Hindsight Experience Replay - report

Sebastian Jost

December 19, 2022

Abstract

Reinforcement learning is a powerful machine learning technique, but sparse rewards can be very problematic for traditional techniques. Hindsight Experience Replay (HER) addresses this problem by learning from unsuccessful episodes more efficiently by generating additional experiences. In this study I replicated several results shown in the publication that introduced HER [And+17] and investigated the interaction with shaped rewards in more detail. My experiments suggest that HER is a powerful and widely applicable technique to speed up learning, enable agents to solve more difficult problems, even when using binary rewards. Contrary to [And+17] I found that HER can still improve by using shaped instead of binary rewards.

Contents

1 Introduction and motivation

Machine learning is quickly becoming more and more popular, being applied to a vast range of tasks. Yet the most popular technique so far, supervised learning, requires massive amounts of data being available for any task. Reinforcement learning is a technique, that generates it's own data, allowing agents to learn on their own with very little domain specific knowledge required. Hindsight Experience replay was introduced in [And+17] to improve RL-agent's performance on tasks with very sparse rewards by being able to learn from undesirable outcomes more efficiently.
While sparse 0-1 rewards are usually easy to define, it is obvious that any initially random agent will have difficulty ever achieving a positive reward in such a setting. Therefore more detailed rewards can be defined to guide the agent towards the goal. Since shaped rewards work well for regular DQN agents and can be used to incorporate prior knowledge about desirable solutions, it is important to investigate how they interact with Hindsight Experience Replay (see Experiment 3).
In this report I will test Hindisght Experience Replay on two problems introduced in Section 3.2 to see if it actually improves performance and how it interacts with shaped rewards. I will also explain the process of replicating experiments from [And+17].

2 Background

I will use the following notation:

Action space: A with actions $a, a' \in A$	Action-value function/ Q-function: $Q : S \times A \to \mathbb{R}$
State space: S with states $s, s' \in S$	Reward function: $R : S \to \mathbb{R}$ or a similar variation
Goal space G with goals $g \in G$	Observed reward: $r \in \mathbb{R}$
Policy $\pi : S \to A$	

2.1 Q-learning

Q-learning is a relatively simple but powerful technique that is commonly used in Reinforcement Learning. The algorithm works by learning an action-value function $Q : S \times A \to \mathbb{R}$ to estimate the future reward after taking action $a \in A$ in state $s \in S$. A simple greedy policy can then be used to control the agent based on this Q-function: $\pi(s) = \text{argmax}_{a \in A} Q(s, a)$.
The Q-values are learned using the Bellmann equation:

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha \left(R(s,a,s') + \gamma \max_{a' \in A} Q_t(s',a') \right) \tag{1}$$

Where $\alpha \in \mathbb{R}^+$ is the learning rate or step size and $\gamma \in [0,1]$ is a discount factor that controls how much the agent cares about immediate and future rewards. $R : S \times A \times S \to \mathbb{R}$ is the reward function, which is often simplified to $R : S \to \mathbb{R}$. t indicates the current iteration of the algorithm or how often this particular value has been updated. A common initialisation is $Q_0(s,a) = 0$, but other initialisation can be used to introduce prior knowledge.
Note that in non-deterministic environments the next state s' is typically not known before taking the action, but can be observed and used for the update afterwards.

2.2 Experience Replay

In Reinforcement Learning it is common to use a replay buffer to store the $k \in \mathbb{N}$ most recent experiences in the form of transitions (s, a, r, s'). The agent then samples from this buffer to train it's neural network. This improves the agent's sample efficiency since an agent can learn from the same experience multiple times.

2.3 Multi-Goal Reinforcement learning

In many scenarios it is desirable for an agent to be able to reach more than one goal. For example a robot being able to navigate not just to one location, but to any given location.
The Reinforcement Learning algorithm we will use here: Q-Learning assumes a fixed reward function $R : S \to \mathbb{R}$. This reward also defines the goal $g \in S$ the agent tries to achieve by $g = \text{argmax}_{s \in S} R(s)$. While there can be multiple states with maximum reward, we have no way of controlling towards which of those the agent goes. To still be able to direct the agent towards various goals, we can extend the state to include the goal. This allows different reward values depending on what the current goal is - effectively combining different reward functions that each lead to one specific goal.

