plays the role of our future prediction at time $t+k$.

(ii) Targets and Target Network. The target network is an observation encoder f_ϕ whose parameters are an EMA of the online network's parameters θ. It outputs targets $z_{t+k} = f_\phi(o_{t+k}) \in \mathbb{R}^N$ that are used to train the online network. After each training step, the target network's weights are updated via an EMA update $\phi \leftarrow \alpha\phi + (1-\alpha)\theta$ where α is the target network EMA parameter. A sketch of the neural architecture is provided in Fig. 1, with more details in App. B.

(iii) Online Network Loss Function. Suppose our RL agent collected a batch of trajectories $\left((o_t^j, a_t^j)_{t=0}^{T-1}\right)_{j=0}^{B-1}$, where $T \in \mathbb{N}^*$ is the trajectory length and $B \in \mathbb{N}^*$ is the batch size. Then, the loss $\mathcal{L}_{\texttt{BYOL-Explore}}(\theta)$ to minimize is defined as the average cosine distance between the open-loop future predictions $\hat{z}_{t,k} = g_\theta(b_{t,k}^j)$ and their respective targets $z_{t+k} = f_\phi(o_{t+k}^j)$ at time $t+k$:

$$\mathcal{L}_{\texttt{BYOL-Explore}}(\theta, j, t, k) = \left\|\frac{g_\theta(b_{t,k}^j)}{\|g_\theta(b_{t,k}^j)\|_2} - \texttt{sg}\left(\frac{f_\phi(o_{t+k}^j)}{\|f_\phi(o_{t+k}^j)\|_2}\right)\right\|_2^2,$$

$$\mathcal{L}_{\texttt{BYOL-Explore}}(\theta, j, t, k) = \left\|\frac{\hat{z}_{t,k}^j}{\|\hat{z}_{t,k}^j\|_2} - \texttt{sg}\left(\frac{z_{t+k}^j}{\|z_{t+k}^j\|_2}\right)\right\|_2^2,$$

$$\mathcal{L}_{\texttt{BYOL-Explore}}(\theta) = \frac{1}{B(T-1)}\sum_{j=0}^{B-1}\sum_{t=0}^{T-2}\frac{1}{K(t)}\sum_{k=1}^{K(t)}\mathcal{L}_{\texttt{BYOL-Explore}}(\theta, j, t, k),$$

where $K(t) = \min(K, T-1-t)$ is the valid open-loop horizon for a trajectory of length T and $\texttt{sg}$ is the stop-gradient operator.

(iv) World Model Uncertainties The uncertainty associated to the transition $(o_t^j, a_t^j, o_{t+1}^j)$ is the sum of the corresponding prediction losses:

$$\ell_t^j = \sum_{p+q=t+1} \mathcal{L}_{\texttt{BYOL-Explore}}(\theta, j, p, q),$$

where $0 \leq p \leq T-2$, $1 \leq q \leq K$ and $0 \leq t \leq T-2$. This accumulates all the losses corresponding to the world-model uncertainties relative to the observation o_{t+1}^j. Thus, a timestep receives intrinsic reward based on how difficult its observation was to predict from past partial histories. We use normalized versions of the world model uncertainties ℓ_t^j as intrinsic rewards $r_{i,t}^j$ see App. B.2.

2.3 Reminder of Diffusion Models

Fundamentally, one can see training a diffusion model [13, 21, 26] as training a denoiser g_θ. Provided a context c and a noisy input $\tilde{z}$ of the original input z, the goal of the denoiser is to output z or equivalently the noise ϵ. The context c provides additional information to the denoiser in order to denoise $\tilde{z}$ into z. In practice, we use noisy inputs of the form:

$$\tilde{z} = \sqrt{\overline{\alpha}_l} z + \sqrt{1-\overline{\alpha}_l}\epsilon, \tag{1}$$

where $\epsilon \sim \mathcal{N}(0,1)$ is Gaussian noise, $\overline{\alpha}_l = (1-\beta)^l$ is the amount of noise controlled by the parameters $\beta \in [0,1]$, strength of the noise, and l, number of iterations of the noise which is sampled uniformly between $[0, L]$ with $L \in \mathbb{N}^*$. The denoiser g_θ takes as inputs $\tilde{z}$, the number of iterations l and the context c and outputs $\hat{\epsilon} = g_\theta(c, \tilde{z}, l)$. The loss used to trained the denoiser g_θ is:

$$\mathcal{L}_{\texttt{DIFF}}(\theta) = \mathbb{E}\left[\|\epsilon - g_\theta(c, \tilde{z}, l)\|_2^2\right]. \tag{2}$$

