REPRODUCING: CURIOSITY DRIVEN EXPLORATION BY BOOTSTRAPPING FEATURES

Audrow Nash & Thomas Huang
University of Michigan
{audrow, thomaseh}@umich.edu

ABSTRACT

The paper had one experiment on several environments, and we attempted to replicate this experiment on two environments: the Atari games Seaquest and Pong. We were able to see policy improvement for Seaquest using the author's method, but not for Pong. The authors did not provide code for this experiment.

1 INTRODUCTION

The paper we are seeking to reproduce proposes an exploration method called *Curiosity by Bootstrapping Features* or CBF, which uses intrinsic reward derived from the error of a dynamics model operating in feature space to drive exploration. The authors assert that this method can lead to exploration, especially in environments where rewards are sparse or absent. They compare CBF against several methods. We choose to compare our implementation against *Fixed Random Features* (FRF), which doesn't update the embedding network, and a random policy (RAND).

2 EXPERIMENTS

2.1 CODE USED

We were unable to find code for the paper that we were reproducing. For computing the Proximal Policy Optimization (PPO) loss, we slightly modified the PPO implementation found in Dhariwal et al. (2017). For the networks, we followed the implementation in Pathak et al. (2017).

2.2 CODE WRITTEN

As code was not available, we implemented the entire architecture from scratch. This involved constructing several networks, including an embedding network, a policy network, and a forward dynamics network. Most details about the networks' parameters and hyperparameters were specified, but some are left out. We noted our values for these unknown hyperparameters in Appendix A. The embedding network was not specified, and we found a similar work to our paper, Pathak et al. (2017), which we used to define our embedding network. The embedding network in Pathak et al. (2017) had some hard-coded numbers, we assume, that were hand-tuned for stability while training. We found that adjusting these numbers could stabilize our network during training.

2.3 ENVIRONMENTS

For our experiments, we used two Atari environments from Brockman et al. (2016), Pong and Seaquest.

2.4 DETAILS

For a detailed explanation of our preprocessing, network architectures, and hyperparameters, see Appendix A.

2.5 Stability

One of our major challenges in reproducing this work was stabilizing the learning process. Specifically, the loss of the forward dynamics model and the value function error would exponentially approach infinity. In the end, we stabilized the network, but perhaps at the expense of improving our network. This section will describe what we have tried and what we observed.

To determine the cause of instability, we first confirmed the correctness of our modified PPO implementation and both the embedding network and the policy network. To do this, we removed the forward dynamics model and replaced the intrinsic reward with the extrinsic rewards from the environment. We found that when we ran these together, the networks were able to learn and achieve increasingly better results on both environments in a stable manner. From this ablation study, we determined that the instability was somehow related to the interactions between the aforementioned networks and the forward dynamics network.

Next, we ran FRF with intrinsic rewards and found that the learning process was stable. However, the method proposed, CBF, updates the embedding network with the policy. This suggests the joint learning process of policy and embedding in combination with the learning of the forward dynamics model greatly destabilizes the learning process.

We found that we were able to stabilize training with CBF through two methods: (i) by adding a hyperparameter to scale down the loss of the forward dynamics network, and (ii) by introducing parallel environments into our implementation. The scaling parameter for the forward dynamics loss was inspired by a hard-coded constant in the implementation of Pathak et al. (2017), which used a similar setup. We could not find justification for this parameter in the paper, but modifying it allowed us to stabilize our network. We found that using parallel environments allowed us to scale the forward dynamics loss less severely while maintaining stability.

2.6 Seaquest

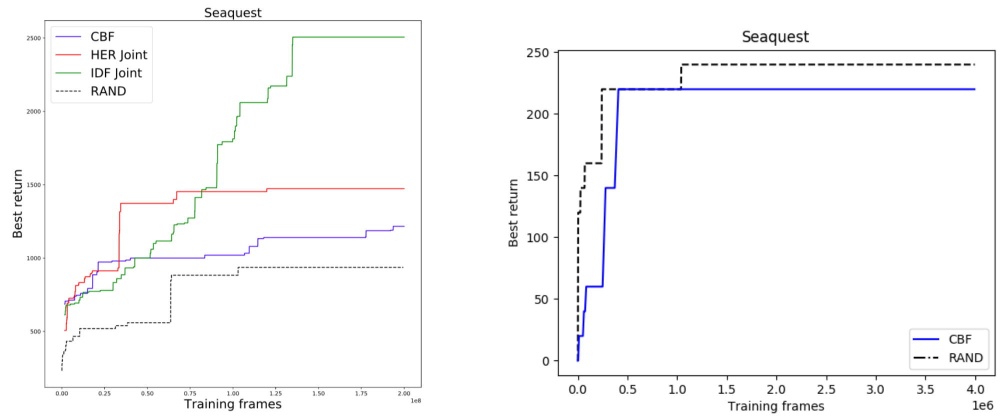


Figure 1: The best score in Seaquest obtained in a single episode for (Left) the paper that we are aiming to reproduce and (Right) our attempt.

Through looking at the best rewards in Fig. 1, it is not clear if our implementation of CBF is improving. However, by considering the average episode score in Fig. 2, we see that our CBF implementation is learning, and that CBF performs better on average than the random policy. This result would be strengthened with additional iterations.

