INTRINSIC CURIOSITY IN REINFORCEMENT LEARNING BY IMPROVING NEXT STATE PREDICTION

by

PAUL LEWIS LOBO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science at
The University of Texas at Arlington
May 2020

Arlington, Texas

Supervising Committee:

Deokgun Park, Supervising Professor
Manfred Huber
Won Hwa Kim

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Deokgun Park for his immense support and guidance throughout my entire two years here at the University of Texas at Arlington. I would also like to thank Dr. Manfred Huber and Dr. Won Hwa Kim for being a part of my committee. Finally, I would like to thank Dr. C.Y Choi, Dr. Michael Ward and Peace Ossom-Williamson for giving me amazing opportunities to work with them during my time here.

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

INTRINSIC CURIOSITY IN REINFORCEMENT LEARNING

BY IMPROVING NEXT STATE PREDICTION

PAUL LEWIS LOBO, M.S.

The University of Texas at Arlington, 2020

Supervising Professor: Deokgun Park

In Reinforcement Learning, an agent receives feedback from the environment in the form of an extrinsic reward. It learns to take actions that maximize this extrinsic reward. However, to start learning, the agent needs to be able to get feedback from the environment by using random actions. This works in environments with frequent rewards, however, in environments where the rewards are sparse the probability of reaching any reward even once becomes very low. One way to explore an environment efficiently is for the agent to generate its own intrinsic reward by using the prediction error from a model that is trained to predict the next state based on the current state and action. This intrinsic reward is like the phenomena of curiosity and leads the agents to revisit states where the prediction error is large. Since predicting the next state in pixel space is not a trivial task, efforts have been made to reduce the complexity by using different ways to extract a smaller feature space to make the prediction on. This thesis explores a couple of ways to stabilize the training when using a Variational Autoencoder (VAE) to reduce the complexity of the next state prediction. It looks at using a memory to train the VAE so that it does not overfit to a batch, it uses a recurrent layer to improve the next state prediction and it integrates the concept of Learning Progress so that the agent does not get stuck trying to predict something it cannot control.

TABLE OF CONTENTS

CHAPTER 1:

INTRODUCTION

Reinforcement Learning algorithms have been proven to work well when the environment provides frequent feedback to the agent. This feedback that is received from the environment is called an Extrinsic reward. In order for the agent to maximize the reward that it receives from the environment, it first takes random actions until it can reach the reward enough times to be able to improve its policy to by increasing the probability of taking actions that lead to these rewards. However, in environments where the rewards are sparse, the probability of reaching the reward initially becomes very low if we take random actions. This leads to the problem of how to explore the environment efficiently. One way to do so is for the agent to supplement the extrinsic reward with a reward that it generates itself that would lead it to revisit new states more often. This kind of reward generated by itself is called an Intrinsic reward [18] and is like the phenomena of curiosity. There have been many different approaches to defining an intrinsic reward by using prediction error in a compressed pixel space using a Variational Auto Encoder [19, 20]. Instead of predicting the next state in pixel space, the paper [12] introduced a way to extract features that are dependent only on the action taken. They showed that this works well in a maze that had various textures on the wall which tend to distract agents trying to predict the next state in pixel space. However, [15] showed us that this kind of Inverse Dynamics model is not able to perform well when encountering white noise. They introduced a TV with static on it and gave the agent an action to turn off the TV. Since the agent still gets distracted by the white noise on the TV, it shows that the Inverse Dynamics model is not able to completely remove the coupling between features that do not affect which action should be taken.

Since, predicting the next state in pixel space can be very complex due to the number of variables that need to be predicted for an image, methods [19, 20] have shown that we can use a Variational Auto Encoder [5]

to compress an image and reduce the size of the problem. This thesis investigates three methods that could improve the performance of the intrinsic agent using prediction error in compressed latent space. The first is to stabilize the training of the Variational Auto Encoder by providing a replay memory like that used in [13]. The second is to add a recurrent layer to the prediction module. This simple technique has been overlooked in [15]. Finally, we see whether we can incorporate the notion of Learning Progress [17] to prevent a curious agent from getting distracted by trying to learn to predict things that it cannot control.

In Chapter 2 we start with the basic introduction to Reinforcement Learning concepts. In Chapter 3 we look at some of the main model free Reinforcement Learning algorithms. Chapter 4 introduces us to the problem of sparse rewards, and we look at some of the approaches used to solve sparse environments. In Chapter 5 we look at how we can improve the performance of an intrinsic agent using a Variational Auto Encoder to reduce the complexity of predicting the next state in pixel space. And finally, in Chapter 6 we compare the results on the My Way Home task [21].

CHAPTER 2:

REINFORCEMENT LEARNING FUNDAMENTALS

In a Reinforcement Learning problem, an agent learns to interact with an environment by performing actions and receiving feedback from the environment in terms of a reward signal. Since we do not tell the agent what actions to take, it is up to the agent to try different actions in different scenarios and learn what actions to take to maximize the reward signal from the environment. It does this by increasing the chance of taking an action that leads to higher rewards and reducing the chance of taking an action that leads to low rewards. We rely heavily on [9] for some of the concepts mentioned below.

2.1 State

A state encompasses all the information required for an agent to decide what action it should take. For example, in a game it could be the pixels that are viewed on the screen or it could be a vector representation of some numbers that each contain some information used to distinguish one state from another. The current state is denoted as S_t.

2.2 Action

An action is used to affect the environment. Different actions at the same state could lead to different outcomes. An action could be of different types. The most common and discrete or continuous.

2.3 Policy

A policy can be thought of as a mapping from a state to an action. It tells us what action a to take when in a state s. The policy could be deterministic where we always take the current best action, or it could be

stochastic where it contains a probability distribution over all the actions that could be performed in the given state.

Deterministic Policy: $\pi(s)$

Stochastic Policy: $\pi(s|a) = \mathrm{P}[A_t = a|S_t = s]$

2.4 Reward

A reward is single number received from an environment when an action a is performed in state s. The goal of the agent is to maximize the total reward received by the agent over time. The reward signal helps us determine what are good and bad actions. If the reward that is received in the sequence after taking an action is low, it allows the agent to change its policy to reduce taking such an action in that state. Whereas if the reward received when taking an action is high it allows the agent to alter its policy to increase the chance of taking that action in that state. When taking an action A_t at state S_t we receive reward R_{t+1} and next state S_{t+1}.

2.5 Cumulative Reward

In Reinforcement Learning it is important to note that the agent does not just maximize the reward at the current state, it is necessary to maximize the reward over an entire episode or trajectory. This is since an agent should not try to achieve a short-term reward that could lead it to get stuck or not be able to reach future rewards that are higher. The cumulative reward is represented as G_t.

Cumulative Reward: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T = \sum_{k=0}^{\infty} R_{t+k+1}$

2.6 Discounted Cumulative Reward

Another vital concept in reinforcement learning is that of discounting rewards. The reason to this is so that an agent gives slightly less importance to rewards it receives further in the future. This is done as it is difficult to predict very far into the future. If an agent could receive the same reward 10 steps into the future

vs 100 steps into the future it will try give more priority to the reward that could be received within 10 steps. The discount factor is given by gamma or γ. The value of gamma is generally < 1. If it is q then there is no discounting and if it is 0 it will not consider future any rewards.

