CURIOSITY-DRIVEN EXPLORATION BY BOOTSTRAPPING FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Curiosity by Bootstrapping Features (CBF), an exploration method that works in the absence of rewards or end of episode signal. CBF is based on intrinsic reward derived from the error of a dynamics model operating in feature space. It was inspired by (Pathak et al., 2017), is easy to implement, and can achieve results such as passing four levels of *Super Mario Bros*, navigating *VizDoom* mazes and passing two levels of *SpaceInvaders*. We investigated the effect of combining the method with several auxiliary tasks, but find inconsistent improvements over the CBF baseline.

1 INTRODUCTION

Modern reinforcement learning methods work well for tasks with dense reward functions, but in many environments of interest the reward function may be sparse, require considerable human effort to specify, be misspecified, or be prohibitively costly to evaluate. In general it is much easier to find environments that we *could* train an agent to act in than it is to find sensible reward functions to train it with. It is therefore desirable to find ways to learn interesting behaviors from environments without specified reward functions.

Pathak et al. (2017) introduced an exploration strategy that leads to sophisticated behavior in several games in the absence of any extrinsic reward. The strategy involves

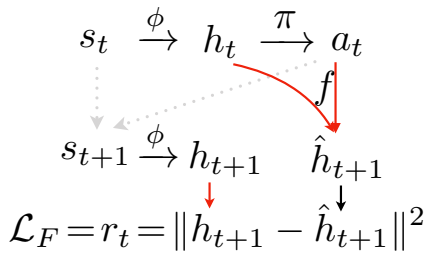
1. Learning features using an inverse dynamics prediction task,
2. training a forward dynamics model in the feature space, and
3. using the error of the forward model as an intrinsic reward for an exploration agent.

Inspired by this result we wondered if it was possible to improve the method by using a different task for learning the features in step 1. To our surprise we found that the choice of feature-learning task didn't matter much. In fact when skipping step 1 altogether, we often obtained comparable or better results. As a result we obtained a method that is simple to implement, and shows purposeful behavior on a range of games, including passing over four levels of *Super Mario Bros* without any extrinsic rewards or end of episode signal (see video *here*). Previous work reported making significant progress on the first level of this game. In addition we report the results of using our method on *VizDoom* maze environment, and a range of Atari games.

2 RELATED WORK

A family of approaches to intrinsic motivation reward an agent based on error (Schmidhuber, 1991b; Stadie et al., 2015; Pathak et al., 2017; Achiam & Sastry, 2017), uncertainty (Still & Precup, 2012; Houthooft et al., 2016), or improvement (Schmidhuber, 1991a; Lopes et al., 2012) of a forward dynamics model of the environment that gets trained along with the agent's policy. As a result the agent is driven to reach regions of the environment that are difficult to predict for the forward dynamics model, while the model improves its predictions in these regions. This adversarial and non-stationary dynamics can give rise to complex behaviors.

Self-play (Sukhbaatar et al., 2017) utilizes an adversarial game between two agents for exploration. Smoothed versions of state visitation counts can be used for novelty-based intrinsic rewards (Bellemare et al., 2016; Fu et al., 2017; Ostrovski et al., 2017; Tang et al., 2017).



ϕ - embedding
π - policy
f - forward dynamics model
r_t - intrinsic reward at step t
$\mathcal{L}_F$ - loss for the dynamics model

Figure 1: Main components of the Curiosity by Bootstrapping Features (CBF) method. s_t and s_{t+1} are environment states at times t, $t+1$, $a_t \sim \pi(\phi(s_t))$ is the action chosen at time t. The red arrows indicate that the input of that arrow is treated as a constant — no gradients are propagated through it.

Other methods of exploration are designed to work in combination with maximizing a reward function, such as those utilizing uncertainty about value function estimates (Osband et al., 2016; Chen et al., 2017), or those using perturbations of the policy for exploration (Fortunato et al., 2017; Plappert et al., 2017).

Schmidhuber (2010) provides a review of some of the earlier work on approaches to intrinsic motivation.

