The Andrew and Erna Viterbi Faculty of ELECTRICAL & COMPUTER ENGINEERING

TECHNION Israel Institute of Technology

Final Report - Project B

RL with Trajectory Feedback

Students: Hagay Michaeli and Uri Gadot
Supervised by: Guy Tenenholtz

CRML Control Robotics and Machine Learning Laboratory

Semester: Winter, 2021
Date: 10/10/2021

Contents

List of Figures

1 Abstract

For many RL problems an informative reward function can improve the learning process of a sparse reward environment, though it can be much harder to define. RL With Trajectory Feedback is an addon for regular RL problems, which enables to use an observer feedback on agent's performance to learn an informative reward function.

First, we showed that an informative reward function (per state, action pair) can be learned using a sequence reward function, which was represented by labeled sequences.

In addition, we showed that by using merely 200 user labeled short sequences, we could learn a reward function that brings unsolvable sparse-reward environment to converge.

2 Introduction

RL - Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

Designing an RL problem requires to define the "state space", "action space" and reward:

- State space - describes all the possible situations of the world (environment).
- Action space – describes the agent possible actions within the environment and/or the agent restrictions.
- Reward – feedback given by the environment to the agent to the agent for each action in each state.

The objective of the RL algorithm, is to create a Policy, in which the agent would choose the best action in order to increase the total cumulative reward given by the environment for those actions. For many RL tasks, deciding on the best "Action space" and "State space" to represent the problem in the most efficient way can be a difficult assignment.

In addition, deciding on a specific reward per action can be a hard task as well (*e.g. judging the quality of a motion of a robotic arm in a* $\frac{1}{30}$ *of a second would be a hard task*). Easier approach would be to give a reward to a longer period of actions ("window" of actions), in which the outcome of the agent selected actions are more clear. When the Action and State spaces are in the continuous domain the problem gets even harder.

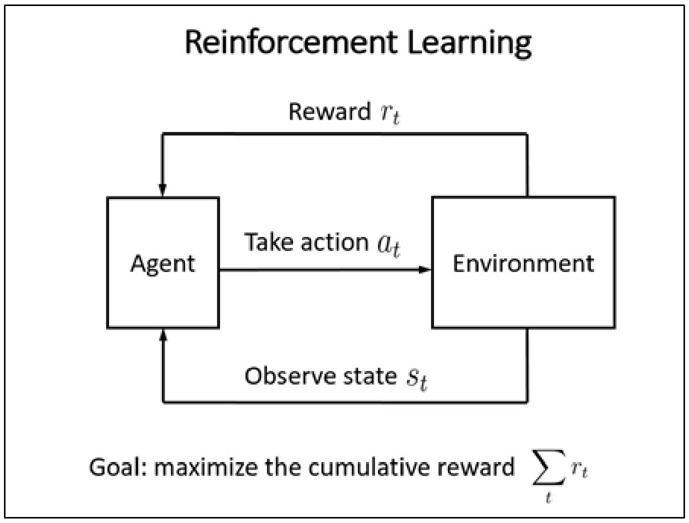


Figure 1: Basic concepts of Reinforcement Learning

Therefore, many RL environments are described with a 'Sparse reward', in which most of the agent actions would receive no reward (scalar value of 0). Except for an action that would lead the agent to the final goal. (*e.g. an agent traveling in a maze would get 0 for every action except for reaching the end of a maze.*)

A lot of RL algorithms can have difficulties getting to a good result in those kinds of environments, especially when reaching the informative reward (*e.g. the end of the maze*) can be a hard task without any rewards received along the way. This is Because the agent has to act "randomly" at this point.

In this project we will describe an approach dealing with these issues to make RL algorithms to converge more easily under those conditions.

