Interpretable Hierarchical Reinforcement Learning

Generalization through hierarchical learning

Divyat Mahajan (14227) Harsh Sinha (14265)

divyatm@iitk.ac.in *harshsin@iitk.ac.in*

Supervisor: Prof. Vinay Namboodiri

Abstract

We present a method for adding interpretability in reinforcement learning by the use of hierarchical learning and combining it with an information maximization scheme. We use latent variables correlated with different trajectories to help in hierarchical learning in a multiple goal environment, while banking on this hierarchy to help us have generalization capability on unseen states and goal. In this direction we also try to use ideas from value function approximation to hasten training on unseen goals. Finally we present experiments on a custom gym environment to designed to validate the proposed method.

Keywords: Reinforcement Learning, Meta Learning, Multi-goal Generalization, Hierarchical Learning

I. Introduction

In the recent years Reinforcement Learning has proven itself by producing one impressive result after another. Reinforcement Learning has been used to out perform the world champions in the game of Go [1] which is a fully observable strategy game, defeating the reigning world champions in the team collaboration based real-time strategy game of DOTA-2 [2], performing complex robotic tasks [3], and learning to do complex dexterous hand manipulation [4]. However the training time does not scale well with the number of parameters, for instance OpenAI five trained for millions of hour of game play, though trough modern advances in modern computation it was compressing 180+ years of gameplay everyday of training. This is in part a result of the general approach taken in Reinforcement Learning, training each task from scratch. Humans, and other animals on the other hand, use the knowledge from one task while learning another related task, this prior information lets us learn in a much smaller time. There have been many leaps towards sample efficient reinforcement learning recently [5, 6, 7], yet without the use of prior knowledge there will be a limit on the effectiveness of such methods.

There are many methods which allow for the use of prior knowledge while learning, these include Imitation Learning [8, 9, 10], where the agent is given expert demonstration on tasks; Meta Learning [11, 12, 13], where the agent essentially has two parts, the planner and the worker, where the planner chooses which worker to activate and each worker does some small part of the overall task; Hierarchical Reinforcement Learning [5, 14, 15]. Furthermore, we

note here that value functions used in reinforcement learning which stand for the utility of a particular state in completing the given task also have been expected to have similar functions when generalizing on unseen goals [16].

In that spirit we have attempted here to use hierarchical learning in combination with information maximization approaches in order to generalize on unseen goals in an environment. The goal is to compare and try to incorporate Value function approximation techniques to this hierarchical architecture thereby allowing us to learn sub-tasks which maybe shared between different goals and thus be able to generalize on unseen goals quickly. The information maximization technique would allow us to have control over the sub-tasks which are learned and make these sub-tasks interpretable. For the information maximization reinforcement learning, we turn to InfoRL [17]. The idea is to have a latent variable which would capture the approach to solving a task in a particular way, for instance a latent code could be used to control the speed at which an agent moves in an environment. Thus the latent code has the ability to disentangle multiple policies for solving a particular task. Thus, the latent code is used to exert control over the sub-tasks and it is also the entity that introduces interpretability to our system. Since InfoRL needs no special supervision and only works on reward functions as is standard in reinforcement learning we can easily switch between different algorithms, from tabular-Q learning [18] to DDPG [19] for continuous control problems. We would be discussing these in greater details in the coming sections.

Since our work is a combination of ideas from various previous works we would first be looking at the relevant work done in the past II, some of which have already been introduced in this section, and try to lay down the exact problem statement III. We will then take a look at the background IV required for this article. Then we present our methodology in detail V post which we give out the details for experiments VI.

II. **Literature Review**

The pre-existing work which we most heavily draw on is Hayat et al.'s InfoRL [17]. In this work the authors utilize the information maximization techniques as used in InfoGAN [20] and InfoGAIL [21] in order to introduce disentanglement between near optimal trajectories in complex environments. InfoRL uses sampled latent codes to generate trajectories which are then used to predict a latent code using a posterior network, the reconstruction loss between the predicted and the sampled latent code is then used to add a posterior reward for the agent to maximize. Further, the information theoretic concept of mutual information maximization is applied between the state action pair and the latent code. This ends up ensuring that the latent codes correspond to specific trajectories, and by the maximization of the standard environment reward, it is ensured that the latent codes correspond to specific near optimal trajectories for achieving the given task. This approach requires that the environment is complex enough for it to have multiple near-optimal trajectories for the task, and thus have trajectories to disentangle, i.e. if there aren't enough paths for the agent to choose from, one might already have enough information about the environment, thereby rendering interpretability useless.