2.1 Dynamics Driven Curiosity

The forward dynamics model takes as input the current state and action and outputs a point estimate of (or a distribution over) the next state

$$s_t, a_t \to \hat{s}_{t+1} \approx s_{t+1}. \tag{1}$$

Alternatively the dynamics model could operate in a space of features of the raw observations:

$$\phi(s_t), a_t \to \widehat{\phi(s_{t+1})} \approx \phi(s_{t+1}) \tag{2}$$

where ϕ is some embedding function that takes as input a state and outputs a feature vector. One advantage of the latter approach is that it may be computationally more efficient to train a predictive model in a lower dimensional feature space. Another is that a good choice of ϕ will exclude irrelevant aspects of the state. Examples of this approach include Stadie et al. (2015) where they learn ϕ by using an autoencoding objective and Pathak et al. (2017) where they use an inverse dynamics task to learn the embedding. The latter work is the closest to our approach. We outline the main differences in Appendix 10.

3 Methods and Notation

3.1 Environment

We consider a discounted, infinite-horizon Markov Decision Process (MDP) given by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p_0, p, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $p : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$ is the environment transition function, p_0 is the initial state distribution, and γ is the discount factor. Normally we would try to produce a policy $\pi : \mathcal{S} \to P(\mathcal{A})$ maximizing the expected discounted returns, but we treat the reward function as being unknown, and instead produce a policy maximizing a surrogate reward function. We will write s_t, a_t, r_t for the state, action, and the surrogate reward received at time t.

3.2 Networks and Parameters

Our method employs the following components:

- Embedding network $\phi : \mathcal{S} \to \mathcal{H} \subseteq \mathbb{R}^d$ parametrized by θ_ϕ, where $\mathcal{H}$ is the embedding space,

- policy network $\pi : \mathcal{H} \to P(\mathcal{A})$ parametrized by θ_π that outputs probability distributions over $\mathcal{A}$,
- forward dynamics model $f : \mathcal{H} \times \mathcal{A} \to \mathcal{H}$ parametrized by θ_f, and
- (optionally) an auxiliary prediction task for learning ϕ implemented by a network g with parameters θ_g. The exact functional form will depend upon the prediction task.

3.3 Losses and Gradients

The intrinsic reward that we replace the extrinsic reward with is a function of the embeddings $h_t = \phi(s_t)$, $h_{t+1} = \phi(s_{t+1})$ of consecutive states s_t, s_{t+1} and the chosen action a_t. Given the output of the forward dynamics model $\hat{h}_{t+1} = f(h_t, a_t)$ the reward is defined as $r_t = \|\hat{h}_{t+1} - h_{t+1}\|_2^2$ (see figure 1). This reward is identical to the *loss* of the forward dynamics model. The policy attempts to maximize the reward by finding unexpected transitions, while the dynamics model tries to minimize its prediction error (identical to the reward) with respect to its own parameters. We optimize this dynamics loss only with respect to the parameters of the dynamics model θ_f and not with respect to the parameters of the embedding network θ_ϕ to avoid issues with features incentivised to become predictable by collapsing to a constant.

The loss for the policy network depends upon what reinforcement learning algorithm we use. In this paper we use the Proximal Policy Optimization (PPO) algorithm described by Schulman et al. (2017). The loss is as described there using the rewards r_t defined above. When we do joint training of features and policy we optimize the policy loss with respect to both θ_ϕ and θ_π. Otherwise we take gradients only with respect to θ_π.

We explore several choices of auxiliary losses to train ϕ with. These are detailed in Section 6. These losses get optimized with respect to θ_ϕ and θ_g.

With this setup the parameters of the embedding function ϕ get optimized with respect to a combination of the auxiliary loss (if any) and policy network loss (when training jointly). In the case where there is no auxiliary task and we are not training the features jointly with the policy, ϕ is simply a fixed function with randomly initialized parameters θ_ϕ.

The forward dynamics model would prefer to have features of small norm to reduce its error, and the policy network would prefer to have features of large norm to increase its reward. It is important to note however that the forward model's loss does not optimize the parameters of the embedding network, and the policy network can only optimize with respect to the actions it takes, not directly manipulate the reward function by increasing the norm of the features.

The overall training process is described in Algorithm 1.

4 Challenges

4.1 Stochasticity

The intrinsic motivation coming from the errors of a deterministic forward dynamics model could be inadequate in some stochastic environments. Previous works have discussed the problem of "TV static". The "TV static" refers to stochastic transitions in the environment (like the white noise static of a TV) that have no causal relationship with the past or the future. In the presence of such stochasticity the forward dynamics model will mispredict the next state, and the agent will be drawn to explore the stochastic subset of the environment, possibly at the expense of other aspects that it could explore instead. Predicting the dynamics in feature space can alleviate the problem if the features don't contain information about the irrelevant aspects of the environment that are stochastic.

