Multi-Step Prediction for Curiosity Driven Exploration

Kushantha U. Attanayake
kushan@umich.edu
University of Michigan

Ruchir Aggarwal
aggarwr@umich.edu
University of Michigan

Dennis Li
dennli@umich.edu
University of Michigan

Julio Soldevilla
jsolde@umich.edu
University of Michigan

Poorani Ravindhiran
poorani@umich.edu
University of Michigan

Abstract

Reinforcement learning has made great strides in exploring virtual environments, especially those of video games because of their well structured reward systems. However, it struggles in sparse reward scenarios. Curiosity driven exploration [3], where the agent is intrinsically motivated to explore the environment, has been proposed as a solution to this issue. However, current implementations of curiosity based learning only predict one time step into the future. As a result, if a catastrophic event were to happen two time steps away, the agent would be none the wiser. We propose an improvement to the proposed algorithm my generating predictions multiple steps into the future, and by using all predictions in our definition of curiosity. Project website at **github/aggarwalRuchir**

1 Problem Statement

Several reinforcement algorithms aim to learn policies used to achieve tasks by maximizing a reward provided by the environment. In real world scenarios, such extrinsic rewards (provided by the environment) are very sparse. It is, in this situation, that intrinsic rewards become useful to learn policies used to achieve tasks. In a series of papers [1], [3], Deepak et. al. proposed a method to generate an intrinsic reward based on the agent's curiosity (defined as the error in an agent's ability to predict consequences of its own actions). The results are very promising, but predicting only one time step into the future may be too short-sighted. Indeed, we can easily think of many scenarios in both the real world and the game world in which seeing something truly curious may necessitate boredom in the immediate future. Currently, the proposed architecture utilizes an embedding of the state (s_t) at time t into a feature space and the action (a_t) taken at time t by the agent to generate a prediction of the embedding of the state at time $t+1$ ($\hat{\phi}(s_{t+1})$) into feature space. This predicted value is used to generate the intrinsic reward of the agent, its curiosity. Some of the questions that arise from this approach are:

1. What would happen if the agent also tried to predict states $s_{t+2}, s_{t+3}, ..., s_{t+n}$ and compared these predictions to reality?
2. What would happen if we used the embedded state $\phi(s_t)$ to make the prediction of $\hat{\phi}(s_{t+1})$?

In this paper, we attempt to answer these questions by presenting modifications to the proposed architecture in [3] and we believe that this will allow the agent to explore the environment faster and make further approaches during the game, improving its performance.

23nd Conference on Neural Information Processing Systems (NeurIPS 2019), Ann Arbor, Michigan.

2 Previous Work

In formal Reinforcement Learning algorithms, an agent tries to maximize its rewards in response to its action in the environment. These rewards are usually external in nature. But this approach requires a careful designing of the reward function. Moreover, this reward function is very specific to the environment the agent interacts with and cannot be generalized to other environments. Thus, one of the current research areas in reinforcement learning is *intrinsic motivation*. In intrinsic motivated reinforcement learning, the agent internally generates a small reward for every good action it takes, according to its policy, and penalizes itself for every bad action.

Through this project, we are particularly interested in exploring the use of "curiosity" as an intrinsic reward. Babies are curious about what happens if they move their fingers in certain way, creating little experiments that lead to initially novel and surprising but eventually predictable sensory inputs. Our "curious" agent can be thought of in a similar way where it tries to generates such little experiments (make predictions) to keep itself surprised(or curious). An agent is encouraged to explore novel states and to make actions that will eventually reduce the error in the agent's prediction of the consequence of certain actions. Many versions of reward function based on curiosity has been proposed over time. Schmidhuber did extensive work on confidence-based curiosity and the ideas of exploration and shaping rewards. In his paper [4], he defined the curiosity proportional to the difference between how many computational resources(storage space and time) the agent needs to encode the data sequence before and after learning. Still and Precup in their paper [5] used information-theory to define the reward function as the gain in the information about the environment by the agent. Satinder et al. [2] derived their inspiration for reward function from the novelty response of dopamine neurons.

In [3], Pathak et. al. defined *curiosity* as 'the error in an agent's ability to predict the consequence of its own actions in a visual feature space learned by a self-supervised inverse dynamics model'.