Apart from this approach most of the interpretable reinforcement learning is based on hierarchical learning. The core idea in hierarchical reinforcement learning is that there are various levels in the agent and the higher ones learn about the task as a whole whereas

the lower hierarchies do not know about the exact nature of the task but are relegated some sub-task from the upper levels. Thus, the lower levels learn how to do these sub-tasks accurately, which may be repeated over different tasks, and the upper levels learn how to choose which lower level to engage and when to do so. This is the approach used in Hinton et al.'s Feudal Reinforcement Learning [14], where the upper hierarchies hide not just the overall task but also actual environmental reward from the lower ones and reward the lower levels for doing their biding irrespective if the actual goal was achieved or not. Only the lowest levels are allowed to act and all the levels above them set the goals for the levels below. This means once the lower levels have learned how to act in the environment they may be easily transferred to other tasks in the same (or similar environment) with the upper levels having to learn how to break down the new task so that the lower levels can be used to achieve the goal. Note that the narrower the sub-tasks the more interpretable would the agents action be for us. Similar structure or hierarchy is used by many different papers, such as Frans et al.'s Meta Learning Shared Hierarchies (MLSH) [15]. In MLSH the authors have two sets of parameters at different levels, the shared parameters ϕ_k where k corresponds to lower-level actions allowed, the higher level parameters, termed per-task parameters, θ have to learned for individual tasks while the ϕ_ks remain fixed. The ϕ_k's are learned for specific lower-level k^{th} task, when all other parameters remain fixed and the updates are done to maximize the future reward. Here each parameter vector is encoded using a Neural Network. Hence, quite similar to the Feudal system, we have a network θ choosing the lower level network ϕ_k to be activate. The selection is done for the next T time steps which is done to ensure that the master (higher level) policy functions at a slower timescale than that of the action (lower level) policy. The update scheme is designed to ensure that we learn sub-policies that are optimal for the tasks we trained on and also generalise well to new tasks. It includes a *Warmup period*, in which given the set of shared sub policies ϕ_k, it learns an optimal θ. This is accompanied with a *Joint Update period*, in which it optimises both the shared and task specific parameters to obtain optimal values for shared sub policies ϕ_k. The warmup period is important since we should update the shared sub policies only when the task specific parameters θ are near their optimal value. The argument to generalise over unseen tasks is based on the assumption that the Warmup period will learn the near optimal values of θ for the fixed set of sub policies. Hence, after learning a set of optimal sub policies, the model can generalise to new unseen tasks by using only the Warmup period updates.

Schual et al.'s [16] Universal Value Function Approximators proposes function approximators $V(s, g, \theta)$ or $Q(s, a, g, \theta)$, where s, a, g are states, actions and goals and θ's are the parameters, which can generalize on states and goal alike. They have shown that these approximators can generalize not just on seen goals but also on unseen goals as well by exploiting the inherent structure of the goal space and the state space. Now, given that out of all the states and goals, the agent would only see a fraction, in order to generalize, the authors have used a method similar to matrix factorization. They find out embedding vectors $\hat{\phi(s)}$ and $\hat{\psi(g)}$ by performing low rank matrix factorization of the sparsely filled value function table $V_g(s)$, after this they do separate multivariate regressions to obtain using two networks ϕ and ψ towards the target embeddings $\hat{\phi}$ and $\hat{\psi}$. Now, the reason we believe UVFA is of importance is because if the set of tasks at hand share dynamics and differ in

goals then it would be possible to initialize the $V_g(s)$ with the results from generalization and the goal g be achieved quickly. The tasks undertaken by the MLSH and InfoRL tend to fall into this category, i.e. they share the environment dynamics and have different goals and thus UVFA can be used seed the value functions and thus help with generalization over goals in an even faster manner.

III. Problem Formulation

Thus, overall, our task is to create a system for reinforcement learning which trains on few goals in an environment and then generalizes on different unseen goals quickly, for doing this we are trying to use hierarchical learning methods and in doing this we also aim at having an interpretability in regards to sub-policies which are chosen for different tasks, this is shown graphically in figure 1.

