Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update

Brown University

Veda Sunkara, Aansh Shah, Zachary Horvitz, Naveen Srinivasan, Leonard Gleyzer

Abstract: In this work we seek to analyze and reproduce the results presented in "Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update" by Su Young Lee et al. [1], accepted to NeurIPS 2019. The proposed algorithm, Episodic Backward Update (EBU) is a reinforcement learning algorithm that uses direct value propagation to update Q-values. EBU propagates rewards backwards through all transitions of a sampled episode, as opposed to traditional experience replay-based learning, where single transitions are uniformly sampled. The approach is novel in that it is amenable to environments with sparse and delayed rewards. The authors claim performance improvements in such domains, with both deterministic and stochastic transitions, and prove the theoretical convergence of their algorithm in the context of both deterministic and stochastic MDPs. For the NeurIPS reproducibility challenge, we reproduced Figure 1 (EBU stencil), Figure 2 (demonstrations using the MNISTMaze domain), and Figures 4 and 5 as they related to the Breakout Atari Game. Ultimately, our results indicate that EBU offers modest boosts in early convergence across several domains. However, we were 1) Unable to reproduce stable learning in the Atari Environment, and 2) Observed that the One-Step DQN architecture outperformed the authors' One-step base-line in their MNIST Maze environment.

1. Background

This paper was introduced in order to combat issues of sample efficiency for Q-learning agents. The authors seek to address the fact that, in many common reinforcement learning problems, agents only observe sparse and delayed rewards. As a result, during experience replay, there is a low probability of sampling a transition with an immediate reward. Thus, in the early stages of training, an agent that samples uniformly from its history of transitions makes one-step updates that are not meaningful; the later transitions that actually have discounted rewards are not guaranteed to be updated. Therefore, many of these updates are wholly ineffective in teaching the agent to perform a certain task, rendering the work done to do those updates both ineffective and inefficient.

A Deep Q-Network (DQN) is a deep neural network (DNN) used to approximate a Q-function for the value of taking an action at a given state. DQNs traditionally learn this approximation via experience replay; the model samples from the history of past transitions to train the function values. DQN is a function-approximator, and therefore consecutive states have similar Q-values. If DQN were to sample transitions in a backward order, as is the fundamental novelty of the EBU algorithm, overestimation errors could accumulate. As a result, DQN uses uniform random sampling at the potential cost of correlatory sequential information. The authors seek to address this, also, with their algorithm.

There are a number of notable prior works that attempt to address these issues. Some propose new network architectures (like dueling DQN), while others work on restructuring the uniform random sampling strategy (like prioritized experience replay). The proposed approach by Lee et al. is intended to be simple and not require modifications to the DQN architecture, training scheme, or hyperparameters, but rather just the training targets.

2. Main Contributions

The paper proposes a method of performing tabular Q-learning using Episodic Backward Update (EBU). At a high level, the authors propose performing a Q-update by, rather than sampling a single transition uniformly at random from episodic memory, sampling an entire episode and performing Q-updates using all transitions in that episode in a backward manner. Explicit pseudo-code outlining this high level procedure can be found under *Algorithm 1* in the original paper, and will be further discussed in later sections.

To perform EBU in a DQN scenario, the authors introduce the concept of a β diffusion factor. This is to prevent overestimation errors from correlated states from accumulating over updates. Explicit pseudo-code outlining this high level procedure can be found under *Algorithm 2* in the original paper. Concomitantly, they propose Adaptive EBU, which employs a tuning scheme for the diffusion factor β; the optimal value of β depends on the nature of the domain as well as the stage of network training. The authors also prove that EBU theoretically converges to the optimal Q-function in finite and deterministic MDPs.

The authors demonstrate their algorithm on a number of domains. The first is a 2D MNIST Maze, in which each set of coordinates in the grid corres[pmds to a pair of randomly selected MNIST digits. The authors' results demonstrate that EBU outperforms baselines (namely One-Step and N-Step DQN) across two wall densities in a deterministic context. The authors also demonstrate EBU's performance relative to One Step and N-step DQN in the same domain in a stochastic context with a range of β values.

