Efficient Goal Selection Method for Hindsight Experience Replay

Mollie Bianchi

Abstract

Often in Reinforcement Learning (RL) the reward function is a binary signal that depends on whether or not the goal has been reached. Only if the agent achieves the goal does it receive a reward. This means an extremely large amount of samples are required before the agent is able to learn. Hindsight Experience Replay (HER) is a method that replays past experiences with a new goal. This has increased the sampling efficiency and practicality of RL on problems with sparse rewards. However, currently the new goals are picked based on a hand-crafted heuristic. This project looks at a new method which selects replay goals that the agent is able to learn the most from. This is achieved by selecting goals that maximize the Bellman error. This method was evaluated on 1D and 2D robotic goal search problems and compared against the existing `Future` and `Final` goal selection methods.

I. INTRODUCTION

Reinforcement Learning (RL) is a powerful tool in robotics. It allows for model-free control in potentially complex environments. With RL, an agent learns to accomplish a task through the rewards it receives from interactions with its environments. These rewards are defined according to some reward function. This reward function needs to be carefully designed to reflect the desired task. Often these reward functions are sparse, binary signals that only indicate successful task completion. This is particularly the case when the type of admissible behaviour is unknown and we do not want to limit the agents possible behaviour. With this type of reward function, if an agent performs a series of actions during an episode and does not achieve the goal, it receives no reward.

In the case when there are multiple potential goals to be achieved, it is still possible to learn from failed experiences. For example, if an agent was trying to reach some goal but ends up too far to the right, the agent could learn that that series of actions would have been successful if the goal was slightly to the right. This is more similar to human reasoning and is the basis behind Hindsight Experience Replay (HER) [8]. HER was developed so that agents could learn from failed episodes by replaying the episode with a new goal.

OpenAI [6] gym has created a series of simulations for teaching a robotic manipulator to learn tasks such as reaching, pushing, sliding, and fetching objects. These tasks have sparse reward functions. Their baseline implementations used DDPG and they found that for most of these tasks DDPG without HER was not able to learn the task. If it was able to learn the task, it was able to learn it much faster with HER. In [8], the replay goal was selected based on future states reached in the episode. [6] posed the question of how to select goals that are the most valuable for replay. This project looks at maximizing the Bellman error in order to select goals that would cause the largest improvement during training. The DDPG algorithm is implemented to solve a robotic move to goal problem. The traditional method of HER was compared against the new, proposed `Max` HER method.

II. BACKGROUND

A typical reinforcement learning problem involves an agent interacting with an environment in order to accomplish some goal. The agent can be described by its state s_t. The agent acts according to some policy $a_t = \pi(s_t)$. The probability of the agent being in some next state, s_{t+1}, given its current state and chosen action is modeled by the transition probabilities, $p(s_{t+1}|s_t, a_t)$. The agent receives a reward, $r_t = r(s_t, a_t)$ which is a function of its state and action. The return is defined as a discounted sum of future rewards: $R_t = \sum_{i=t}^{\infty}(\gamma^{i-t} r_i)$. The agent's goal is to maximize its expected return $\mathbb{E}[R_0|s_0]$ given some initial state, s_0. The Q-function for a given policy is defined as $Q^{\pi}(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$. It is the expected return after taking action a_t in state s_t and continuing to follow policy π. An optimal policy, $\pi^\star$, is a policy such that $Q^{\pi^\star}(s_t, a_t) \geq Q^{\pi}(s_t, a_t)$ for all possible states, actions, and policies. An optimal policy has an optimal Q-function meaning it satisfies the Bellman equation:

$$Q^{\pi^\star}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\pi^\star}(s_{t+1}, a_{t+1})] \qquad (1)$$

A. Deep Q Networks (DQN)

Q-learning [1] works by arbitrarily initializing a Q-function. Then the agent follows the greedy policy:

$$\pi(s_t) = \underset{a_t}{\operatorname{argmax}}\ Q^{\pi}(s_t, a_t) \qquad (2)$$

based on the current Q-function. This greedy policy is generally not easily applied to continuous action spaces because it requires an optimization for a_t at each timestep. This step is often too slow for nontrivial continuous action spaces. Using this greedy policy with some noise, episodes are generated. At each step in an episode, the Q-function is updated based on the Bellamn equation according to:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\big(r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t)\big) \qquad (3)$$

