**ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE**
**SCHOOL OF LIFE SCIENCES**

Master project in Life Sciences and Technology

# MAXIMUM LIKELIHOOD ESTIMATOR WITH VARIANCE FOR CURIOSITY-DRIVEN EXPLORATION

Carried out in the laboratory of IDIAP
at EPFL
Under the supervision of Prof. François Fleuret

Done by

**OLESIA ALTUNINA**

Under the direction of
Prof. François Fleuret
In the laboratory of IDIAP

EPFL

External Expert Pierre Baqué, Co-founder & CEO of Neural Concept

LAUSANNE, EPFL 2020

# Contents

# Summary

In reinforcement learning (RL), the external rewards are conventionally continuous. However, in scenarios close to the real world, they can be sparse or altogether absent, which undermines the efficacy of the standard learning algorithms. Following the ideas of curiosity in psychology, to improve learning in environments with sparse rewards, intrinsic motivation was introduced, by either encouraging the agent to explore novel states or by increasing agent's capacity to predict the outcomes of its own actions. Both these approaches turned out to suffer from dimensionality issues, which were intensified in the case of stochasticity. Pathak et al. [8] proposed a new intrinsic curiosity module (ICM), which consists of two models, one, inverse, that is learning meaningful embeddings through predicting actions by state pairs, and the other, forward, that predicted the next embedding from state-action pairs. The discrepancy between the predicted embedding and the actual one is fed to the RL algorithm in use as the internal reward. This algorithm successfully battled dimensionality issues and irrelevant stochasticity issues. Nonetheless, it does not work for environments where the outcomes of agent's actions can be random, which we call relevant stochasticity.

In this work, we reproduce the ICM paired with asynchronous advantage actor-critic RL algorithm from Pathak et al., that was originally written in TensorFlow, in PyTorch. On top of it, we introduce a new environment PicoImaze to test a forward loss that is based on a Gaussian density with unconstrained variance (as opposed to the original approach of fixing it) in the case of relevant stochasticity. We show that our new loss converges to values close to zero or negative, which might illustrate agent 'getting bored' as it learns more about the underlying distribution (both its mean and variance).

# Introduction

## Rewards: Continuous, Sparse, Missing

The field of reinforcement learning (RL) provides algorithms for an agent to learn 'from experience' by optimising its policy to maximise the rewards provided by an environment. Conventionally, non-zero rewards are implied to be continuously supplied to the optimisation algorithm following nearly every agent's action. The example of this might be an Atari game with its increasing score [1] or the angle of the pole in the cart-pole swing-up task [2].

However, in scenarios that are close to the real world, rewards can often be only occasional (sparse), very rare (very sparse) or there can be none at all. In such cases, the

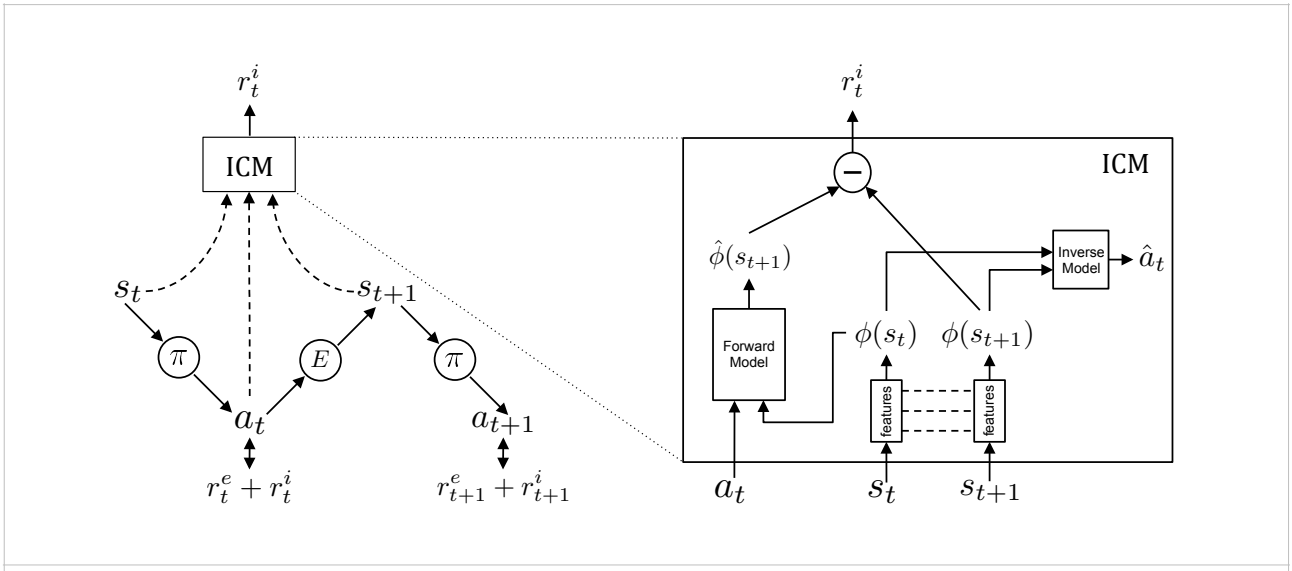



Fig. 1: The diagram from Pathak et al. representing the ICM-A3C model

behaviour of the agents trained with conventional algorithms fails to exhibit the emergent properties expected of a successfully learning system. Mainly, agents cease to systematically explore the arena, which is a problem, given that random exploration is rarely effective for any other than really simple environments.