Another problem is the presence of a "lottery", i.e. a stochastic transition from a particular state to states with meaningfully different possible outcomes (e.g. if the future of the agent depends in some important ways on the outcome of a roll of a die). Since the future of the agent depends on the unpredictable outcome, the different outcomes must be represented differently in feature space. In this case the agent will receive high reward for participating in the "lottery". Such a situation can arise in games when an agent passes a level, and transitions to a level with random positions of obstacles and enemies. Sometimes this effect can be advantageous to exploration, but sometimes

it is not, as discussed in section 5. In such situations a stochastic dynamics model would be more appropriate.

4.2 MEASURING PROGRESS

There is no definite way of measuring the progress of exploration. Some possible goals for exploration include learning about the dynamics of the environment, obtaining policies that can control aspects of the environment, obtaining policies that can be easily fine-tuned to optimize a particular reward function, producing agents that exhibit complex and nuanced behaviors, discovering unexpected aspects of the environment etc. Corresponding to such goals one could think of multiple proxies for measuring the progress on achieving them. In environments with a particular environmental reward function that we choose to ignore at training time, progress at achieving high returns often can be a reasonable measure of exploration. This is because many games and tasks are such that the rewarding behavior is also complex, nuanced, requires knowledge of the environment's dynamics, and the ability to control it. Other measures of progress include counts of visited states, and amount of time staying alive (if staying alive is challenging in the particular environment).

5 EXPLORATION WITH MINIMAL REQUIREMENTS OF THE ENVIRONMENT

Besides using the reward signal, it is common practice to include other environment-specific clues to facilitate learning. We tried to remove such clues, as they can be laborious to specify in a new unfamiliar environment.

In several Atari games such as *Breakout* the game doesn't properly begin until the agent presses fire, and so it is common to add a wrapper to the environment that automatically does this to avoid the agent getting stuck (Mnih et al., 2015). We don't use such wrapper, since being stuck is predictable and unrewarding to our agents.

In addition we switch to a non-episodic infinite horizon formulation. From the agent's perspective, death is just another transition, to be avoided only if it is boring. In some games the end of episode signal provides a lot of information, to the extent that attempting to stay alive for longer can force the agent to win the game even in the absence of other rewards as noted by the authors of (Christiano et al., 2017).

In our experiments we don't communicate the end of episode signal to our agents. Despite that we find that agents tend to avoid dying after some period of exploration, since dying brings them back to the beginning of the game — a region of state space the dynamics model has already mastered.

In some environments however the death is a stochastic transition — the positions of entities in the game might be randomized at the beginning of the game. This poses a challenge for exploration based on deterministic dynamics models, which is drawn to seek out the stochastic transition from the end to the beginning of the game. In the environments that we dealt with it wasn't a problem, so we leave using a stochastic dynamics model for future work.

6 FEATURE LEARNING METHODS

6.1 HINDSIGHT EXPERIENCE REPLAY (HER) (ANDRYCHOWICZ ET AL. (2017))

HER is a recent method for learning a policy capable of going from any state to any state (or more generally any goal). It learns offline from an experience replay buffer of environment transitions. We only use this method as a way of training the embedding network ϕ, not to use it for actually achieving any goals.

The loss for HER is based on the Bellman loss for the goal-conditioned state-action value function with respect to a reward function defined by

$$r(s_t, s_{t+1}; g) = \begin{cases} 1 & s_{t+1} = g \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Given a discount factor $\gamma \in [0, 1]$, the optimal state-action value function $Q^*(s, a; g)$ for a deterministic environment is γ^n where n is the length of the shortest path from state s to goal state g that takes action a in state s.

HER is an off-policy algorithm that learns the state-action values of states with respect to goals that can be chosen from the future of the state's trajectory. When we train our model we first sample a random transition s_t, s_{t+1}, a_t from the replay buffer, then a goal g which is s_{t+1} with probability 0.5 and s_{t+k} with probability 0.5 where k is sampled from -100 to 100 uniformly at random.

In our implementation of HER we use a goal-conditioned state-action value function $q(s, a; g)$ which is a function of $(\phi(s), \phi(g), a)$. The motivation for this method of learning embeddings is that the network is encouraged to represent aspects of the environment that can be controlled and are useful for control.