For continuous state spaces, the Q-function can be approximated by a neural network as done in Deep Q Networks (DQN) [2]. Since the optimal Q-function must satisfy the

Code used for this project can be found at: `https://github.com/MollieBianchi/controls_project`
Accompanying video can be found at: `https://drive.google.com/open?id=1Mail2d5154FZpn89yIgrqrgNunVVXVQK`

Bellman equation, the following loss function is used to the train the network:

$$\mathcal{L} = (y_t - Q^{\pi}(s_t, a_t))^2 \tag{4}$$

where

$$y_t = r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1})). \tag{5}$$

DQN uses a technique known as experience replay [4]. At each time step in an episode, the experience tuple (s_t, a_t, r_t, s_{t+1}) is added to a replay buffer. For each network update, a batch of samples is drawn from the replay buffer and used to compute the loss in (4).

B. Deep Deterministic Policy Gradients (DDPG)

Deep Deteministic Policy Gradients extend DQN to continuous action spaces. This is achieved by using two neural networks. One network approximates the Q-function and is trained as in DQN. This network is referred to as the critic network $Q(s, a|\theta^Q)$ with parameters θ^Q. The second network learns the optimal policy. It is referred to as the actor network $\mu(s|\theta^{\mu})$ with parameters μ. Since the agent's goal is to choose an action that maximizes the Q-function, the actor network is updated using gradient ascent with the policy gradient:

$$\nabla\theta^{\mu} J \approx \nabla_a Q(s_t, a_t = \mu(s_t)|\theta^Q)\nabla_{\theta_{\mu}}\mu(s_t|\theta^{\mu}) \tag{6}$$

Since the critic network that is being updated with (4) is also used to calculate the target value in (5), training is often prone to divergence. So target networks similar those used in [2] are used, that is a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, are created and used for calculating the target values. The weights of the target networks slowly track the learned networks:

$$\theta' \longleftarrow \tau\theta + (1 - \tau)\theta' \tag{7}$$

with $\tau << 1$. DDPG is an off-policy algorithm, so the exploration policy is different from the learned policy. The exploration policy is constructed by adding a noise process to the actor policy: $\mu(s_t|\theta_t^{\mu}) + \mathcal{N}$. This noise process can be selected according to the environment. In the original DDPG paper an Ornstein-Uhlenbeck process [5] was used.

C. Universal Value Function Approximators (UVFA)

DQN and DDPG are formulated to learn a policy for a specific goal. Universal Value Function Approximators (UVFA) [9] extend DQN to the problem when there is more than one goal the agent could be asked to complete. For example, moving to a specified goal location for each episode instead of the same goal location for all episodes. In this formulation, there is a reward function for each possible goal, $r_g(s_t, a_t)$. At the start of an episode, an initial state, s_0 and a goal, g is sampled from some distribution. The policy is now a function of the current state and the goal, $\pi(s_t, g)$, as is the Q-function, $Q^{\pi}(s_t, a_t, g) = \mathbb{E}[R_t|s_t, a_t, g]$. [9] shows that with this formulation it is possible to train a network to approximate the Q-function using the DQN method.

III. Related Work

Hindsight Experience Replay (HER) [8] is an addition to the standard DDPG algorithm. A state transition is composed of the state, action, next state reached, reward, and desired goal at each time step in an episode. Without HER, this transition is added to the replay buffer with the goal that was sampled for that episode. With HER, the transition is added to the replay buffer an additional time with a new desired goal. This new desired goal is selected according to four different strategies: `Final`, `Future`, `Episode`, and `Random`. `Final` uses the last state that was reached in the episode. `Future` uses a random state which comes from the same episode as the transition being replayed and was observed after it. `Episode` uses a random state from the same episode as the transition being replayed. `Random` uses a random state that has been previously encountered at any time during training as the new goal. Measured in terms of highest success rate, the `Future` and `Final` methods performed best. OpenAI was able to reproduce the HER results in [6]. They noted that without HER, DDPG was not able to solve many of the tasks in their simulations. They also noted that the goals for HER are generated using the hand-crafted heuristics noted above. They propose that future works could look into the magnitude of the Bellman error to judge which goals are most valuable for replay. This is the focus of this project.