## Intrinsic Motivation

One way to deal with this problem is to introduce intrinsic motivation into the model. In psychology, intrinsic motivation, or curiosity [3], 'refers to doing something because it is inherently interesting or enjoyable' [4], as opposed to extrinsic motivation, which refers to doing from the place of striving to reach a particular outcome. In RL, extrinsic motivation can be interpreted as the goals put in place for the agent by the environment with a set of extrinsic rewards. As for curiosity, there emerged two major approaches:

1. Encourage discovering the unexplored states [5],
2. Increase agent's capacity to predict the outcomes of its actions [6, 7].

---

### Noisy TV Problem Solved?

If the state-space is continuous high-dimensional vector-space such as images, it is quite demanding to build both a statistical model of states (1) and a dynamical model of an environment (2). Moreover, the stochasticity of the environment that the agent cannot affect, may introduce unnecessary additional complexity to the model, and ultimately make the agent get stuck. This effect was described by Jürgen Schmidhuber [7] as a noisy TV problem: if the agent sees a TV with continuously changing channels showing white-noise, it will get stuck because every state from there on will be novel, and there is no way the agent will predict the next white-noise picture. The other catch here is that besides a noisy TV problem being so hard to tackle, it may also be completely irrelevant to the goal at hand.

To deal with both high-dimensionality and irrelevant stochasticity problem, Pathak et al. [8] proposed their own intrinsic curiosity module (ICM) (Fig. 1). First, the inverse model produces state embeddings $\phi(s_t), \phi(s_{t+1})$ relevant to the agent's actions by estimating them from state pairs $s_1, s_{t+1}$. Second, the forward model predicts next state embeddings

$\hat{\phi}(s_t), \hat{\phi}(s_{t+1})$ from state-action pairs $s_t, a_t$. The discrepancy between the predicted embedding $\hat{\phi}(s_{t+1})$ and the actual one $\phi(s_{t+1})$ constitutes the internal reward $r_i$, which in the end is fed to the RL algorithm in use.

The model described above successfully tackles many tasks with sparse or altogether missing rewards, including Mario Bros. game [8, 9] and Vizdoom [8, 10] reported in the original paper, as well as a plethora of other environments from video games to physics simulations to 3D navigation tasks [11].

---

## Noisy TV in Relevant Stochasticity

There appears, however, to be another kind of a noisy TV problem here. All the above mentioned environments that ICM successfully tackles are deterministic with respect to the agent's actions, i.e. for each action, there is just one possible outcome. In the presence of the relevant stochasticity, the ICM model ceases to explore, just like the agent with high-dimensional curiosity model would in the case of irrelevant stochasticity. This is highlighted in the large-scale study by Pathak et al. [11]. In this study, they test their intrinsic curiosity model on a Unity environment with an improvised TV at some point of the maze, which they give the agent a 'remote control' to. As a result, at the moment the agent sees this TV, it gets stuck randomly switching channels.

We hypothesise that such ICM-RL dynamics in stochastic environments might stem from the chosen density for the forward model output. The authors used mean squared error (MSE) as their loss function, which corresponds to a fixed variance Gaussian density of the underlying distribution. In the stochastic scenario, however, the distribution's variance does change, which we account for in the loss that we test out in this study. To make sure that the ICM model works correctly, we also reproduce the original experiments in Vizdoom, with the code based on TensorFlow, in PyTorch.

# Methods

## New Forward Loss

The outputs of the two ICM models, inverse and forward, can be interpreted as sampled from the distributions $p(a_t \mid s_t, s_{t+1}, \theta_{inv})$ and $p(\phi(s_{t+1}) \mid \phi(s_t), a_t, \theta_{forw})$, where $\{s_t, a_t, s_{t+1}\}$ is a transition tuple and $\theta_{inv}$ and $\theta_{forw}$ are the parameters of the inverse and forward models. The baseline MSE forward loss comes from choosing to minimise the negative log likelihood

$$\mathscr{L}(\theta_{forw} \mid \phi(s_{t+1})) = -\log(p(\phi(s_{t+1}) \mid \phi(s_t), a_t, \theta_{forw})) = -\frac{1}{2\sigma^2}(x-\mu)^2 + \text{const},$$

which corresponds to the fixed-mean Gaussian density. However, if we let the standard deviation vary as well, which we would need in the case of non-deterministic outcomes, we will get the following negative log likelihood

$$\mathscr{L}(\theta_{forw} \mid \phi(s_{t+1})) = -\frac{1}{2\sigma^2}(x-\mu)^2 + \log(\sigma) + \text{const}.$$

In this study, we employ the latter function as a loss (and, potentially, curiosity reward) and test its performance in a newly built environment.

## Implementation

The original implementation of the intrinsic curiosity module (ICM) with the asynchronous advantage actor-critic (A3C) algorithm is publicly available [12]. It uses TensorFlow as its framework for neural networks, and OpenAI Gym's version of Vizdoom [13] (which is currently standalone) and Mario to provide environments for the RL agents.

As a basis for the PyTorch implementation of the model, we used a basic A3C implementation by Ilya Kostrikov [14]. On top of it, we implemented the intrinsic curiosity

module, following the implementation of Pathak et al. The hyper-parameters of the model were taken from [15].