3 Preliminaries: Theory Behind Curiosity Driven Exploration

In [3], Pathak et. al. devise an agent composed by a reward generator which outputs a curiosity-driven intrinsic reward and a policy that outputs a sequence of actions that maximize the reward signal. The policy is represented by a deep neural network $\pi(s_t; \theta_P)$ with parameters θ_P, where the θ_P is optimized to maximize the expected sum of rewards

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] \tag{1}$$

The main contribution of the paper is the design of an architecture to compute this intrinsic reward signal based on prediction error of the agent's knowledge about its environment.

The key insight that allows for the development of a good intrinsic reward is modeling the states in a feature space that distinguishes things that can be controlled by the agent, things that the agent does not control but still affect the agent and things that the agent does not control and that do not affect the agent. To learn such a feature space, the researchers train a deep neural network with two components: the first one will encode the raw state s_t into a feature vector $\phi(s_t)$ and the second one will take as inputs the feature vectors $\phi(s_t), \phi(s_{t+1})$ and predict the action taken by the agent to go from state s_t to state s_{t+1}. This process corresponds to learning a function g, called the *inverse dynamics model*, and defined by

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I) \tag{2}$$

, where $\hat{a}_t$ is the prediction of the action a_t and the parameters θ_I are optimized to solve

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

where L_I is some function measuring the difference between the predicted and actual actions.

Additionally, the authors train another network that will take as inputs a_t and $\phi(s_t)$, and will learn the function f that will be used as the feature encoding of the state at the next time-step

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

where $\hat{\phi}(s_{t+1})$ is the prediction and θ_F are parameters trained to optimize

$$L_F(\phi(s_t), \hat{\phi}(s_{t+1})) = \frac{1}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$$

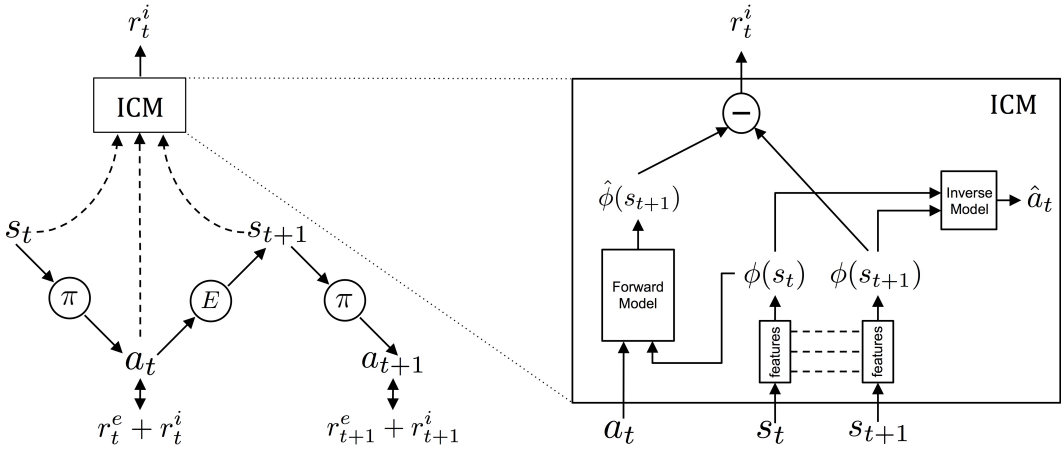


Figure 1: Original Architecture

The authors call this learned function, f, the *forward dynamics model*. With these two models defined as above, the proposed intrinsic reward function is then defined as

$$r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \tag{3}$$

where η is some scaling factor. Finally, since we are interested in finding the optimal reward and the optimal parameters for the network g and f, the overall optimization problem is solved so that the agent learns the following

$$\min_{\theta_P,\theta_I,\theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] + (1-\beta)L_I + \beta L_F\right] \tag{4}$$

In [1],the main contribution of the authors is the study of different ways of obtaining the feature space of the states of the game and understand how the agent learns in these different feature spaces. Fundamentally, they establish the baselines that any good feature space should have. As per that, any good feature space should be compact, sufficient and stable. Some of the feature space transformations discussed include mere pixel observation, random features initialization (obtained by doing a random initialization for the embedding network and fixing this initialization; no training of the network), variational auto-encoders (VAE) and inverse dynamics features (IDF- obtained from the inverse dynamics model described above).