The approach is inspired by that of Prioritized Experience Replay (PER) [10]. Generally, experience transitions are uniformly sampled from a replay memory. In PER instead of adding additional transitions to the replay buffer, the method for selecting experiences from the replay buffer is altered so that experiences deemed more important are prioritized and sampled at an increased frequency. One criterion of importance is how much the RL agent can learn from a transition. A proxy for this measure is given by the Temporal Difference (TD) or Bellman error:

$$\delta = r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1})) - Q^{\pi}(s_t, a_t). \tag{8}$$

Using this prioritized sampling method with DQN, they were able to improve performance achieving a new-state-of-the-art on many Atari games.

A different approach for selecting goals for HER was proposed in [11]. This does not use the Bellman error. Instead it looks at generating goals that are easy to achieve in the short term but also useful for guiding the agent to reach the actual goal in the long term. This is achieved by using an optimization problem to minimize the Wasserstein distance between goals that have already been achieved in an episode and the original goal.

IV. Methodology

The experiments involve a continuous action and state space so the DDPG algorithm is used. As well, the goal is randomly selected at the start of each episode so the critic and actor networks are based on the UVFA formulation, that is the critic network is a function of s_t, a_t, and g and the actor network is a function of s_t and g. This is combined

Algorithm 1: DDPG with Max HER

```
Randomly initialize critic network $Q(s, a, g|\theta^Q)$ and actor $\mu(s, g|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \longleftarrow \theta^Q$, $\theta^{\mu'} \longleftarrow \theta^\mu$
Initialize replay buffer $R$
for episode=1, M do
    Initialize episode_transitions $E = []$
    Reset environment and get initial state $s_t$ and episode goal $g$
    for $t$ = 1, T do
        Select $a_t$ according to exploration factor and current actor policy
        Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1}, g)$ in $R$
        Add transition $(s_t, a_t, s_{t+1})$ to $E$
        Perform one step of optimization:
            Sample a random minibatch of N transitions $(s_t, a_t, r_t, s_{t+1}, g)$ from $R$
            Set $y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}, g|\theta^{\mu'}), g|\theta^{Q'})$
            Update critic by minimizing the loss: $L = \frac{1}{N}\sum_t (y_t - Q(s_t, a_t, g|\theta^Q))^2$
            Update the actor network by doing one step of gradient ascent using the sampled policy gradient:
                $\nabla\theta^\mu J \approx \nabla_a Q(s_t, a = \mu(s_t)|\theta^Q)\nabla_{\theta_\mu}\mu(s_t|\theta^\mu)$
            Update the target networks:
                $\theta^{Q'} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
                $\theta^{\mu'} \longleftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end
    Initialize max Bellman Error $\delta_{max} = 0$ and new goal $g_{new}$
    for $\hat{g} \in E \in s_t$ do
        Compute sum of squared Bellman Error for all transitions in E:
            $\delta_E = \sum(Q(s_t, a_t, \hat{g}|\theta^Q) - (\hat{r_t} + \gamma Q'(s_{t+1}, \mu'(s_{t+1}, \hat{g}|\theta^{\mu'}), \hat{g}|\theta^Q))^2$
        if $\delta_E > \delta_{max}$ then
            $\delta_{max} \longleftarrow \delta_E$
            $g_{new} \longleftarrow \hat{g}$
        end
    end
    Compute $\hat{r}_t$ for all transitions in $E$ using $g_{new}$
    Store new transitions $(s_t, a_t, \hat{r}_t, s_{t+1}, g_{new})$ in $R$
    Decay exploration factor
end
```

with HER to improve sample efficiency. The novel contribution of this project is selecting a goal for HER that maximizes the Bellman error. The complete algorithm implementation is shown in Algorithm 1.

A. Network Architecture

Both the actor and critic networks have a single hidden layer with 4 units for the 1D simulation and 20 units for the 2D simulation. ReLU activations are used. The state, s_t, and goal, g are concatenated and provided as input to the actor network. The final layer of the actor network passes through a $\tanh$ function to constrict the output action to between -1 and 1. The state, s_t, action a_t, and goal g are concatenated and provided as input to the critic network which outputs the Q-value. The networks were implemented in PyTorch and optimized using the Adam optimizer [7].