We know that the RL algorithms require a _reward per each action taken. To deal with that we would assume that we can give a reward on a sequence of states and actions of the problem that correlates better with the problem solution than the sparse reward, given per state, defined by the environment. (*e.g. outside observer that gives a feedback to a sequence of robot moves inside the maze instead of "sparse" reward all the way except for the end*)

3 Literature Review

We used 2 previously written papers to inspire out work:

- The first one - **Reinforcement Learning with Trajectory Feedback** [1] which introduced the concept of learning a single step reward function out of sequence reward function.
- The second one - **Deep Reinforcement Learning from Human Preferences** [2], a work that has similar features, but the scoring method was done by comparing 2 sequences of the agent and deciding on the one with the better performance.
  Their implementation of the user scoring mechanism was very understandable, which inspired our code of the User Interface.

4 Modelling

4.1 RP concepts

As mentioned before, To try and solve this problem let's assume we have an "observer" that knows the desired task and can give a decent feedback on a given sequence (window) of state, action pairs.
We would like to add to the Agent a component which would learn a dense Reward function according to those feedbacks, and adds the learned reward to the environment Sparse reward in order to help the agent learn the problem better. We would call this component "Reward Predictor" (or RP).
The RP would be trained (either on-line, or off-line using "expert" windows), in order to converge into a decent reward function that should help the RL agent converge faster. (training concept displayed in figure 2).

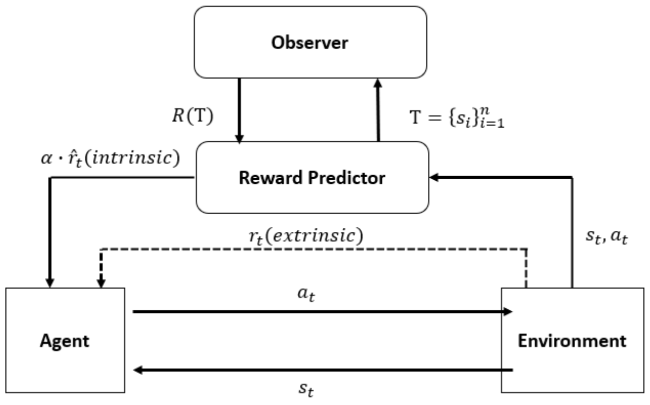


Figure 2: RP Training mechanism, using outside "expert" observer

4.2 Siamese Networks

For a trajectory $\mathbf{T}$ (where $\mathbf{T}$ is a trajectory of n steps), we will assume that the reward for the entire trajectory ($R(\mathbf{T})$) is equal to the Accumulated reward for all of the state-action pairs in the given trajectory:

$$R(\mathbf{T}) = \sum_i^n r_i$$

where $r_i = r_i(s_i, a_i)$.

In order to achieve the desired Module, a solution involving Siamese Networks was selected, in which there are n equal models with **shared weights** and the sum of the outputs would be compared to the labeled window reward. (as can be seen in figure 3).

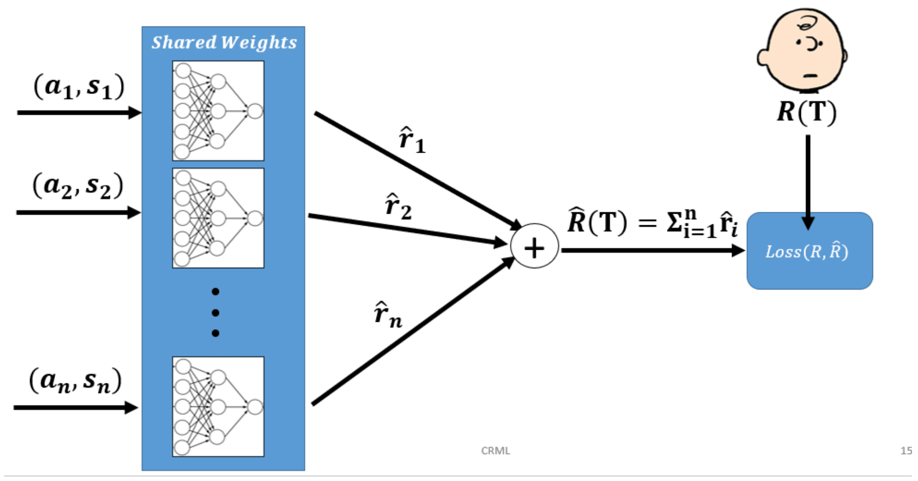


Figure 3: Siamese Networks training mechanism

4.3 Uncertainty

The phenomena of uncertainty in deep learning models should be treated in the RP module for 2 main reasons:
First, the predictions of the RP would be used as the reward of the agent, although they may be incorrect, e.g. in states that were not seen in the RP training.
In addition,in the setting of training RP online, we would like to minimize the number of windows an observer has to label. To do that we would rather skip windows we know that RP is certain about.

