arXiv:2203.00874v1 [cs.LG] 2 Mar 2022

Follow your Nose: Using General Value Functions for Directed Exploration in Reinforcement Learning

Somjit Nath[1], **Omkar Shelke**[1], **Durgesh Kalwar**[1], **Hardik Meisheri**[1], **Harshad Khadilkar**[1, 2]
[1]TCS Research
[2]IIT Bombay
{somit.nath, shelke.omkar, durgesh.kalwar, hardik.meisheri, harshad.khadilkar}@tcs.com

Abstract

Exploration versus exploitation dilemma is a significant problem in reinforcement learning (RL), particularly in complex environments with large state space and sparse rewards. When optimizing for a particular goal, running simple smaller tasks can often be a good way to learn additional information about the environment. Exploration methods have been used to sample better trajectories from the environment for improved performance while auxiliary tasks have been incorporated generally where the reward is sparse. If there is little reward signal available, the agent requires clever exploration strategies to reach parts of the state space that contain relevant sub-goals. However, that exploration needs to be balanced with the need for exploiting the learned policy. This paper explores the idea of combining exploration with auxiliary task learning using General Value Functions (GVFs) and a directed exploration strategy. We provide a simple way to learn options (sequences of actions) instead of having to handcraft them, and demonstrate the performance advantage in three navigation tasks.

1 Introduction

Reinforcement Learning (RL) has shown great advances in the recent years, in solving sequential decision making tasks. Particularly in the field of games, Reinforcement Learning algorithms have managed to achieve superhuman levels of performance [Mnih *et al.*, 2013] [Silver *et al.*, 2016]. In reinforcement learning, the problem of exploration vs exploitation is a persistent problem, particularly in complicated environments. It is a common practice to use ϵ-greedy strategies for exploration and that is what most algorithms use generally. However, for complicated environments such as the game Montezuma's Revenge in Atari [Mnih *et al.*, 2013], the RL agent is unable to learn anything because the state space for fully random exploration is too large.

In Reinforcement Learning, the main goal is to obtain a policy, π that maximises the expected discounted return

$$\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)]$$

from interaction with the environment in the form of states s_t, actions a_t, and rewards r_t. At the beginning, the agent will not have much idea about the environment it is in and thus needs to gather information before it can take useful actions. If the state space is very complicated and/or the reward is very sparse, obtaining useful trajectories (collection of states, actions and rewards) can be difficult. This is the problem of exploration in RL which is still quite an open area for research.

There have been many recent papers that address the problem of exploration like Noisy Networks [Fortunato *et al.*, 2017], Count-based Exploration [Tang *et al.*, 2017], Curiosity Driven Exploration [Pathak *et al.*, 2017], unsupervised learning of goal space for intrinsically motivated exploration [Péré *et al.*, 2018], Go-explore [Ecoffet *et al.*, 2019] which goes to a previously explored state and restarts exploration from that state. One interesting breakthrough for solving hard exploration problems came in the Never Give Up [Badia *et al.*, 2020b] paper, which was included in the Agent 57 [Badia *et al.*, 2020a] paper which is able to successfully solve all the 57 Atari games. This paper introduces two additional rewards for defining directed exploration, the episodic and the intrinsic reward. Agent 57 uses a family of policies with varying degree of exploration trained by Recurrent Experience Replay in Distributed Reinforcement Learning (R2D2) [Revaud *et al.*, 2019] with Retrace (λ) [Munos *et al.*, 2016] to account for off-policy learning. Most of these methods require learning some form of novelty or generating a reward function based on curiosity. None of them are as simple as the traditional ϵ-greedy approach. Recently, [Dabney *et al.*, 2020] came up with a temporally extended version of the ϵ-greedy exploration strategy that can handle complicated environments as well.