We also remark by inverting Eq. (1) that $z = (\tilde{z} - \sqrt{1-\overline{\alpha}_l}\epsilon)\frac{1}{\sqrt{\overline{\alpha}_l}}$. Therefore, we can easily express the predicted original input $\hat{z}$ from the predicted noise $\hat{\epsilon}$ and the noisy input $(\tilde{z}, l)$:

$$\hat{z} = (\tilde{z} - \sqrt{1-\overline{\alpha}_l}\hat{\epsilon})\frac{1}{\sqrt{\overline{\alpha}_l}}. \tag{3}$$

2.4 BLaDE

BLaDE and BYOL-Explore are very similar but differ in two points: (1) the architecture of the predictor g_θ and (2) how the intrinsic reward is derived. For BLaDE, the predictor is a diffusion model with inputs the context $b_{t,k}$ and the couple $(\tilde{z}_{t,k}, l_{t,k})$, which represents the leaked information from the future at time $t+k$. The predictor outputs a predicted noise $\hat{\epsilon}_{t,k} = g_\theta\left(b_{t,k}, (\tilde{z}_{t,k}, l_{t,k})\right)$. The input $\tilde{z}_{t,k}$ is a noisy version of the future latent z_{t+k} and $l_{t,k}$ represents the quantity of noise added. More precisely, for each couple (t,k), we sample a Gaussian noise $\epsilon_{t,k} \sim \mathcal{N}(0, \frac{1}{N})$, a number of iterations $l_{t,k}$ uniformly in $[0, L]$ and uses the future latent z_{t+k} to compute $\tilde{z}_{t,k}$ via Eq.1:

$$\tilde{z}_{t,k} = \sqrt{\overline{\alpha}_{l_{t,k}}} z_{t+k} + \sqrt{1-\overline{\alpha}_{l_{t,k}}}\epsilon_{t,k}.$$

The number of iteration $l_{t,k}$ is passed as a one-hot encoding of size L to the predictor g_θ. It is important to remark that we do not sample the noise according to a centered-normalized Gaussian but by $\mathcal{N}(0, \frac{1}{N})$ where N is the number of features. Then, from the predicted noise $\hat{\epsilon}_{t,k}$ and the noisy input $(\tilde{z}_{t,k}, l_{t,k})$, we can reconstruct the predicted original output using Eq. (3):

$$\hat{z}_{t,k} = (\tilde{z}_{t,k} - \sqrt{1-\overline{\alpha}_{l_{t,k}}}\hat{\epsilon}_{t,k})\frac{1}{\sqrt{\overline{\alpha}_{l_{t,k}}}}.$$

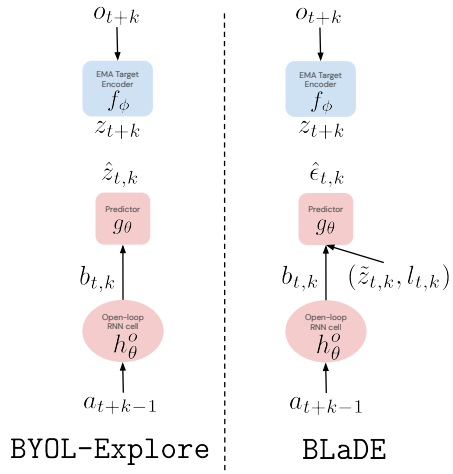


Figure 2: BLaDE's and BYOL-Explore's predictor architectures.

For a batch of data $\left((o_t^j, a_t^j)_{t=0}^{T-1}\right)_{j=0}^{B-1}$, the BLaDE's loss for an element of the batch j, time t, and open-loop step k is similar to the BYOL-Explore loss between predictions and targets:

$$\mathcal{L}_{\texttt{BLaDE}}(\theta, j, t, k) = \left\| \frac{\hat{z}_{t,k}^j}{\|\hat{z}_{t,k}^j\|_2} - \texttt{sg}\left(\frac{z_{t+k}^j}{\|z_{t+k}^j\|_2}\right)\right\|_2^2,$$

$$\mathcal{L}_{\texttt{BLaDE}}(\theta) = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} \frac{1}{K(t)} \sum_{k=1}^{K(t)} \mathcal{L}_{\texttt{BLaDE}}(\theta, j, t, k),$$

Contrary to a diffusion model that uses a L_2 loss between the noise and the predicted noise (see Eq.(2)), here, we use a cosine loss between the original target and its reconstructed prediction from the noise. Those are almost equivalent in theory and we made this choice in practice to make the BLaDE and BYOL-Explore losses similar.