This is reflected below in the reward functions getting both the current state and goal as an input. These could also be combined into a single state $\hat{s} \in S \times G$, but keeping them separate is easier to implement and understand.

2.4 Hindsight Experience Replay

As introduced in [And+17] Hindsight Experience Replay (HER) is a technique to improve the performance of RL-agents on tasks with very sparse rewards. It does so by being able to learn from undesirable outcomes more efficiently. HER adds extra transitions to this replay buffer to learn from mistakes. It does so by additionally storing a goal state g in each transition (s, a, r, s', g) and then adding extra transitions assuming the goal was something else. An obvious choice for these extra goals is to choose states that were actually reached during the episode because we know that whatever actions were taken to get there, would have been successful if that was actually the goal. Note that this is exactly how humans learn in some situations.
In more complex environments like the robot-arm problem described in [And+17], it is useful to choose $G \neq S$ to reduce the input size for neural networks. To implement this with the goal sampling, one can use a map $m : S \to G$ to convert states into goals i.e. by extracting location data of one object in the state.
In [And+17] the authors test three strategies of sampling the additional goals. These are:

1. **random:** choose a random $g \in G$
2. **episode:** choose a random state s reached in the episode as the goal g
3. **future:** when sampling goals for a transition $s \to s'$, choose a random state reached in the episode after this transition as the goal g.

In the experiments here, we only use the future strategy and always sample $k = 4$ additional goals for each episode, independent of the problem size.

3 Experiment description

3.1 Experiment goals

With the following experiments we want to evaluate whether HER improves learning performance. While it is meant to work well with binary rewards, we also want to investigate how HER performs when using shaped rewards. To achieve this we perform three experiments with the following goals:

1. Compare learning speed of DQN's with and without HER.
2. Compare solvability of problems of various difficulty using DQN's with and without HER.
3. Find out how HER interacts with shaped rewards.

3.2 Learning environments

I use two different learning environments to perform all experiments. The first, the Bitflip problem, was also used in [And+17]. The second problem, a gridworld environment, is a typical example problem for Reinforcement Learning since it allows for a wide range of customizations. This environment serves as a replacement for the robot-arm control problems that is introduced in [And+17]. A notable difference is, that here we only investigate discrete problems, whereas the robot-arm control has a contiuous action space and the authors used a slightly more advanced technique - DDQN - to deal with that.
Both problems are easily scalable to increase or decrease their difficulty.

3.2.1 Bitflip problem

The first environment models the Bitflip problem: Starting with a random binary sequence of length $n \in \mathbb{N}$, the agent can flip a single bit at a time to achieve a given goal state. By default the goal is the sequence $(1, 1, ..., 1)$.
This problem was used in [And+17] as a problem that's very hard to learn with regular Q-learning and sparse rewards, because with random actions the agent almost never experiences positive rewards making it unable to learn. It was shown in [And+17], that learning with HER can solve the problem for much larger n.
Input size for the neural networks is $2n$ (current state and goal) both when using HER and when just using DQN and experience replay. Outputs are n neurons representing each possible action. We always choose the action with the largest value.

3.2.2 Gridworld environment

The second environment is a gridworld as introduced in the presentations. An agent can move along the cardinal directions on a cartesian grid with step size 1. The agent starts on one square and the episode ends if it has reached a predefined goal square. There are some randomly placed walls on the grid where the agent cannot move. The edge of the square grid is treated as walls. All actions are legal at all times. If the agent chooses an action that would move it into a wall, it does not move at all instead.
The walls of the gridworld are generated automatically using the procedure described in Appendix 8.2. This makes the problem scalable and allows performing all experiments for this second problem.
The neural network gets four inputs for this problem and has four outputs. Inputs are the current and goal positions as vectors in $[0, 1]^2$ each. The outputs represent the four possible actions (up, right, down, left) and the agent always chooses the action with the highest output value.
For this problem I chose a fixed starting point but varied the goals within each maze between epochs. Any free square is a potential goal.