Discounted Cumulative Reward: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

2.7 Reinforcement Learning Methods

2.7.1 Q Learning

The Q Function can be thought of as a function parametrized by the current state and action and it returns the expected Cumulative Future Discounted Reward from that state until the terminal state. When the state space is small, we can build a table that encompass all states and actions. If the number of states is n and number of actions is m, the q table would be of size n by m.

The Q function is given by: $Q^{\pi} = E[R_{t+1} + R_{t+2} + \cdots | s_t, a_t]$

We learn the Q function by optimizing the bellman equation:

$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_a Q(s',a') - Q(s,a)]$

We then try to maximize our score function by taking the action with the best Q Value.

The entire algorithm from [9] is given as:

$Initialize\ Q, for\ all\ s \in S, a \in A(s), arbitrarily, and\ Q(terminal-state, .) = 0$
$Repeat\ (for\ each\ episode)$:

$Initialize\ S$

$Repeat\ (for\ each\ step\ of\ episode)$:

$Choose\ A\ from\ S\ using\ policy\ derived\ from\ Q\ (e.g.\ \in -greedy)$

$Take\ actino\ A, observe\ R, S'$

$Q(S,A) = Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$

$S \leftarrow S'$

$until\ S\ is\ terminal$

The intuition behind this is that if we are we move from a good state to a state with all bad future estimates, the current Q value would decrease and when we move to a state that has a better Q value then we would increase the current Q Value.

We do not actually have to wait until the end of the episode to calculate the future discounted reward. Instead we use the estimated future discounted reward by looking up our Q Table and taking the maximum value over all possible actions. This approach is called a Temporal Difference Method.

2.7.2 Deep Q Learning

As the number of states grows in a problem it becomes computationally infeasible to build such large tables for the Q table. With the advent of Neural Networks, it becomes possible to use a Neural network as a function approximator for the state space. The input to the neural network is the current state and it outputs the Q value for each possible action. The paper Playing Atari with Deep Reinforcement Learning [13] showed that this approach works on multiple different Atari games. The additional difference compared to Q Learning is that you generally add a memory called an experienced replay so that the neural network does not overfit to the current trajectory.

The Deep Q Learning algorithm from [13] is given as:

$Initialize\ replay\ memory\ D\ to\ capacity\ N$

$Initialize\ action-value\ function\ Q\ with\ random\ weights$

for episode $= 1, \mathrm{M}$ do

$Initialize\ sequence\ s_1 = \{x_1\}\ and\ preprocessed\ sequenced\ \emptyset_1 = \emptyset(s_1)$

$for\ t = 1, T\ do$

$With\ porbability\ \in selct\ a\ random\ action\ a_t$

$Otherwise\ selct\ a_t = max_a Q^*(\emptyset(s_t), a; \theta)$

$Execute\ action\ a_t\ in\ emulator\ and\ observe\ reward\ r_t\ and\ image\ x_{t+1}$

$Set\ s_{t+1} = s_t, a_t, x_{t+1} and\ preprocess\ \emptyset_{t+1} = \emptyset(s_{t+1})$

$Store\ transition\ (\emptyset_t, a_t, r_t, \emptyset_{t+1}) in\ D$

Sample random minibatch of transitions $(\emptyset_t, a_t, r_t, \emptyset_{t+1})\ from\ D$

$$Set\ y_i = \begin{cases} r_j & for\ terminal\ \emptyset_{j+1} \\ r_j + \gamma\ max_{a'} Q(\emptyset_{j+1}, a';\ \theta) & for\ non-terminal\ \emptyset_{j+1} \end{cases}$$

$Perform\ a\ gradient\ descent\ step\ on\ (y_i - Q(\emptyset_j, a_j;\ \theta))^2$

$end\ for$

$end\ for$

2.7.3 Policy Gradients

One of the drawbacks of Value based methods is that when the number of actions is continuous, it is impossible to learn a Value for each possible action in a continuous scale. To overcome this, we use Policy Gradient based methods which directly learn a probability distribution for which action to take given a state.

One of the basic policy gradient algorithms is called REINFORCE which is given by

Input: a differentiable policy parametrization $\pi(a|s,\theta)$

Initialize policy parameter $\theta \in R^{d'}$

Repeat forever:

Generate an episode $S_0, A_0, R_1 \dots S_{T-1}, A_{T-1}, R_T$

For each step from $t = 0, \dots T-1$:

$G \leftarrow return\ from\ step\ t$

$\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta \ln \pi(A_t, |S_t, \theta)$

2.7.4 Actor Critic

In policy gradient methods we have shown that we can directly learn a probability distribution for each action given a state. To maximize the score, we need to maximize the following equation:

$$\nabla J(\theta) = E_\pi[\nabla \log(\pi(\tau)) r(\tau)]$$

This method is a Monte Carlo method as we need the reward for an entire trajectory before we can make an update. Since different trajectories are taken from one step it suffers from high variance. Some of these drawbacks can be overcome by using Actor Critic methods. To do so we replace the rewards from the trajectory with a Q Value estimate effectively converting the Monte Carlo approach to a Temporal Difference approach. The optimization function becomes:

$$\nabla J(\theta) = E_\pi[\nabla \log(\pi(\tau)) Q(s_t, a_t)]$$

To get the Q Value, we need to run two networks. One to output the probability distribution for each action and one to output the Q Value for each action.

The Advantage Actor Critic method reduces the variance further by using a baseline.

Here the advantage is the Q value of an action given a state minus the average Value of that state. It is basically tells us how much better or worse that action is compared to the average Value of the state.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

Instead of having to use three networks, one for the policy, one for the Q function and one for the Value function, we can replace the Q function with the current reward plus the discount Value function of the next state.

The advantage becomes:

$$A(s_t, a_t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Finally, the Advantage Actor Critic optimization function becomes:

$$\nabla J(\theta) = E_\pi[\nabla \log(\pi(\tau)) A(s_t, a_t)]$$

2.7.5 Proximal Policy Optimization

The main goals behind the paper "Proximal Policy Optimization" [2] was to first Limit the updates to ensure stable learning, second to use Off Policy updates to increase sample efficiency and third to be simple to implement. The problem with Advantage Actor Critic is that it is still very prone to the step size when doing gradient descent. If the step size is too small it takes very long to learn while if the step size is large, it causes very bad drop offs in performance due to high variability in trajectories. The paper Trust Region Policy Optimization [14] (TRPO) addressed this problem by adding a constraint on the KL divergence between the old policy and new policy. However, this method is complex to implement, does not scale to large scale networks and does not allow sharing of layers between the Value function and Policy function. With that in mind, the goals of the Proximal Policy Optimization paper were to first limit the updates at each step to ensure stable training, second to use off policy samples to increase sample efficiency and third to be simple to implement. They introduced a new clipped surrogate objective function which is simple to

implement. Like TRPO, they use the ratio between the new policy and old policy. If the Ratio is greater than 1 it means that the current action is more likely under the new policy, if it is negative it means the action is less likely. However, unlike TRPO instead of using the KL divergence to constrain how large an update can be, the use a simple clip function that limits how far the ratio can vary from 1. For example, we could limit the ratio between 0.8 to 1.2. Due ensures much more stable training, is simple to implement and scales well to large problems.