B. Exploration Policy

The chosen action is based on an exploration factor, ϵ. ϵ starts at 1 and decays by a factor of 0.999 each episode until it reaches a minimum value of 0.05. If a uniformly sampled random number between 0 and 1 is less than ϵ, a random action is uniformly chosen from the action space. Otherwise, the output from the actor network, μ, is used as a_t. This exploration policy was found to result in the most exploration over the state while still allowing the methods to converge.

C. Goal Selection

For the proposed `Max` HER method, after an episode is completed, each of the reached states, s_t is evaluated as a potential new goal, $\hat{g}$. The new goal that is selected, g_{new}, will be the one from which the critic network would receive the largest update. In other words, it is the goal that maximizes the sum of the squared Bellman error for all the states in that episode. For each potential new goal, $\hat{g}$, the new reward, $\hat{r}_t$, for each transition is recomputed based on $\hat{g}$. The sum of the squared Bellman errors can then be computed as:

$$\delta = \sum \big(Q(s_t, a_t, \hat{g}|\theta^Q) \tag{9}$$

$$- (\hat{r}_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}, \hat{g}|\theta^{\mu'}), \hat{g}|\theta^Q)\big)^2 \tag{10}$$

The $\hat{g}$ that results in the largest δ is selected as the new goal, g_{new}. Each transition is modified to be $(s_t, a_t, \hat{r}_t, s_{t+1}, g_{new})$ and is added to the replay buffer. The original transition with an unmodified goal and reward was also added to the replay buffer previously during the episode.

V. Experiments

Two simulations were used for evaluation. The first was a simple 1D robot tasked with travelling to a goal position on a line from a random starting position. The state space was the robots position along a line between -5 units and 5 units and the action space was its desired velocity normalized from -1 unit/s to 1 unit/s. The robot's dynamics were then $s_{t+1} = s_t + a_t\Delta t$. The second simulation extends this task to 2D. The robot is holonomic, and its state is its 2D position within a 3x3 grid. Its action space was a 2D velocity with a maximum component value of 1. The reward function used in both cases was -1 for every time step where the robot's state was greater than 0.05 from the goal and +100 if the robot reached the goal. The `Max` HER method proposed here was compared against DDPG without HER, DDPG with HER using the `Final` method, and DDPG with HER using the `Future` method.

VI. Results

A. 1D Simulation

The evolution of the Q-function output by the critic network during training using each the four methods is shown in Figure 5. For plotting purposes, the goal input to the network was fixed at 0. The a_t input to the network is indicated by the y-axis and the x-axis indicates the s_t input. As expected, all the networks eventually converged to output the highest Q-value when s_t is equal to the goal. For states to the left of the goal, the Q-values are higher if the a_t input is positive (i.e. move to the right). Similarly, for states to the right of the goal, the Q-values are higher if the a_t is negative (i.e. move to the left). The policy trained using the `Max` HER method converges to this expected Q-function the fastest while the `Future` and `Final` methods take slightly longer. The critic network trained without any HER takes the longest to converge to the optimal Q-function. The output from all methods had the correct shape even early in the training. This is likely why there was not significant differences in the time to took to learn a successful policy.

The success rate of the policies learnt using each of the four methods evaluated without noise on 10 randomly generated episodes after each training episode is shown in Figure 1 and the average episode return of each policy is shown in 2. All policies had similar performance but the ones trained with `Future` HER and without HER were able to converge the fastest. The `Max` HER method has a lower success rate and average return in comparison to the other methods despite converging the Q-function faster.