3.3 Experiment setup

3.3.1 Learning parameters

In [And+17, Appendix A], it is stated, that the neural network used for the bitflip problem had a single hidden layer with 256 neurons. The other parameter settings are described later in [And+17, Appendix A] and I used mostly the same parameters. The major difference is that I trained the networks for only 10 to 25 epochs instead of 200. I also used the same settings of 50 cycles of 16 episodes and 40 optimisation steps per epoch.
The only parameter I did not find is which goal sampling strategy the authors used for the Bitflip environment. Since they found the future strategy with $k = 4$ additional goals to be the best, I used that in all experiments.
I used the same settings for the gridworld problem.
I initially ran all experiments with a replay buffer of size $10,000$ (compared to 10^6 used in [And+17]. However, due to the much lower number of epochs, this led to poor performance since early experiences were not replaced fast enough such that the agent kept trying unsuccessfully to learn from the first few episodes where the agent was still acting mostly randomly. Therefore I repeated all experiments with a smaller replay buffer of size 2048, which enabled the agent to solve the bitflip problem for larger n.

3.3.2 Reward functions

For investigating goal 3, I used the following reward functions. The bitflip environment uses the 0-1, MSE and MAE reward while the gridworld environment uses the 0-1 and remaining two rewards.
In this implementation all reward functions depend only on the current state and the goal state. Since the goal state is not constant when using HER, that is also passed to the reward functions. Since rewards are typically maximized in Reinforcement Learning, we add a negative sign to the last four reward functions. Here let $s \in S$ be a state and $g \in S$ the desired goal state.
0-1 reward: Reward 1 if goal is reached, 0 otherwise:

$$R_{01}(s,g) = \begin{cases} 1 \text{ if } s = g \\ 0 \text{ else} \end{cases}$$

MSE reward: Use mean squared error between state and goal as reward:

$$R_{\text{MSE}}(s,g) = -\frac{1}{|s|}\sum_{i=1}^{|s|} |s_i - g_i|^2$$

MAE reward: Use mean absolute error between state and goal as reward:

$$R_M(s,g) = -\frac{1}{|s|}\sum_{i=1}^{|s|} |s_i - g_i|$$

Euclidean reward: Use Euclidean distance between state and goal as reward:

$$R_E(s,g) = -\sqrt{\sum_{i=1}^{|s|} (s_i - g_i)^2}$$

Manhattan reward: Use Manhattan distance between state and goal as reward:

$$R_M(s,g) = -\sum_{i=1}^{|s|} |s_i - g_i|$$

3.3.3 Measurements during training

After each epoch of training, the current success rate is calculated from the total number of successful episodes so far:

$$SR = \frac{\text{Nbr. of successful episodes}}{\text{Nbr. of played episodes}}$$

Training is stopped when the success rate is higher than 99.5% to speed up the experiments.

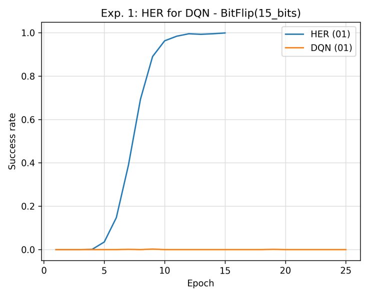
3.3.4 Experiments

Experiment 1: train a DQN and DQN with HER once on a given problem size and plot the successrate over the epoch number. This experiment uses only 0-1 rewards.
Experiment 2: train the DQN and DQN with HER once each for various problem sizes and record the final success rates. Then plot the final success rates over the problem sizes to compare which problems can be solved with the current training parameters. This experiment uses only 0-1 rewards.
Experiment 3: train the DQN and DQN with HER once each for three different reward functions as mentioned in Section 3.3.2
In all experiments I made sure to use the same gridworld environment (same walls and starting position) for both the regular DQN and the one using HER. The walls and starting positions are usually different between different reward functions and problem sizes though, which makes these results slightly less reliable.
All experiments can be run by executing `main_combined.py` in [Jos]. On my laptop with 8 CPU cores this took about 2h to complete. Running the experiments shown in the appendices requires changing the `buffer_size` in the `config_dqn` calls in `main_combined.py`. The plots are then saved in the `plots` subdirectory.