Advantage Actor Critic Loss:

$$L^{PG}(\theta) = \widehat{E_t}[\nabla_\theta \log \pi_\theta(a_t, s_t)\widehat{A_t}]$$

Like TRPO, instead of using the log of the policy we can use the ratio between the new policy and new policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t,s_t)}{\pi_{\theta old}(a_t,s_t)}, \textit{here } r_t(\theta) \textit{ denotes the ratio between the new policy and old policy.}$$

The surrogate objective function that is maximized in TRPO is:

$$L^{CPI}(\theta) = \widehat{E_t}\left[\frac{\pi_\theta(a_t,s_t)}{\pi_{\theta old}(a_t,s_t)}\hat{A}_t\right] = \widehat{E_t}[r_t(\theta)\hat{A}_t]$$

The Clipped Objective Function introduced in PPO becomes:

$$L^{CPIP}(\theta) = \widehat{E_t}[\min(r_t(\theta)\,\widehat{A}_t, clip(r_t(\theta), 1-\in, 1+\in)\,\widehat{A}_t)]$$

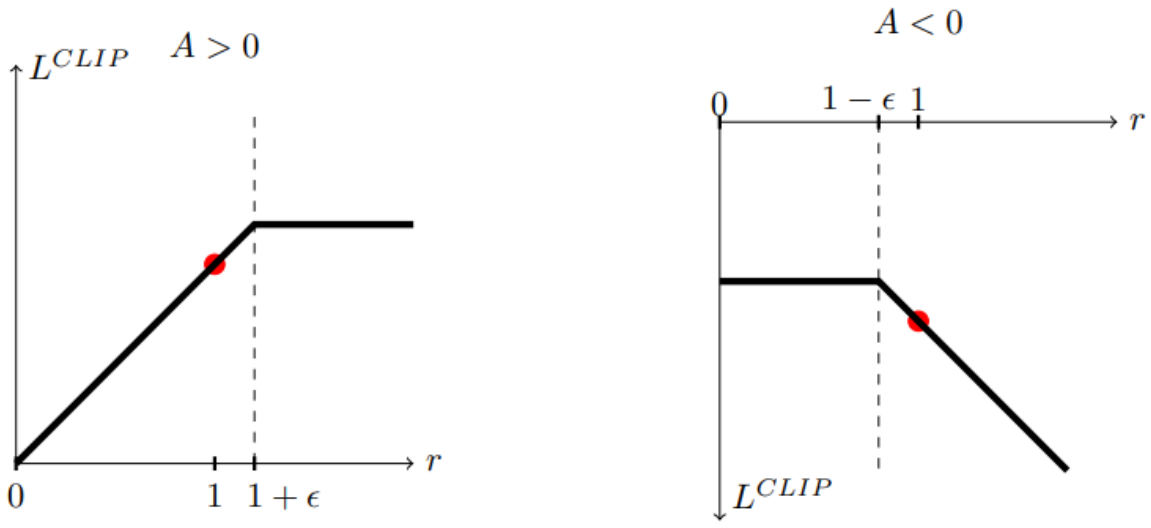

Figure 1: Image from [2] that shows the simple clipping approach to ensure stability.

Since we take the minimum between the clipped and unclipped objective function, we can consider two cases. In the first case when the advantage is greater than zero, the maximum that it can to it 1 + epsilon. Whereas when the advantage is negative, the minimum that it can go to is 1 – epsilon.

Comparing this with TRPO, it effectively optimizes the same objective function however instead of adding a complex KL divergence constraint, it adds a simple clip function that is efficient and works very well.

CHAPTER 3:

NEED FOR INTRINSIC MOTIVATION

3.1 Sparse Rewards

Model free Reinforcement Learning algorithms have been proven to work well in various tasks when the rewards received from the environment are frequent. Since they constantly receive feedback from the environment, they can learn quite quickly. However, in environments where rewards are sparse, the agent needs to try many sequences of actions in order to receive any reward at all. As the environment gets more complex using random action would not lead to any results as the probability of reaching the reward by just taking random actions can be very low. Even if the agent can reach the reward once, it is not enough as it needs to be able to do so a few times to make any decent progress.

A very simple example of this is the Continuous Mountain Car problem in Open AI's Gym [10]. In this problem, a car needs to build up momentum in order to reach the flag at the top of the mountain. Since the car is underpowered, by just taking the action right it is not able to reach the top. The car also receives a negative reward proportional to the amount of energy it uses at each step. Since using random actions does not lead to reaching the reward, instead of learning to keep exploring the environment, it learns to reduce the amount of energy spent and eventually stops taking any actions. Even some of the most advanced model free algorithms like Proximal Policy Optimization [2] struggle to solve this task.

3.2 Reward Shaping

The Mountain Car problem mentioned above can be solved by changing the reward structure, instead of keeping a single reward at the top of the mountain we could give the agent a reward proportional to the distance it is from the flag. This simple change in the reward structure would make the environment simple to solve. This change is the reward is called reward shaping and is a tool that can be used to reduce the complexity of an environment. However, we cannot possibly do this for every environment, especially if we want the agent to come up with its own method to reach the goal.

3.3 Extrinsic vs Intrinsic Rewards

To overcome such a problem, it is necessary to introduce a better way to efficiently explore an environment instead of just taking random actions. One way to do so is for the agent to generate a reward based on it reaching new interesting states. This kind of reward generated from within the agent is called an Intrinsic reward and it can be likened to the curiosity of a human when interacting with something new. Extrinsic reward on the other hand is something external to the agent and this is generally the reward received from the environment. We are now going to look at some of the different methods of formulating an intrinsic reward.

3.4 Different Formulations of Intrinsic Reward

3.4.1 Next State Prediction in Pixel Space

A very simple approach to the Intrinsic reward, is to use the prediction of the next state based on the current state and action taken. For new states, the agent will not be able to predict what happens when it tries out all the possible actions for that state. As the agent reaches the same state more often its prediction of what happens based on what action it takes gets better and the intrinsic reward is reduced. As the reward is reduced, the agent tries to reach new states that gives it a better reward. We could use the prediction error

between the next state and the predicted next state as the intrinsic reward. By using pixel space, we use the prediction error of the difference in pixel values between the actual next state and the predicted next state.

3.4.2 Next State Prediction Using a VAE

The problem with using Pixel space for the intrinsic reward is that the number of variables that need to be predicted is very large. For example, even for very small colored images of size 64 by 64 would mean predicting 64 * 64 * 3 or 12,288 variables. This is not a trivial task, and the performance generally degrades in more complex environments. One way to improve this method is to use a Variational Auto Encoder [5] (VAE) to compress each image. A VAE can be used to compress an image into a much smaller latent representation.