6.2 Inverse Dynamics Features (IDF)

Given a transition (s_t, s_{t+1}, a_t) the inverse dynamics task is to predict the action a_t given the previous and next states s_t and s_{t+1}. Features are learned using a common neural network ϕ to first embed s_t and s_{t+1}. The intuition is that the features learned should correspond to aspects of the environment that are under the agent's immediate control. This feature learning method is easy to implement and in principle should be invariant to certain kinds of noise (see Pathak et al. (2017) for a discussion). A potential downside could be that the features learned may be myopic, that is they do not represent important aspects of the environment that do not immediately affect the agent.

6.3 No Auxiliary Task

Instead of using auxiliary tasks to learn embeddings of observations, we could instead learn the features only from the optimization of the policy, or not change them at all after the initialization. We refer to the exploration method using an embedding network that is fixed during training as Fixed Random Features (FRF). We refer to the algorithm that uses features that learn only from the policy optimization Curiosity by Bootstrapping Features (CBF).

We speculate that Curiosity by Bootstrapping Features works via a bootstrapping effect where the features encode increasing amounts of relevant aspects of the environment, whereas the forward model learns to predict the transitions of those relevant features. Initially the features encode little information about the environment. By trying to find unpredictable transitions of those vague features, the policy has to attend to some additional aspects of the environment that were not previously encoded in the features. These aspects of the environment now become part of the feature space, and hence the dynamics model now has to predict their transitions as well. Even more aspects of the environment then become relevant for finding surprising transitions in the feature space, and the cycle continues.

One fixed point of this process is when features are constant, and the dynamics model has to predict constant transitions. Empirically we find that this fixed point is in fact unstable — if features initially contain some amount of information about the environment, eventually they start encoding increasingly more aspects of the environment (as evidenced by looking at pixel reconstructions from a separately trained decoder trained to decode the features into the pixel space).

7 Experimental Results

For details on the experimental setup such as hyperparameters and architectures see Appendix 9. We implemented the models using Tensorflow (Abadi et al. (2015)) and interfaced to the environments using the OpenAI Gym (Brockman et al. (2016)).

7.1 Evaluation

We use the mean (across three random runs) best extrinsic return of the agent as a proxy for exploration. In general there is no *a priori* reason why extrinsic return should align well with an intrinsic measure of interestingness based on the dynamics, but it often turns out to be the case.

Algorithm 1 CBF with optional auxiliary losses

$N \leftarrow$ number of rollouts
$N_{\text{opt}} \leftarrow$ number of optimization steps
$K \leftarrow$ length of rollout
$t = 0$
Sample state $s_0 \sim p_0(s_0)$
for $i = 1$ **to** N **do**
  for $j = 1$ **to** K **do**
    sample $a_t \sim \pi(a_t|s_t)$
    sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    calculate intrinsic reward $r_t = \|\phi(s_{t+1}) - f(\phi(s_t), a_t)\|^2$
    add s_t, s_{t+1}, a_t, r_t to replay buffer R and to optimization batch B_i
    t += 1
  end for
  for $j = 1$ **to** N_{opt} **do**
    optimize θ_π and optionally θ_ϕ wrt PPO loss* on batch B_i
    sample minibatch M from replay buffer R
    optimize θ_f wrt forward dynamics loss on M
    optionally optimize θ_ϕ, θ_A wrt to auxiliary loss
  end for
end for
*PPO loss is modified so as not to include the end of episode signals or 'dones'.

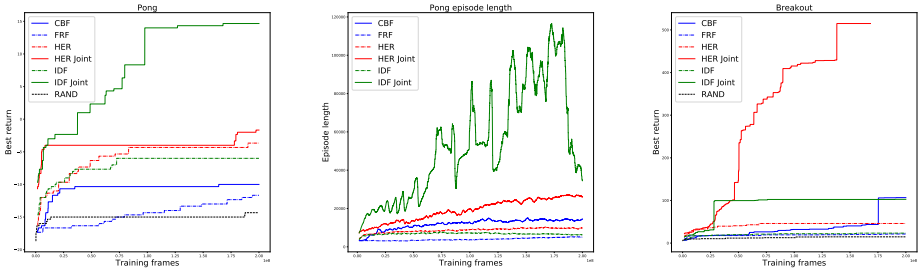


Figure 2: Left: *Pong* results measured by game score. Center: *Pong* results measured mean episode length. Right: *Breakout* results measured by game score..