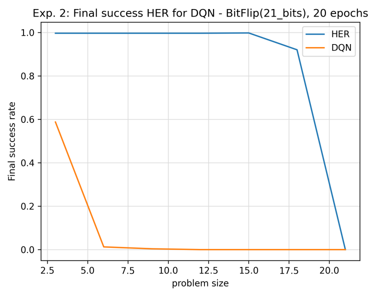
4 Implementation efforts

In this section I describe the process of implementing the experiments.
I initially tried replicating the Bitflip environment and DQN with HER from scratch using python and the Tensorflow library. However, I could not get my networks to learn anything with this program. Therefore I used code found on GitHub (see [Her]). I split that code into several files, added all the experiments described above and implemented a new environment and interface for potential future environments. This also required making a few changes to the Bitflip environment defined in the code from GitHub. I did not change the core functionality that implemented the DQN agents. HER algorithm or replay buffer.
Checking [And+17, Appendix A] showed that the provided code already used the same parameters as in the original experiments. While I played around with them initially, I ended up using very similar ones for my experiments here with the differences mentioned above.
My code used for the final experiments is publicly available on Github [Jos].

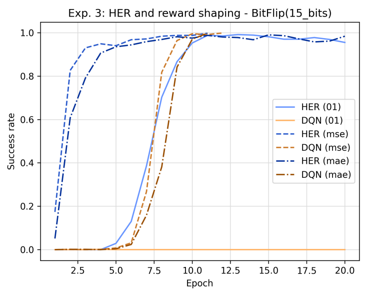
5 Experiment results



(a)

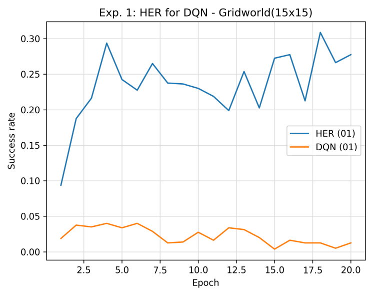


(b)

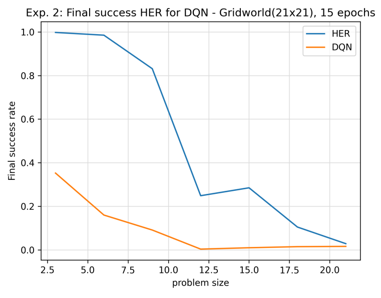


(c)

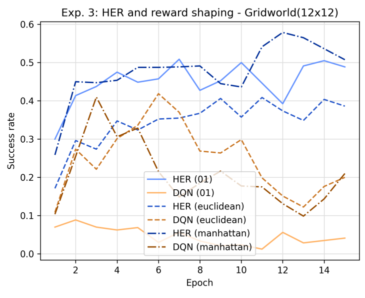
Figure 1: Bitflip experiments with Batch size 2048. (1a) Exp. 1: Comparison of DQN and DQN with HER on a bit flip environment with a problem size of 15, over 25 epochs. (1b) Exp. 2: Comparison of the size of bit flip problems that can be solved by DQN and DQN with HER, over 20 epochs. The problem size increases by 3 in each step. (1c) Exp. 3: Comparison of DQN and DQN with HER on a bit flip environment with a problem size of 15, over 20 epochs, using different reward functions.