As explained in [Dabney *et al.*, 2020], the various other exploration strategies developed keeping difficult environments in mind tend to over-fit to those challenging environments and do not work as well in environments requiring lesser exploration. The authors further explain how temporally extended epsilon greedy is a sound strategy that can work well in a variety of environments. In this paper, we aim to demonstrate the following contributions:

1. We address the problem of balancing exploration and exploitation in a general sense, in contrast to prior studies that focused only on hard exploration environments.

Figure 1: High-level architecture of the DEZ-exploration policy with M GVFs, one main policy, and one uniform policy. The reward for training each GVF is computed using handcrafted functions f. The logic for choosing action a is described in Algorithm 1.

2. We extend previous exploration strategies by using auxiliary task learning with the help of General Value Functions (GVFs) to perform directed exploration, thereby further improving state space coverage during exploration. Our strategy improves on temporally extended ϵ-greedy exploration with sub-policies from the General Value Functions to perform better exploration.

3. The move to a specifically trained set of GVFs has major advantages, particularly with respect to better representation learning.

2 Background

Temporally Extended EZ-greedy Exploration: In generic ϵ-greedy exploration, the agent can either take an action that maximizes its own state-action values with probability 1-ϵ or a random action with probability ϵ. Thus given infinite time, it would explore the entirety of the state space as it would be taking random actions infinite times provided we always have a non-zero ϵ always. This form of exploration is generally very slow in covering different parts of the state space. Thus temporally extended ϵ-greedy [Dabney *et al.*, 2020] fixes a random action and keeps taking the action for a certain number of time steps, called persistence (Z) of the random action. So instead of taking one random step with probability ϵ, it executes a sequence of same random actions. The persistence Z is not a constant value throughout the exploration process but it is uniformly sampled from 0 i.e. no exploration to a maximum number of time-steps (a hyper-parameter, which would depend on the environment). This is why the algorithm is also known as EZ greedy.

In the paper, the authors mostly used the same random action repeated K times, but we can easily replace a sequence of repeated actions with a hand-crafted or learned option [Sutton *et al.*, 1999]. These options could help to reach parts of state-space that could be novel or would help in solving the main goal. In this paper, we formulate a framework to learn such options by using General Value Functions.

General Value Functions: A value function provides an estimate of the discounted total expected return that can be obtained given a certain state-action pair and a policy π. Similarly, General Value Functions (GVFs) [Sutton *et al.*, 2011] estimate a user-specified environment property, optimizing over the user-specified reward function given a policy and a discount factor. Thus, if the discounted returns be G_t and for a trajectory length T,

Relates to GVF cumulant

collected by π

$$Q^{GVF}(s, a; \pi, \gamma, r) = \mathbb{E}[G_t | s_t, a_t, A_{t+1:T-1} \sim \pi].$$

The input to the GVFs would thus be a policy, discount, reward function (also called cumulants). The 'question' asked of a GVF is the estimate of the discounted return for the specified cumulants. For each question, GVFs can be trained with normal RL algorithms (like Q-learning) and thus can provide useful information about the environment and the transition dynamics. In the Horde architecture [Sutton *et al.*, 2011], the authors show the usefulness of answering predictive questions about the environment as well as the capability of learning multiple control policies from a single behaviour policy.

The cumulants can also be viewed as providing auxiliary tasks the agent needs to perform before it can solve the main task. Often such auxiliary tasks would have no reward signal associated with them otherwise. The tasks may be definde using handcrafted cumulants [Jaderberg *et al.*, 2016] or they may themselves be learnt on their own [Veeriah *et al.*, 2019]. In this paper, we define the GVF reward functions and termination functions by hand, and learn the GVF values off-policy i.e. they are learned with respect to their own greedy policies. Thus, maximising for Q^{GVF} generates a different sub-task oriented policy which can be viewed as an option, to be used for a short persistence period. Thus, by following the policy for a short period we can generate useful options.