3.1 Network details

The intrinsic curiosity module consists of the forward and the inverse model. The inverse model uses a series of four convolution layers, each with 32 filters, kernel size 3x3, stride of 2 and padding of 1. ReLU non-linearity is used after each convolution layer. The dimensionality of the output of the fourth convolution layer is 288. Two such outputs for consequent time are concatenated into a single feature vector and passed as inputs into a fully connected layer of 256 units followed by an output fully connected layer with 4 units to predict one of the four possible actions. The forward model is constructed by a sequence of two fully connected layers with 256 and 288 units respectively.

4 Method: Multi-Step Prediction

We propose two major changes to the implementation provided in [3]. First, we modify the forward dynamics model to generate $\hat{\phi}(s_{t+2}), \hat{\phi}(s_{t+3}), ..., \hat{\phi}(s_{t+n})$ in addition to generating $\hat{\phi}(s_{t+1})$, and we redefine the reward signal to use both of these predictions such that

$$r_t^i = \frac{\eta}{2}\sum_{i=1}^{n}(w_p^i||\hat{\phi}(s_{t+i}) - \phi(s_{t+i})||)$$

where w_p and η are scaling factors, $\hat{\phi}(\cdot)$ are the outputs generated by the forward dynamics model, and $\phi(\cdot)$ are the output generated by the feature extractor. In addition, for now, we choose to use 'Random Feature Initialization' as our feature learning method in the feature extractor. We take our embedding network (CNN) and fix it after random initialization. Since the network is fixed, the features are stable.

To accommodate this change, we also have to change the way the policy was defined. In the original implementation, the forward dynamics model needed a_t and $\phi(s_t)$ to generate $\hat{\phi}(s_{t+1})$, thus if it were to generate $\hat{\phi}(s_{t+n})$, it would need $a_{t+(n-1)}$ and $\phi(s_{t+(n-1)})$. We can circumvent the requirement for $\phi(s_{t+(n-1)})$ by using $\hat{\phi}(s_{t+(n-1)})$, which is generated by the forward dynamics model, but the only way to acquire $a_{t+(n-1)}$ is through the policy network, which requires $s_{t+(n-1)}$ as an input.

Thus, the second change we propose is to change the policy network such that the input to it would be the features of s_t, i.e. $\phi(s_t)$. This would allow us to use $\hat{\phi}(s_{t+(n-1)})$ as an alternative in the forward dynamics model, allowing us to acquire an action for a predicted state, thus allowing us to generate $\hat{\phi}(s_{t+n})$.

Our hypothesis is that the performance of the agent will improve because of the increased horizon. With the new definition of the reward, the agent is disincentivized from visiting states that are adjacent to already well known states, in addition to being disincentivized from visiting already visited states. As a result, it would even further avoid dangerous regions that could lead to it's death, and choose to explore the state space even further,

One of the major concerns was the proposed change to the policy. Because our implementation necessitates acquiring the next action using the features of the current state as the input, instead of deriving it directly from the observed state, there was concern that the agent performance would decrease, or at the very least depend heavily on the method used to extract features form the observed state. This concern has been alleviated, at least partially, which is explained in greater detail in the current progress section.