(a)



(b)



(c)

Figure 2: Gridworld experiments with Batch size 2048. (2a) Exp. 1: Comparison of DQN and DQN with HER on a gridworld environment with a problem size of 15x15, over 20 epochs. (2b) Exp. 2: Comparison of the size of gridworld problems that can be solved by DQN and DQN with HER, over 15 epochs. The problem size increases by 3 in each step. (2c) Exp. 3: Comparison of DQN and DQN with HER on a gridworld environment with a problem size of 12x12, over 15 epochs, using different reward functions.

5.1 Solvability

We see throughout all experiments that the DQN with HER performs significantly better than the DQN without HER, The difference is especially large in fig. 1b, where the DQN barely solves the problem for 3 bits (8 possible states) whereas HER solves it for up to 18 bits (262144 possible states). It should be noted that the DQN can perform much better than this when properly adjusting the replay buffer size, batch size, epoch number and size of the hidden layer. With those adjustments I have been able to get the DQN to solve the Bitflip problem for 8 bits. But it is clear that HER makes it much easier to find suitable parameters.

5.2 Learning speed

Due to the very bad performance of the DQN, figures (1a) and (2a) don't allow us to draw meaningful conclusions about the learning speed, since the DQN does not learn to solve the problems at all.
However figures (2c) and especially (1c) do show some useful information. Here the DQN with HER clearly learns the problem faster than the DQN without HER.

5.3 Interaction with shaped rewards

Figures figures (1c) and (2c) also show a pretty clear trend for the interaction with shaped rewards. Contrary to what was stated in [And+17, Sec. 4.4], these plots show that HER still benefits from using the shaped rewards and for all reward functions outperforms the corresponding DQN without HER in both learning speed and maximum success rate.
The plots in [And+17, Fig. 5] analysing the interaction with shaped rewards are also interesting considering the fact that they do not show any improvements from using shaped rewards and no task is solved when using them. Therefore it is possible that that reward function was just not good for the task, even though the authors claim that this was the best among the shaped rewards that they tried.
While I was initially able to create a setting where HER with MSE reward only got up to about 50% success rate for the bitflip problem with 8 bits while the DQN without HER solved the same problem using the MSE reward, I was not able to recreate this later on. At the time this result was consistent over multiple runs and showed how HER can be hindered by shaped rewards.

5.4 Single vs. Multi-Goal RL

In the Bitflip problem the goal was fixed and the starting position randomized. In the Gridworld on the other hand, the starting position was fixed and the goal randomized. HER outperformed the regular DQN in both environments suggesting that it is a valuable technique even for single-goal tasks.

6 Further questions

I only replicated experiments for the future goal sampling strategy and a single parameter k for the number of goals. It would be interesting to try out different, more complex goal sampling strategies and trying to replicate the experiments for different k.
My experiments also showed reason for much more optimism regarding the interaction with shaped rewards than what was shown in [And+17]. Therefore further experiments with other reward functions and environments would be useful to produce actionable guidelines on when shaped rewards work well with HER and when they don't.
Finally the interaction of HER with other learning enhancement techniques can be investigated. For example some environments allow incrementally increasing the difficulty during training by starting close to the goal at first and then slowly increasing the approximate distance to the goal during training. This technique has been shown to be successful in [Ago19], which I have also partially replicated myself.

7 Conclusion

In conclusion, the results of these experiments show that Hindsight Experience Replay (HER) can significantly improve learning performance in reinforcement learning tasks. The DQN with HER consistently demonstrated faster learning and the ability to solve more difficult problems than the regular DQN. Additionally, HER was able to effectively use shaped rewards, as demonstrated by the higher success rates when using HER compared to the regular DQN for all three reward functions. However, the benefit of using shaped rewards seems smaller than when not using HER. These results suggest that HER is a valuable technique for improving learning performance in a wide variety of reinforcement learning tasks.

References

[Ago19] McAleer S. Shmakov A. et al. Agostinelli F. "Solving the Rubik's cube with deep reinforcement learning and search." In: *Nat Mach Intell 1, 356–363 (2019)* (2019). last accessed: 2022-12-19. URL: https://doi.org/10.1038/s42256-019-0070-z.