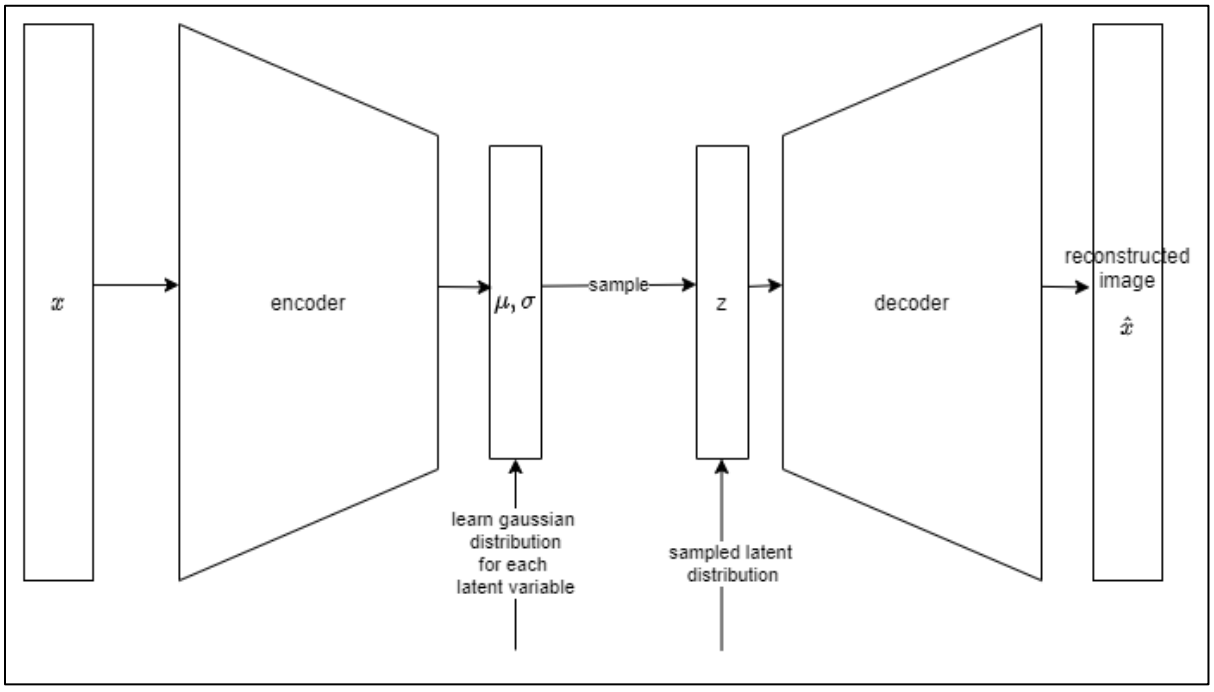

Figure 2: Variational Auto Encoder

However, unlike normal Auto Encoders that just compress the data into smaller dimensions, in a VAE we also regularize the latent representation to be similar to a unit gaussian distribution. In a plain Auto Encoder, close points in the latent space could be complete dissimilar once decoded. There could also be points that

do not represent anything once decoded. Since A VAE's latent space is regularized to be similar to a unit gaussian, variables close to each other in latent space would be similar once decoded.

By encoding the data into a much smaller space, we reduce the complexity of predicting the next state drastically. For example, using the same image of 64 by 64 by 3 we could compress it into a vector of just 64. So instead of having to predict 12288 variables we just need to predict 64 variables.

3.4.3 Exploration by Random Network Distillation

The paper "Exploration by Random Network Distillation" [11] showed that the prediction error between a neural network with that of a fixed randomly initialized neural network works quite well and is also computationally very efficient. They showed very good results on tasks like Montezuma's revenge which is traditionally proven to be a very difficult task for Reinforcement Learning agents to solve. Their approach is that neural networks tend to give lower prediction errors on examples that they have been previously trained on. So, by using a fixed network as the target, novel experiences would tend to have a larger error and thus a bigger intrinsic reward.

3.4.4 Inverse Dynamics

The main goal of the paper "Curiosity-driven Exploration by Self-supervised Prediction" [12] was to reduce how much the intrinsic reward depends on features of the environment that do not affect the agent's actions. In this paper the authors show another way to formulate an intrinsic reward that does not depend on predicting the next state in pixel space. Their idea is to use a feature space which is only affected by what is necessary for an agent to take an action.

In order to do this, they use an inverse dynamics model to output the agents action given the current and next state. The feature space learned while predicting the action has no influence of factors not necessary to take the action. They then use this feature space to train a model that predicts the feature representation of the next state given the current state and action.

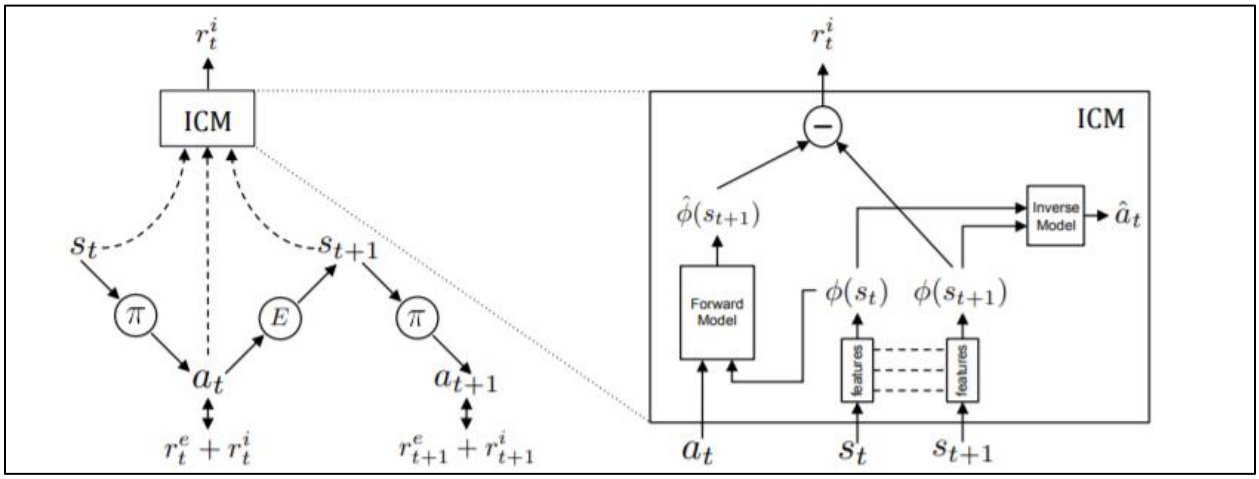

Figure 3: Formulation of Curiosity using an Inverse Dynamics model from [12]

In the diagram above, the Intrinsic Curiosity Module first consists of the inverse dynamics model that learns to predict what action the agent took when in state St that resulted in state St+1. This can be broken up into two steps. First St and St+1 is passed into the same sub module that encodes them into the feature space. They are then both used to predict the action at.

This amounts to training a Neural Network parametrized by weights I as below:

$\hat{a}_t = g(s_t, s_{t+1};\ \theta_I)$

With Loss function:

$min\ L_I(\hat{a}_t, a_t)$

The forward dynamics model can be expressed as:

$\hat{\varphi}(s_{t+1}) = f(\varphi(s_t), a_t;\ \theta_F)$

With Loss function:

$L_F = \frac{1}{2}[\hat{\varphi}(s_{t+1}) - \varphi(s_{t+1})]_2^2$

CHAPTER 4:

IMPROVING NEXT STATE PREDICTION

When using a next state prediction model for intrinsic curiosity, we run into problems whenever the environment is stochastic in nature. For example, if an agent encounters a tree that has leaves blowing in the wind, it is impossible to accurately predict the movement of each leaf. Similarly, if an agent encounters a TV with static noise which is completely random, the agent will not be able to predict the next state. The agent would continuously receive high rewards in these scenarios and would not stop exploring the rest of the environment. The paper "Curiosity-driven Exploration by Self-supervised Prediction" [12] showed that by using an inverse dynamic model it is possible to train a feature set that removes factors that do not affect the policy. As explained in section 4.4.4, they train the inverse dynamics model to predict the action taken by using the current state and next state. They then use the feature space from the inverse dynamics model of the current state and the actual action taken to predict the feature space of the next state. This prediction error is used as the intrinsic reward.