Contrary to BYOL-Explore, we do not compute the uncertainties l_t^j directly from the losses that trains the world model. Instead we use a slightly tweaked loss:

$$\mathcal{L}_{\texttt{BLaDE}}^{\kappa}(\theta, j, t, k) = \left\| \frac{\hat{z}_{t,k}^{j,\kappa}}{\|\hat{z}_{t,k}^{j,\kappa}\|_2} - \texttt{sg}\left(\frac{z_{t+k}^j}{\|z_{t+k}^j\|_2}\right)\right\|_2^2,$$

where the predictions $\hat{z}_{t,k}^{j,\kappa}$ have been computed from noisy inputs $\tilde{z}_{t,k}^{j,\kappa}$ with a fixed amount of noise iterations $l_{t,k}^{j,\kappa} = \kappa$ where κ is an integer in $]0, L]$. Then, we compute the uncertainties l_t^j using those fixed-amount of noise losses:

$$\ell_t^j = \sum_{p+q=t+1} \mathcal{L}_{\texttt{BLaDE}}^{\kappa}(\theta, j, p, q),$$

where $0 \leq p \leq T-2$, $1 \leq q \leq K$ and $0 \leq t \leq T-2$. The idea behind using a fixed amount of noise κ for the intrinsic rewards is to be able to control how much variance of the future targets is added in the intrinsic rewards (as explained in details in App. A). Controlling this variance allows us to avoid stochastic traps and not get attracted to them. Indeed, with κ small we provide an input $\tilde{z}_{t,k}^{j,\kappa}$ to the predictor that is very similar to its target z_{t+k}^j and that can help the predictor disambiguate between different future predictions if the environment is stochastic. On other terms, we are providing/leaking noisy information about the future to the predictor to allow it disambiguate between different future outcomes and therefore reduce the variance added to the intrinsic rewards. Naturally, the more stochastic the environment is, the lower the threshold κ should be to control the added variance. It is totally possible to provide future information to the diffusion model as the intrinsic rewards are computed in the learner and not at acting time.

3 Experiments

We run our experiments on BLaDE with the threshold $\kappa = 10$, the amount of noise $\beta = 0.05$ and the number of noise iterations $L = 100$. We provide details on the neural network architectures in App. B as well as more hyperparameters values in App.D.

Stochastic Grid World. We first give an illustrative example on a small, partially-observable gridworld domain. We create a simple maze, with bouncing doors (see fig. 3 left). The agent sees a small window of radius 2 around it. The first red door is a stochastic bouncing door, where it moves left or right uniformly randomly, while the other doors bounce back and forth deterministically. We ran BYOL-Explore and BLaDE as pure exploration (no extrinsic reward) and track how many doors the agent is able to visit. As fig. 3 right shows, BYOL-Explore is only able to pass the first door, and we see that it actually gets stuck moving around the first door. On the other hand BLaDE is able to get pass and explore all four doors consistently. This result highlights that stochastic dynamics is an important limitation of BYOL-Explore, and that BLaDE can be completely robust to it.

Atari Learning Environment [3]. This is a widely used RL benchmark, comprising approximately 50 Atari games. These are 2-D, fully-observable, (fairly) deterministic environments for most of the

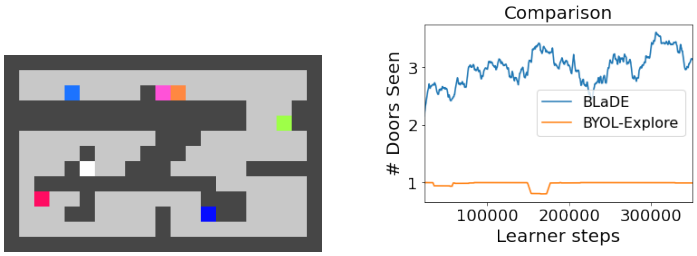


Figure 3: **Left:** Gridworld Maze. The white box is the agent. The red block is a stochastic door block that randomly moves left and right. The dark blue, green, and orange blocks are deterministic door blocks that bounce back and force. The purple and light blue blocks are just there for tracking purposes. **Right:** `BLaDE` is able to visit all four doors while `BYOL-Explore` is stuck at the first one.

games but have a very long optimization horizon (episodes last for an average of 10000 steps) and complex observations (preprocessed greyscale images which are 84×84 byte arrays). We select one of the hardest exploration and probably the most well-known games [2] to conduct our experiments, namely `Montezuma's Revenge`. To show the robustness of `BLaDE`, we add three types of noise to the environment. First, a non-actionable additive noise which is a 84×84 random array of integer values $\{-1, 0, +1\}$ that are independently and uniformly chosen at each time-step. This array is simply added to the 84×84 image byte array. It is important to remark that the addition is done at the byte level which means that $-1 + 0 = 255$ and $255 + 1 = 0$. Second, an actionable additive noise which is a 84×84 random array of integer values $\{-1, 0, +1\}$ that are uniformly chosen at each beginning of the episode and at each time the action `no-op` is taken. This array is also simply added to the 84×84 image byte array. It is important to remark that this noise is added at each time-step but changes (is resampled) only at the reset of each episode and when the action `no-op` is taken. Third, a noise that fundamentally changes the dynamics and that has been previously introduced in [17] called *sticky-actions*. In an environment with sticky actions, the probability of the action chosen by the agent a to be the action A_t executed at time t is $1 - p$ and with probability p the action a_{t-1} executed at time $t - 1$ is also executed at time t:

$$A_t = \begin{cases} a, & \text{with probability } 1 - p, \\ a_{t-1}, & \text{with probability } p. \end{cases}$$

We set p to be 0.1 in our experiments.