7.2 Abbreviations

In the figures we use abbreviations for the method names: IDF = inverse dynamics auxiliary task without joint training, IDF Joint = inverse dynamics auxiliary task with joint training, HER = Hindsight Experience Replay auxiliary task without joint training, HER Joint = Hindsight Experience Replay auxiliary task with joint training, CBF = no auxiliary task with joint training, FRF = no auxiliary task without joint training, RAND = random agent.

We have chosen most hyperparameters for our experiments based on open-sourced implementations of PPO and DQN (Hesse et al., 2017). We have changed the entropy bonus coefficient and number of optimization steps for PPO, as well as algorithm-specific learning rates after some initial results on Pong and Breakout. We used those chosen hyperparameters on all the other environments (see appendix 9 for more details).

We test various approaches on the games *Super Mario Bros* and *VizDoom* (Kempka et al. (2016)) considered in Pathak et al. (2017), as well as some additional Atari games from the Arcade Learning Environment (Bellemare et al. (2013)).

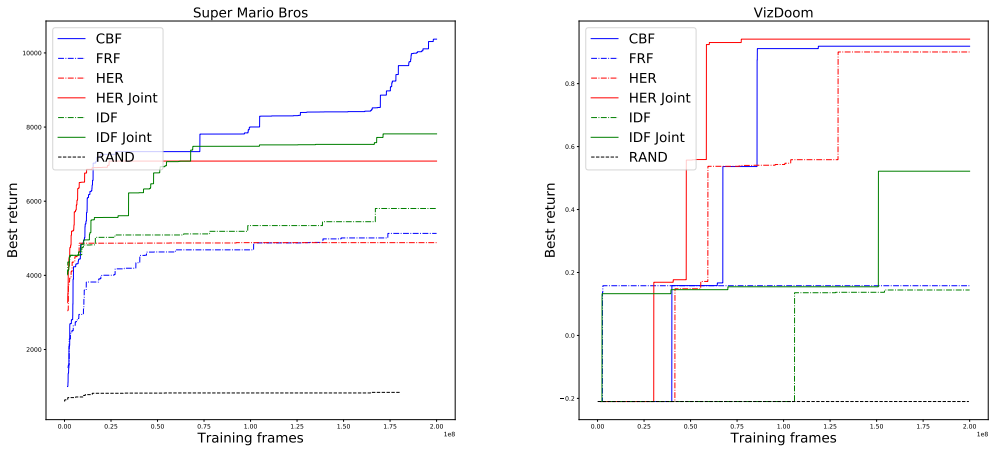


Figure 3: Left: results on *Super Mario Bros* measured by maximum distance travelled. Right: results on *VizDoom* measured by game score.

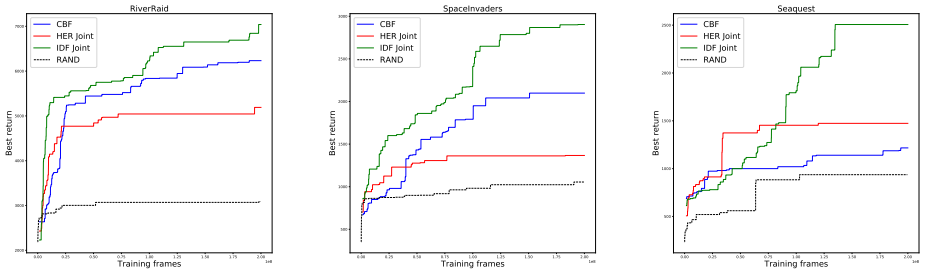


Figure 4: Left: Results on *Riverraid* measured by game score. Center: Results on *SpaceInvaders* measured by game score. Right: results on *Seaquest* mesaured by game score.

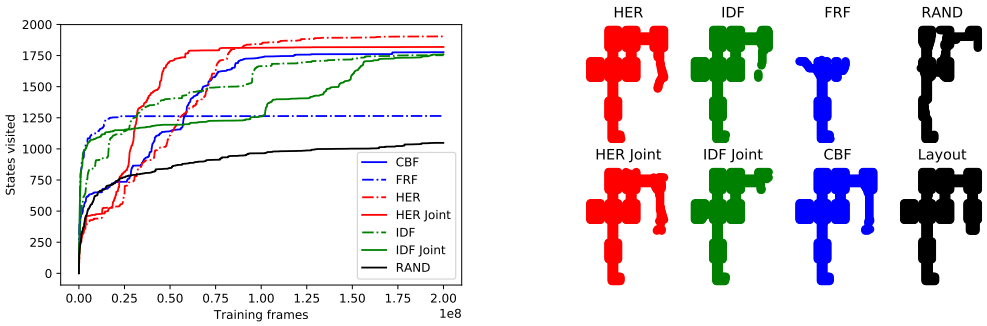


Figure 5: Additional *VizDoom* results. Left: Results as measured by number of unique locations visited. Right: Results shown as all locations visited by all of the three runs.