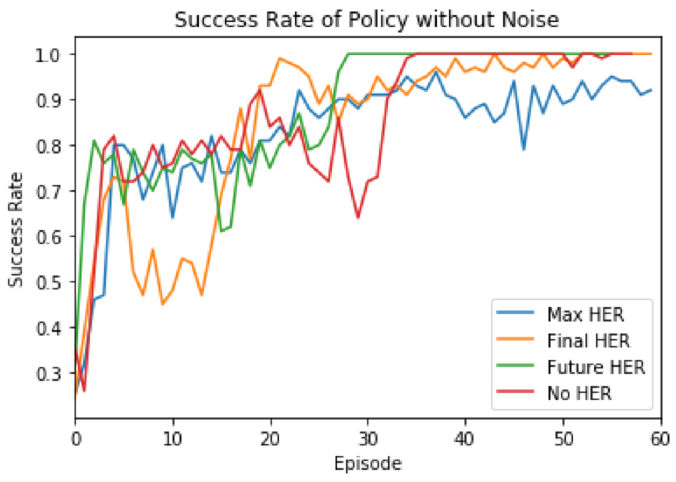

Fig. 1: Success Rate for 1D Simulation

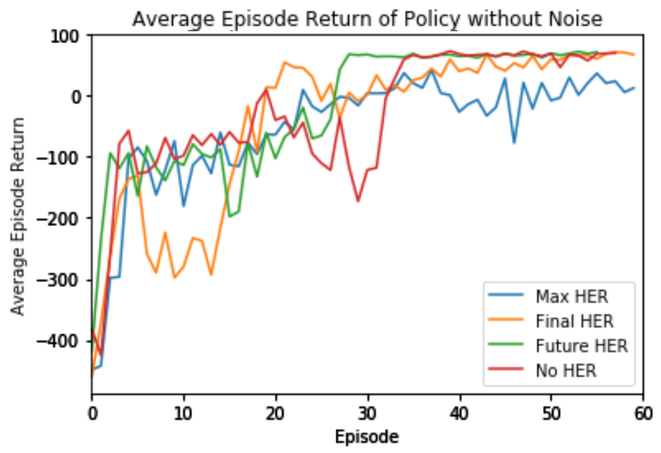

Fig. 2: Average Episode Return for 1D Simulation

B. 2D Simulation

Figure 6 shows $Q(s_t, a_t = [0,0], g = [0,0])$ during training with each of the four methods. The axes indicate the value of the input, s_t. The optimal Q-function should have the largest value at $s_t = g$. The `Final` and `Future` methods were able to converge fastest. The output from critic network trained with `Max` HER converged slower, but all three methods were eventually able to converge to the correct Q-function. The critic network trained without HER was not able to learn the correct Q-function.

The success rate and average return of the policies on the previous ten episodes encountered during training is shown in Figure 3 and 4 respectively. The policy trained using `HER`, shown in blue, reaches an 100% success rate and a higher return earlier than the other policies, but it diverges and does not stay converged as the other HER based policies do. Both the `Future` and `Final` HER based policies were able achieve consistent success during training.

With the 1D simulation all the policies were able to learn a successful policy after approximately 35 episodes. With the 2D simulation, only the policies trained using HER were able to learn a successful policy. Nearly 100 episodes were required for the `Final` method to learn a successful policy and almost 140 episodes for the `Future` HER method. This increase in required episodes was expected as the 2D simulation is a harder problem. Using a purely random exploration policy, it would be difficult to ever reach the goal whereas in the 1D simulation it is much easier to reach the goal from random actions. This is likely why the policy trained without HER succeeded in the 1D case but failed in the 2D case.

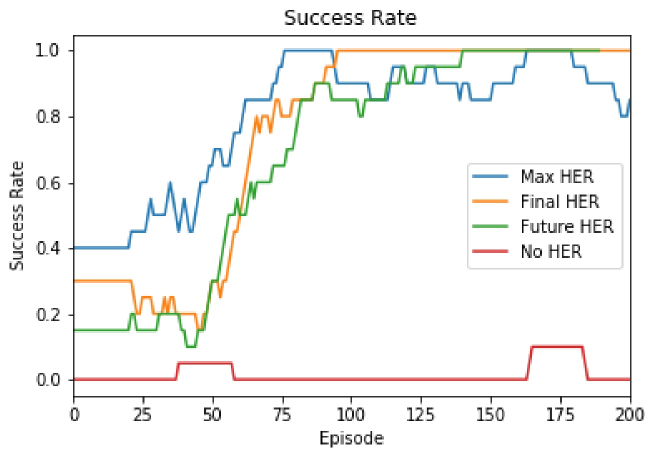

Fig. 3: Success Rate of the last 10 Episodes During Training for the 2D Simulation