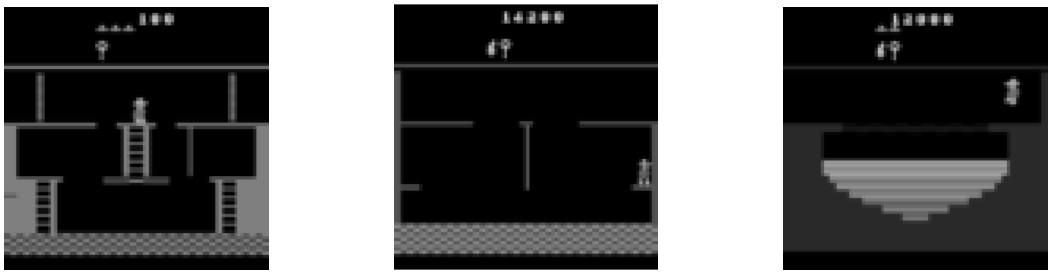


Figure 4: Examples of the classical `Montezuma's Revenge` game without noise.

Comments on the choices of noise. The non-actionable additive noise is independent of the dynamics and models a noise at the observation level due to imperfect sensors for instance. It has been chosen to artificially increase the observation space and render ineffective naive exploration methods based on counting the number of visits or predicting a random function of a given observation such as `RND`. The actionable additive noise increases the stochasticity of the dynamics of the `no-op` action, it should therefore be challenging for any prediction-error based method such as `BYOL-Explore` and `ICM`. In addition, it also artificially increases the observation space by choosing the `no-op` action which should render `RND` ineffective. Finally, sticky-actions increases the stochasticity of the dynamics for every action which is problematic for prediction-error based methods such as `BYOL-Explore`.

Figure 5: Examples of additive noise added to the classical `Montezuma's Revenge` game.

Experimental Setup. In addition to `BLaDE`, we also run baselines, namely `BYOL-Explore`, `RND` and `ICM`. We run experiments on two different evaluation regimes. The first regime uses a mixed reward function $r_t = r_{e,t} + \lambda r_{i,t}$ which is a linear combination of the normalized extrinsic rewards $r_{e,t}$ and intrinsic rewards computed by the agent $r_{i,t}$ with mixing parameter λ. This may be the most important regime for a practitioner as we can see if our intrinsic rewards help improve performance, with respect to the extrinsic rewards, compared to the pure RL agent. The second regime is fully self-supervised where only the intrinsic reward $r_{i,t}$ is optimized. This regime gives us a sense of how pure exploration methods perform in complex environments.

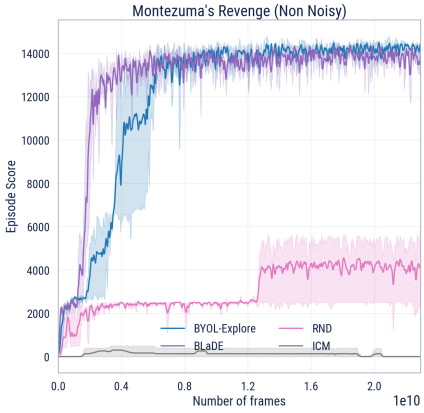
Choice of RL algorithm. We use VMPO [29] as our RL algorithm. VMPO is an efficient on-policy optimization method that has achieved strong results across both discrete and continuous control tasks, and is thus applicable to all of the domains we consider. Further details regarding the RL algorithm setup and hyperparameters are provided in Appendix.

Performance Metrics. In the mixed regime, we evaluate performance in terms of the agent score at a number of observations/frames t, $\texttt{Agent}_{\texttt{score}}(t)$, as measured by undiscounted episode return [9, 1]. The number of frames t corresponds to all the frames generated by all the actors by interacting with the environment, even the skipped ones. Frames/observations can be skipped if there is an action repeat which is the case in `Atari` where the action repeat is of 4. In the intrinsic regime, we evaluate performance in terms of the total number of rooms visited at a number of observations/frames t since beginning of training. Note that accessing later rooms requires navigating complex dynamics such as collecting keys to open doors, avoiding enemies, and carefully traversing rooms filled with traps such as timed lasers. Finally, we follow the classical 30 random no-ops evaluation regime [20, 30], and average performance over 10 episodes and over 3 seeds. In the figures, we show best seed, average performance and worst seed.