2.7 Pong

Unlike in Seaquest, we were unable to find improvement with CBF in the Pong environment to reproduce the paper's results. We tried modifying several hyperparameters, such as the learning rate, scaling of the forward dynamics, number of rollouts, and number of optimization steps performed.

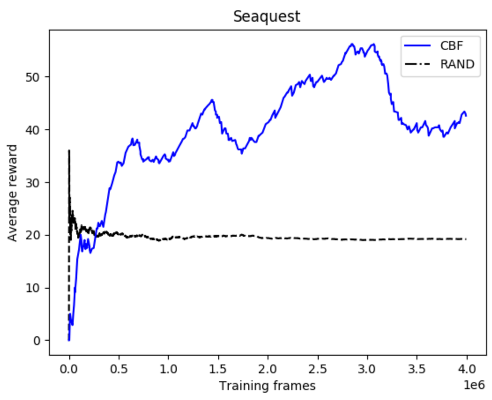
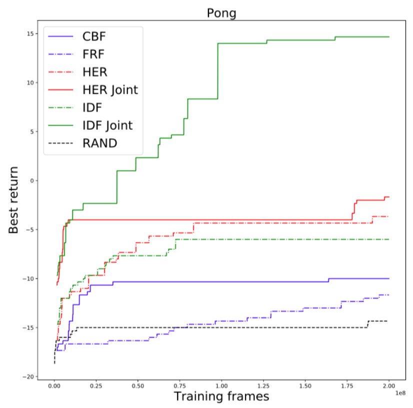


Figure 2: The mean score our CBF implementation returned while playing Seaquest.



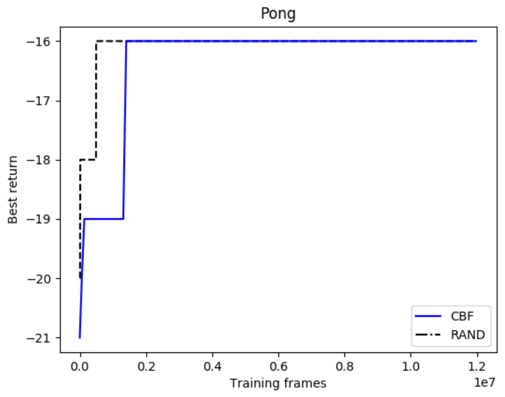


Figure 3: The best score in Pong obtained for (Left) the paper that we are aiming to reproduce and (Right) our attempt.

We also tried running several environments in parallel. Through these methods we were able to improve stability, they were not able achieve noticeable improvement.

3 Results and Conclusion

In the end, we were able to use the described algorithm, CBF, to improve a policy using an intrinsic notion of rewards for Seaquest, as seen by average reward, but not for Pong. We found that the stability of CBF was extremely sensitive to the choice of hyperparameters, not all of which were fully specified. It is possible that our approach to stabilize our implementation of CBF during training inhibited policy improvement.

References

Anonomous. Curiosity Driven Exploration By Bootstrapping Features. pp. 1–12, 2018.

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

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

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL `http://arxiv.org/abs/1705.05363`.

A Implementation Details

A.1 Preprocessing

For preprocessing of the Atari games, we used the wrappers used in the DQN implementation in Dhariwal et al. (2017) for both experiments. We modified it slightly to not automatically ”press fire” on reset, same as Anonomous (2018). We also wrapped the environment so end-of-life in the environment is end-of-episode, since this helps value estimation.

A.2 Architectures

Our implementation of PPO was heavily based on the baseline implementation in Dhariwal et al. (2017).

Our embedding network has three convolutional layers followed by a linear layer all with ReLU activations, with the convolutional layers exactly the same as the CNN policy used in the DQN implementation in Dhariwal et al. (2017). Our linear layer has 288 outputs.

Our policy network has one linear layer with ReLU activation, and two output heads for action probabilities and values, which maps the previous layer to a vector of length size of action space and to a single scalar, respectively. We followed the implementation in Pathak et al. (2017) closely.

Our forward dynamics network concatenates the state embedding with a one-hot representation of action followed by two linear layers with a residual connection (sum of squared difference between predicted and actual next state embedding) to the output. Again, we followed the implementation in Pathak et al. (2017) closely.

A.3 Hyperparameters

We used mostly the same hyperparameters as specified in Anonomous (2018), however not all were explicitly stated. We tried our best to infer the rest.

Table 1: General Hyperparameters

Hyperparameter	**Value**	**Description**
Learning rate for forward dynamics model	1×10^{-5}	Stated in paper
Replay buffer size	1000	Timesteps stored in memory buffer per environment. Stated in paper
Minibatch size	128	Size of sample from replay buffer for forward dynamics and auxiliary loss training step. Stated in paper
Size of embedding space	288	From Pathak et al. (2017)
Length of rollout	256	From PPO in Dhariwal et al. (2017)
Number of optimization steps	4	Determined by testing
Scaling Forward Dynamics Error	1/288	Determined by testing

Table 2: PPO Hyperparameters

Hyperparameter	**Value**	**Description**
clip_param	0.2	Clipping parameter epsilon
entcoeff	0.001	Entropy coeff. Stated in paper
optim_epochs	8	Number of optimization steps per epoch. Stated in paper
optim_stepsize	1×10^{-3}	Step-size for optimization
optim_batchsize	64	Size of minibatch of rollout
gamma	0.99	Gamma in advantage estimation
lam	0.95	Lambda in advantage estimation