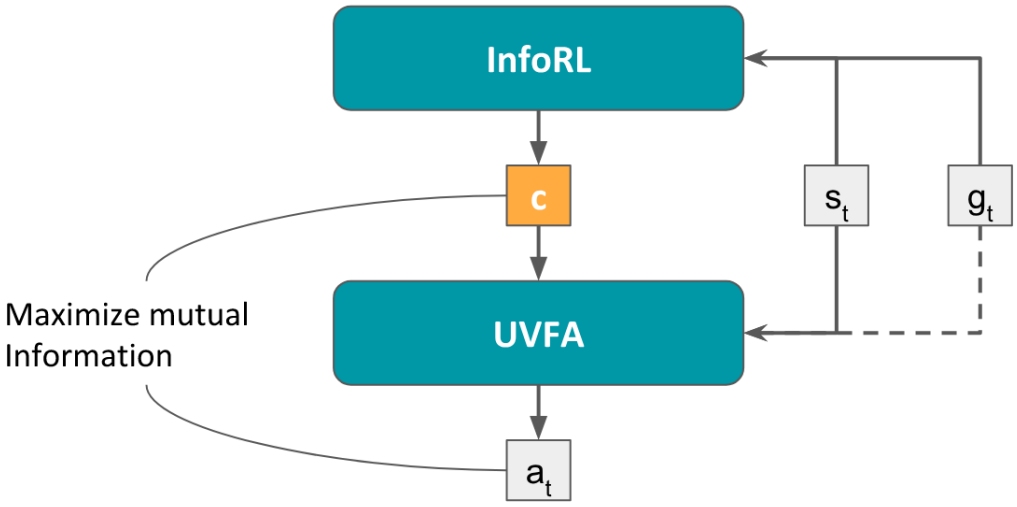

Figure 1: The problem statement.

IV. Background

A. *Reinforcement Learning*

We use the standard formulation of the reinforcement learning problem. For which we assume an infinite horizon Markov Decision Process (MDP), represented by the set $(S, A, T, R, S_0, \gamma)$, where S is the set of all state, A is the set of all actions, $T : S \times A \times S \to \mathbb{R}$ is the state transition probability distribution, $R : S \to \mathbb{R}$ is the reward function, S_0 is the distribution of initial states, and the $\gamma \in [0, 1]$ is the discount factor for the MDP. Note that we modify the reward function based on the InfoRL, as will be described in detail in the coming sections.

Furthermore since we would be working in a multi-goal setting, let us also introduce the following: G the set of all goals, R_g the reward function corresponding to goal $g \in G$ and

$\gamma_g : S \to [0, 1]$, where $\gamma_g(s \neq s_g) \in (0, 1]$ serves as discount factor and at s_g, i.e. the state corresponding to the goal g, $\gamma_g(s_g) = 0$, functioning as a soft termination of the MDP. The goal as usual is to maximize the discounted future rewards, $R_t = \sum_{i=t}^{t_g} \gamma_g^{i-t} r_i$, where r_t is simply the reward obtained at the time step t, which could be give by $R_g(s_t)$.

$\pi : S \times A \to [0, 1]$ is a policy under which the actions are taken, and the values it takes represents the probability of selection of that particular action. Note that there maybe a deterministic version of this policy $\pi : S \to A$. We will also be utilizing the Q-value and Value functions' standard formulation:

$$Q_{g,\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{t=0}^{\infty} R_g(s_{t+1}) \prod_{k=0}^{t} \gamma_g^i(s_k) \right]$$

$$V_{g,\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[\sum_{t=0}^{\infty} R_g(s_{t+1}) \prod_{k=0}^{t} \gamma_g^i(s_k) \right]$$

$$where,\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim T(s_{t+1}|s_t, a_t)\ and\ t \geq 0$$

B. Mutual Information

When looking at the case of nearly optimal trajectories for achieving a task, we would probably like to have a variable or a vector thereof, which would both tell us which trajectory has been chosen and allow us to have control over which trajectory gets chosen. Now, in real world and thereby in robotics tasks there are many nearly optimal options. Take the example of a navigation task, in case of different paths to the goal where the goal is to move away from the center, it would be great if we could also learn to move in specific directions based on some variable, similarly if we could associate some different variable with the speed of motion, we could have a master network which can use these two variables to easily perform any navigation task.

The idea of information is taken from the Information theory, where the concept of information is essentially to quantify the amount of surprise one might get from the results of an experiment. Mutual Information Maximization in simple terms refers to an idea which is roughly the inverse of uncertainty principle, i.e if mutual information is maximized between two quantities, and you have high probability corresponding to one of the quantities, you are less likely to be surprised by the results of an experiment which measures the value of the other quantity.