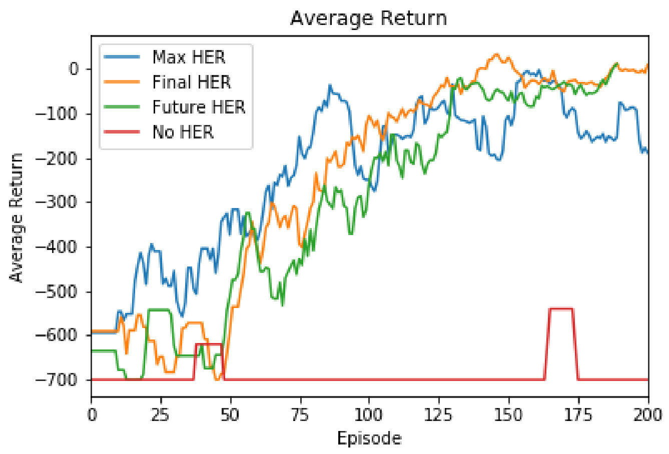

Fig. 4: Average Return of the last 10 Episodes During Training for the 2D Simulation

VII. Conclusions and Future Work

Training with any of the HER methods improved sampling efficiency and reduced the number of episodes necessary to learn a successful policy in both the 1D and 2D simulations. In the 2D simulation, the method trained without HER was unable to learn. The use of HER would likely be even more important for complicated tasks with sparse reward functions where it is difficult to achieve the goal with using a random exploration policy.

The proposed method of `Max` HER was able to speed up learning the Q-function for the 1D simulation. However, for this simple simulation all methods were able to learn the correct Q-function so there were no significant advantages seen in performance or number of episodes need to learn a successful policy. The performance of `Max` HER in terms of success rate and average return was actually lower than the other methods at the end of training. For the 2D simulation, the critic network trained with `Max` HER was not the fastest to converge. The `Future` and `Final` methods were able to converge faster. Ultimately, there does not seem to be an advantage of `Max` HER over the other methods of goal selection.

There are still lots of areas for future work on this topic. For example, one could look into training a goal generator to output a goal given an input transition and current policy or some sort of curriculum based learning that selects easier goals that eventually lead to the desired goal.

References

[1] Watkins, Christopher JCH and Dayan, Peter. Q-learning. Machine learning, 8(3-4):279–292, 1992.
[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[3] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[4] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.
[5] Uhlenbeck, George E and Ornstein, Leonard S. On the theory of the brownian motion. Physical review, 36(5):823, 1930.
[6] Plappert, Matthias, et al. "Multi-goal reinforcement learning: Challenging robotics environments and request for research." arXiv preprint arXiv:1802.09464 (2018).
[7] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[8] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5055–5065.
[9] Schaul, Tom, et al. "Universal value function approximators." International conference on machine learning. 2015.
[10] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. arXiv preprint arXiv:1511.05952.
[11] Ren, Zhizhou, et al. "Exploration via Hindsight Goal Generation." Advances in Neural Information Processing Systems. 2019.