Results We run our experiments in the mixed and intrinsic regimes in `Montezuma's Revenge` for `BLaDE` and the baselines in the non-noisy classical environment as well as the non-actionable and noisy, actionable and noisy and sticky-action environments. As expected when additive noise is added, `RND`'s performance is flatlining as shown in Fig. 8 and in Fig. 7. However, we can see that `BYOL-Explore` performs well in the non-actionable noisy case (see Fig. 7) as the active representation learning seems to filter irrelevant features to perform efficient exploration. However, in the actionnable noisy case presented in Fig. 8, we see the performance of `BYOL-Explore` dropped and with more variance. This is also expected as it is a method based on prediction error that is subject to stochatics traps generated by stochastic dynamics. We see that this problem is aggravated in the sticky action case (see Fig. 9) where `BYOL-Explore` completely flatlined. Finally, we see that `BLaDE` is able to perform well across all types of noise in the mixed and intrinsic regime.

4 Limitations

Even if our method improves `BYOL-Explore` and seems more robust than state-of-the art baselines such as `RND`, there are still several limitations. First, we use diffusion models with only one step of denoising to build our future predictions whereas classical diffusion models use hundreds of denoising steps. We have not tried increasing the number of denoising steps since we were looking for a computationally efficient method but this trade-off between efficiency (fast) and more compute (better samples) needs to be studied in the future. Second, leaking information about future outcomes might



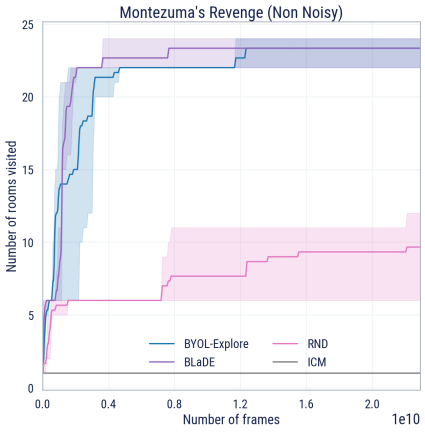
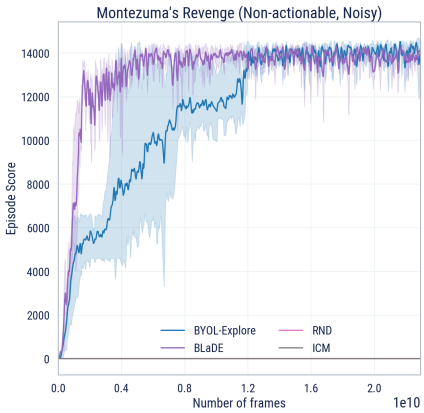


Figure 6: Results on classical non-noisy environment. **Left**: figure shows the agent score $\texttt{Agent}_{\texttt{score}}(t)$ for the mixed regime. **Right**: figure shows the number of rooms visited since beginning of training in the intrinsic regime.



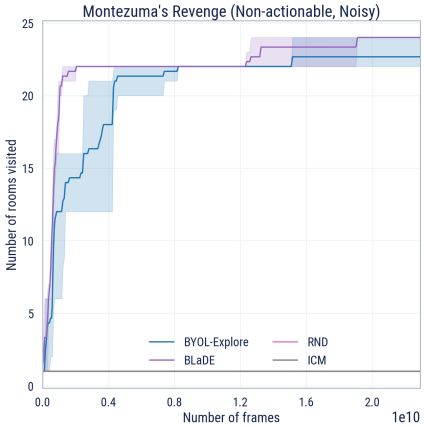
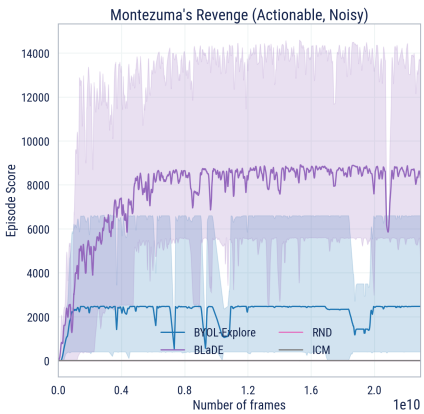


Figure 7: Results on non-actionable and noisy environment. **Left**: figure shows the agent score $\texttt{Agent}_{\texttt{score}}(t)$ for the mixed regime. **Right**: figure shows the number of rooms visited since beginning of training in the intrinsic regime.