There are 2 types of Uncertainty when it comes to function modeling:

1. **Aleatory uncertainty** - The internal randomness of phenomena. [3]

2. **Epistemic uncertainty** - the scientific uncertainty in the model of the process, usually it is due to limited data and knowledge

To deal with the epistemic uncertainty, We use a method called **Model Ensemble**

4.3.1 Model Ensemble

Ensemble modeling is a process where multiple diverse base models are used to predict an outcome. The motivation for using ensemble models is to reduce the generalization error of the prediction [4]

Another Reason, is that using model ensemble helps reducing the Epistemic uncertainty of our RP. Basically we try to see if several models agree on the results of a certain step or not, which help us to reduce our trained windows to a smaller number.

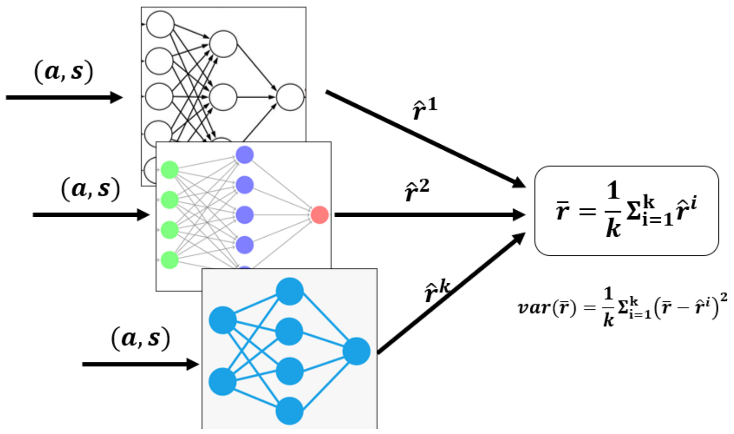

(a, s)
(a, s)
(a, s)
$\hat{r}^1$
$\hat{r}^2$
$\hat{r}^k$
$\bar{r} = \frac{1}{k}\Sigma_{i=1}^{k}\hat{r}^i$
$var(\bar{r}) = \frac{1}{k}\Sigma_{i=1}^{k}(\bar{r} - \hat{r}^i)^2$

Figure 4: Basic concept of Model Ensemble

The variance of the ensemble predictions can be an indication of the models agreement over certain reward.

4.4 Training

As mentioned before, we designed 2 methods of Training for the Reward Predictor Module:

1. Online Training
2. Offline Training

4.4.1 Online Training

In Online Training, the agent saves it's previous visited states and performed actions in a "Window Buffer" (*marked as* W).
Each time step, the current RP model would estimate the reward for the last single state, action pair, (s_t, a_t) using Forward propagation $\big(\hat{r_t} = RP(s_t, a_t)\big)$.
Then, a method of uncertainty is used to decide if the current Window Buffer was estimated with either high or low level of certainty.
If the level of uncertainty is high, the current W is added to the RP training Buffer (*which full of windows*).

In order to label those windows we can use several scoring techniques:

- **Automatic method** - Using a previously known reward function over sequences (*e.g. the dense reward function, or any other function*)
- **Manual method** - The desired technique, in which outside observer rate the inserted window with a score

While the manual method is the desired outcome it has a major downside in terms of time needed to rate each window, and accuracy of scoring function (*since again, humans cannot be accurate enough*).

The complete algorithm for the Online training is mentioned in algorithm 1. An illustration of the algorithm Block Diagram can be seen in figure 5.