4.1 Forward Dynamics Model Change

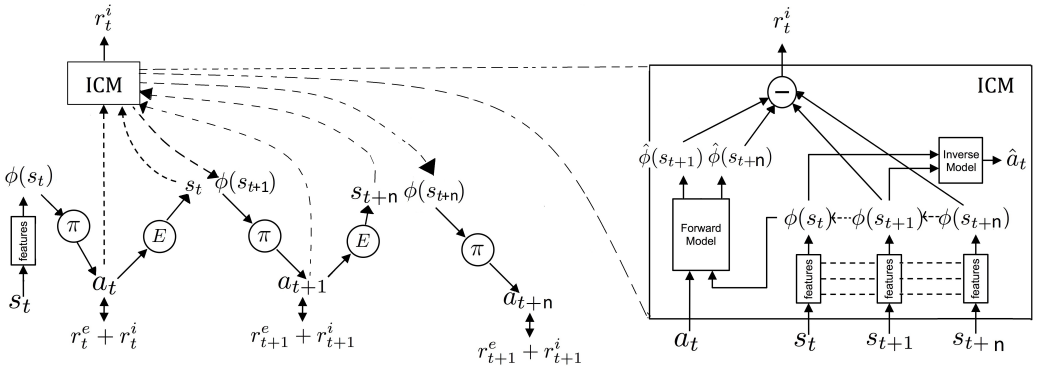


Figure 2: Updated Architecture - The agent in state s_t with features $\phi(t)$ interacts with the environment by executing an action a_t sampled from its current policy π and ends up in the state s_{t+1} with features $\phi(t+1)$. These features are then used to predict the action for the next timestep and this continues for n steps. The policy π is trained to optimize the sum of the extrinsic reward (r_t^e) provided by the environment E and the curiosity based intrinsic reward signal (r_t^i) generated by the Intrinsic Curiosity Module (ICM). ICM encodes the states s_t, s_{t+1}... s_{t+n} into the features $\phi(s_t)$, $\phi(s_{t+1})$... $\phi(s_{t+n})$. The forward model takes as inputs the features ($\phi(s_t)$, $\phi(s_t)$ ... $\phi(s_t + (n-1))$) and a_t and predicts the feature representation $\hat{\phi}(s_{t+1})$ of s_{t+1} and for all n time steps. The prediction error in the feature space is used as the curiosity based intrinsic reward signal.

Having changed the policy to take input from the feature/latent space rather than state space, it is now possible to generate action for the n-th time step, a_{t+n}, without having to wait for the n-th -1 state. This, in turn, allows us to make a prediction for the feature embedding of the second state using the forward dynamics model at the first time step itself. Similarly, we can make the predictions for

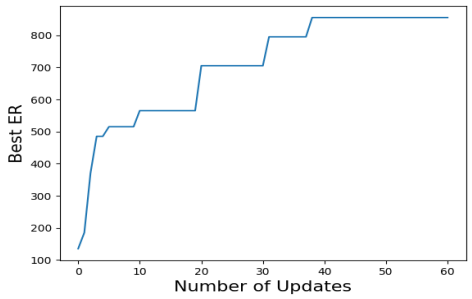
features of all states from the current state until state $t+n$. Using this fact, we made the modifications to the proposed architecture such that the intrinsic reward (curiosity) is computed as a weighted average of the state prediction errors for all states from the current state to t+n. Mathematically, it is given by equation 5.

$$r_t^i = \frac{\eta}{2} \sum_{i=1}^{n} (w_p^i || \hat{\phi}(s_{t+i}) - \phi(s_{t+i}) ||) \tag{5}$$

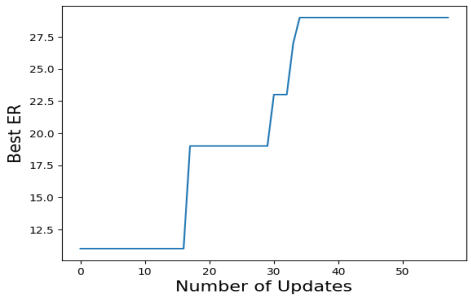
where w_p and η are scaling factors, $\hat{\phi}(\cdot)$ are the outputs generated by the forward dynamics model, and $\phi(\cdot)$ are the outputs generated by the feature extractor.

5 Experiments and Discussion

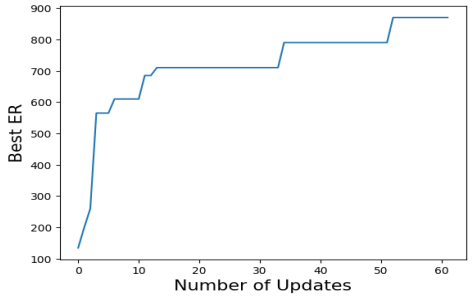
5.1 Policy Architecture Change



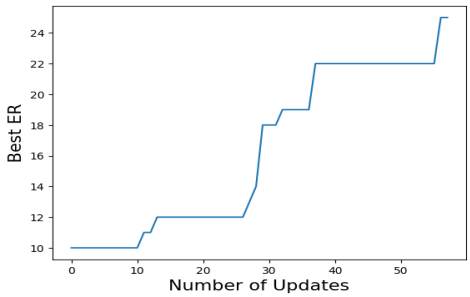
Space Invaders best external reward



Breakout best external reward



Space Invaders best external reward



Breakout best external reward

Figure 3: Preliminary results from the initial run of the algorithm. The top row shows the results for the original architecture and the second row shows the results of the updated policy architecture.