[And+17] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017). last accessed: 2022-11-22. URL: https://arxiv.org/pdf/1707.01495.pdf.

[Her] Alex Hermansson. *Bitflip problem with HER using Pytorch*. last accessed: 2022-12-19. URL: https://github.com/AlexHermansson/hindsight-experience-replay/blob/65468d523bb803123d7aab9bb83abc7a3d bitflip.py.

[Jos] Sebastian Jost. *My source code for the experiments*. URL: https://github.com/simulatedScience/hindsight-experience-replay.

8 Appendix

8.1 Appendix A: learning parameters

For the main experiments depicted in fig. (1a) to (2c) I used the following parameters:

- discount factor (γ): 0.98
- learning rate (α): 0.001
- batch size: 128
- buffer size: 2048
- goal sampling strategy: future with $k = 4$

The Artificial Neural Networks use only dense layers with the input and output dimensions specified in the environment definitions and a single hidden layer with 256 neurons. The hidden layer uses ReLU as an activation function, the output layer uses linear activations.
The program uses two Neural Networks for each agent: One network mapping $S \times G \to \mathbb{R}$ to estimate state values and one $S \times G \to \mathbb{R}^{|A|}$ estimating the advantage $Q(s, a) - \frac{1}{|A|} \sum_{i=1}^{|A|} Q(s, a_i)$. These are defined in `ddqn_agent.py` in the class `DuelingMLP`.

8.2 Appendix B: Maze generation

To generate the walls for the gridworld environment I developed and used the following algorithm:
The start and goal positions are chosen randomly, but the goal position is chosen such that it is at least a certain distance away from the start position (determined by the width and height of the maze). The maze is initially generated with a specified percentage of walls placed randomly. A random path from the start to the goal is then generated using an ϵ-greedy policy and the Manhattan distance to the goal. Walls along the path are removed to guarantee the existence of a solution, and any inaccessible free positions are removed through a cleaning process using breadth first search. The resulting maze, start position, and goal position are returned.
To use the maze for the neural networks, the positions are scaled to be pytorch tensors containing 32-bit floats in range $[0, 1]$. When performing an action the state is converted back into integers to be modified. This should prevent any numeric errors from accumulating by modifying states by adding other floats.

8.3 Appendix C: experiments with larger batch size

The following experiments use the same parameters as described above, but with buffer size 10^5 instead of 2048. Each experiment was run twice to estimate the random variations. We see that while the overall results are slightly different than with the smaller batch size, the difference in performance of HER and the regular DQN is still basically the same. These also align with most previous runs I did to test the programs suggesting the results to be quite reliable.