Algorithm 1 Reward Predictor Online Training

```
1: at time step $t$: RP gets $(s_t, a_t)$ to evaluate
2: Evaluate reward $\hat{r}_t(s_t, a_t)$ and "confidence" (ensemble variance)
3: Update current agent running window
4: if $average\ window\ var > thresh$ then
5:     Get observer input on window
6:     Add $(window, observer\ input)$ to RP train buffer
7: end if
```

4.4.2 Offline Training

In this Training method, we divide the RL algorithm into 2 separate executions. In the first one, we use a "Dense" function to train the agent, and collect many windows while the agent preforms.
Then, we rate the windows (*using one of the techniques mentioned in section 4.4.1*).
After all of the windows are scored, we Train the RP model until it converges to a Reward function with certain level of error.
Then we begin the second RL algorithm execution, in which we load the *"pre-trained"* RP model. In this run no Window buffer is needed to be stored and

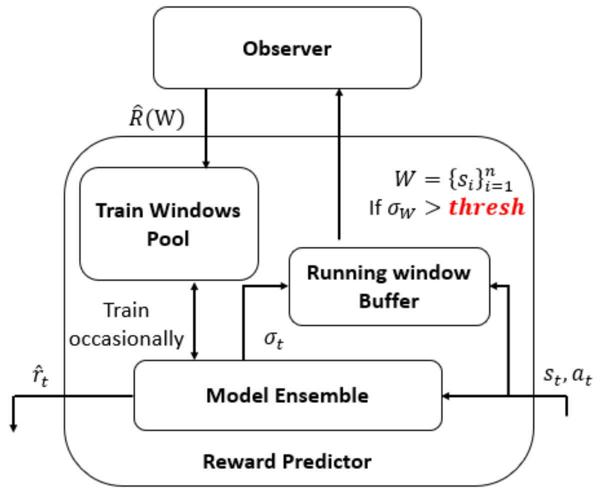


Figure 5: RP Online Train blocks diagram

no Online scoring is need to be done, since the RP model is already trained to the desired reward function.

5 Simulation Details

5.1 Environments details

5.1.1 Gridworld

A basic version of our own GridWorld environment with multiple rooms where the agent needs to reach a certain point in the grid.
Details:

- Agent can move up, down, left, right [Discrete]
- Agent goal is to reach an end-point
- Sparse Reward function is as follows:

$$R(s,a) = \begin{cases} 1 & if\ S = goal \\ 0 & else \end{cases}$$

In all simulations we used a 4 rooms map at size 30×30. For training speedup and increasing expolration we used $\times 8$ vectorized environments.

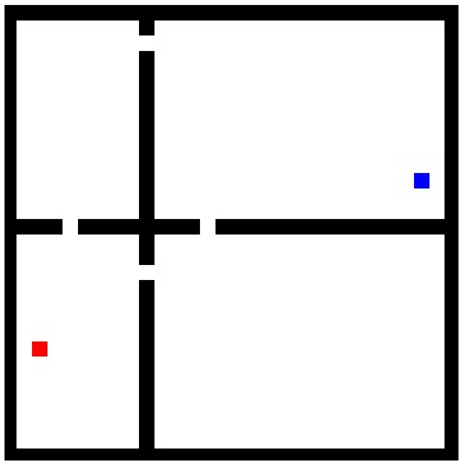


Figure 6: GridWorld env image

5.1.2 Mujoco

We chose one the basics environments in Mujoco, Hopper to be our POC environment for continuous action and state spaces.
Details:

- Action space is the control of agent joints [Continuous]
- Agent goal is to travel as much distance as possible
- Sparse Reward function is as follows:

$$R(s,a) = \begin{cases} 1 & if\ distance > X \\ 0 & else \end{cases}$$

For Mujoco environment we used openai gym implementation [5]. We trained the agent using Stable Baselines algorithm implementations [6]. We trained Rooms tasks using PPO [7], and Mujoco tasks with SAC [8].