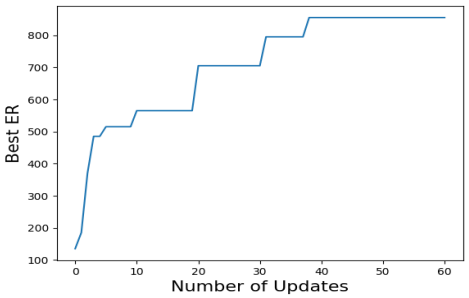
Changing the policy network to operate on the feature space of the state instead of the raw observations seemed to have a slight impact in agent performance, but the differences between the two architectures are not large enough to raise concerns.

We ran the simulation for only 63 updates which is far less than actually required (more than 1000) because of the time it takes to train the agent. From Figure 2. we can say the following:

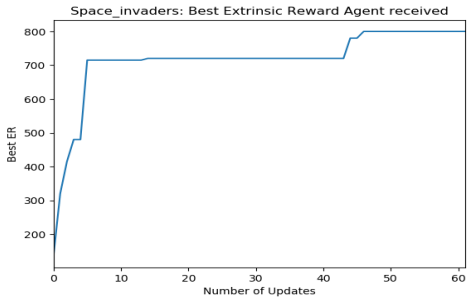
- The average agent advantage follows a similar curve as with the original architecture. We reason that this maybe because of the fewer number of updates and we may see deviations as the number of updates increase.
- The best external return of the agent are slightly different. In the case of Space Invaders (column 1), the best external return is higher for our updated architecture around update 40 [1]. Similarly, for Breakout (column 2), the external return is higher in the updated architecture than in the original architecture.

[1] simulated with default parameters and not with optimum parameters for the environment

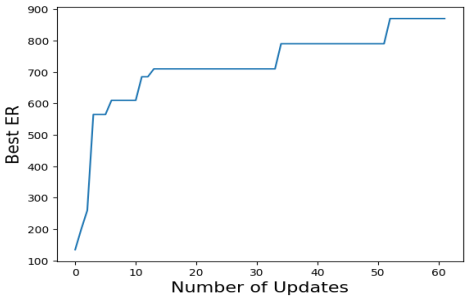
After validating the change in policy network input, we ran experiments on a variety of environments using our updated reward function which was a linear combination between suprisal in timestep t+1 and timestep t+2. We ran experiments with various settings for w_p^1 and w_p^2, the coefficients of the surprisal for t+1 and t+2 in calculating the reward respectively. We compare our results to two baselines: (1) the performance using the architecture originally described in [1] and (2) the performance of our updated architecture using $w_p^1 = 1$ and $w_p^2 = 2$, both being run on our own machines. Besides w_p^1 and w_p^2, we used the same hyperparameters as in [1]. We used random features as our feature embedding network in all experiments. Performance was measured and compared using extrinsic reward or in-game progress. We ran experiments on a Nvidia Tesla P4 and generally did not run experiments for longer than 24 hours.We found that $w_p^1 = 1$ and $w_p^2 = .2$ gave the best results in a variety of environments. However, we found that other weight combinations gave reasonable results. For example, in Space Invaders, we interestingly see that using $w_p^1 = .2, w_p^2 = 1$ gave results comparable to $w_p^1 = 1, w_p^2 = .2$ and the baseline, as shown in figure 4. We found that in simple environments, w_p^2 could be high and not significantly diminish our results, while in complex environments the agent would learn extremely slowly if at all. We hypothesize that this is because complex environments are more difficult to predict, so the agent will have a hard time getting started in making any progress.



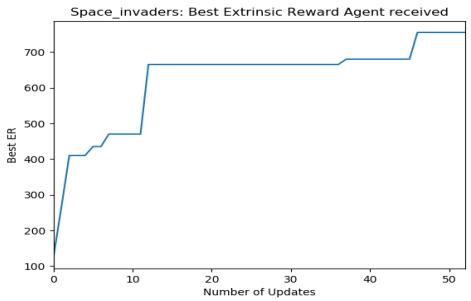
Results from original code in [3] (baseline)