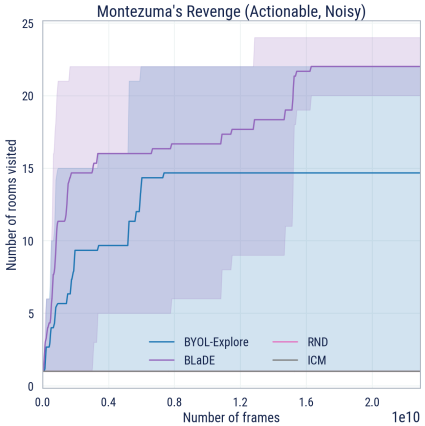


Figure 8: Results on actionable and noisy environment. **Left**: figure shows the agent score $\texttt{Agent}_{\texttt{score}}(t)$ for the mixed regime. **Right**: figure shows the number of rooms visited since beginning of training in the intrinsic regime.

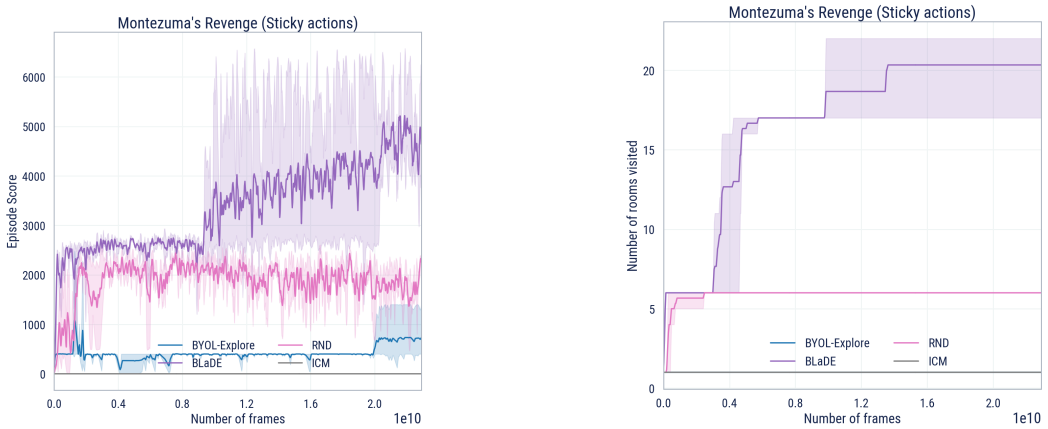


Figure 9: Results on environment with sticky actions. **Left**: figure shows the agent score $\texttt{Agent}_{\texttt{score}}(t)$ for the mixed regime. **Right**: figure shows the number of rooms visited since beginning of training in the intrinsic regime.

lead to worst representation because the context may not be used as much as in the `BYOL-Explore` loss, however we do not observe this in terms of final performance in our `Atari` experiments. Third, we use a fixed global threshold κ irrespective of the dynamics. Ideally, κ should be adaptive and could be high for states with deterministic dynamics and low for states with stochastic dynamics.

Finally, `BLaDE` inherits some limitations from `BYOL-Explore`. First, we are still learning with short-term predictions which can be uninformative if not much changes happen in the environment in few steps. Second, mixing intrinsic and extrinsic reward is not adaptive and should be tuned while learning. Thirdly, we use Gated Recurrent Units (GRUs) [5] as working memories which have some deficiencies regarding handling properly long-term dependencies.

5 Conclusion

We have shown that state-of-the art exploration methods could fail when real-world noise is introduced in the fairly deterministic benchmark they have been previously tested on, namely `Atari` games. More precisely, Random Network Distillation (`RND`) fails in the presence of stochasticity at the observation-level and `BYOL-Explore` fails when the underlying dynamics is stochastic. However, `BYOL-Explore` is still robust to the observation-level noise that does not affect the dynamics. To tackle this problem, we have introduced a simple and scalable modification of `BYOL-Explore`, called `BLaDE`, that uses a diffusion model instead of regression to predict future targets. This allows to not only model the expectation of the future-target distribution but the entire distribution. To avoid stochastic traps inherent to predicting the expectation, we provide information about the future outcomes to help the diffusion model predict the correct future outcome which allows to control the variance of future targets and produces intrinsic rewards that are not attracted to states with high variance future outcomes. In our experiments, we show with different types of noise that `BLaDE` is a robust exploration method and that performs as well as `BYOL-Explore` in the classical deterministic setting. However, `BLaDE` has still some limitations such as the noise threshold κ that is fixed for all the states. In the future, we would like to find a way to make it adaptive to the level of stochasticity of each state.