5.2 Baseline Simulation

In order to get a POC we tested the following steps:

5.2.1 Sparse vs Dense Reward

We tested Rooms environment with 2 types of rewards:

- Sparse reward: $R_s(s,a) = \begin{cases} 1 & if\ S = goal \\ 0 & else \end{cases}$

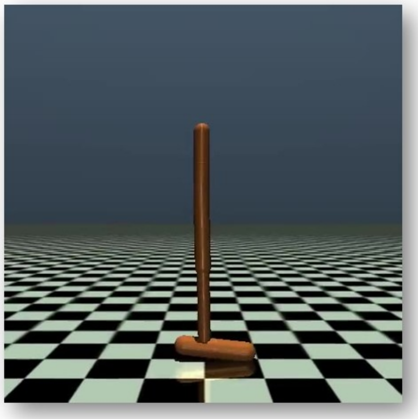


Figure 7: Mujoco Hopper env

- Dense reward: We added to the sparse reward R_s a component R_d which has inverse ratio to the shortest path distance from the goal.

This step helped us find an environment in which additional dense reward, such as RP, can improve the training with sparse reward only. The simulation showed that in a map at size 30×30 with 4 rooms and initialization in the farthest rooms from the goal, there is a significant gap between sparse and dense reward performance: dense reward converge after about 20K steps, while sparse reward did not improve at all 8(a).

5.2.2 RP POC

We tested the possibility to train a reward predictor with labeled windows as a reward function offline (with collected data) and online (during the agent's training). In both approaches the reward of a window was defined as the summation of the dense rewards of all steps in the window sequence. In this setting we also had to chose the window length. Figure 9 shows offline convergence of RP using windows with different length. We chose length 10 for the rest of the experiments. We could see in both simulations a convergence of RP to a function similar to R_d. We were also able to use the RP in both experiments to make the agent converge. In Online experiment we saw a convergence after about 60K steps 8(b).

5.3 Real User Simulation

In the RP POC setting there are 2 main problems which needs to be solved before training RP with real user input:
The first one is the number of windows required for training; In Baseline POC we used all the windows during the agents training as training data for RP. When

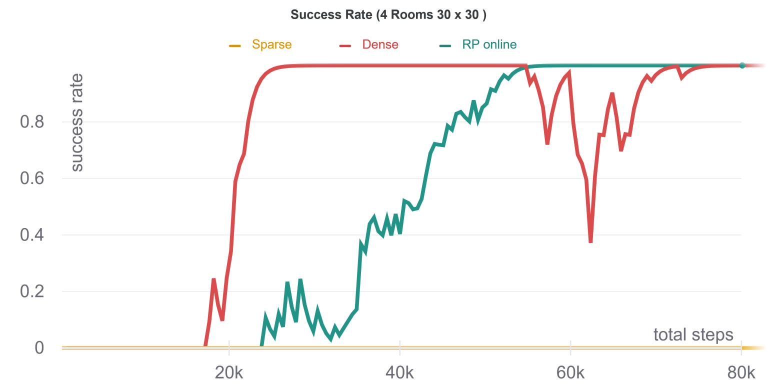


Figure 8: 4 Rooms 30 × 30
(a) Sparse (yellow) Vs Dense (Red) rewards Training comparison. Dense converge after 20K steps while sparse doesn't improve.
(b) RP POC - with RP trained online agent converge after 60K steps.

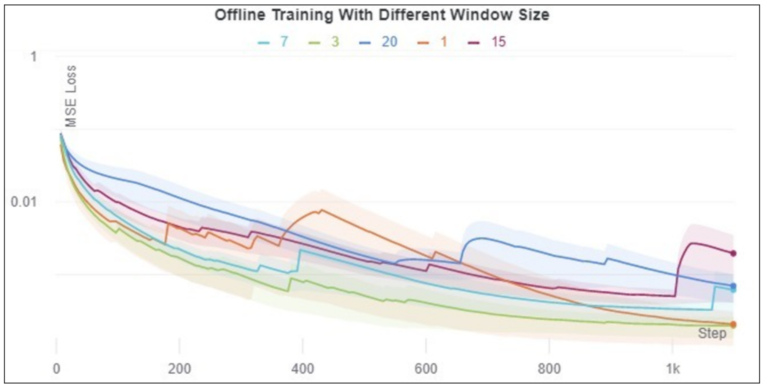


Figure 9: 4 Rooms 30 × 30 - RP Offline Training different several window sizes

using real user feedback we would like to minimize this number of windows, to reduce the amount of labeling work. The second problem is that the summation of dense reward is a complex function which doesn't represent the precision of human's feedback. We would like to simulate real user feedback by simpler reward function.