Space invaders with weights w_p^1 = .2, w_p^2 = 1



Single-step prediction only (baseline): w_p^1 = 1, w_p^2 = 0



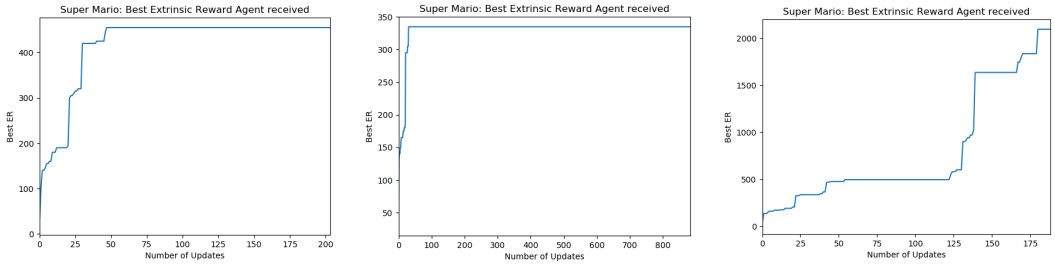
Space invaders with weights w_p^1 = 1, w_p^2 = .2

Figure 4: Comparison of external reward achieved in Space Invaders using the baselines (left two) and our updated double-step model using various settings of w_p).

We found that in almost all environments, incorporating surprisal from timestep t+2 led to the agent learning faster than in the baseline. Our model achieved similar performance (extrinsic reward or in-game progress) as the baselines.

Results for Mario are shown in figure 5. $w_p^1 = 1, w_p^2 = .2$ were the best hyperparameters for Mario. Agents were generally able to reach level 3. No models (including both baselines) were able to pass level 3 in the maximally allotted 24 hours.

In PacMan, after 2200 updates, our modified architecture achieves 440 points using w_p^2 = .2, while the baseline of w_p^2 = 0 achieves 500 points. Gameplay videos of our modified agents and baselines are available here.



$w_p = 0.2$ $w_p = 0.5$ $w_p = 0$

Figure 5: Comparison of external reward achieved in Super Mario Bros using the baseline (left) and our updated double-step model using various settings of w_p^1 and w_p^2)

6 Future Work

Our experiments focused only on making two predictions from any given state, i.e. our loss was only

$$\frac{\eta}{2}(||\hat{\phi}(s_{t+1}) - \phi(s_{t+1})|| + ||\hat{\phi}(s_{t+2}) - \phi(s_{t+2})||)$$

due to time constraints, and our results indicate that the performance of the agent does improve by predicting the features of an additional step into the future. An interesting avenue of future work would be to see if the performance of the agent improves even further by making even more predictions at each state, and at which point making additional predictions stop having an impact on agent improvement.

Another key finding in our results is that the additional predictions seem to have a bigger impact on complex environments than in simple environments. It would be interesting to see if this trend continues to even more complicated environments that Super Mario Bros., such as 3 Dimensional environments, or environments with a large and complex action space, such as MuJoCo.

An alternate approach to using curiosity in exploration would to have multiple sub-agents each generating predictions to a set but unique depth, at each time step and then use an aggregate of their predictions to decide on the action to take on the state. For example, at every time step, we could have one agent make predictions for the next two states, another make predictions for the next three states, etc. and then use a weighted sum of the errors for all the agents to update the policy. This is only one possible avenue of using multiple predictions, there are a myriad of other ways to leverage the ability to make predictions for multiple future states to improve agent performance, and it would be a very interesting avenue of future research. In general, we find the concept of searching for an uncertain future to be exciting in both a technical and human sense, so we are excited to see what explorations of this concept may uncover.

In the original implementation the authors tried several different methods of extracting features from the observations, but due to time constraints we only utilized randomly initialized features for our experiments. The authors of the original paper concluded that the random features are sufficient for many popular RL game benchmarks, but learned features appear to generalize better for more complex environments, however they do take longer to train. Additional work should be done to explore if this observation continues when combined with multi-step prediction.

7 Division of Work

All five members of the team met up at the beginning of the project to outline goals, and with the help of Professor Lee and GSI Ruben Villegas, came up with the architecture for the new Forward Dynamics Model.

Ruchir and Dennis set up environments on Google Cloud Platform for training and testing experiments. Ruchir, Dennis, Julio and Kushantha ran several agents on various environments to get baseline readings to compare our changes against.

Kushantha implemented the proposed changes to the policy network architecture, and ran experiments to compare agent performance against original architecture.

Ruchir wrote script to visualize data from training runs as graphs.

All members of the team contributed to the midterm report.

Dennis identified key parts of the original code that had to be changed to implement our improvement to the Forward Dynamics Model. Ruchir and Kushantha implemented the change to the Forward Dynamics model to generate two step predictions, updated the loss and reward function for the agent.