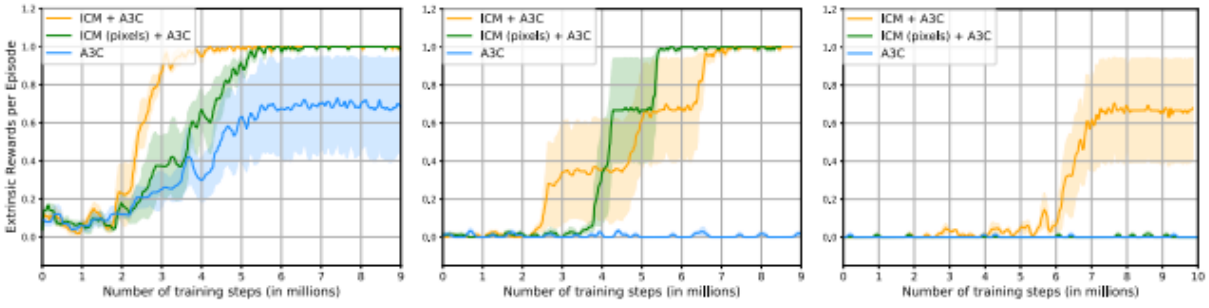

Figure 4: Results from [12] showing how Intrinsic Motivation is used in increasingly difficult tasks.

The results from the paper show that their version of Intrinsic Curiosity works on increasingly difficult tasks. The task shown in the figure shows three versions of the Viz Doom "My Way Home" task where the agent needs to navigate rooms to find an amour chest which gives it an extrinsic reward. Each room/corridor has textures on the walls then tends to distract agents using a next state prediction model as a form of intrinsic reward. In the Dense version of the task, the agent is spawned in random rooms. As it often spawns quite close to the reward it is not considered a sparse task. In the Sparse version of the task the agent is forced to always spawn in room 13 which is approximately 270 steps away from the goal whereas in the Very Sparse Setting, the goal is 350 steps away from the goal. By increasing the distance between the agent and the goal we increase the sparseness of the environment.

However, when the paper compared different intrinsic motivation models, they claimed that using a Variational Auto Encoder for the next step prediction proved to be unstable. They then went on to compare the model to a model using next step prediction error in pixel space. Since the number of variables to predict are very high in pixel space, its performance degrades on the sparser environments.

In the Very Sparse setting, on average, their agent reaches the goal just over 60 percent of the time. In the following section we are going to see if we can improve on the performance of this inverse dynamics model.

Another interesting finding in the paper "Large-Scale Study of Curiosity-Driven Learning" [15] showed that the Inverse Dynamics model did not do very well when solving an environment that had a TV displaying static. The agent was not able to ignore the stochastic nature of the static on the TV and would get drawn to it and get stuck.

4.1 Architecture

We first define a common architecture for all our experiments. We then compare the performance of adding a memory for the VAE to stabilize its training. We then add a recurrent layer in the next step prediction

model. Finally, we integrate the concept of learning progress. The initial architecture uses a frame stack of 4. This is to give the agent context into which direction it is moving. Each frame is sent to a Variational Autoencoder to compress the image into a latent representation of size 32. The stacked latent representations of each frame are concatenated with a one hot encoded representation of the action and is sent to the prediction module. The Same stacked latent representation of the Next state is used to calculate the prediction error. We use the L2 norm. Since we have 128 latent variables to be predicted, we add an Intrinsic coefficient so that the cumulative sum of the intrinsic rewards over an episode do not become larger than the extrinsic reward. It should also be noted that we do not share the latent representation between the Variational Autoencoder and that of the network trying to maximize the score.

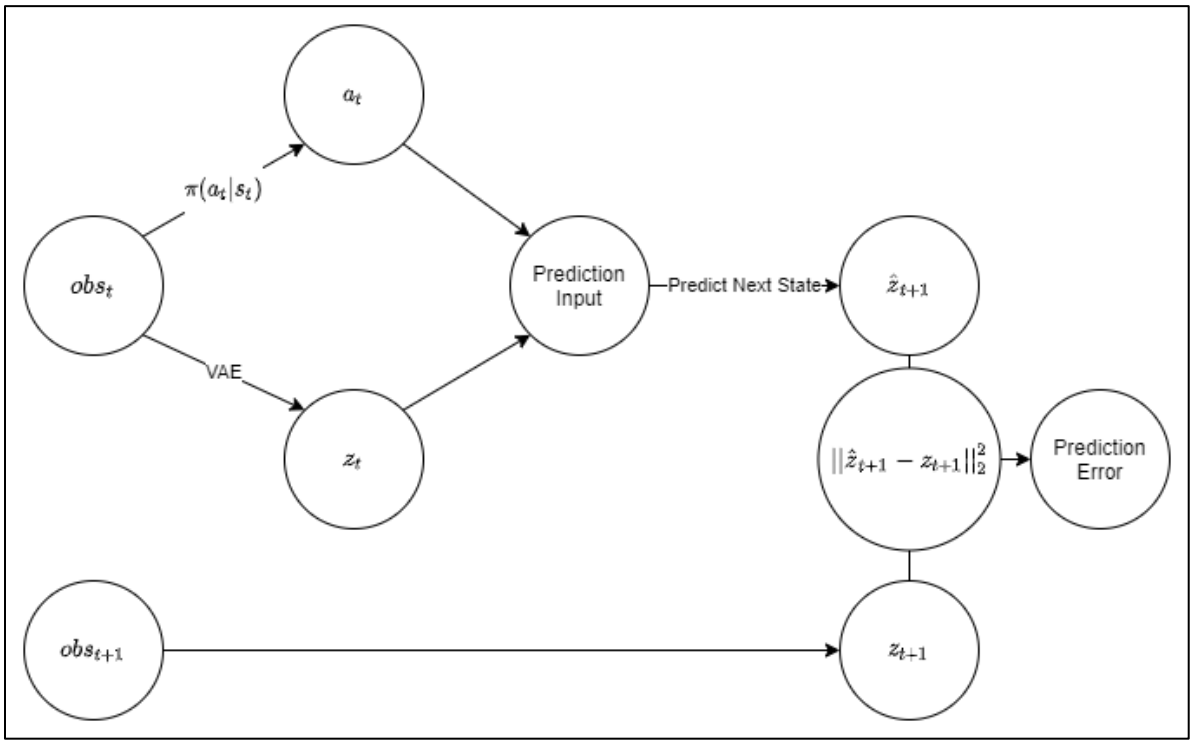

Figure 5: Architecture for Intrinsic Motivation using Next State Prediction.

Each observation is obtained by reducing the size of each image to 64 by 64. An action repeat of 4 is used and the final observation is obtained by stacking the last 4 states.

The VAE encoder consists of 4 Convolutional layers and a single dense layer. The output of each layer is then passed to a Batch Normalization layer. The Convolutional layers all have relu activation functions. The Decoder consists of a Dense layer and then 4 Convolutional transpose layers. Each having relu activation functions and the output of each layer is then sent to a batch normalization layer.

The Next State Prediction module consists of a Dense Layer of size 256 then it passed through a LSTM layer with hidden size 512 and tanh for its activation function. Then it is passed to two more Dense Layers of size 256. Finally, it is passed through another Dense layer with size 128 to represent the 4 stacked VAE encoded frames.