3 Directed EZ Greedy Algorithm

Directed EZ Greedy learns auxiliary tasks in the form of GVFs and uses sub-policies obtained by greedy action selection from the General Value Functions learned as options for temporally extended ϵ-greedy algorithm. With probability ϵ it selects a random option which can be sub-policies from any of the randomly chosen GVFs at that particular state and keeps taking the greedy action with respect to that GVF. Thus, it explores different parts of the state space by following a specific GVF policy which makes the exploration directed, hence the name Directed EZ Greedy. The entire algorithm is portrayed in Algorithm 1.

The algorithm learns M+1 Value Functions as shown in Figure 1 where M is the number of GVFs. These Value Functions are learned from a shared common representation through which the losses for both the main Q values and the GVFs propagate. In Algorithm 1, we sample an action from M+1 ([0, M]) possible options. In all our experiments, we include the option of repeating a random action as one additional option as well. This is added to make sure that theoretically, it is still possible to visit all states in the state space and we have asymptotic full state space coverage.

There are 3 main advantages of this approach:

1. There is no requirement for handcrafted options, rather DEZ-greedy can take sub-polices defined by the GVFs. Thus the input to generate the options can be as simple is

Algorithm 1: DEZ-greedy exploration strategy

```
Function DEZGreedy(ϵ, Z_max):
    Countdown timer z ← 0
    Uniformly sampled random action w ← −1
    Selected GVF index g ← 0
    while True do
        Observe state s
        if z == 0 then
            // Timer ran out: choose controlling policy
            if random() < ϵ then
                // Explore
                Sample duration: z ∼ [1, Z_max]   → persistence
                Sample GVF: g ∼ [0, M]   → policy
                if g == 0 then
                    Sample action: w ← U(A)
                    a ← w   uniform
                else
                    a ← argmax(Q_g^GVF)
            else
                // Exploit
                a ← argmax(Q^Main)
        else
            // Continue previous exploration policy
            if g == 0 then
                a ← w
            else
                a ← argmax(Q_g^GVF)
            z ← z − 1
        Take Action a
```

defining a reward and a termination function as opposed to a sequence of actions.

2. Normally GVFs have been used to give predictive knowledge of the environment and this directly helps in learning good representations. In Figure 1 as well, we employ a similar structure whereby the GVFs not only influence the exploration policies but also help in learning better representations.

3. Learning GVFs while following the behavior policy leads to divergence if we do not use importance sampling ratios as the greedy policy of the GVFs will be off-policy. However following the GVF policy initially helps to keep the behavior policy closer to the greedy GVF policy (at least during the exploration phase). Following that even if the GVFs diverge, we do not use it explicitly because we anneal ϵ gradually. This reduces the impact of off-policy divergence.

4 Environments

Most of our experiments were tested in the gridworld setting with hand-crafted cumulants. We mainly considered following environments:

- **Two Rooms Environment:** Here, the goal of the agent is to reach the adjacent room by passing through a corridor. As shown in Figure 2, the agent spawns in the bottom left corner and it has to find its way to the green dot on the top right (the red dot is not relevant in this environment). This seems like an easy task, but for larger dimensions it becomes difficult as the agent has to explore for a long time before it reaches the goal. The agent receives the (x,y) coordinates of its position as its state. The agent gets a reward of +1 on reaching the green dot and a step reward of -0.01 with maximum time step of 300 after which the episode ends. For this environment, we used only one GVF which gets a cumulant of 1 for crossing the corridor.

Figure 2: Two Rooms & SubGoal Two Rooms Environment. The red dot (SubGoal) is represented by R, and the main goal (for both environments) is the green dot shown here as G.

- **SubGoal Two Rooms Environment:** This is a modification of the Two Rooms Environment with similar dynamics, except the agent has to collect the red dot and then go to the green dot. The reward for reaching the green dot without collecting the red dot is +0.1, whereas it is +1 after collecting the red dot. The other parameters are exactly the same. For this environment, we append a boolean value of whether the agent has visited the red flag to the (x,y) coordinates to make the states Markov. In this environment, we used one more GVF in addition to the one used in Two Rooms, which receives a cumulant of +1 if the agent visits the red dot.