Mathematically, if we have random variables X and Y, then the mutual information (I) between them can be written as a function of the entropies (H) :

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X), \quad where$$

$$H(X) = -\sum_{x \in \mathbb{X}} p(x) log_e(p(x))$$

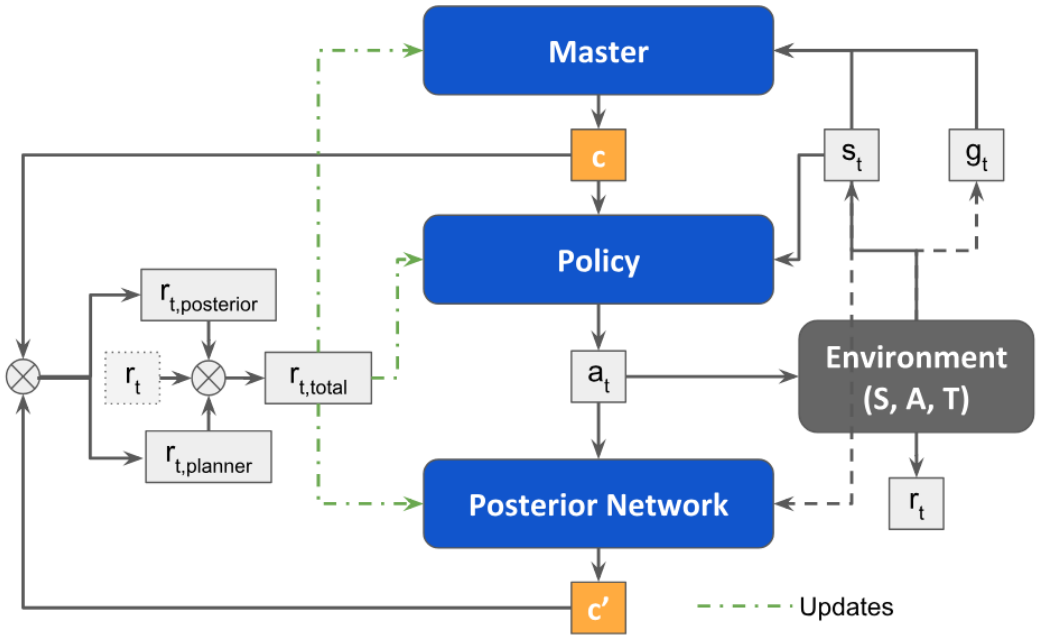

Figure 2: System Architecture: change this figure

V. Methodology

Our core architecture is as shown in figure 2. As explained in the II subsection we have added the posterior reward to maximize the mutual information between the actions and the latent code. Thus, the master policy along with the posterior network stand for the InfoRL module from 1 and UVFA module from the same has not been included in the pipeline shown as it is not required to be done at training-time. Since we are aiming to use the matrix factorization based method for UVFA, we can collect the data regarding the values and corresponding states at training time and then post that we can use matrix factorization to generalize on unseen goals and states. Let's look at the algorithm is detail.

We have three networks, or equivalents thereof for instance Q-tables, namely Master, Policy, and Posterior. The master network, generates the latent code based on the current state and the goal, thereby selecting what sort of lower level actions would be taken. At this point there maybe two paths, fixing the latent code from the master at the start of the episode or updating the latent code every few steps. The policy network then learns a policy based on the latent code and its interactions with the environment. The updates to the policy are done after every time step and the reward used for the updates is the sum of posterior, planner, and the environment reward. The posterior network uses the generated action state pair to guess the latent code generated from the master and this is used as a posterior reward in updates to master and the posterior, this way the choice of the latent code and the state, action pair can become correlated.

Algorithm 1: Interpretable Hierarchical Reinforcement Learning

initialize master(ϕ_m), policy(π_θ), and posterior(ϕ_p);
repeat
  sample goal $g \sim \mathbb{G}_T$;
  $s \sim S_0$;
  c $\sim$ master_network(s, g);
  repeat
    sample trajectories $\tau, s, r \sim \pi_\theta(c, s)$;
    sample state action pairs $\xi \sim \tau$;
    $c' \sim \phi_p(\xi)$;
    $r = r + posterior\_reward(c, c')$;
    update_policy(r);
    update_master(r);
    update_posterior(c, c');
  until *convergence*;
until *N times*;

VI. Experiments

We perform experiments in a multiple goal grid environment as show in the figure 3. The goals can be set at in the environment at any time. We choose a set of N goals before the experiment and train our networks on these four goals, after which we show the generalization on different goals. In the environment, given that it is a discrete one, there are four allowed actions, Up, Down, Right, and Left.