4.2 VAE Memory

The use of a Variational Auto Encoder is generally unstable when being used to predict the next state in Reinforcement Learning. Consider the example of a maze, the agent generally receives high rewards when encountering a new room. As the agent explores the new room, most of the frames that are generated are from the new room as the agent's policy will constantly take the agent to that room. Since the distribution of images is imbalanced it leads to stability problems when training the Variational Auto Encoder. To overcome this problem, it is possible to use a memory mechanism like that of an experienced replay which was introduced in [13] which was used to stabilize the performance of a Q Learning agent. Like the experienced replay, we can use a memory that adds all the frames that are sampled from the environment. We then sample mini batches from the memory to make updates to the Variational Auto Encoder model. It should be noted that the intrinsic reward is generated from the Next State Prediction model which cannot be trained using a memory as that would reduce the need to revisit states. The prediction module should be

improved only when the agent revisits states with high prediction error. The use of the memory greatly improves the stability of training the Variational Auto Encoder.

4.3 Recurrent Layer for Next Step prediction

Using a recurrent layer when designing the Next State Prediction module has often been overlooked. The World Model [1] used a Variational Auto Encoder to compress the current state into a latent representation and then used a Recurrent Network with a Mixture Density Network to learn how to predict the next state. They managed to attain state of the art results in the Gym Car Racing [3] environment and even showed that they could train an agent inside a model's dreams. The use of a recurrent layer is often overlooked when designing a next state prediction model to be used in an intrinsic reward formulation. Especially in Partially observable environments, the agent needs to be able to remember its surroundings to predict the next state. For example, if the agent goes close to a wall that has a repeated texture on it and then tries to go right, it is impossible to say whether it will still see the same pattern or reach the end of the texture.

4.4 Learning Progress

One of the problems with using an intrinsic reward, is that an agent could be drawn to areas in the environment that are stochastic in nature. In the example of leaves moving in a breeze or the static on a TV, the next step prediction module would always have a high error as it is not possible to predict these events accurately. The agent gets drawn to these sources of high prediction error and does not explore the rest of the environment. To overcome this issue, we need to integrate learning progress to the prediction module. The paper "Intrinsic Motivation Systems for Autonomous Mental Development" [16] formulated that the learning progress is the reduction in prediction error on consecutive visits to the same state. So, if the error reduces that means that the agent can learn something from revisiting the state, however if the error increases or stays constant it means that the prediction module is unable to gain any information from visiting the state. However, the method shown to incorporate learning progress stores all trajectories in

memory. They use a method to approximate similar states and then calculates whether the error reduces when revising a grouped state. This method is not scalable to more complex environments. Another method that is mentioned in [17] is to use the decrease in error between the output of the prediction module before and after an update is made. For this to work, the learning rate of the prediction module needs to be small. This form of learning progress can be easily implemented when using a neural network to make the prediction of the next state.

CHAPTER 5:

EXPERIMENTS

5.1 Environment

The Environment, Doom My Way Home was created by [21]. It is based a setting in the game doom where the agent needs to navigate through multiple rooms to reach a piece of armor. On reaching the armor the agent in rewarded with a score of 1.0. There is a time limit of 2100 steps. This environment tends to distract agents that try to predict the next state in pixel space due to the numerous textures on the walls of the rooms. Since it is difficult to predict exactly how the textures move when the agent's view changes. In the original task there are 17 rooms attached by corridors, and the agent is spawned randomly in any of the rooms. Since the agent sometimes spawns close to the reward, it can learn quite quickly, and the task is considered a dense task. However, [12] showed that by forcing the agent to always spawn far from the reward, the task is converted into a sparse task. There are two sparse tasks, one sparse where the agent is spawned 270 steps away from the reward and a very sparse task where the agent is spawned approximately 350 steps away from the reward.

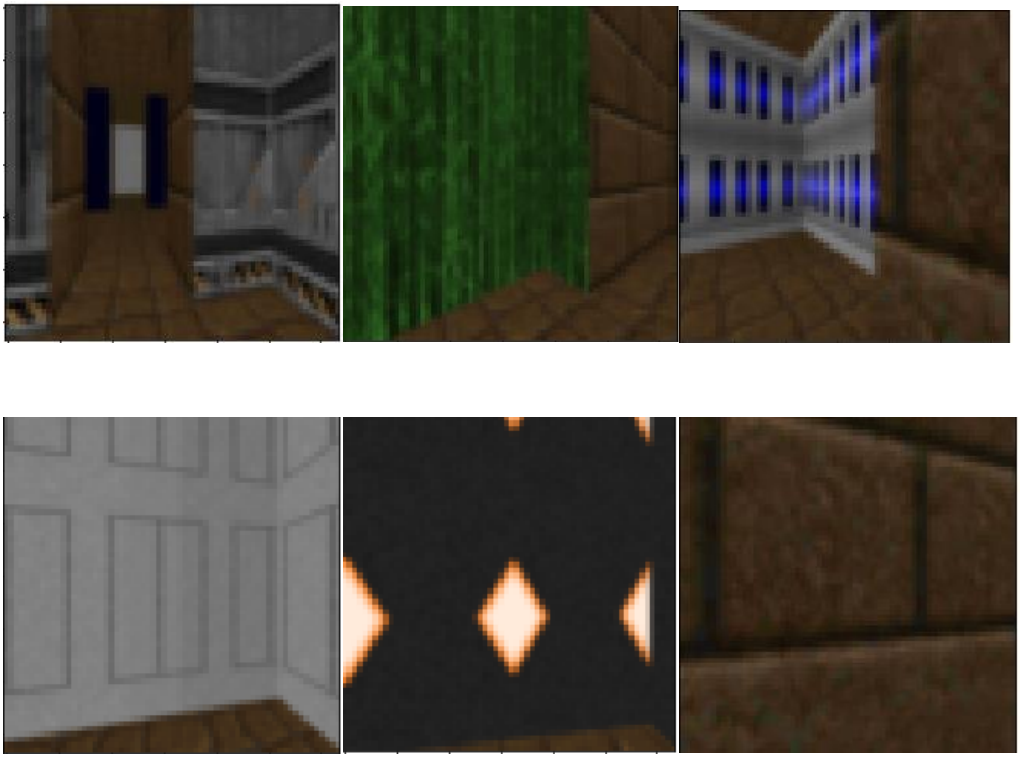

Figure 6: Images from the Doom My Way Home Environment depicting various textures on the walls

We first look at how our architecture compares against a Vanilla PPO agent on Dense, Sparse and Very Sparse Doom My Way Home Tasks.

5.2 Intrinsic vs Vanilla PPO

5.2.1 Dense Task

In the Dense task, the agent encounters the reward much more often, so we can increase the entropy parameter to 0.01 instead of using 0.001 like the other experiments.