7.3 PONG

The results for the *Pong* experiments are shown in Figure 2 on the left. We see that joint training with IDF performs the best (see a video *here*). From watching the rollouts we observe that the policy seems to be optimizing for long rallies with many ball bounces rather than trying to win (see figure 2 in the center). We also see that joint training methods do better than their respective non-joint analogues. Some of the runs have encountered an unexpected feature of the environment: after 9 minutes and 3 seconds of continuous play in one episode, the ball disappears and the background colors start cycling in seemingly random order (see video *here*). We suspect this is a bug of the Atari emulator that we used, but we haven't investigated the issue further.

7.4 BREAKOUT

The results for the *Breakout* experiments are shown in Figure 2 on the right. We see that the methods without joint training perform little better than random agent, with the exception of the HER feature learning method. HER with joint training performs very well, coming close in some runs to a perfect score (see a video *here*).

7.5 SUPER MARIO BROS

We investigated using the same version of *Super Mario Bros* used in Pathak et al. (2017) namely Paquette (2016), but found that we could not run the environment as fast as we would like. For this reason we switched to using an internal version with an action wrapper replicating the action space used in Pathak et al. (2017). We also ran the released code for Pathak et al. (2017) on both versions of the game and found that the agent was in both cases able to make progress on, but unable to pass, the first level, consistent with the authors' reported results.

The metric we use for monitoring progress is the total distance the agent travels to the right over the course of the game. Larger returns correspond to passing more levels and getting further in those levels. The results are shown in Figure 3 on the left. Every method (apart from random agent) considered was able to pass the first level. Our best method CBF was able to pass the first 4 levels, defeating the boss Bowser and moving onto the second world in the game (see video *here*). The same agent also found the secret warp room on level 2 and visited all of the pipes leading to different worlds. All of the methods without joint training performed relatively poorly.

7.6 VIZDOOM

We use the same setup as in Pathak et al. (2017) with 'DoomMyWayHomeVerySparse' environment, a *VizDoom* scenario with 9 connected rooms. The agent is normally given an extrinsic reward of -0.0001 per timestep, a timelimit of 2100 steps, and a reward of $+1$ for getting the vest. We use the 'very sparse' version where the agent always spawns in the same room maximally far from the vest. There are five actions: move forward, move backwards, move left, move right, and no-op.

Looking at Figure 3 on the right, we see that all of the methods (but not random agent) are able to reach the vest in at least one of the runs, and that HER, joint HER and CBF methods are able to do so reliably (see the video *here*). To get a finer-grained notion of progress we recorded the (x, y) coordinates of the agents over the course of training. We then binned these into a 100×100 grid. In Figure 5 you can see a visualization of the locations visited by each method. We see that most of the methods achieve good coverage of the maze, even random agent performs surprisingly well. Training on fixed random features however results in poor exploration. In addition in Figure 5 on the left we show how the number of unique bins visited increases with training for each method. Overall the success of random agent indicates that harder tasks should be used to evaluate exploration in future work.

7.7 RIVERRAID

In Figure 4 on the left we see that all methods are considerably better than random, but IDF performs the best (see the video *here*).

7.8 SPACEINVADERS

We see in Figure 4 in the center that all methods are considerably better than random, IDF performs the best, one of the better runs was able to pass almost 4 levels (see the video *here*).

7.9 SEAQUEST

We see in Figure 4 on the right that all methods are better than random agent and IDF performs the best. The agents pursue an interesting strategy that we were not previously aware was possible: by hovering immediately below the water's surface, the agent is able to survive indefinitely without running out of air while still being able to shoot at enemies (see the video *here*).

8 CONCLUSIONS AND FUTURE DIRECTIONS

Our experiments have shown that any of the joint training methods can work well for exploration. The fact that a method as simple as CBF performs so well, however, suggests that the success of the method of Pathak et al. (2017) comes to a great extent from a feature-bootstrapping effect, and the utility of an auxiliary task, if any, is to stabilize this process.