Dennis, Kushantha, Julio and Poorani worked on the poster for the presentation. Ruchir and Kushantha presented the project at the poster session.

Julio integrated the Mario environment with the agent code base and added the ability to save training runs and to generate videos. Ruchir added the ability to get statistics for Mario training runs.

Kushantha extended Forward Dynamics model to generate multi-step predictions in addition to simply one-step or two-step predictions.

Dennis, Ruchir, Kushantha and Julio ran experiments to generate results and videos for the final report.

Ruchir created and published the website for the project and combined the final videos.

Dennis, Ruchir, Kushantha and Julio contributed to the final report.

8 Conclusion

Our results show that multi-step prediction improves the learning rate of the agent in both simple and complex environments. It achieves this result using only randomly initialized features to obtain a representation of the environment, a method of feature extraction the original authors deemed sufficient for most Reinforcement Learning benchmark environments, but not necessarily optimal in complicated environments.

Furthermore, due to the inherently unpredictable nature of generating a future prediction using a previous prediction, placing too much trust in the second prediction leads to worse performance simply using only the first prediction. This shows that while there is additional information to be gained from additional predictions, they also tend to be quite noisy, and as such latter predictions should be weighted lower than earlier predictions.

Our team did not have sufficient resources or time to fully explore the capabilities of an agent augmented with the ability to predict multiple steps into the future as opposed to simply predicting one step, but our results show very promising trends that would indicate that ultimate agent performance would be better with our augmentation as opposed to simply predicting a single future step. Overall, curiosity based exploration utilizing multi-step show strong results, and could serve as a platform for further research and experimentation.

9 Acknowledgements

We would like to thank and acknowledge Ruben Villegas, who originally had the idea of extending [1] to multiple timesteps and for providing us with lots of guidance in this research. We would also like to thank Honglak Lee for his help and mentor-ship in this project and throughout this course.

References

[1] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

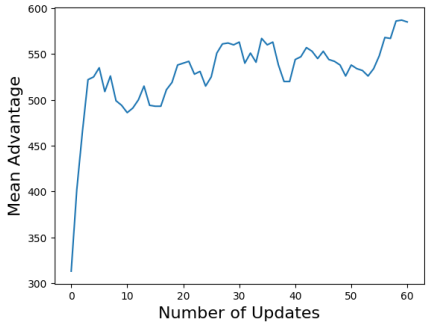
[2] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

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

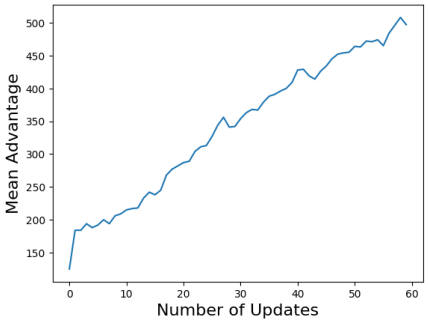
[4] Jürgen Schmidhuber. A formal theory of creativity to model the creation of art. In *Computers and creativity*, pages 323–337. Springer, 2012.

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

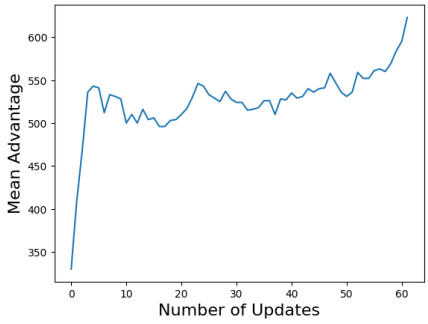
10 Appendix



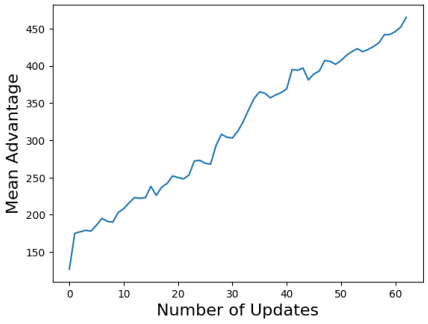
Space Invaders advantage



Breakout advantage



Space Invaders advantage



Breakout advantage

Results with updated policy architecture

Figure 6: Preliminary results from the initial run of the algorithm. The top row shows the results for the original architecture and the bottom row shows the results of the updated architecture.