Table 1: Hyperparameters for Dense Task

PPO Hyperparameters	GAMMA = 0.99 N_STEPS = 128 ENTROPY_REG = 0.01 LEARNING_RATE = 1e-4 VALUE_COEFFICIENT = 0.5 MAX_GRAD_NORM = 0.5 LAMBDA = .95 MINI_BATCHES = 4 NO_PT_EPOCHS = 3 EPSILON = .2 EPOCHS = 100000000 N_ENVS = 20
VAE Hyperparameters	VAE_LEARNING_RATE = 0.0001 VAE_Z_SIZE = 32 VAE_KL_TOLERANCE = 0.5
Next State Prediction	PREDICTION_LEARNING_RATE = 0.001 INTRINSIC_COEFFICIENT = 0.000005 EXTRINSIC_COEFFICIENT = 10.0

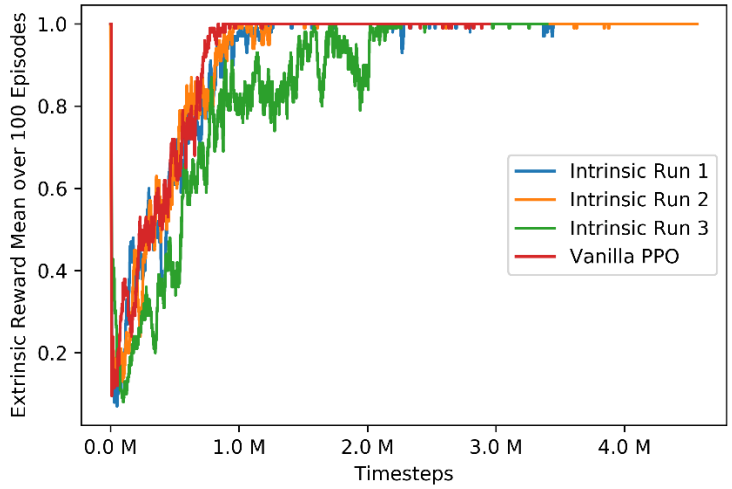

Figure 7: A comparison of our Intrinsic Formulation vs a Vanilla PPO algorithm on the Dense Doom My Way Home Task

5.2.2 Sparse Task

Table 2: Hyperparameters for Sparse Task

PPO Hyperparameters	GAMMA = 0.99 N_STEPS = 128 ENTROPY_REG = 0.01 LEARNING_RATE = 1e-4 VALUE_COEFFICIENT = 0.5 MAX_GRAD_NORM = 0.5 LAMBDA = .95 MINI_BATCHES = 4 NO_PT_EPOCHS = 3 EPSILON = .2 EPOCHS = 100000000 N_ENVS = 20

VAE Hyperparameters	VAE_LEARNING_RATE = 0.0001 VAE_Z_SIZE = 32 VAE_KL_TOLERANCE = 0.5
Next State Prediction	PREDICTION_LEARNING_RATE = 0.001 INTRINSIC_COEFFICIENT = 0.000005 EXTRINSIC_COEFFICIENT = 10.0

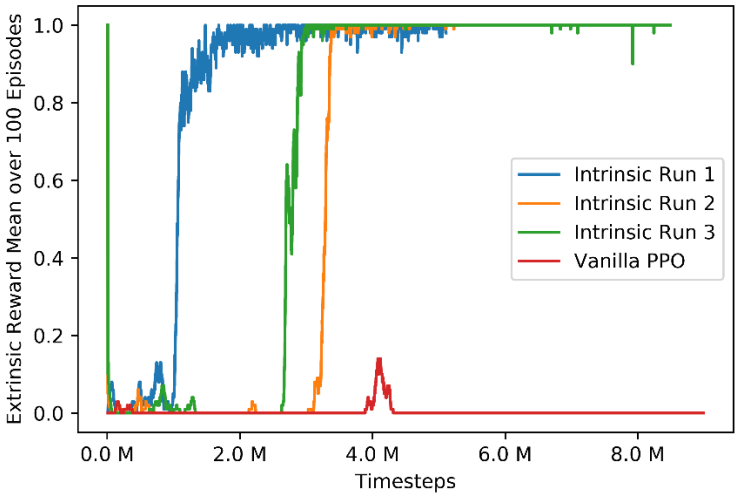

Figure 8: A comparison of our Intrinsic Formulation vs a Vanilla PPO algorithm on the Sparse Doom My Way Home Task

5.2.3 Very Sparse Task

Table 3: Hyperparameters for Very Sparse Task

PPO Hyperparameters	GAMMA = 0.99 N_STEPS = 128 ENTROPY_REG = 0.01 LEARNING_RATE = 1e-4 VALUE_COEFFICIENT = 0.5

	MAX_GRAD_NORM = 0.5 LAMBDA = .95 MINI_BATCHES = 4 NO_PT_EPOCHS = 3 EPSILON = .2 EPOCHS = 100000000 N_ENVS = 20
VAE Hyperparameters	VAE_LEARNING_RATE = 0.0001 VAE_Z_SIZE = 32 VAE_KL_TOLERANCE = 0.5
Next State Prediction	PREDICTION_LEARNING_RATE = 0.001 INTRINSIC_COEFFICIENT = 0.000005 EXTRINSIC_COEFFICIENT = 10.0

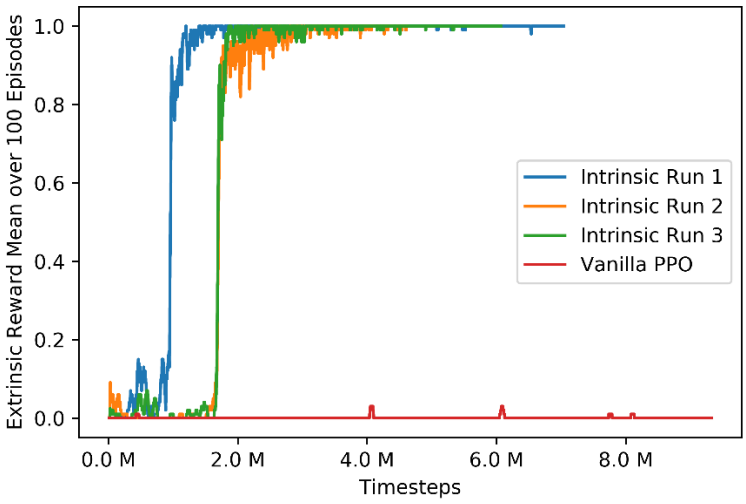

Figure 9: A comparison of our Intrinsic Formulation vs a Vanilla PPO algorithm on the Very Sparse Doom My Way Home Task

We can see that in the Dense Task, there is very little difference between a Vanilla PPO agent and an Intrinsic agent. This is because the rewards are frequent, and the agent can learn quite quickly. In the Sparse and Very Sparse task we see a significant difference. The Very Sparse task tends to converge faster than the Sparse task. This can be attributed to the fact that in the Sparse task the agent is spawned in a room whereas in the Very Sparse task the agent is spawned in a corridor that does not have any textures on the walls.