For the experiments we take a neural network for the posterior and make q-tables for the master and the policies. The master is a table of size $|S| \times |G| \times |c|$. The latent codes are selected from this table using argmax. The selected code and the current states are then fed to the policy network, which is a table to size $|S| \times |c| \times |A|$. The master network is trained at fixed intervals (and consequently at slower time scale) and we alternate between the policy network and the posterior updates. As seen from figure 2, the reward used to update the networks is created by summing up the posterior, planner and the environment rewards. The planner reward is added in order to penalize the master network for too much variance in the set of latent codes generated for the whole episode. Unlike the MLSH, we are allowing the master and the policy networks at the same times, only adding the planner reward to ensure correlation between the trajectories and same latent codes.

We did another set of experiments, this time the master and the policy network along with the posterior network were all neural networks. Due to their inherent nature, we expect them to perform better in generalization than any other method, but on the flip side they do require more training time. Due to a time crunch an the fact that we weren't able to get them to learn the environments using the neural networks we couldn't finish the experiments using this path.

A. Results

In the figures below from figure 4 to 6, we show the results obtained.

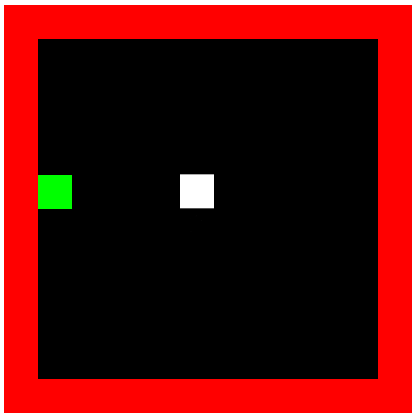

Figure 3: Grid Environment,
boxes are of grid size, green – goal, white – initial position

The figure 4 is generate by training on four goals, with 4 latent codes. Similarly 5 does the same with 8 latent codes and 8 goal. We later on generalize on the same 8 goal environment in the figure 6.

B. Discussion and Conclusion

From the plots above we can say that we are able to establish the correlation between the latent codes and the various trajectories for different goals and are successful in avoiding any kind of catastrophic forgetting as well in case of multiple goal environments. In addition to this, in the figure 4 and figure 5, we can see that even though master was allowed to select latent codes are every step, the rewards function correctly to lead to a single latent code per trajectory.

The generalization results as shown in in figure 6, are of great importance, these plots are generated by testing the performance on all of the possible goals in the grid, using the training from the 8-goal case. The sub-plot (a), shows that in general the unseen goals take very few steps (updates) to be solved, and the sub-plot (b), shows that the agent also targets the goals in most cases. In the sub-plots (c) and (d), we see that the as more and more training is done on the unseen goals, both the win-rate and the path-length improve. If we were to train these from scratch (as is the case for the original 8-goals here), it takes a minimum of a few hundred updates, where as when under our method, it take roughly around 20 updates only.

Also, the results can be further improved by the use of matrix factorization techniques on the value function, in a similar way to that done in universal value function approximators [16]