- **Doorkey Environment:** This is popular environment for benchmarking RL algorithms [Chevalier-Boisvert *et al.*, 2018], with different tasks. For our experimentations we chose Doorkey environment with 6×6 and 8×8 grid-size. Board configuration is very similar to Two Rooms environment however, position of corridor and initial position of agent are random. In addition, to enter another room, agent needs to pickup a key, unlock and open the door. Additionally, the actions for going left and right are a bit different than just going left or right. The agent has to first turn to the direction it needs to move in and then go forward. This can be really problematic for EZ-greedy methods as action repetition would get nowhere and the agent would basically keep on rotating in the same state. Thus, to make the comparison fair, we included options for going left and right for these envi-

Figure 3: Different Algorithms compared on 19x19 TwoRooms and 11x11 SubGoal Two Rooms Environment across 10 runs ($Z_{max} = 30$)

(a) (b)

Figure 4: (a) Parameter Sensitivity of EZ-greedy and DEZ-greedy strategies with respect to the persistence, Z_{max} in SubGoal Two Rooms Environment (b) Comparison of Performances of epsilon greedy and directed epsilon greedy ($Z_{max} = 30$) for various grid sizes of SubGoal Two Rooms Environment

ronments. Additionally, for this environment, we used 2 GVFs. The first GVF gets a cumulant of +1 for picking up the key and the next GVF, gets a cumulant of +1 for unlocking the door.

5 Results

In this section, we will compare our approach with ϵ-greedy strategy that is widely used as well as the temporally extended ϵ-greedy strategy [Dabney *et al.*, 2020]. The detailed hyperparameters for reproduction of the results are listed in the Appendix B which is in the supplementary materials provided. In addition, upon acceptance, we would make our code public.

Algorithms compared: For our experiments, we compare 5 different algorithms.

- The RL algorithm used for our experiments is the DQN algorithm [Mnih *et al.*, 2013], i.e. all the GVFs and the main Q values are learned using DQN. The baseline to compare against would thus be DQN. DQN uses ϵ-greedy exploration with annealing ϵ and that is what we used for this experiment as well. Details about the variants of DQN used for each of the environments are presented in Appendix B.
- EZ-DQN is the temporally extended ϵ-greedy version of DQN with EZ-greedy exploration. The version with repetitive actions is used Two Rooms and SubGoal Two Rooms Environments. But for DoorKey, we used options for going left, right and up instead of repetitive actions. EZ-DQN with repetitive actions for DoorKey is denoted by EZ-DQN(Vanilla).
- DQN+GVF is the algorithm that uses GVFs only for representation learning via the common representation which is modified by both the main DQN agent as well as all the GVFs. This still uses ϵ-greedy exploration like DQN.
- EZ-DQN+GVF uses the exact architecture as above except the behaviour policy uses EZ-greedy exploration strategy.

Figure 5: Training plot for DoorKey environment for 5 seed values ($Z_{max} = 5$)

Figure 6: DEZ greedy performance for different persistence values on Doorkey environment. The Y-axis on the left represents the average reward across the last 2k games while the axis on the left displays the number of episodes DEZ-DQN+GVF took to reach 90% of the final convergence value of the reward. Error bar denote standard error across 5 random seeds

- DEZ-DQN+GVF is our algorithm that uses GVFs not only for learning better representations but also for learning options which can help in directed exploration.

Note on Baselines: For our paper, we did not compare any state of the art exploratory methods as they were the basis of comparison for [Dabney *et al.*, 2020]. Thus we only used their baseline in addition to epsilon greedy, which is the widely used exploration strategy. We ideally want to ask, what would be a good exploratory strategy that can work well across a bunch of different kinds of environments (both simple and challenging).

5.1 Two Rooms & SubGoal Two Rooms

The Figure 3 portrays the learning curves of the algorithms across 10 runs over 2000 episodes. The shaded regions are the standard errors across the 10 seeds for all the training curves. the The Z_{max} for the temporally extended versions of exploration are set to 30.

From Figure 3 it is evident that EZ-greedy helps a lot in exploring the state space much much quickly and there is a marked difference between algorithms that use it and normal ϵ-greedy. However, adding directed exploration with the use of GVFs makes the RL agent learn even faster because of exploration of useful state-spaces as the GVFs are designed to stimulate goal-driven exploratory behaviour. Another interesting thing worth noting is that even adding GVFs without using any exploratory strategy helps in better performance since the DQN+GVF (blue) is slightly better than DQN (red). Even in the case of using EZ-greedy with GVFs there is an improvement in performance in the SubGoal Two Rooms environment. This is likely due to GVFs learning useful representations which can improve performance.

Sensitivity with respect to Persistence Z_{max}: The performance of the algorithm will be determined by the persistence values chosen. For example, a persistence value of 1 for EZ exploration reduces to ϵ greedy exploration. In Figure 4 (a), we plot the sensitivity of the algorithms across different persistence values. For smaller values the performance degrades due to less exploration, however, for directed exploration, the drop in performance is significantly less as even with smaller persistence values the agent can construct meaningful options which makes DEZ-greedy much more robust.

Robustness to difficulty of exploration: One main aspect that we wanted to target is better performance for both simple and hard domains. In Figure 4 (b) we plot the mean average

reward for ϵ-greedy and DEZ-greedy method. With smaller grid size, ϵ-greedy does really well but it fails to work for higher grid sizes. However, DEZ-greeedy not only works for the higher grid sizes, but also beats ϵ-greedy in smaller domains as well. This is an interesting result because even it shows how this strategy is robust to the hardness of the environment with respect to exploration. Thus directed EZ Greedy can be used universally for all types of environments.

5.2 DoorKey

In Figure 5 we present training results for 2 grid sizes of the Doorkey environment from Gym MiniGrid [Chevalier-Boisvert *et al.*, 2018]. For 6×6 we can observe DEZ-DQN+GVF converging early significantly compared to other algorithms. We also note that for smaller grid size there is minor variations across random seeds for all the algorithms as denoted by shaded region. For 8×8 size, we see higher variance across random seeds for EZ-DQN+GVF and DQN+GVF and certain seeds fail to converge. DEZ-DQN+GVF has relatively low variance and shows similar trend to 6×6 providing evidence to robustness across grid-sizes, as observed earlier.

It is also interesting to note the decrease in performance for the GVF variants of both DQN and EZDQN for 8×8 DoorKey. We believe this is due to off-policy divergence that can happen when the greedy policy with respect to each of the GVFs is very different from the behaviour policy of the main RL agent. This phenomenon discourages use of GVFs for representation learning particularly for Deep RL methods, because the GVFs values diverge resulting in degradation in performance. However, since our method periodically take actions from the GVFs, it helps in avoiding diverging values. This off-policy behaviour is explored more along with useful heatmaps of Q values in Appendix A.

Asymptotic Performance: Addition of GVFs with different tasks that influences representation can often lead to sub-optimal behaviour. But since we use the GVFs primarily for exploration and also have reduced off-policy divergence, we empirically show that the performance even for higher values of persistence in a small environment is not bad. Consistent with the theme in the SubGoal Two Room Environments, Figure 6 portrays the performance of DEZ-DQN+GVF on 6×6 and 8×8 DoorKey with respect to increasing Z_{max}. The spread of values across the left Y-axis (blue) is very minimal for both the plots. The only thing that changes is the time to get to 90% of convergence value of the reward. This suggests that the directed EZ greedy method explores more in the beginning due to higher Z_{max}, but ultimately settles down to the optimal policy. Even if we increase the value of Z_{max} to something much greater than what the environment calls for, the performance does not dip greatly.

Quantifying Exploration: Having a higher reward is one way of ascertaining that the agent has explored important parts of the state space faster and as a result is able to converge much faster. However, a direct comparison of the exploration strategies can be made by using visitation frequency of unique states. Figure 7 presents a comparison of various exploration strategies directly by plotting the farthest distance

Figure 7: This plot tries to quantify exploration by comparing distance and unique states visited by normal epsilon greedy and directed epsilon greedy method. The graph is plotted with aggregating values for 100 episodes.

from the initial state and the number of unique states visited. Distance from the initial state is also quantification of exploration for this environment as the distance would increase when the agent can unlock the door and visit the next room. We can observe that although initially, EZ-DQN is visiting more unique states, it fails to get to reach farther from the initial state. Hence directed EZ greedy streamlines this by visiting more unique yet far away states. As expected, only repetitive actions do not work well in this environment, evident by the poor exploration of Vanilla EZ-DQN.

6 Conclusion and Future Work

Exploration is one of the critical aspects for good performance for Reinforcement Learning, especially in games. EZ-greedy [Dabney *et al.*, 2020] has been shown to perform better overall in a variety of Atari games compared to curiosity driven methods which specialize in exploration for some specific games. DEZ-greedy incorporates aspects of both EZ-greedy and auxiliary task learning along with learning richer representations and is more robust as well. Thus, it should work well in a variety of games.

From the experiments, Directed EZ greedy seems to learn much much faster because of a better exploration strategy along with improved representation learning. Intuitively as well, this makes sense as options generated from the GVFs take the agent into states which are useful for optimizing towards the main goal. Additionally, this architecture is very flexible as we only require a scalar reward to learn the GVFs.

The next steps would be to try to adapt it to environments which can learn the cumulants automatically from the environments [Veeriah *et al.*, 2019]. Additionally, DEZ-greedy can be combined with any intrinsic motivation based exploration strategy, as well, if they add exploratory reward functions to the main reward. This along with goal-driven GVFs would further improve exploration for complicated environments. It can also be used for transfer learning in procedurally generated environments as the GVFs learned can help in generating useful options for newer tasks as well.

References

[Badia *et al.*, 2020a] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark, 2020.

[Badia *et al.*, 2020b] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies, 2020.

[Chevalier-Boisvert *et al.*, 2018] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[Dabney *et al.*, 2020] Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended ϵ-greedy exploration. *CoRR*, abs/2006.01782, 2020.

[Ecoffet *et al.*, 2019] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019.

[Fortunato *et al.*, 2017] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017.

[Jaderberg *et al.*, 2016] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks, 2016.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[Munos *et al.*, 2016] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning, 2016.

[Pathak *et al.*, 2017] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017.

[Péré *et al.*, 2018] Alexandre Péré, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. *CoRR*, abs/1803.00781, 2018.

[Revaud *et al.*, 2019] Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2d2: Repeatable and reliable detector and descriptor, 2019.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[Sutton *et al.*, 2011] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, page 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[Tang *et al.*, 2017] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Exploration: A study of count-based exploration for deep reinforcement learning, 2017.

[Veeriah *et al.*, 2019] Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Richard Lewis, Janarthanan Rajendran, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks, 2019.

A Appendix A

One aspect that is crucial while using GVFs is off-policy learning. Learning GVFs on-policy with respect to the behaviour policy might not yield as useful options (sub-policies) as learning them off-policy. In the DEZ-greedy exploration strategy since we sample actions greedily from the GVFs often, the resulting behaviour policy is closer to the greedy policy of the GVFs. This is particularly true in our case as most of our GVFs are goal-driven. We find that EZ-DQN+GVF second best performing algorithm has divergent Q values whereas DEZ-DQN+GVF are bounded

Figure 8 shows us the Q values for each of the individual actions for the GVF that gets a reward of +1 on collecting the red dot in the SubGoal Two Rooms Environment. It is evident that for EZ-DQN+GVF, the best performing algorithm apart from ours, has divergent Q values, since we never sample any actions greedily from the GVFs, the GVFs diverge to nonsensical values. However, the values for DEZ-DQN+GVF are much more bounded. In addition, following the optimal policy of this GVF will lead the agent into places close to the red dot thereby increasing the chances of picking up the red dot during exploration.

B Appendix B

All the necessary hyper-parameters required for the necessary reproduction of the experiments are mentioned in the following table.

Note on DQN variants: For the Two Rooms and SubGoal Two Rooms environment, we used a vanilla DQN with target networks and experience replay. The DQN variants used in DoorKey have an additional soft updating parameter, τ which basically keeps the target network from changing too much. At every target network update, we perform a weighted update of the target network parameters, with the weighting on the current train network weights being τ.

Also, for the MiniGrid environments, the weights were not updated at every time step, but after every 10 time-steps, which explains the high batch size used in these domains. These all modifications were used to get DQN to work properly for these environments, in an acceptable number of time-steps, making the use of as much parallel compute as possible.

(a) EZ-DQN+GVF

(b) DEZ-DQN+GVF

Figure 8: Off-policy divergence with generic GVF algorithms for 11x11 SubGoal Two Rooms. The table shows Q values of the GVF that gets a reward of +1 on collecting the red dot. The values are much more bounded for DEZ-greedy exploration.

Table 1: Hyper-Parameters

Environment	Algorithm	Learning Parameters	Model Parameters
Two Rooms	DQN DQN+GVF EZDQN EZDQN+GVF DEZDQN+GVF	runs = 10 episodes = 2000 batch = 64 γ = 0.99 α = 0.001 ϵ_{start} = 1.0 ϵ_{stop} = 0.001 ϵ_{decay} = 1e-3**(1/episodes) Persistence, Z_{max} = 30 GVFs = 1	Architecture = [16 units, 16 units] Replay Buffer Size = 10000 Target Network Update = 100 steps
Sub Goal Two Rooms	DQN DQN+GVF EZDQN EZDQN+GVF DEZDQN+GVF	runs = 10 episodes = 2000 batch = 64 γ = 0.99 α = 0.001 ϵ_{start} = 1.0 ϵ_{stop} = 0.001 ϵ_{decay} = 1e-3**(1/episodes) Persistence, Z_{max} = 30 GVFs = 2	Architecture = [16 units, 16 units] Replay Buffer Size = 10000 Target Network Update = 100 steps
6x6 DoorKey MiniGrid	DQN DQN+GVF EZDQN EZDQN+GVF DEZDQN+GVF	runs = 5 episodes = 10000 batch = 2048 γ = 0.95 α = 0.0001 τ = 0.05 ϵ_{start} = 1.0 ϵ_{stop} = 0.01 ϵ_{decay} = (ϵ_{stop} ** (ϵ_{start} / (episodes * 0.9))) Persistence, Z_{max} = 5 GVFs = 2	Architecture: Conv1(C=16,F=2); MaxPool1(F=2,S=2); 3 FC layer [60,60,6] Replay Buffer Size = 50000 Target Network Update = 1000 steps
8x8 DoorKey MiniGrid	DQN DQN+GVF EZDQN EZDQN+GVF DEZDQN+GVF	runs = 10 episodes = 30000 batch = 4096 γ = 0.95 τ = 0.05 ϵ_{start} = 0.9 ϵ_{end} = 0.01 $\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \times \exp(-t/10^3)$ Persistence, Z_{max} = 5 GVFs = 2	Architecture: Conv1(C=16,F=2); MaxPool1(F=2,S=1); Conv2(C=16,F=2); MaxPool2(F=2,S=1); 4 FC layer [120,60,10,6] Replay Buffer Size = 60000 Target Network Update = 5 episode