Some immediate future research directions include trying CBF on environments with continuous action spaces, and investigating feature-bootstrapping for count-based exploration methods such as Ostrovski et al. (2017). We would also like to research exploration of environments with greater amounts of stochasticity.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv:1703.01732*, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems*, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013.

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*, pp. 1471–1479, 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym, 2016.

Richard Y Chen, John Schulman, Pieter Abbeel, and Szymon Sidor. UCB and infogain exploration via q-ensembles. *arXiv:1706.01502*, 2017.

Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv:1706.03741*, 2017.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *arXiv:1706.10295*, 2017.

Justin Fu, John D Co-Reyes, and Sergey Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. *arXiv:1703.01260*, 2017.

Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. OpenAI baselines. https://github.com/openai/baselines, 2017.

Rein Houthooft, Xi Chen, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational information maximizing exploration. In *Advances In Neural Information Processing Systems*, pp. 1109–1117. 2016.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348, Santorini, Greece, Sep 2016. IEEE.

Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, pp. 206–214, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, pp. 4026–4034, 2016.

Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *arXiv:1703.01310*, 2017.

Philip Paquette. Super Mario Bros. https://github:ppaquette/gym-super-mario, 2016.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv:1706.01905*, 2017.

Jürgen Schmidhuber. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pp. 1458–1463. IEEE, 1991a.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers, 1991b.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *Advances in Neural Information Processing Systems Workshop*, 2015.

Susanne Still and Doina Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv:1703.05407*, 2017.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 2017.

9 Appendix: Experimental Details

Preprocessing

We followed the standard preprocessing for Atari games (see wrappers used in the DQN implementation in Hesse et al. (2017)) for all of our experiments, except for not using the automatic "press fire" in the beginning of the episode wrapper. For Mario and VizDoom we downscaled the observations to 84 by 84 pixels, converted them to grayscale, stacked sequences of four frames as four channels of the observation, and used a frame skip of four. We also used an action wrapper replicating the action space used in Pathak et al. (2017).

Architectures

Both our implementation of HER and PPO were based on the code in Hesse et al. (2017) with HER following the implementation of DQN.

The embedding network ϕ consisted of three convolutional layers followed by a dense layer similar to the DQN implementation in Hesse et al. (2017).

The policy network π consisted a dense layer followed by two output heads for action probabilities and values.

The forward dynamics head f concatenates the state embedding with a one-hot representation of action and is followed by two dense layers with a residual connection to the output.

The auxiliary DQN head for the HER task concatenates the embeddings of the state and goal followed by a dense hidden layer followed by an output layer.

The auxiliary IDF head concatenates the state and next state embeddings, followed by two dense layers and an output softmax layer.

Hyperparameters

We used the same hyperparameters for all experiments. We used the default hyperparameters from the PPO implementation in Hesse et al. (2017) except for the entropy bonus, which we decreased to 0.001 and the number of optimization steps per epoch, which we increased to 8.

For HER we used a discount factor of 0.99. We used stabilization technique for the target value in the Bellman loss: we used a Polyak-averaged version of the value function with decay rate 0.999. The learning rate for the HER task was 10^{-4}.

The experience replay buffer contained 1000 timesteps per environment (of which there are 32 by default).

The learning rate for the forward dynamics model was 10^{-5}.

The minibatch size for the forward dynamics and auxiliary loss training step was 128.

Each training run consisted of $50e6$ steps which is $200e6$ frames since we used the standard frame-skip of 4.

10 Appendix: Comparison with Pathak et al. (2017)

Besides the use of a different set of auxiliary losses for learning the features, we note some of the salient differences with the work (Pathak et al., 2017).

- We don't propagate the gradients from the dynamics model to the parameters of the embedding network.
- We use PPO as our reinforcement learning algorithm instead of A3C (Mnih et al. (2016)) since in our experience PPO requires little tuning.
- We used 32 parallel environments rather than 16. We initially ran experiments with only 8 or 16 environments and got qualitatively similar results, but in general found that more parallel environments resulted in stabler learning.
- We trained our next state predictor and any auxiliary losses on an experience replay rather than online. We used a small experience buffer (only 1000 timesteps per environment) that may have decorrelated the data and stabilized the learning dynamics.
- We used a CNN for our policy rather than an LSTM. We judged that a small amount of partial observability in the environment was worth the tradeoff since LSTMs are typically slower and more difficult to train.