5.3.1 Reducing the number of asked windows

One of the hardest issues we faced during this phase was the enormous amount of windows that the RP marked with a high level of uncertainty.
The line of thought, was to reduce this number as much as we can, since in the next phases we wanted to replace the automatic scoring function with a human, so reducing the amount of overload was necessary.

To achieve that we wanted to look for the right value of the "Variance Threshold" that the RP uses to consider a window "Trainable", higher values would mean less windows but can also reduce significantly the number of trainable windows, which in results would make the RP model unstable, or not even close to a good reward function estimator, thus causing the RL agent to not converge, or not succeed in it's task.
In figure 10 we can see the experiment with several values of the "Variance Threshold" and how changing it effects the number of asked windows.
In figure 11 we can see the effect of changing the variance threshold over the RP model convergence (*in terms of Loss*).
In figure 12a we can see the effect of changing the variance threshold over the RL agent convergence (*in terms of Distance from goal cell in GridWorld*).

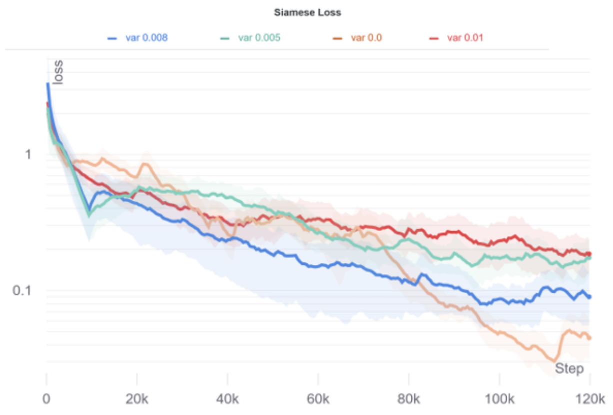


Figure 10: Variance Threshold experiments - RP loss through time

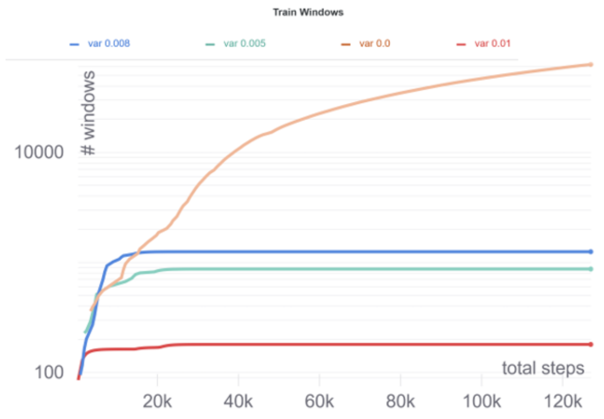
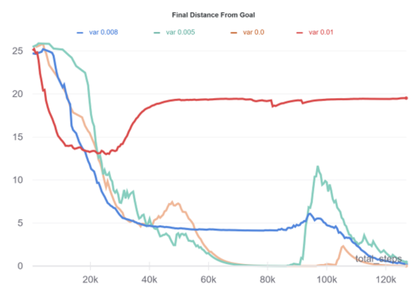
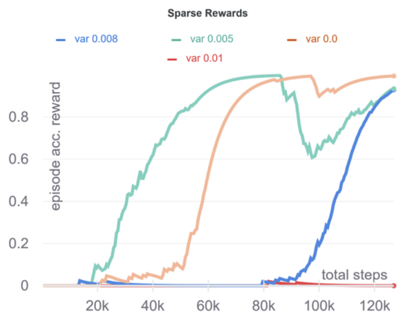


Figure 11: Variance Threshold experiments - number of windows which are "trainable"



(a) Final distance from goal cell



(b) Average sparse Reward

Figure 12: Variance Threshold experiments - Agent Success

5.3.2 Online "Discrete" Training

Since giving a manual estimation of the agent performance on a given trajectory is both a discrete process, but also a very noisy one. Since human capabilities of staying consistent are limited.
Therefore, we changed the trained reward to be Both **discrete** (a set of 10 possible values) and also **noisy**. We wanted to see the effects of this change on the results from previous experiments.
In figures 13, 14 and 15 We can see that discretizing the Windows score causing the RP to converge faster, thus reducing the number of windows significantly, while preserving the same performance in terms of RL agent convergence.
We are still not sure what was the reason, but those findings are certainly very interesting.