Acknowledgments and Disclosure of Funding

We would like to thank Abbas Abdolmaleki, Florent Altche, Bernardo Avila Pires, Arunkumar Byravan, Adrià Puidomenech Badia, Daniele Calandriello, Jean-Bastien Grill, Tim Harley, Steven Kapturowski, Thomas Keck, Daniel Jarrett, Jean-Baptiste Lespiau, Kat McKinney, Remi Munos, Kyriacos Nikiforou, Georg Ostrovski, Razvan Pascanu, Miruna Pislar, Doina Precup, Alaa Saade, Satinder Singh, Hubert Soyer, Pablo Sprechmann, Corentin Tallec, Yunhao Tang, and Michal Valko for their support and advice in developing this work.

References

[1] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[2] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*, pages 1471–1479, 2016.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Seventh International Conference on Learning Representations*, pages 1–17, 2019.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Mayank Daswani, Peter Sunehag, and Marcus Hutter. Q-learning for history-based reinforcement learning. In *Asian Conference on Machine Learning*, pages 213–228. PMLR, 2013.

[8] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[10] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020.

[11] Zhaohan Daniel Guo, Shantanu Thakoor, Miruna Pîslar, Bernardo Avila Pires, Florent Altché, Corentin Tallec, Alaa Saade, Daniele Calandriello, Jean-Bastien Grill, Yunhao Tang, et al. Byol-explore: Exploration by bootstrapped prediction. *arXiv preprint arXiv:2206.08332*, 2022.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

[16] Marcus Hutter et al. *Feature reinforcement learning: Part I. unstructured MDPs*. De Gruyter Open, 2009.

[17] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[18] R Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Machine Learning Proceedings 1995*, pages 387–395. Elsevier, 1995.

[19] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391, 2021.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[21] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[22] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[23] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29:4026–4034, 2016.

[24] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[25] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International Conference on Machine Learning*, pages 5062–5071. PMLR, 2019.

[26] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[27] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[28] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

[29] H Francis Song, Abbas Abdolmaleki, Jost Tobias Springenberg, Aidan Clark, Hubert Soyer, Jack W Rae, Seb Noury, Arun Ahuja, Siqi Liu, Dhruva Tirumala, et al. V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. *arXiv preprint arXiv:1909.12238*, 2019.

[30] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[31] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We are not including theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] We are not including theoretical results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code is proprietary
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specify the main training details in the paper and we include a full list of hyperparameters description in the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report error bars in learning curves of the agent score for every agent we run.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We include all the information regarding the compute in the appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use the ALE and `DM-HARD-8` and we cite the creators.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing.

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:

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

As the intrinsic rewards are derived from the prediction loss, we see that when the loss is minimized, we introduce the (expected-conditional) 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.

One way to reduce the variance of future targets $\texttt{Var}[z_{t+1}|(b_t, a_t)]$ is to provide more information to the predictor to predict more accurately z_{t+1}. Indeed, we know from the general law of total variance that for any random variable couple (X, Y) we have:

$$\texttt{Var}(Y) = \mathbb{E}[\texttt{Var}[Y|X]] + \texttt{Var}[\mathbb{E}[Y|X]].$$

Therefore the more information we provide to the predictor, the lower the (expected-conditional) variance is going to be. Let's note $\tilde{z}_{t+1}$ the additional information provided to the predictor then we have by the law of total variance:

$$\mathbb{E}\left[\texttt{Var}[z_{t+1}|(b_t, a_t)]\right] \geq \mathbb{E}\left[\texttt{Var}[z_{t+1}|(b_t, a_t, \tilde{z}_{t+1})]\right].$$

Of course if $\tilde{z}_{t+1} = z_{t+1}$ then the expected-conditional variance becomes null as we provide all the information to the predictor to predict z_{t+1}. Providing too much information will solve the problem of stochastic traps as there is no more residual variance but will forbid the predictor to understand the dynamics between (b_t, a_t) and z_{t+1} as it could directly predict z_{t+1} from $\tilde{z}_{t+1}$. Therefore $\tilde{z}_{t+1}$ should contain enough information to reduce the (expected-conditional) variance but not too much to allow the predictor to understand the dynamics between z_{t+1} and (b_t, a_t) and learn to smoothly reduce the epistemic uncertainty.

B General `BYOL-Explore` and `BLaDE` 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: $f_\phi : \mathcal{O} \to \mathbb{R}^N$