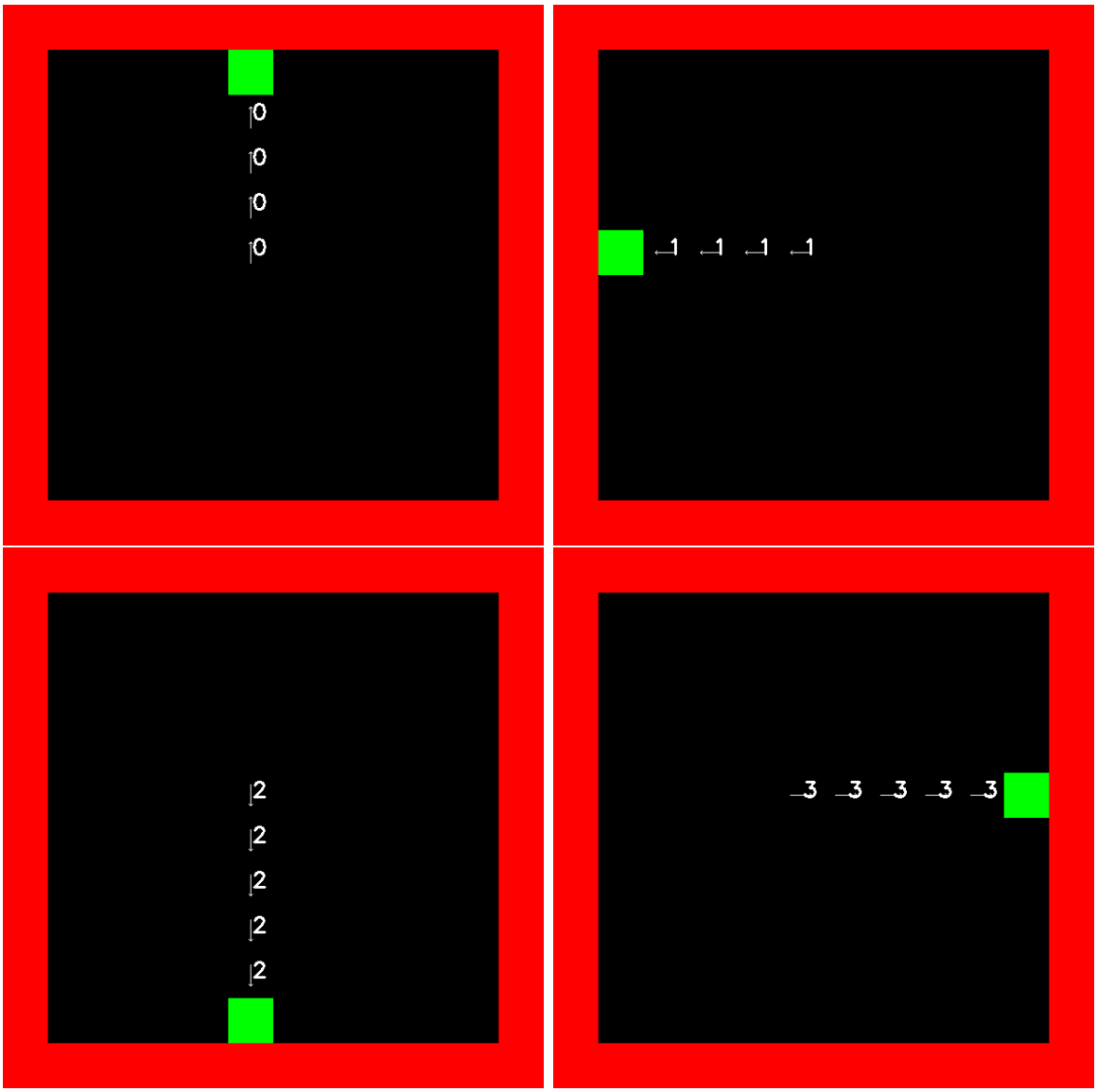

Figure 4: 4 Goal Results

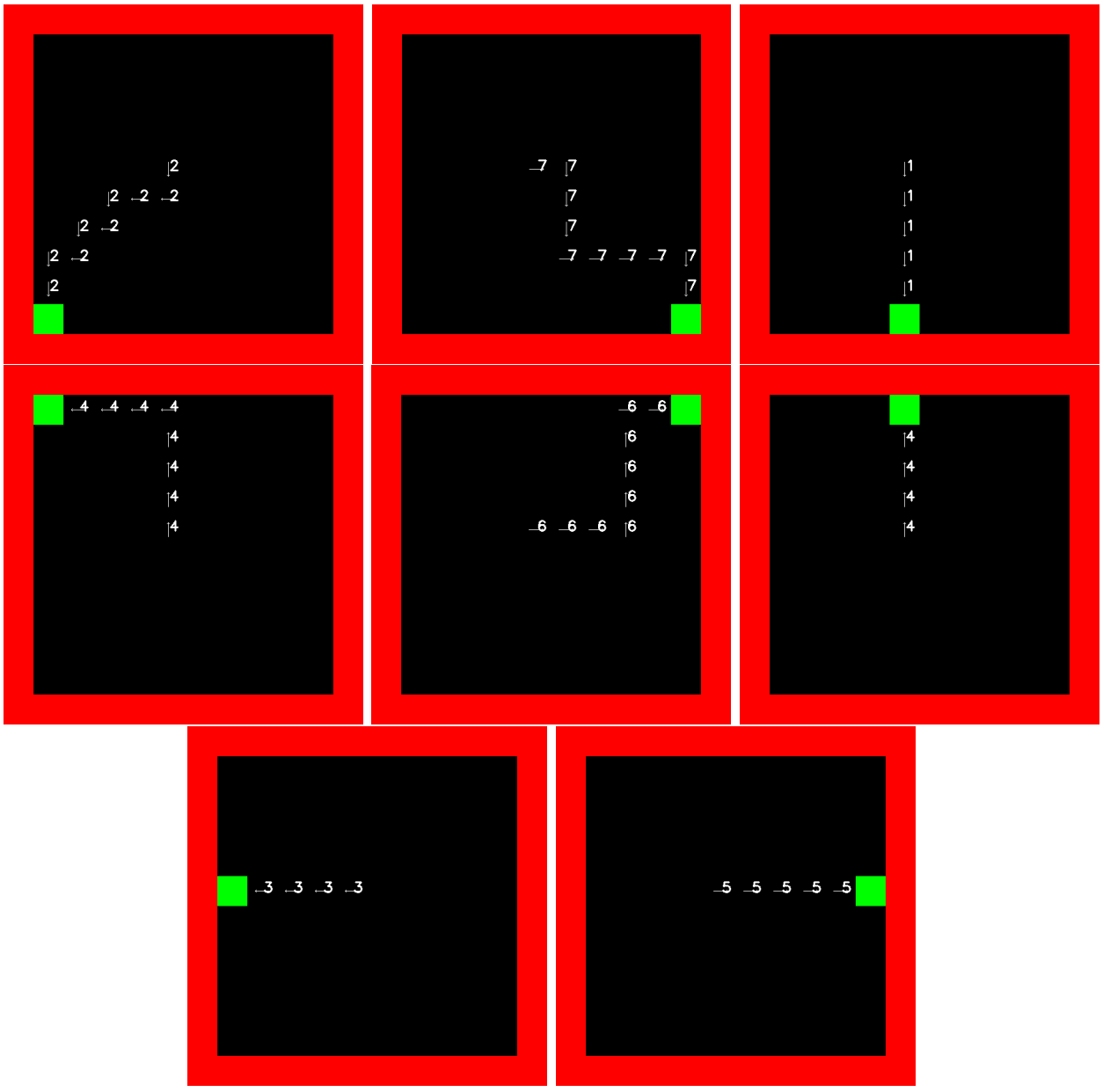

Figure 5: 8 Goal Results

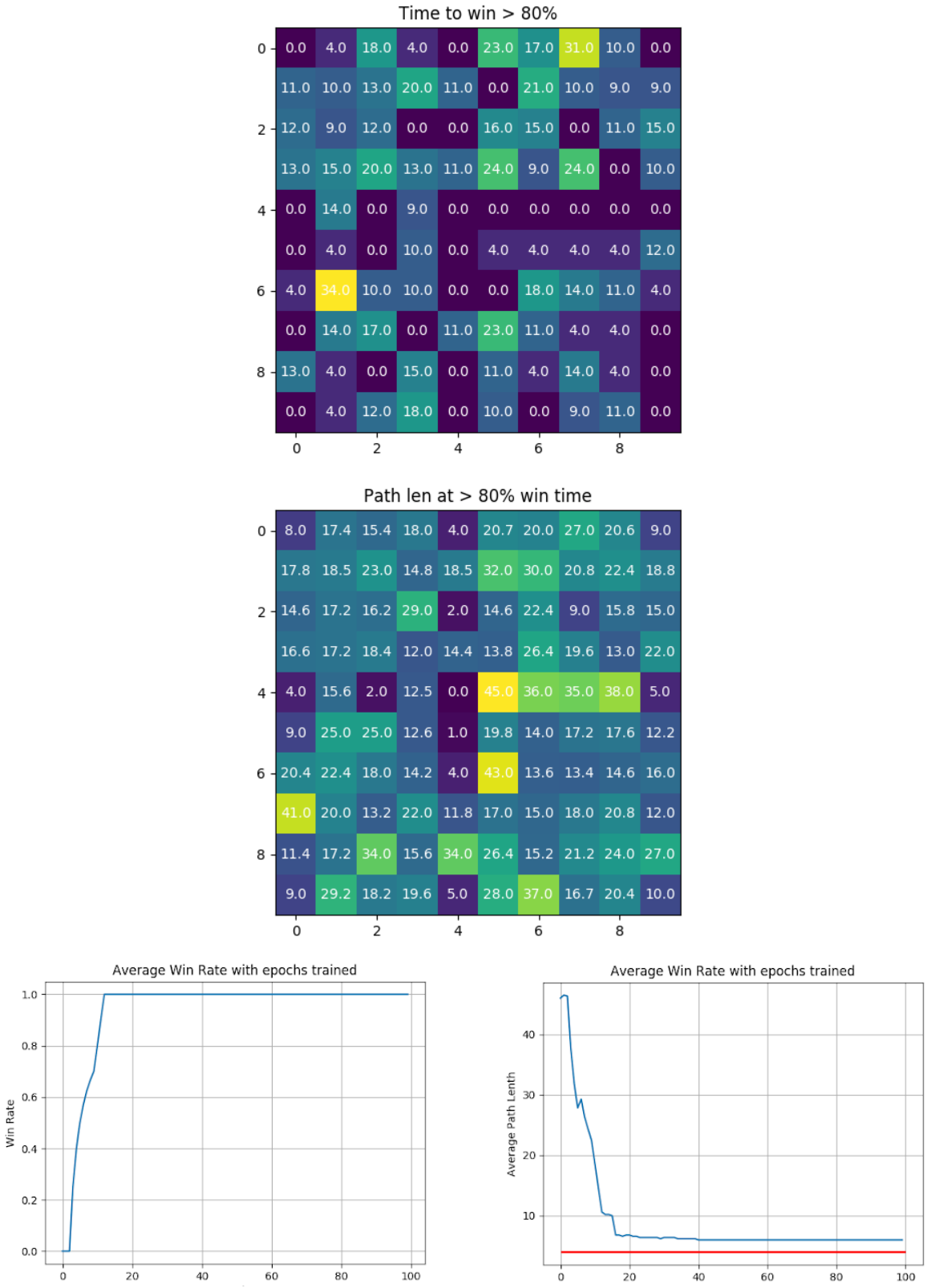

Figure 6: 8 Goal Generalization Results

VII. **Future Work**

As discussed in the VI section, the next logical steps for this project would be go in the direction to testing on increasingly complicated environments. We had begun experimentation on a robotic hand based environment similar to the navigation environment as shown here, comparison on that environment with other methods which have similar goals as ours would be next.

Furthermore, it would be interesting to try and use this method to trajectories and use them in a transfer setting, say trying to learn quickly in an environment where the physics has changed in which we originally trained.

VIII. **Acknowledgements**

Besides the vital suggestions from our supervisor Prof. Vinay Namboodiri, we are thankful to Aadil Hayat (17111001) and Utsav Singh (16511261) for their help and support.

IX. References

[1] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484.