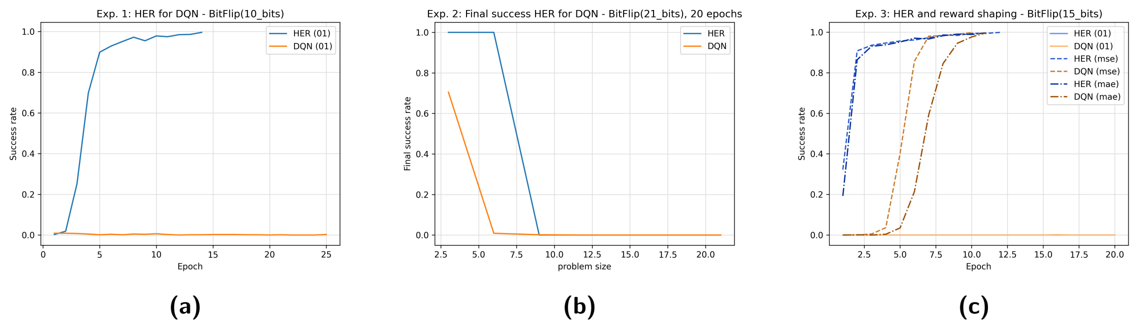


Figure 3: BitFlip experiments with Batch size 10^5

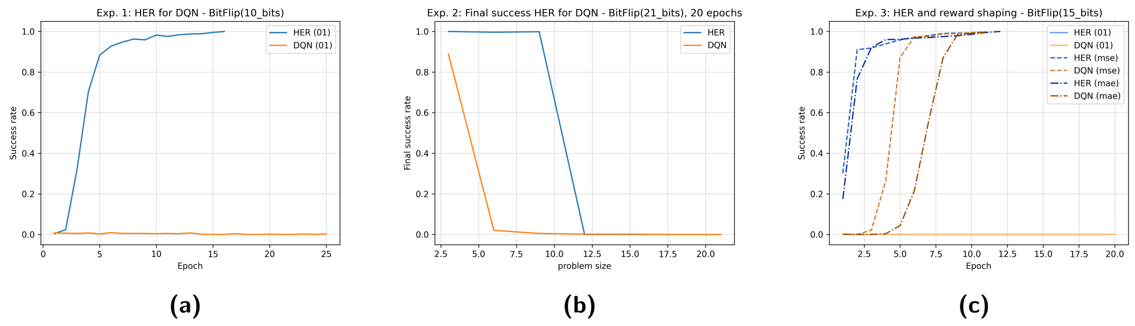


Figure 4: BitFlip experiments with Batch size 10^5 (repeated)

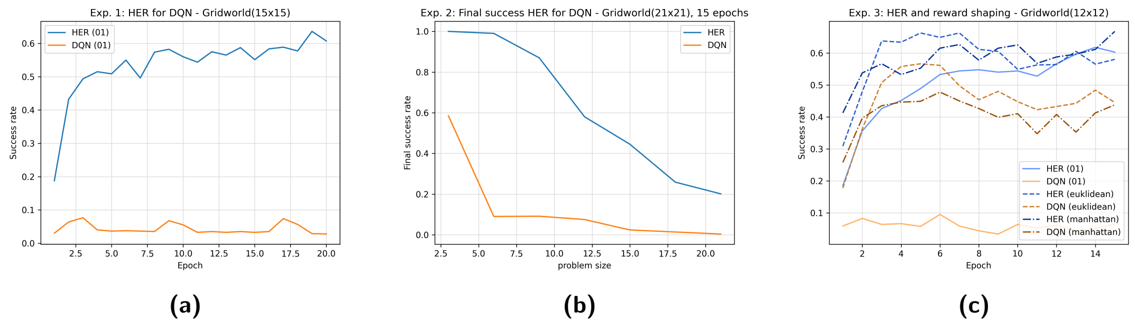


Figure 5: Gridworld experiments with Batch size 10^5

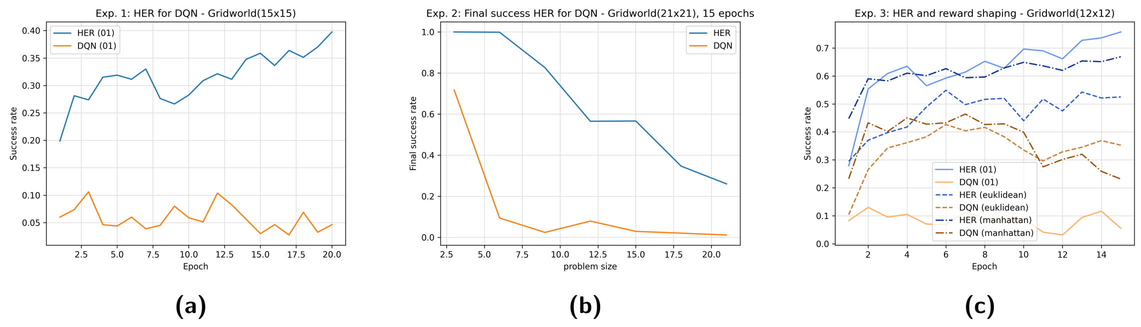


Figure 6: Gridworld experiments with Batch size 10^5 (repeated)