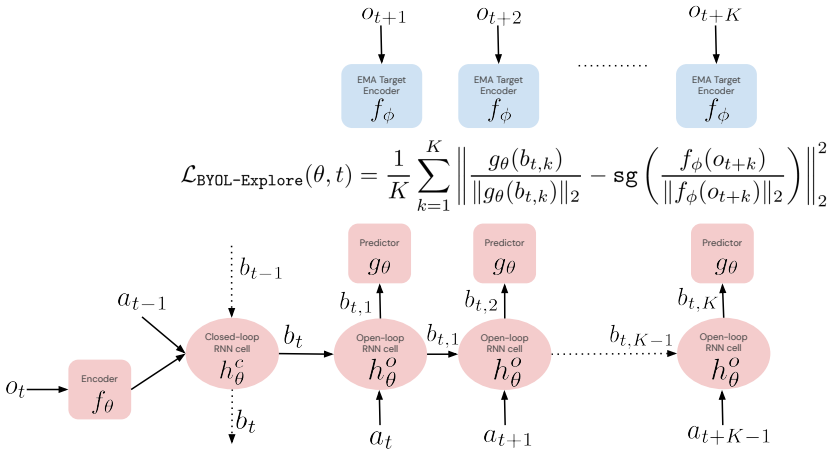


Figure 10: `BYOL-Explore`'s Neural Architecture.

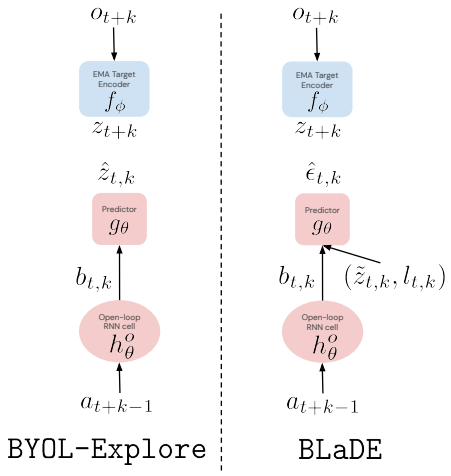


Figure 11: `BLaDE`'s and `BYOL-Explore`'s predictor architectures.

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: $h_\theta^c : \mathbb{R}^M \times \mathbb{R}^N \times \mathcal{A} \to \mathbb{R}^M$ 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: $h_\theta^o : \mathbb{R}^M \times \mathcal{A} \to \mathbb{R}^M$ 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 $((\ell_t^j)_{t=0}^{T-2})_{j=0}^{B-1}$ 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 $((\ell_t^j)_{t=0}^{T-2})_{j=0}^{B-1}$, we compute the batch mean $\overline{r_c}$ and the batch mean of squares $\overline{r_c^2}$:

$$\overline{r_c} = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} \ell_t^i, \qquad \overline{r_c^2} = \frac{1}{B(T-1)} \sum_{j=0}^{B-1} \sum_{t=0}^{T-2} (\ell_t^i)^2.$$

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:$r_{i,t}^j = \frac{\ell_t^i}{\sigma_r}$.

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,

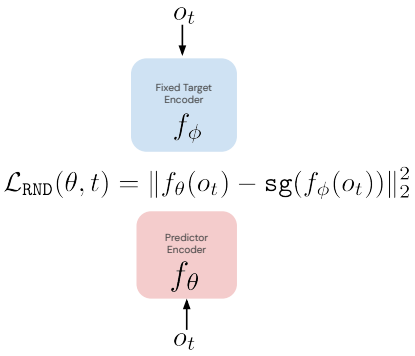


Figure 12: RND's Neural Architecture.

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 :$r_{i,t}^j = \frac{\ell_t^j}{\sigma_r}$.

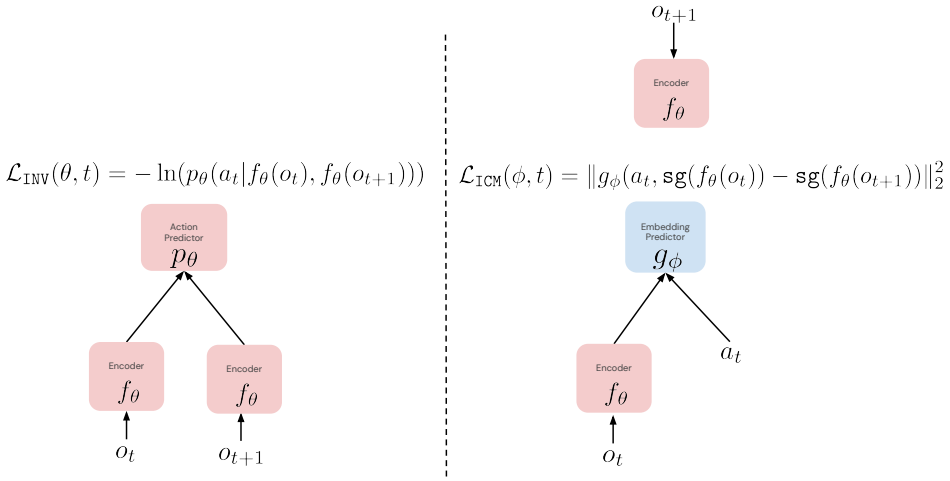


Figure 13: ICM's Neural Architecture.

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].