[2] OpenAI, "OpenAI Five" URL: https://openai.com/five/.

[3] Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." The Journal of Machine Learning Research 17.1 (2016): 1334-1373.

[4] Rajeswaran, Aravind, et al. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." arXiv preprint arXiv:1709.10087 (2017).

[5] Nachum, Ofir, et al. "Data-efficient hierarchical reinforcement learning." Advances in Neural Information Processing Systems. 2018.

[6] Buckman, Jacob, et al. "Sample-efficient reinforcement learning with stochastic ensemble value expansion." Advances in Neural Information Processing Systems. 2018.

[7] Gruslys, Audrunas, et al. "The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning." (2018).

[8] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[9] Ho, Jonathan, and Stefano Ermon. "Generative adversarial imitation learning." Advances in Neural Information Processing Systems. 2016.

[10] Peng, Xue Bin, et al. "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills." ACM Transactions on Graphics (TOG) 37.4 (2018): 143.

[11] Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[12] Mishra, Nikhil, et al. "A simple neural attentive meta-learner." arXiv preprint arXiv:1707.03141 (2017).

[13] Duan, Yan, et al. "RL ^{2}: Fast Reinforcement Learning via Slow Reinforcement Learning." arXiv preprint arXiv:1611.02779 (2016).

[14] Dayan, Peter, and Geoffrey E. Hinton. "Feudal reinforcement learning." Advances in neural information processing systems. 1993.

[15] Frans, Kevin, et al. "Meta learning shared hierarchies." arXiv preprint arXiv:1710.09767 (2017).

[16] Schaul, Tom, et al. "Universal value function approximators." International conference on machine learning. 2015.

[17] Hayat, Aadil, et al. "InfoRL: Interpretable Reinforcement Learning using Information Maximization." To Be Published.

[18] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

[19] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[20] Chen, Xi, et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." Advances in neural information processing systems. 2016.

[21] Li, Yunzhu, Jiaming Song, and Stefano Ermon. "Infogail: Interpretable imitation learning from visual demonstrations." Advances in Neural Information Processing Systems. 2017.