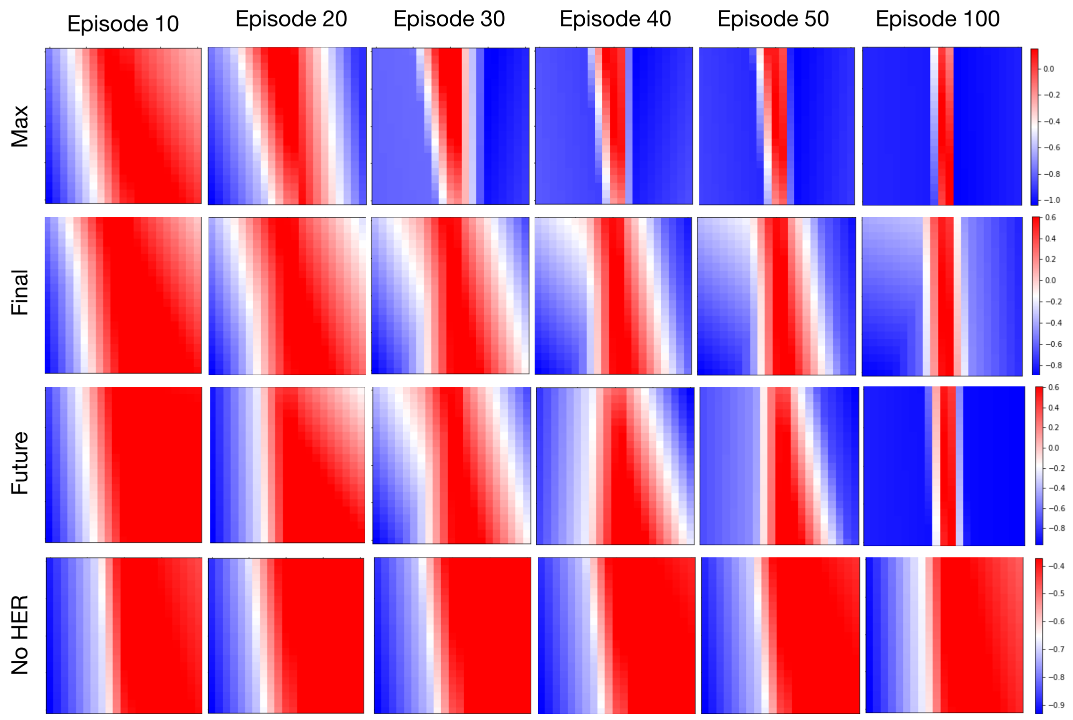

Fig. 5: Output of the critic network, $Q(s_t, a_t, g = 0)$, during training where s_t is indicated by the x-axis and a_t by the y-axis.

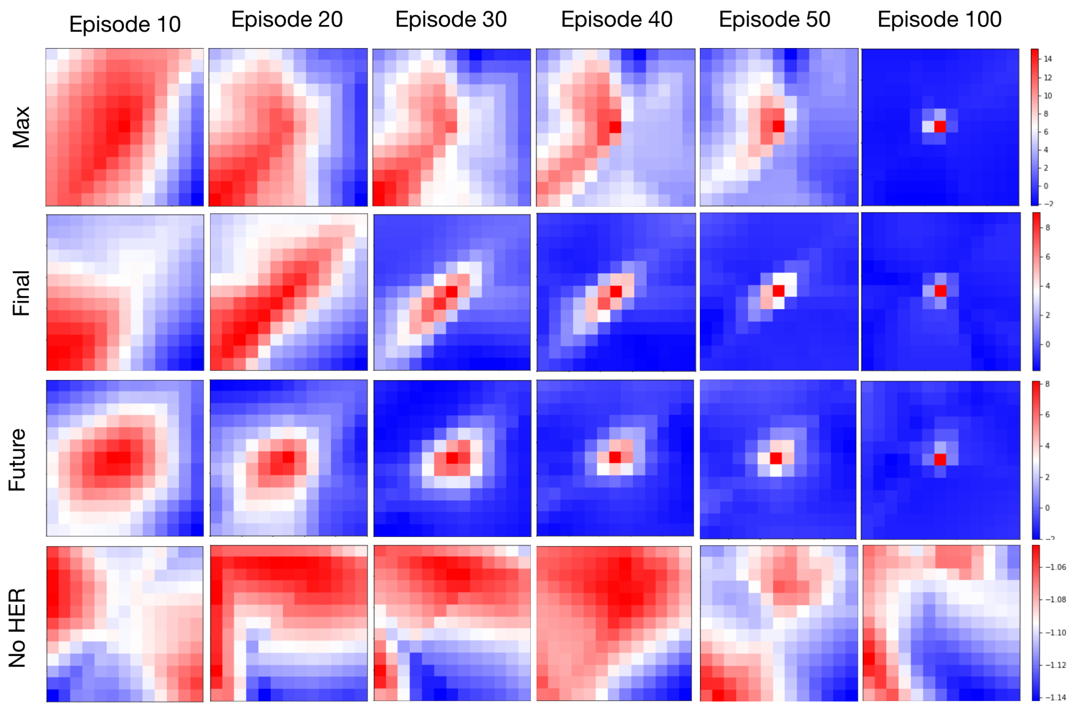

Fig. 6: Output of the critic network, $Q(s_t = [x, y], a_t = [0, 0], g = [0, 0])$, during training where s_t is indicated by the x-axis and y-axis.