In addition, adding the noise to the already discrete scoring function, causing a significant rise in the number of windows while also hurting the total outcome of the RL agent convergence rate.
These results are encouraging in terms of continuing our experiments to the next phase of involving human in the process of rating the windows performance.

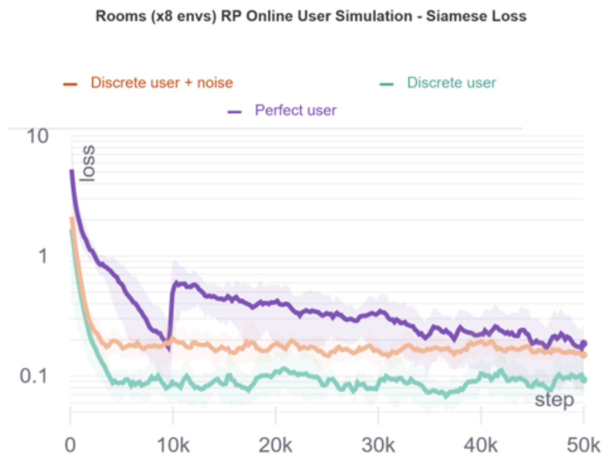


Figure 13: Continuous Vs Discrete Vs Discrete + Noise - RP loss through time

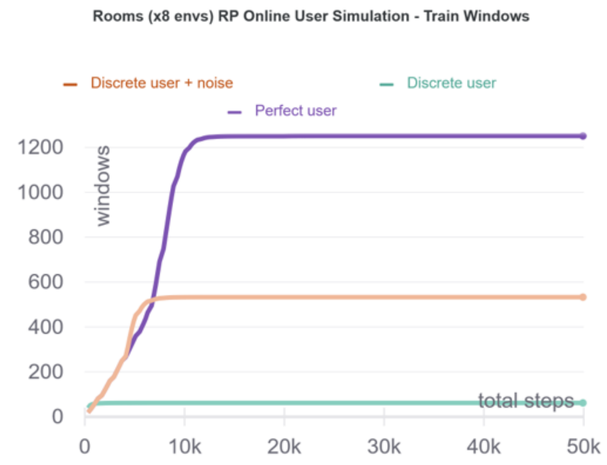


Figure 14: Continuous Vs Discrete Vs Discrete + Noise - number of windows which are "trainable"

5.4 Offline Manual Training

Finally, We tried to reach a training where we can find a small amount of Windows, give a Human based feedback and train the RP Off-line with this data. Then, We trained the agent on the sparse environment with the pre-trained RP module and compared the results to the basic sparse-only reward. The results can be seen in figure 16, in which the RP which were trained with more windows, helps the agent reach a higher level of success, and definitely improving the sparse only trained agent.

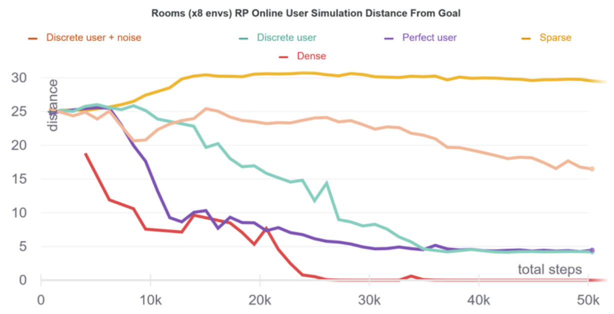


Figure 15: Continuous Vs Discrete Vs Discrete + Noise - success of RL agent using the trained RP model

Another interesting fact is the comparison between the "per-step" predicted reward Vs the real dense reward of those Offline-trained RPs. In figure 17a and 17b We can see 2 RPs. One was trained with only 50 windows and the second one was trained with 200 windows.
We can see that the predicted reward and real reward of the 200 windows trained RP have the same pattern, while the 50 windows trained RP have different patterns in the more latter stages of the training. This perhaps can be related to the fact the number of different states it was trained on was not sufficient to understand the reward function thus, hurting it's predicted reward and the agent training.