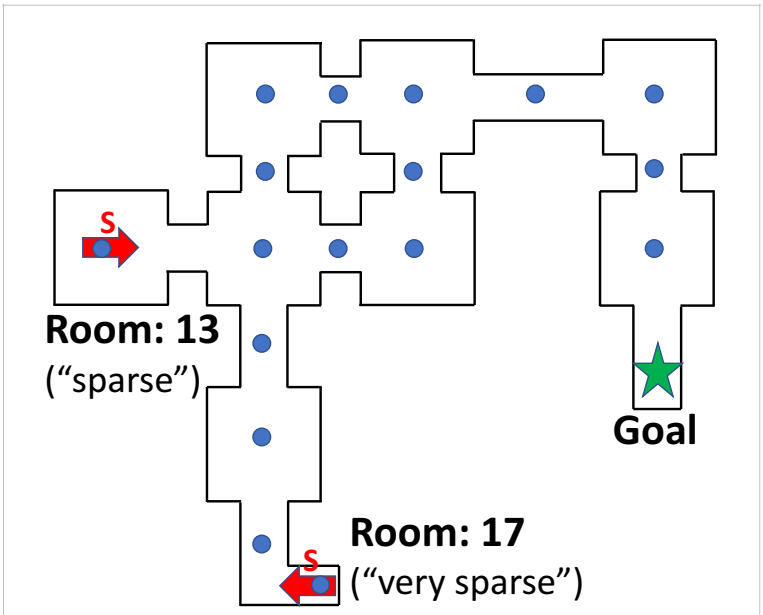



Fig. 2: Vizdoom environment from Pathak et al. Blue dots are spawning locations

We were aiming at reproducing Pathak's original TensorFlow result on Vizdoom in PyTorch. It was important to use the updated standalone version of Vizdoom though, to keep the code up to date. To couple Vizdoom's native API and Gym's API, we used Simon Hakenes's vizdoomgym, Vizdoom wrapper for Gym [16]. The scenarios were again taken from Pathak's implementation, as well as some of the environment wrappers.

To record the test performance during training in Vizdoom, we used its recording function, which stores all the actions and critical parameters for each run into a separate file that can later be used to reconstruct the video of the run. For this, we modified the following code by Sharada Mohanty [17].

The code allows for three types of games: Atari, Vizdoom and Picolmaze (the environment created to test the new loss). It is also possible to monitor and create videos for Atari games through Gym's monitor wrapper.

The implementation is organised in an asynchronous manner after Ilya Kostrikov. The file main*.py parses the arguments and creates train (train*.py) and test (test*.py) processes. The train processes run in an infinite loop while the test process wakes up and does a test run every once in a [parametrised] while, and kills all the processes after the preset number of runs. There are different train and test files for various configurations (curiosity

Fig. 3: Picolmaze environment. Left: Original pictures placed on the 3x3 grid. Right: Coloured pictures placed on the grid, 2 colours to chose from when hit ‘play’

and A3C, curiosity only, A3C only). This is done to not overload the looping functions with ‘if’ statements. The test process also logs all the necessary metrics using TensorBoard and logger (modified from [18]), which also keeps all the crucial configuration information.

## AWS and Vizdoom Issue

The training was happening on Amazon Web Services (AWS) EC2 servers with Deep Learning AMI (Ubuntu 16.04, V26.0) with preinstalled PyTorch. We chose c5.4xlarge compute optimised instances, which have 16 vCPUs with 32 GB of memory. After launching an instance, we installed all the necessary packages (the protocol was the same each time and took around 5 minutes). It was quite challenging to successfully install Vizdoom on the machine, which turned out to be due to the lack of rights to update the local cmake copy. Given that Vizdoom had to be installed using the local pip, the local copy of cmake could not build it correctly because of their incompatibility. We had to remove the local copy of cmake and update the global one in order to be able to install Vizdoom. The final solution looks like this:

```
sudo rm -r /usr/local/bin/cmake
```

```
sudo /home/ubuntu/anaconda3/envs/pytorch_p36/bin/pip install vizdoom
```

---

## Environments

Vizdoom [10] is a package by Marek Wydmuch that ‘allows developing AI bots that play Doom using the visual information (the screen buffer)’. The environment constitutes a 3D maze, which the agent can learn to navigate intelligently. The framework allows to create environments with custom scenarios, rewards and action spaces. In this study, we used the custom scenarios created by Pathak et al., 3 versions of ‘DoomMyWayHome-v0’: original, with 17 spawning locations (Fig. 2, coloured blue), sparse (spawning in location 13 only), and very sparse (spawning in location 17). In this case, Vizdoom is considered a 3D navigation task, the aim of which is to collect the west at the far location of the map (green star in Fig. 2). Discrete actions ‘forward’, ‘left’, ‘right’, ‘wait’ are available to the agent.

Apart from Vizdoom, we also created a new simple environment to test the new loss against the original one from Pathak et al. The name of the environment is Picolmaze (maze of coloured pictures), it consists of a square grid, in each cell (room) of which there is a different picture of some colour (Fig. 3). We transform the colours of the image using eighteen matplotlib colour schemes. The space of the grid is open, i.e. there are no walls as in a conventional maze. The default boundary conditions are periodical, i.e. if the agent is on the left edge of the grid and goes left, it appears on the right age of the grid. We also tried to run a model with fixed boundary conditions though, and got an interesting effect there. 4, 9, and 16 room setting were used throughout all the training in Picolmaze.

The actions available to the agent are ‘left’, ‘right’, ‘up’, ‘down’, and ‘play’. The ‘play’ action samples uniformly from the set of several colours assigned to the room. There is a special parameter function required at the initialisation stage that sets the number of options to each room. The number of options to sample from in each room defines that

room's entropy. For this study, we tried ascending entropies (number of options from 1 to the room index), zero entropy (the only option for each room) and two levels of the setup with the same number of options in each room (8 and 16).

We should note that some colour schemes are very similar, which might have lead to a situation where the room colour sample is not entirely uniform and the embeddings (the output of the inverse model) are not equidistant from each other. This does not hugely affect the overall result, but it can be one of the sources of noise in this environment.

For the training of the Picolmaze, we used uniform policy as the simplest one. We first trained only the inverse network of the ICM module, then froze it and trained the forward model. This way, we could assure that both networks converge independently by taking a look at the corresponding crucial metrics. No RL was happening in this case. Given that the A3C algorithm is notoriously tricky to fine-tune, we did not want to risk trying to do it on a completely new environment.

Initially, the idea was to baseline on the noisy TV example Unity environment from the large-scale study of curiosity-driven exploration by Burda et al., but it turned out to not be publicly available [19]. If the parameters were in-place, it would not take a long time to fine-tune the model, and baselining on something already existing would definitely be more convincing, but alas, we were not able to do it.

---

## Training Times

The training the model on the AWS machines in Vizdoom took ~4h 30m for ~19,000,000 training steps (including rollout) in the A3C-only setup (~1200 FPS), ~4h 25m for ~10,000,000 steps in the ICM-A3C setup (~630 FPS), and ~1h 40m for ~6,000,000 steps in the case of ICM-only (~1000 FPS). We always used 16 training subprocesses when running those experiments. In the case of Picolmaze, the training took ~40m for ~265,000

steps for both inverse and forward models. We only used a single training subprocess here, but launched 6 runs at a time on one machine.

Our code is available on Github [20].

# Results

## Vizdoom

Fig. 4 shows Vizdoom learning curves for 3 reward settings: dense (A), sparse (B) and very sparse (C). The blue lines represents learning curves for the A3C-only (no curiosity) model, and the yellow lines — ICM-A3C (RL + curiosity). For each of the settings, there are just two (A3C and ICM-A3C) training sessions shown. The mean and variance in the upper two rows are the ones of the distribution sampled from 10 independent test runs for each model timestamp saved during training. The lower row is taken from Pathak et al. [8], and each line there represents the mean over 3 independent training runs. Although the plots show a little bit different things, they are nevertheless comparable.

As we can see from Fig. 4, our model implemented in PyTorch shows dynamic that is similar to that of Pathak's model. From this, we can conclude that our model is implemented plausibly enough to yield a result similar to the original paper.

Now that we showed that we had successfully reproduced the model, let us move on to the experiments in the new environment.

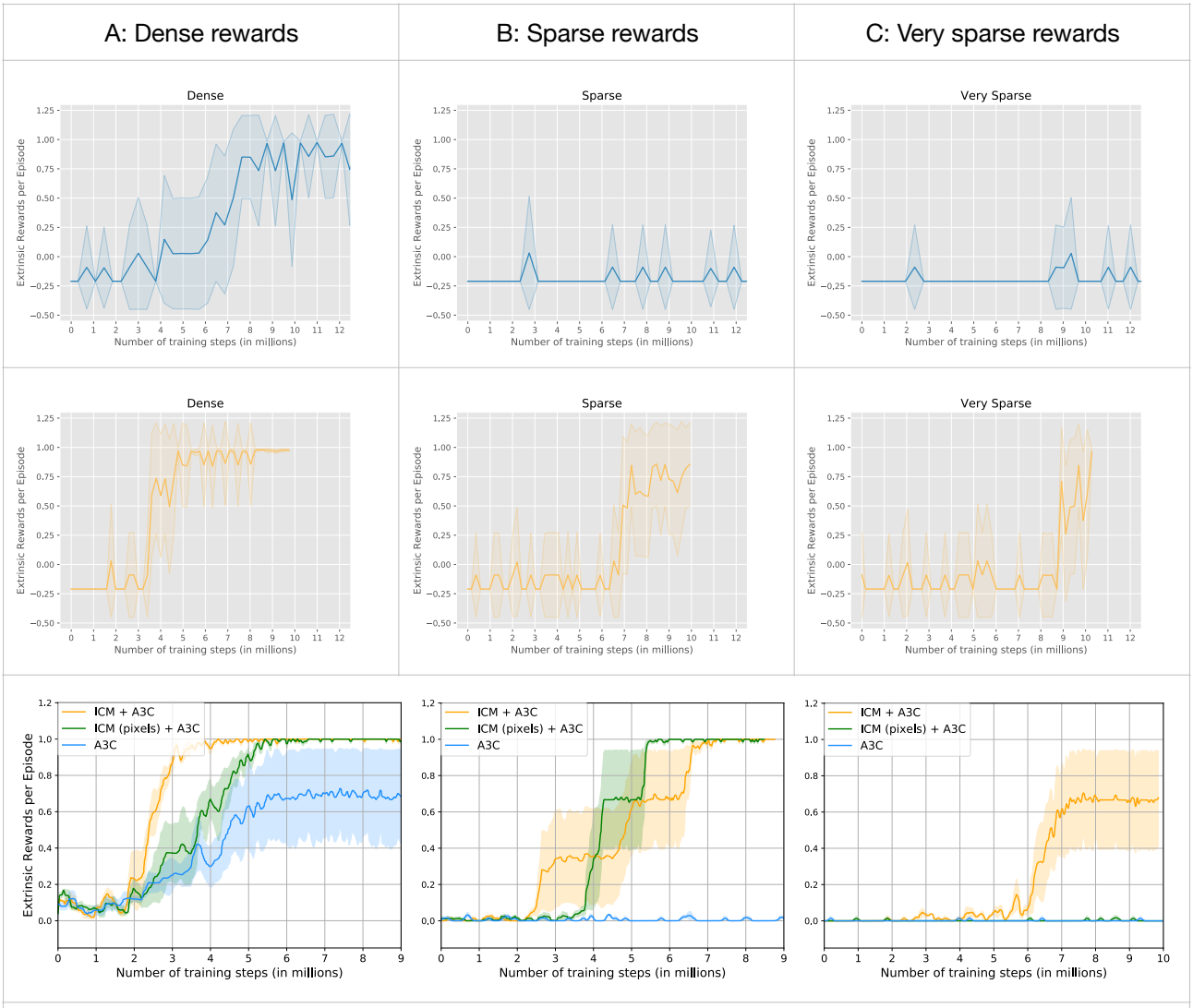



Fig. 4: Test results for the models saved throughout the training process for A3C-only (blue) and ICM-A3C (green) models and dense (A), sparse (B) and very sparse (C) environments. For each recorded model, the test was run 10 times. Filling shows a 1-std span. The tests were run in dense reward environment for all training settings. There is just one training run for each setting, which was picked from a number of train runs. Third row contains similar plots from Pathak et al.

## PicoImaze

Let us first take a look at the training metrics of the inverse model (Fig. 5). We identified three important metrics here: loss (C), max softmax (B) and accuracy (A). Loss here is the inverse loss, which is expressed as cross-entropy between the predicted action and the true one. Max softmax is the maximum value of the softmax output of the action prediction, i.e. the 'certainty' of the most likely action. Finally, accuracy is the percentage of actions predicted correctly.

One trend in these graphs one can clearly observe is that the curves for the deterministic setting negatively stand out comparing to the stochastic ones. Initially, we hypothesised that the accuracy for every setting would be quite high (and the losses — comparable). For the deterministic setting, because in order to predict 'play', the network would only need to assure that the two states are exactly the same (since when the agent hits 'play' in the deterministic setting, the state stays the same). For the stochastic setting, because the correct 'play' prediction would require successful comparison of two pictures that have very similar intensity landscape, but have different colours. As for the movements predictions, they are pretty straightforward in both cases: the network would be required to 'build a spatial map' of the arena by comparing each two dissimilar pictures.

To take a closer look into the inverse model, we created the learning curves that show the percentage of incorrect guesses by action (Fig. 6). In these diagrams, we observe that the amount of incorrect predictions for the movement actions converges very quickly and stays the same throughout all settings for each number of rooms. E.g., for 4 rooms it reaches ~0.1, for 9 rooms — ~0.07, for 16 rooms — ~0.05. As for the 'play' action, it is upper-bound by about twice as high values in the case of the deterministic setting, as opposed to all the other, stochastic, settings, where it converges to levels that are comparable or way lower than those of the movement's actions.

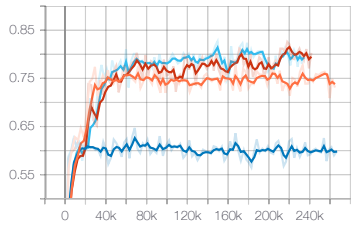
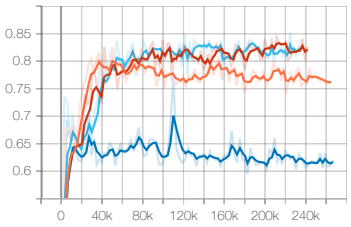
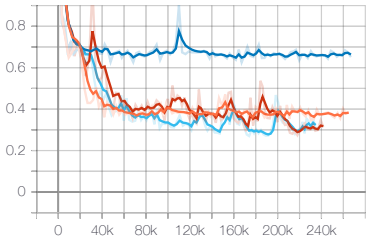
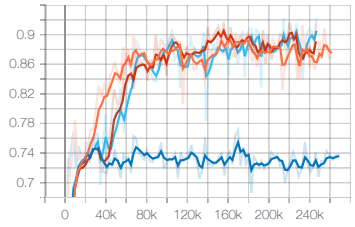
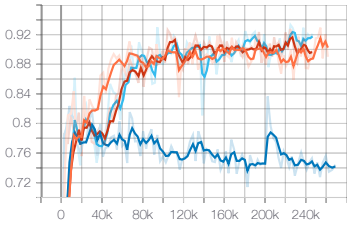
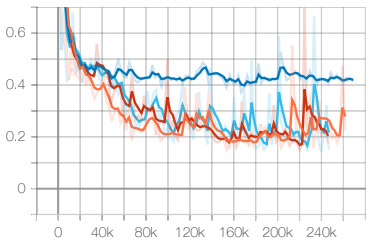
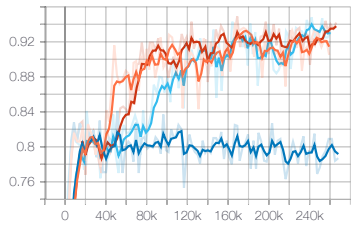
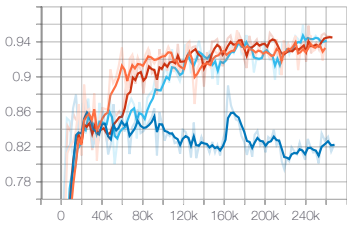
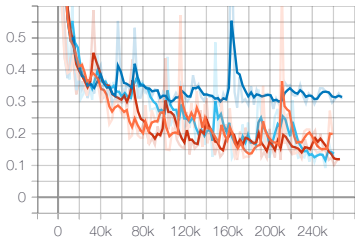
| Rms | A: Accuracy | B: Max softmax | C: Loss |
|---|---|---|---|
| 4 |  |  |  |
| 9 |  |  |  |
| 16 |  |  |  |

Fig. 5: Inverse model training metrics (A, B, C) as a function of the number of steps (uniform policy, periodic boundary conditions) for different numbers of rooms (4, 9, 16)

Legend: **Ascending entropies** **Deterministic setting** **16 options per room** **8 options per room**


Fig. 6: Fraction of misclassified actions (inverse model, uniform policy, periodic boundary conditions) for different numbers of rooms (4, 9, 16)

We first thought that this might be due to the fact that the the training and testing environments were created independently in the respective subprocesses, and thus the pictures in them had the same spatial locations but distinct colours. We ran the same experiment in two other more plausible scenarios: one where the two environments, training and testing, were exactly the same, and the other one where the training environment was ‘hard reset’ (created anew with different colours) at the end of each episode (1000 steps). However, the trend persisted.

One possible explanation for this is that the majority (4/5) of the input state-pairs, which corresponds to the movement actions, were dissimilar in both their contents and colours. This might have biased the network against the ‘play’ action predictions, which were similar in either just the colour or both the colour and contents. And one likely reason for the convergence of the incorrect ‘play’ predictions to almost zero in the cases of stochasticity (comparing to the movement predictions that stay at the same level) might be the following. Whenever there are two pictures that have the same landscape but different colours, it is only possible that the action is ‘play’. But in the case of the movement actions, the ground truth for two completely different images can be any of the four: ‘left’, ‘right’, ‘up’ or ‘down’.

Now let us move on to the forward model (Fig. 8) and compare the learning curves for the baseline forward loss (Fig. 8A) and the proposed Bayesian loss (Fig. 8B). First, in the 16-room setting, we can clearly see that in stochastic environments, the baseline loss converges to ~1000, while the Bayesian loss reaches 100 and below. This can be interpreted as the agent getting the more bored the more it learns about the environment’s state distributions. In the deterministic setting, however, the baseline loss converges to zero, and the Bayesian loss goes way below. This can be explained by the properties of the Bayesian loss function $(x-\mu)^2/2\sigma^2 + \log(\sigma)$ (Fig. 7). When the value

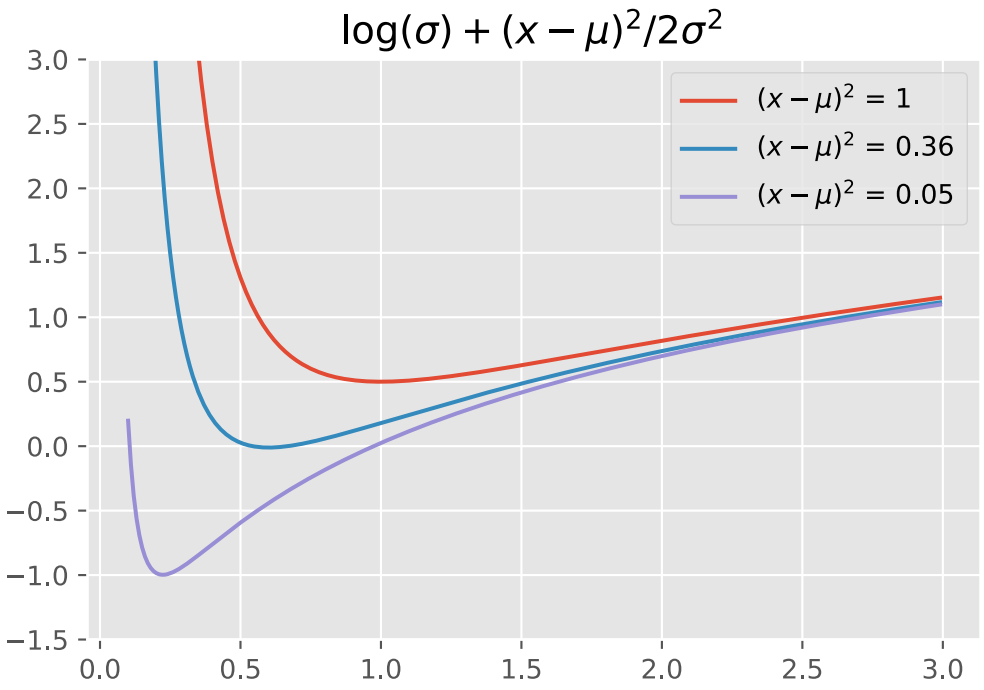



Fig. 7: The shape of the Bayesian loss function w.r.t. variance

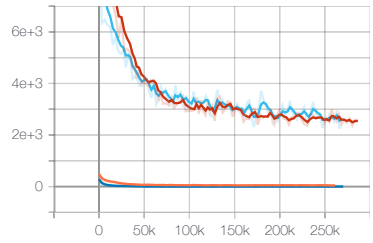
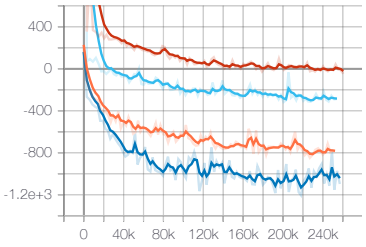
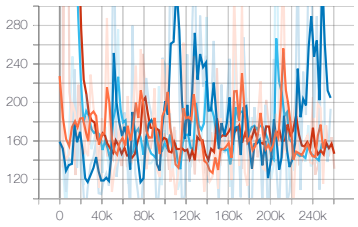
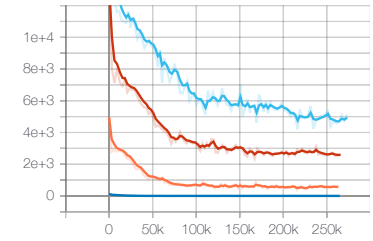
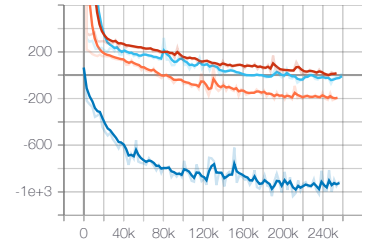
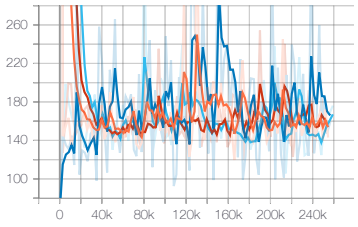
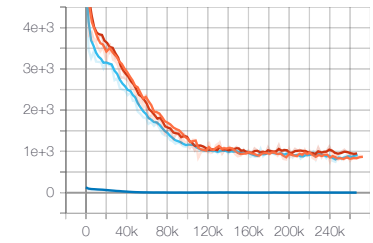
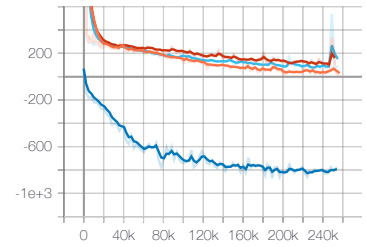
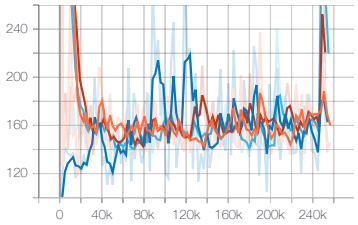
| Rms | A: Baseline forward loss | B: Bayesian forward loss | C: Bayesian forw. loss w/o regular. |
|---|---|---|---|
| 4 |  |  |  |
| 9 |  |  |  |
| 16 |  |  |  |

Fig. 8: Forward model training metrics (A, B, C) as a function of the number of steps (uniform policy, periodic boundary conditions) for different numbers of rooms (4, 9, 16)

Legend: **Ascending entropies** **Deterministic setting** **16 options per room** **8 options per room**

of the $(x - \mu)^2$ part goes below ~0.36, i.e. when the absolute difference between the true and predicted mean is less than 0.6, the function can take negative values in sigma in the interval (0, 1). Thus, when we see the Bayesian loss going negative, it means that there are at least some of the variances of the embedding vector that are less than one.

Another interesting observation is that the more rooms there are in the environment, the closer together are the stochastic learning curves. In the case of the loss this, again, stems from the properties of the function. In Fig. 9B, C, we can note that in the case of 16 rooms and stochastic environments, both the mean and the standard deviation curves are close together, with the mean difference being quite high, ~0.6, and variances converging to ~2.25. We can observe the same rule for the other cases as well.

The relative mean values acquired from training the forward network with the new loss are consistent with that of the MSE loss scenario (Fig. 9A, B). Here, we can also note the metrics clustering w.r.t. the number of rooms. It seems to be an artefact of the environment, but, unfortunately, we did not have enough time to look into it closer.

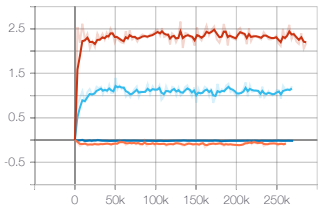
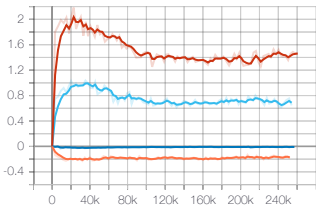
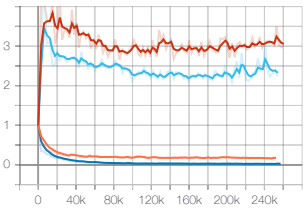
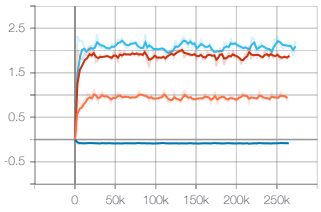
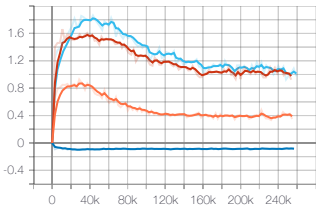
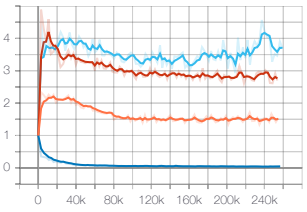
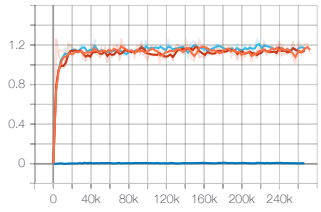
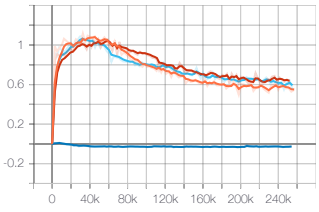
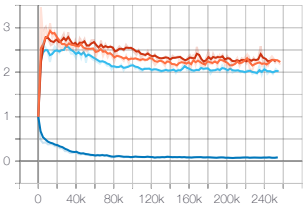
| Rms | phi2 | A: Baseline forward predicted mean | B: Bayesian forward predicted mean | C: Bayesian forward predicted std |
|---|---|---|---|---|
| 4 | 2.3<br>1.3<br>+0.<br>0<br>+0.<br>0 |  |  |  |
| 9 | 2.1<br>1.9<br>+0.<br>0<br>-0.0 |  |  |  |
| 16 | 1.2<br>1.2<br>1.1<br>+0.<br>0 |  |  |  |

Fig. 9: Forward model training metrics (A, B, C) as a function of the number of steps (uniform policy, periodic boundary conditions) for different numbers of rooms (4, 9, 16)

Legend: **Ascending entropies** **Deterministic setting** **16 options per room** **8 options per room**

# Conclusion

In this work, we reproduce an intrinsic motivation module paired with asynchronous advantage actor-critic RL algorithm from Pathak et al. [8], that was originally written in TensorFlow, in PyTorch. On top of it, we introduce a new environment Picolmaze (square grid of coloured pictures, Fig. 3) to test a forward loss that is based on a Gaussian density with unconstrained variance (as opposed to the conventional approach of fixing it) in the case of relevant stochasticity (when the outcomes of agent's actions are random). We show that our new loss converges to values close to zero or negative, which might illustrate agent 'getting bored' as it learns more about the underlying distribution (both its mean and variance).

The negative rewards are common practice in RL, so we can use the new Bayesian loss as a reward, too. In fact, the decision of Pathak et al. to use the factorised MSE loss as intrinsic reward is quite empirical: there are no formal constraints or requirements what rewards should look like. In the case of negative curiosity reward, we can think of an experiment [21] where humans struggled to entertain themselves when left alone in a room, and some even chose to give themselves a mild electric shock that was placed in the room. It is as if not learning the new information about the environment is more 'painful' than getting an electric shock.

In this study, we did not train an RL agent to explore the environment with the Bayesian loss. Training such an agent would probably require factorising the rewards, possibly, by different amounts the positive and the negative.

It might also be interesting to take a closer look at the Picolmaze embeddings to understand why it is that the more rooms there are in the environment, the closer together stochastic means and variances get. Moreover, why is the inverse model's accuracy for

the deterministic setting way below that of the stochastic settings'? We offered possible explanations for these effects, but did not check them directly.

While reproducing the ICM-A3C model, we tried to follow the original paper and code as closely as possible, and although our respective models converge a bit (~1-2 million training steps) slower than those in Pathak's implementation, the paces are quite similar. This might be due to computational efficiency difference in TensorFlow and PyTorch, although it is hard to know for sure.

# References


1. Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. Nature, 2015.

2. Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. ICLR, 2016.

3. Silvia, Paul J. Curiosity and motivation. In The Oxford Handbook of Human Motivation, 2012.

4. Ryan, Richard; Deci, Edward L. Intrinsic and extrinsic motivations: Classic definitions and new directions. Contemporary Educational Psychology, 2000.

5. Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In NIPS, 2016.

6. Mohamed, Shakir and Rezende, Danilo Jimenez. Variational information maximisation for intrinsically motivated reinforcement learning. In NIPS, 2015.

7. Schmidhuber, Jürgen. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). IEEE Transactions on Autonomous Mental Development, 2010.

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

9. Paquette, Philip. Super mario bros. in openai gym. https://github.com/ppaquette/gym-super-mario, 2016.

10. Kempka, Michał, Wydmuch, Marek, Runc, Grzegorz, Toczek, Jakub, and Jaśkowski, Wojciech. Vizdoom: A doom-based AI research platform for visual reinforcement learning. In IEEE Conference on Computational Intelligence and Games, 2016.

11. Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, Alexei A. Efros. Large-scale study of curiosity-driven learning. arXiv:1808.04355, 2018.

12. https://github.com/pathak22/noreward-rl, 2020.

13. https://github.com/mwydmuch/ViZDoom, 2020.

14. https://github.com/ikostrikov/pytorch-a3c, 2020.

15. https://github.com/pathak22/noreward-rl/blob/master/src/constants.py, 2020.

16. https://github.com/shakenes/vizdoomgym, 2020.

17. https://gitlab.crowdai.org/vizdoom/vizdoom-subcontractor/blob/master/generate_video.py, 2020.

18. https://github.com/ikostrikov/pytorch-a3c/pull/14/commits/02d6f658304dedc3282c27e2b1ed79136363dfe4, 2020.

19. https://github.com/openai/large-scale-curiosity, 2020.

20. https://github.com/utanashati/curiosity-recast, 2020.

21. Wilson, Timothy D.; Reinhard, David A.; Westgate, Erin C.; Gilbert, Daniel T.; Ellerbeck, Nicole; Hahn, Cheryl; Brown, Casey L.; Shaked, Adi. Just think: the challenges of the disengaged mind. Science, 2014.