5.3 Effect of VAE Memory and RNN

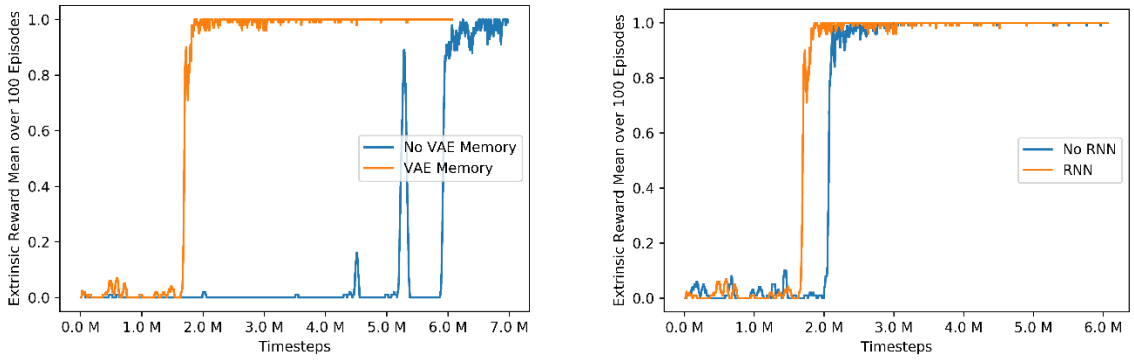

Figure 10: A comparison of the effects of A VAE Memory and a Recurrent Layer on the Very Sparse Doom My Way Home Task

We can see that removing the VAE memory causes the agent to take much longer to consistently reach the target. This can be attributed to the VAE being unstable when overfitting to a batch. Removing the Recurrent layer also causes a slower convergence although it is not as much of a difference as the VAE memory.

5.4 Effect of Learning Progress

Table 4: Hyperparameters for Intrinsic reward with Learning Progress

PPO Hyperparameters	GAMMA = 0.99 N_STEPS = 128 ENTROPY_REG = 0.001 LEARNING_RATE = 1e-4 VALUE_COEFFICIENT = 0.5 MAX_GRAD_NORM = 0.5 LAMBDA = .95 MINI_BATCHES = 4 NO_PT_EPOCHS = 3 EPSILON = .2 EPOCHS = 100000000 N_ENVS = 20
VAE Hyperparameters	VAE_LEARNING_RATE = 0.0001 VAE_Z_SIZE = 32 VAE_KL_TOLERANCE = 0.5
Next State Prediction	PREDICTION_LEARNING_RATE = 0.0001 INTRINSIC_COEFFICIENT = 0.0005 EXTRINSIC_COEFFICIENT = 10.0

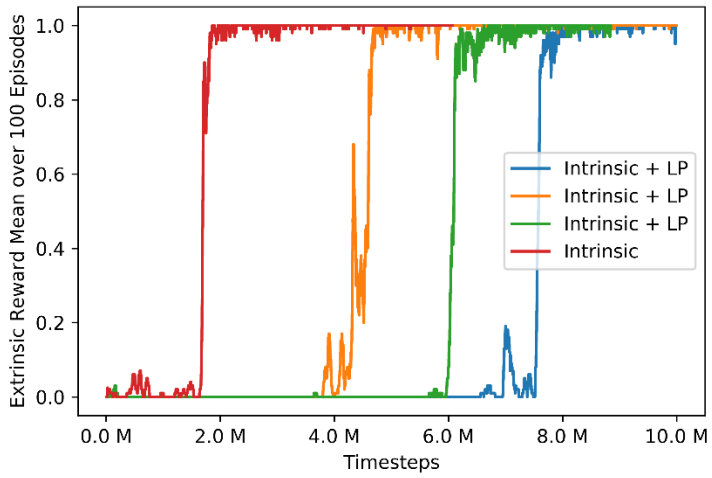

Figure 11: Integrating Learning Progress with the Very Sparse tasks

This shows that although this implementation of Learning Progress works, it takes longer to converge due to the need to reduce the learning rate of the prediction model. If we keep the Learning rate of the prediction model too high, the model tends to overfit to initial trajectories that lead to big reductions in error. The model finds it difficult to get out of this as the entropy of the policy drops too low. We can also infer that when we use just the prediction error, exploring a new room would have a much higher reward compared to when integrating learning progress. When using Leaning Progress, we will spend longer exploring a room as we will explore it until the reduction in error is lower than the reduction in error received from exploring a new room.

CHAPTER 6:

CONCLUSION

From the results, we first see how stabilizing the training of the Variational Auto Encoder can speed up the convergence of a Reinforcement Learning agent. We then see how adding a Recurrent Layer in the Next State Prediction module can give the agent context to its surroundings and helps improve prediction and thereby results in faster convergence. Finally, we look at how we can incorporate the concept of Learning Progress when training the Next State Prediction module. Although this slowed down convergence, it is a necessary element for a curious Reinforcement Learning agent to prevent the agent from constantly getting drawn to stochastic areas of the environment where the prediction module cannot be improved.

CHAPTER 7:

FUTURE WORK

There are a couple of ways that this work can be extended. The first is to use a similar architecture like the World Model [1] that uses a Mixture Density Network [6] to predict the next state. This can help improve models in more complex environments that have more possible outcomes of the next state. For example, learning to predict the movement of a character in the environment. We can then test the model on an environment that contains sources of constant unpredictability like adding a TV with white noise being displayed on it. Finally, we could integrate the option architecture [9] that allows temporal abstraction by using an intrinsic reward to learn sub tasks in an environment.

CHAPTER 8:

REFERENCES

[1] David Ha and Jurgen Schmidhuber, World Models, March 2018. URL https://arxiv.org/abs/1803.10122.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization, July 2017. URL https://arxiv.org/pdf/1707.06347.pdf

[3] Car Racing, https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py

[4] VizDoom, https://github.com/mwydmuch/ViZDoom

[5] Diederik P Kingma, Max Welling, Auto-Encoding Variational Bayes, Dec 2013, https://arxiv.org/abs/1312.6114

[6] Christopher M. Bishop, Mixture Density Networks, Feb 1994, https://publications.aston.ac.uk/id/eprint/373/1/NCRG_94_004.pdf

[7] Nikolaus Hansen, The CMA Evolution Strategy: A Tutorial, April 2016, https://arxiv.org/abs/1604.00772

[8] Vijay R. Konda John N. Tsitsiklis, Actor-Critic Algorithms, https://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf

[9] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998, https://mitpress.mit.edu/books/reinforcement-learning

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, Open AI Gym, 2016, https://arxiv.org/abs/1606.01540

[11] Yuri Burda, Harrison Edwards, Amos Storkey, Oleg Klimov, Exploration by Random Network Distillation, 2018, https://arxiv.org/abs/1810.12894

[12] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, Trevor Darrell, Curiosity-driven Exploration by Self-supervised Prediction, 2017, https://pathak22.github.io/noreward-rl/

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari with Deep Reinforcement Learning, 2013, https://arxiv.org/abs/1312.5602

[14] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel, Trust Region Policy Optimization, 2015, https://arxiv.org/abs/1502.05477

[15] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, Alexei A. Efros, Large-Scale Study of Curiosity-Driven Learning, 2018, https://arxiv.org/pdf/1808.04355.pdf

[16] Pierre-Yves Oudeyer, Frédéric Kaplan, and Verena V. Hafner, Intrinsic Motivation Systems for Autonomous Mental Development, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4141061&tag=1

[17] Schmidhuber J, Curious model-building control systems. In Proceedings of the International Joint Conference on Neural Networks, https://mediatum.ub.tum.de/doc/814953/file.pdf

[18] P.Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. Frontiers in neurorobotics, 2009, https://www.frontiersin.org/articles/10.3389/neuro.12.006.2007/full

[19] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration, 2016, https://arxiv.org/abs/1605.09674

[20] Shakir Mohamed and Danilo J. Rezende, Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning, 2015, https://arxiv.org/abs/1509.08731

[21] Gym Doom My Way Home https://github.com/ppaquette/gym-doom