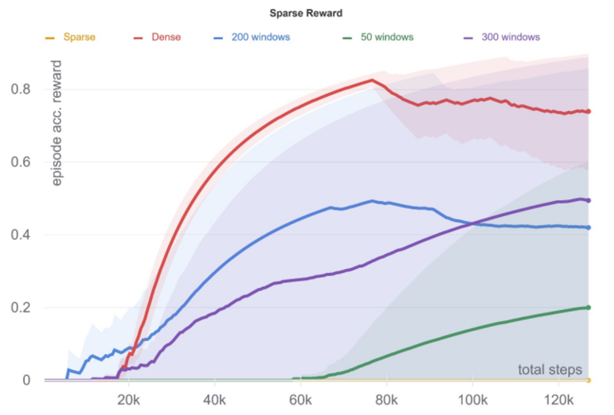
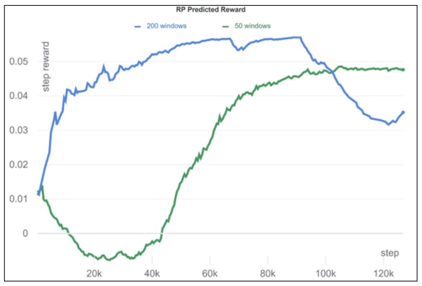
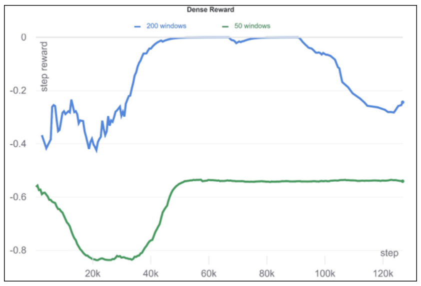


Figure 16: Manual Offline RP training - Success rate of RP's trained with different number of windows



(a) Reward Predictor predicted reward



(b) Real Dense (Dijkstra) reward

Figure 17: Manual Offline RP training - Predicted Vs Real reward (per step)

6 Implementation

All of our code can be found in: https://github.com/Ugadot/RL_with_trajectory_feedback

7 Future Work

In the future, some resources are needed in order to conduct an Online training - where the observer scores the agent windows while the agent is training.

Succeeding with this experiments, can lead to a very interesting conclusions, and perhaps help solve other unsolvable problems.

In addition, adding an NLP module, in order to make the user scoring capabilities more realistic (such as giving verbal feedback instead of discrete scalar), can also help with the Online, scoring mechanism.

8 Summary

We showed several important conclusion from our work:

1. Generating reward function can be hard task
2. Learning reward function from trajectory based input is possible
3. Training a module (**Reward Predictor**) successfully Online (while the agent is learning) is also possible.
4. Combining the **Reward Predictor** as intrinsic reward to a sparse reward environment can improve performance of agent convergence, and perhaps help with unsolved problems.

References

[1] Y. Efroni, N. Merlis, and S. Mannor, *Reinforcement learning with trajectory feedback*, 2021. arXiv: 2008.06036 [cs.LG].

[2] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, *Deep reinforcement learning from human preferences*, 2017. arXiv: 1706.03741 [stat.ML].

[3] A. D. Kiureghian and O. Ditlevsen, "Aleatory or epistemic? does it matter?" *Structural Safety*, vol. 31, no. 2, pp. 105–112, 2009, Risk Acceptance and Risk Communication, ISSN: 0167-4730. DOI: https://doi.org/10.1016/j.strusafe.2008.06.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167473008000556.

[4] V. Kotu and B. Deshpande, *Deep Learning*. 2019, ISBN: 9780128147610.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. eprint: arXiv:1606.01540.

[6] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, *Stable baselines3*, https://github.com/DLR-RM/stable-baselines3, 2019.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG].