Q-MAP: A CONVOLUTIONAL APPROACH FOR GOAL-ORIENTED REINFORCEMENT LEARNING ICLR 2019 REPRODUCIBILITY CHALLENGE

Shishir Sharma, Raviteja Chunduru, Surya Penmetsa
School of Computer Science
McGill University
Montreal, Canada
{shishir.sharma,raviteja.chunduru,surya.penmetsa}@mail.mcgill.ca

ABSTRACT

We have chosen to reproduce the results of the paper *Q-map: a Convolutional Approach for Goal-Oriented Reinforcement Learning* from ICLR 2019 submissions as part of the Reproducibility challenge. The paper is motivated by the fact that current exploration algorithms in Reinforcement Learning (RL) such as ϵ-greedy, take a completely random action at any time with ϵ probability and rely on random walks to discover new transitions. This approach for exploration is not ideal and frequently produces poor transitions. The main idea of the paper is an exploratory algorithm which replaces the random actions in ϵ-greedy by random goals. The algorithm is proposed for spatial environments where on screen locations are potential goals and are represented with their spatial coordinates. The paper proposes a convolutional auto-encoder-like neural network architecture called Q-map which simultaneously represents the General Value Functions (GVF) for all the goals. This allows to efficiently share weights and to identify correlations between visual patterns such as walls, enemies or ladders and patterns in the distance towards on-screen locations.

1 INTRODUCTION

Mnih et al. (2015) demonstrated that by approximating the optimal action-value function using a deep convolutional neural network, their proposed Deep Q-Network (DQN) agent was able to achieve the performance level of a professional human games tester on many Atari 2600 platform games. While the agent uses the deep convolutional network for exploiting, it uses ϵ-greedy algorithm to explore the environment. The ϵ-greedy algorithm relies on a probability ϵ to take a random action at any time instead of the greedy action determined by the agent. The parameter ϵ is usually decayed linearly to ensure a smooth transition from complete exploration to almost complete exploitation. A major issue with this approach is that using random walks to discover new transitions can generate many repetitive and unnecessary or even destructive transitions, resulting in poor exploration and thus the agent being unable to discover the reward signal required for training.

1.1 GOAL ORIENTED REINFORCEMENT LEARNING

In a typical RL setting, an agent attempts to learn to solve a particular task by striving to maximize the corresponding reward signal. The concept of goal-oriented learning extends this by specifying goals that the agent tries to achieve. While the reward signal is considered sufficient to describe any task in the environment, providing the agent with the task description in the form of specific goals to reach is often found to be advantageous. The concept of goal-oriented RL is formalized by General Value Functions (GVFs) which create a specific value function per goal. Thus, instead of employing action-value functions $Q^{\pi}(s, a)$ specific to the rewards defining the task, the GVFs introduce $Q^{\pi}_{g}(s, a)$ action-value functions specific to goals. For such an RL paradigm, efficiently reusing past experience to update estimates towards several goals at once becomes desirable but usually requires independent updates per goal. Another challenge in such goal-parameterized policies

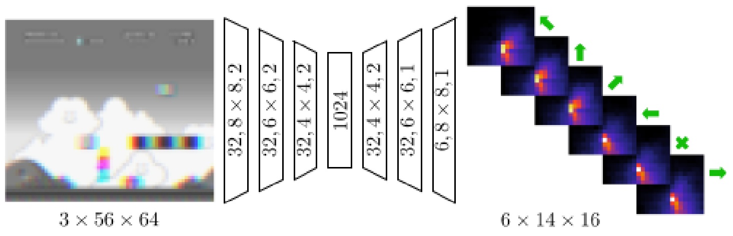

Figure 1: Q-map neural network architecture. A stack of greyscale frames as input, with 3 convolutions, 1 hidden layer and 3 transposed convolutions to output a stack of Q-maps, one for every possible action choice. Figure from Pardo et al. (2018)

is to make the best use of the data collected so far, to be able to generalize well to goals that were not necessarily selected when the data was collected. This motivates the proposed goal-oriented agent called Q-map, that utilizes an autoencoder-like neural network to predict the minimum number of steps towards each goal in a single forward pass. The goals for the agent are represented by spatial coordinates such as on-screen location of the character in ATARI or SNES games.

2 Q-MAP

The standard RL framework, formalizing the sequential interactions of the agent with its environment as a Markov Decision Process (MDP) with state space S, action space A, reward function $r(s, a, s')$ and state-transition function $p(s'|s, a)$ is employed. At each time step t, the agent provides an action at based on the current state s_t and sampled from its policy $\pi(a_t|s_t)$. The environment responds by providing a new state s_{t+1} and a reward r_{t+1}. For a terminal state, no more interaction is possible after reaching it. Thus, it can be simply considered as a deadlock state that only transitions to itself, providing no reward. The action-value function of the policy is represented as

$$Q^{\pi}(s, a) = E_{s' \sim p(s'|s,a), a' \sim \pi(a'|s')} r(s, a, s') + \gamma Q^{\pi}(s', a') \tag{1}$$

The Q-value function indicates the quality of each possible immediate action when further actions are taken in accordance with policy π. Using the Q-learning algorithm Watkins & Dayan (1992) the action-value function of the optimal policy π^* is iteratively approximated by updating the estimated Q-values as :

$$Q_{t+1}(s, a) \leftarrow (1 - \alpha)Q_t(s, a) + \alpha(r + \gamma \arg\max_{a'} Q_t(s', a')) \tag{2}$$

using previously experienced transitions (s, a, s', r) and a learning rate α.

As the target $r + \gamma \arg\max_{a'} Q_t(s', a')$ does not rely on the policy used to generate the data, Q-learning is able to learn off-policy efficiently by re-using previous transitions stored in a replay buffer. Such kind of experience replay lets online RL agents remember and reuse experiences from the past. Also, instead of uniformly sampling experience transitions from the replay memory, Schaul et al. (2015) demonstrate how the experiences can be prioritized, so as to replay important transitions more frequently, and therefore learn more efficiently.

2.1 PROPOSED AGENT

Pardo et al. (2019) propose an agent called Q-map which has a convolutional auto-encoder-like neural network enabling the agent to simultaneously represent the GVFs of all possible on-screen locations for all possible actions from the raw screen pixels. This allows for efficient weights sharing and identification of correlations between visual patterns such as walls, enemies or ladders and patterns in the distance towards on-screen locations. Thus once the agent has learned to perform a specific task, for example how to navigate ladders, this knowledge can be transferred to other ladders later in a game.

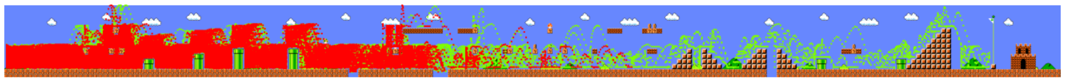

Figure 2: Comparison of the exploration of the agent when using a DQN with ε-greedy (red) and when using the proposed exploration strategy (green). This result is from the original paper. We see that with the proposed exploration strategy, the agent can explore more of the level, especially the later stages. Figure from Pardo et al. (2019)

2.2 METHODOLOGY

The Q-map takes a stack of greyscale game frames as input and produces a stack of 2D Q-frames as output. The agent extracts a notion of the distance γ^{k-1} between the current state and the desired goal in terms of the number of actions. A single voxel in the output stack of Q-frames at column x, row y and depth a represents this distance from the current state to the on-screen coordinates (x, y) given an action a. A great advantage of the Q-map architecture is that it allows one to query the estimated distance towards all goals in a single pass through the network and to efficiently create a learning target from these estimates with just an additional pass using a 3D version of the Q-learning algorithm.

Given a transition (s, a, s', x', y'), the value $Q(s, a, g)$ for any coordinates g can be updated using a variant of Q-learning, with the following target:

$$y = \begin{cases} 1, & \text{if } (x', y') = g \\ 0, & \text{else if } s' \text{ is terminal} \\ \gamma \arg\max_{a'} Q_t(s', a'), & otherwise \end{cases}\Bigg\}$$

Where γ is a discount factor used for the Q-map, not necessarily the same as the one discounting the rewards from the environment. The pseudo-code for the training algorithm is given below and the training methodology is also detailed in figure 3.

Algorithm 1 Training a Q-map network Q

```
for iteration = 1, 2, ... do
    Sample a batch $b$ of transitions from a replay buffer B
    for each transition $(s, a, s', x', y'$ , term$)^{(i)}$ in $b$ do
        if term = True then
            $Q^{(i)} \leftarrow 0$
        else
            $Q^{(i)} \leftarrow \gamma$ clip($\arg\max_a Q(s')[:, :, a']$, 0, 1)
            $Q^{(i)}[y', x'] \leftarrow 1$
        end if
    end for
    Update $Q$ to reduce $\sum_i ||Q^{(i)} - \mathcal{Q}(s^{(i)})[:, :, a^{(i)}]||_2^2$
end for
```

2.3 Q-MAP + DQN AGENT

To contrast the proposed Q-map exploration strategy with ϵ-greedy, the authors propose an agent that uses Q-map to explore and a Deep Q-Network by Mnih et al. (2015) with double Q-learning van Hasselt et al. (2015) and dueling to exploit. At each time step, the action selection follows the decision tree:

- With probability ϵ_r a completely random action can be selected to ensure pure exploration for Q-map and DQN.
- If no random action has been selected and no goal is currently chosen, a new one is chosen with probability ϵ_g

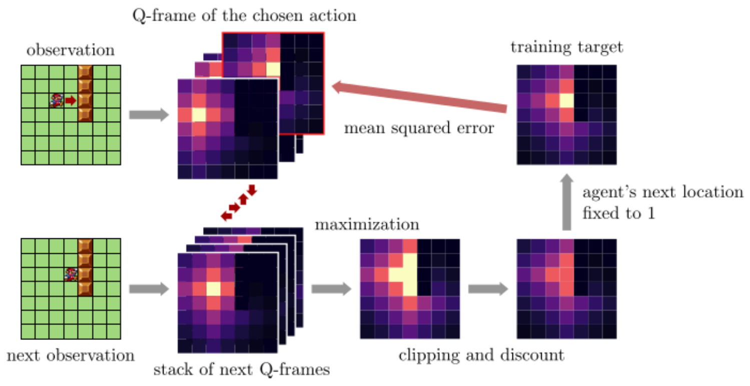

Figure 3: Training process for the Q-map agent. Figure from Pardo et al. (2019)

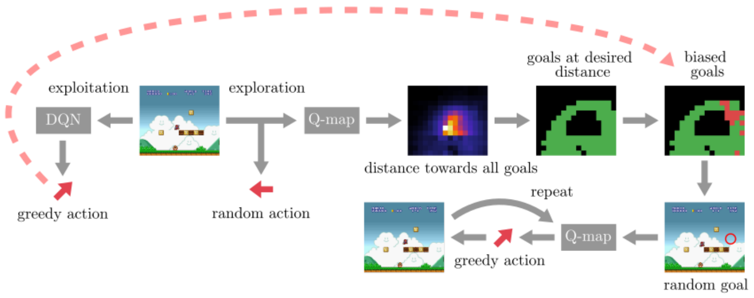

Figure 4: The Decision tree employed by the Q-MAP + DQN agent. Figure from Pardo et al. (2019)

- To choose a new goal, first the Q-map is queried for the predicted distance towards all the goals, and the goals are filtered to remove the ones that are too close or too far away (based on two hyper parameters specifying the minimum and maximum number of steps). Out of the remaining goals, we consider the goals for which the first action we need to take to reach them is the same as the action predicted by the DQN for that state. We choose the new goal randomly from this set of goals. A time limit to reach the chosen goal is set based on the predicted distance to this goal.
- If a goal is currently chosen, the Q-map is queried to take a greedy action in the direction of the goal and the time allotted to reach it is reduced.
- Finally, if no random action or goal has been chosen, DQN is queried to take a greedy action with respect to the task.

This decision tree is depicted in figure 4. As the number of steps spent following goals is not accurately predictable, ensuring that the average proportion of exploratory steps i.e. random and goal-oriented actions, follow a scheduled proportion ϵ is difficult. As an approximation, the probability ϵ_g of selecting new goals is dynamically adjusted by using a running average of the proportion of exploratory steps $\tilde{\epsilon}$ and increasing or decreasing ϵ_g to ensure that $\tilde{\epsilon}$ approximately matches ϵ.

2.4 Environment

The paper presents the results of Q-MAP + DQN agent on Montezumas Revenge and Super Mario All-Stars environment from OpenAI Gym. We have chosen to experiment on the Super Mario All-Stars environment and have ensured that the environment used is exactly the same as used by the authors in order to be able to compare the reproduced results.

3 RESULTS

In this section, we provide the results we have obtained from our experiments with the methods presented in the paper, and we compare these results with those presented in the original conference submission. We further provide a brief analysis of these results.

3.1 REPRODUCING RESULTS FROM THE ORIGINAL PAPER

First we investigated the claims made by the paper in regards to the performance of the proposed agent and the comparison with the baseline on level 1.1 of Super Mario environment. Thus, we tried to verify that the proposed method which employs a Q-map for exploration and a DQN for exploitation performs better than the reported baseline which uses a DQN with ε-greedy exploration. After allowing each approach to run for 5 million timesteps, we plot the sum of the rewards on the y-axis as a function of timesteps on the x-axis. Even with GPU acceleration, the proposed method took approximately 3.5 days and the baseline took approximately 2.5 days to reach 5 million timesteps . We did not test how often the flag at the end of the level was reached. The results claimed by the paper are shown in figure 5 and the results we obtained are shown in figure 6.

We obtain different results than what are presented in the paper. From our experiments, the baseline DQN with ε-greedy seems to do better than the proposed method. However, unlike the paper, we were unable to perform multiple runs with different seeds and average the results since one run takes a collective time of 6 days to finish execution. The presented results are consistent across our implementation as well as the authors'. Our codebase can be found here[1]

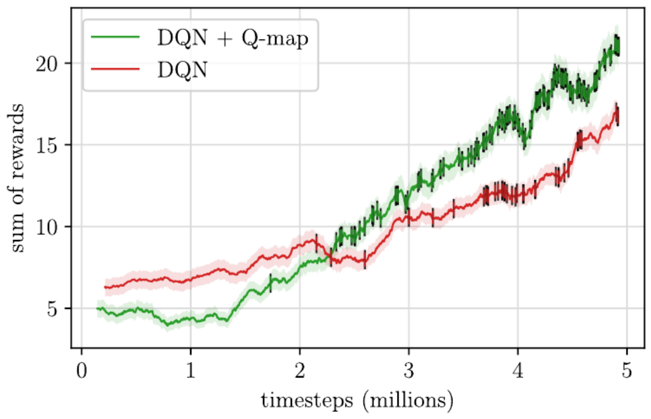

Figure 5: Comparison of DQN + Q-map with DQN. These results are from the original paper. The vertical lines represent Mario reaching the flag at the end of the level. Figure from Pardo et al. (2019)

3.2 EVALUATING GENERALIZATION ACROSS UNSEEN LEVELS

In the original paper, the authors claim that the Q-map generalizes well to unseen levels, and show results for this only for a simple grid world environment and not on the more complex Mario environment. In this section, we analyze how well Q-maps generalize to unseen levels in Mario. We also analyze the performance of the proposed algorithm on completely new and unseen levels.

3.2.1 GENERALIZATION OF Q-MAP

The authors claim that Q-map generalizes well to unseen levels as shown in figure 7 from the original paper. After running the proposed algorithm for 5 million timesteps on level 1.1 of Super Mario All-Stars and testing it on different levels, the Q-map generalization across different levels are summarized in figure 8.

[1]`https://github.com/shishir13sharma/QMAP_ICLR_reproducibility_2019`

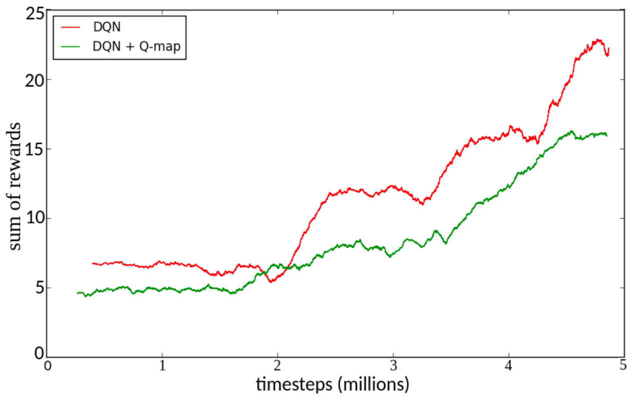

Figure 6: Comparison of DQN + Q-map with DQN. These are the results we obtained.

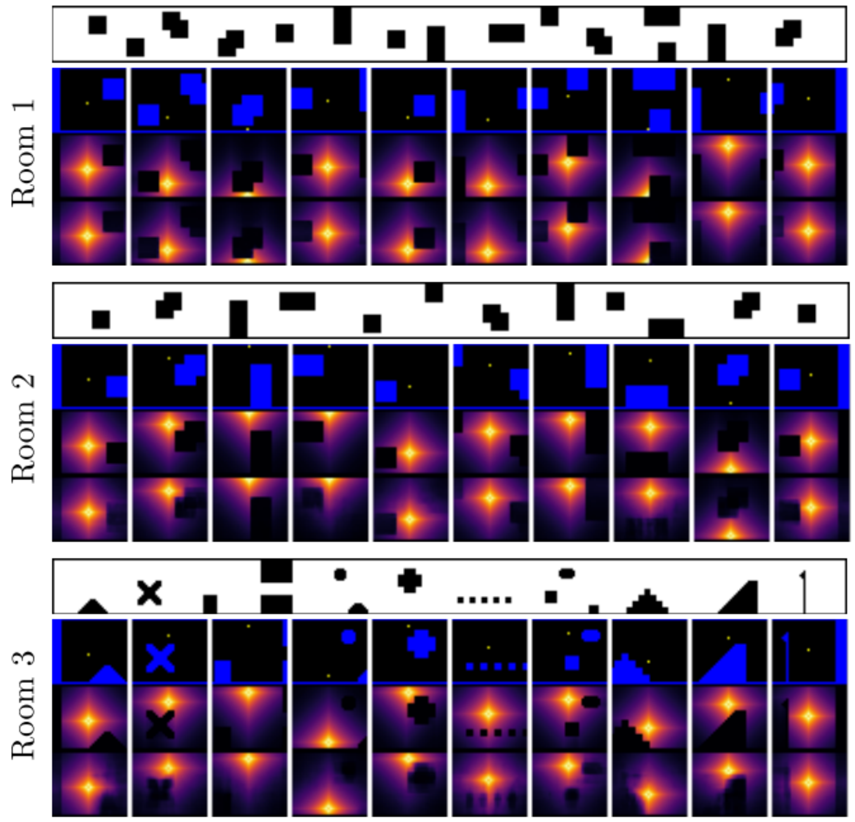

Figure 7: Q-map generalization results from the original paper. For each room, there are four rows which correspond to: room layout, observations given to the agents, the ground truth Q-frames, and the predicted Q-frames. Although only room 1 was used for training, the predictions for room 2 and room 3 are also accurate. Figure from Pardo et al. (2019)

As seen in figure 8, levels which are similar to level 1.1 have accurate Q-map predictions, but for levels which have a different colored background or have a different backdrop, the Q-map predictions are somewhat poor.

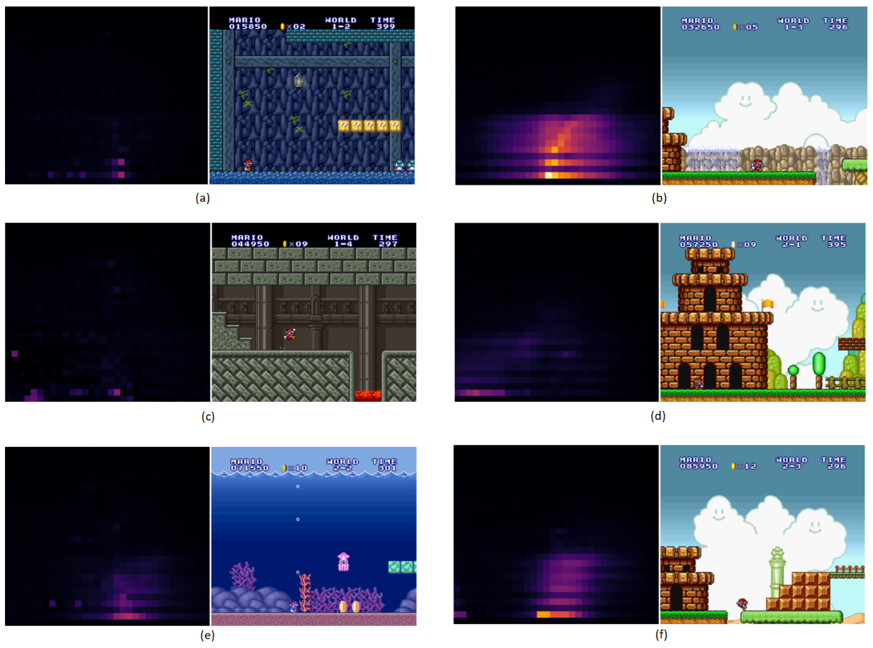

Figure 8: Q-map generalization results on different states in different levels of Super Mario All-Stars. On the right of each sub-figure, the state is shown and on the left, the predicted Q-frame is shown (a) Q-frame for level 1.2, (b) Q-frame for level 1.3, (c) Q-frame for level 1.4, (d) Q-frame for level 2.1, (e) Q-frame for level 2.2, (f) Q-frame for level 2.3

3.2.2 Generalization of proposed method

Although the authors discuss Q-map generalizing to unseen levels, the ability of the Q-map and DQN agent to generalize as a whole to unseen levels is not investigated. With Q-map being an improved approach for exploration, we expect the DQN to have learned enough about the environment so as to be able to generalize. We explored the performance of the proposed method on unseen levels with the agent only exploiting using the trained DQN. However, since the levels are quite different from level 1.1 on which the agent was trained, the performance on other levels is understandably bad and in pure exploitation mode, the agent does not perform too well on unseen levels. Often, Mario fails to get past even the first monster or jump. The bad generalization to the unseen levels may be explained by the color scheme of the level, or due to the change in how the obstacles looked. For example, level 1.2 contains a blue background while level 2.2 has brown plants in place of green pipes, in addition to a blue background because the level is underwater. Level 1.3 has inclined jumps instead of flat level jumps as in level 1.1. Thus, although the agent learns to navigate flat jumps and jump onto elevate platforms to break bricks in level 1.1, it does not seem to be able to put the two together and navigate elevated jumps in level 1.3. While purely exploiting in level 1.3, the agent fails at the first elevated jump.

Even on level 1.1, on which the agent was trained, the reward is quite low. Figure 9 is a snapshot of the agent executing in level 1.1 in pure exploitation. Note that this is after Mario has executed actions for 5 million timesteps on the same level. At the above point in the level, Mario gets stuck against the pipe because the DQN continuously tells him to move towards the right, which does not get him anywhere and does not change his state. Thus, since this is pure exploitation, Mario is stuck in this state till the timer runs out, and he no longer moves forward. In the training phase, when there was an exploration component, the Q-map provided the information that Mario should move up from this position to achieve goals on the right. However, since we do not use the Q-map in the pure exploitation phase, Mario is stuck in this state and does not gain any more reward. Hence the poor performance on the level even on which the agent spent so much time on.

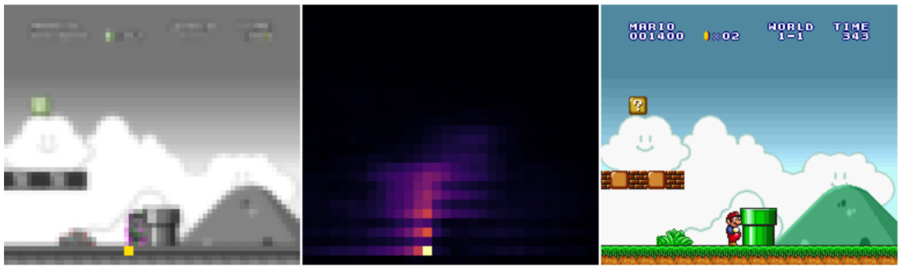

Figure 9: When exploiting, the agent gets stuck in level 1.1 because we use only the DQN. As we can see, the Q-map tells the agent to move up, but since we don't use the Q-map when exploiting, this information is ignored.

A trained Q-map gives us extremely valuable information, which instead of discarding, should be integrated into the decision making process. Similar to how biased goals were chosen in the exploration phase to ensure that the goals were sensible and reduced the 'cancelling-out' effect, the information from the Q-map should be utilized in the exploitation phase to prevent the agent from getting stuck when the DQN keeps telling the agent to execute the same action repeatedly without any change in the state.

Also, even when the agent is in the training phase and is still exploring, as the exploration rate decreases, it may be better to explore more using the Q-map, rather than exploring with a random action. This is because a random action could be perilous to the agent and could cause it to terminate, while the Q-map will more likely lead the agent to a safer path, while simultaneously forcing it to explore.

4 Conclusion

The original paper proposes a new Q-map based method for exploration which the authors claim perform better than using just ε-greedy. From the results using the authors' code as well as our implementation, we could not verify this. The results we have obtained do not match the stated results in the original paper, as evidenced by figure 2 and figure 3 above. Additionally, we conclude that when exploiting, it might be beneficial to include the valuable information being provided by the Q-map into the decision making process and use the Q-map along with the DQN to choose the next action to execute.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Fabio Pardo, Vitaly Levdik, and Petar Kormushev. Goal-oriented trajectories for efficient exploration. *CoRR*, abs/1807.02078, 2018. URL `http://arxiv.org/abs/1807.02078`.

Fabio Pardo, Vitaly Levdik, and Petar Kormushev. Q-map: a convolutional approach for goal-oriented reinforcement learning, 2019. URL `https://openreview.net/forum?id=rye7XnRqFm`.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015. URL `http://arxiv.org/abs/1511.05952`.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL `http://arxiv.org/abs/1509.06461`.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.