With respect to Atari Games, the authors demonstrate generally the faster convergence of Adaptive EBU over Nature DQN for 49 games across 10 million frames (namely the 49 games of Atari 2600 domain[2]). For two specific games, Gopher and Breakout, they report the test episode score and the respective mean Q-values of all transitions within the testing episode. They show that β values closer to 1 have a tendency to result in overestimated Q-values. The authors explain that the recursive application of the max operation can lead to more and more overestimated Q-values. Without being appropriately attenuated with a relatively small diffusion factor, this method can lead to unstable learning.

For the purposes of this challenge, we sought to reproduce the MNIST Maze results (Figures 1 and 2), as well as the Atari results for the game of Breakout (Figures 4 and 5).

3. Analysis and Goals

Our goal for this challenge was to evaluate the author's claims about EBU's performance relative to baselines by attempting to reproduce their results across a subset of the tasks described in the original paper. We reproduced Figures 1 and 2, as well as subsections of Figures 4 and 5.

For Figure 1, our goal was simply to better understand the EBU algorithm, and compare it to a uniform sampling baseline on a simple task (finite and deterministic MDP). For Figure 2, our goal was to assess the author's claims regarding EBU's performance relative to one step and n step DQN in both deterministic and stochastic contexts. For Figures 4 and 5, our goal was to assess EBU's performance relative to DQN on the Atari game Breakout. As discussed, the authors provided results for several more games, but we limited our scope to a single game due to compute and time limitations.

The authors prove the theoretical convergence of their algorithm to the optimal Q-function for both deterministic and stochastic MDPs. Thus, our goal was to demonstrate their algorithm's performance in both of these contexts.

4. Reproducibility

4.1. Figure 1: Tabular EBU

Reproducing Figure 1 was fairly straightforward. We simply implemented the pseudo-code provided by the authors. One point of confusion was the approach the authors employed for determining the "probability of learning the correct path". While Figure 1 displays the probability of the optimal sequence being learned, the researchers to do not indicate how they computed these values. For example, we do not know if this was computed theoretically, or empirically. To produce our trials, we approximated the probability by running 10000 episodes.

4.2. Figure 2: 2D MNIST Maze

Reproducing Figure 2 was relatively difficult. First, rather than being a well documented task, the MNIST Maze environment seems to have been constructed for this paper. We were unable to find significant references to it in the literature. **This fact alone posed several distinct challenges:**

1. In order to reproduce the authors' work, we first had to write our own version of this environment.
   In the MNIST Maze task defined by the authors, the agent negotiates a 10 by 10 maze, where each state representation is provided by randomly sampling two MNIST digits that correspond to the agent's coordinates. For example, the coordinates: **(1,2)** are represented by the stacked pixels of random 1 and 2 MNIST digits. In the stochastic setting, the agent has a 10% of moving in each orthogonal direction (for example, if the agent takes the 'up' action, it has a 10% chance of going left and a 10% chance of going right)
   We generated our mazes by randomly initializing a Numpy array, where the probability of a wall at each coordinate was set via a wall density parameter. We then solved each candidate maze via A^* to compute the optimal length. If a maze was not solvable, we discarded it. For our MNIST digits, we used the MNIST training corpus. For each trial, we generated a new maze.
   Because we wrote our own environment, the possibility remains that it differs in some way from the environment used for the results in the paper. For example, the authors did not specify what algorithm they used for maze generation.
2. In addition, because MNIST Maze is not a canonical task, the lack of existing baselines made debugging our algorithm difficult. Our difficulties were compounded by slow training times. It took roughly 64 hours to reproduce Figure 2 with 40 trials for each experiment (8 minutes per trial on an NVIDIA $K80$ machine). Because of these time constraints, we only ran 40 trials per experiment, versus the authors' 50.

In addition, we also struggled to interpret the author's implementation details for their models:

1. In the Appendix D the authors discuss their network architecture and hyperparameters. One of these is a minibatch size of 350. While the minibatch parameter is the number of sampled transitions for the vanilla DQN, it was unclear how it applied to EBU.
   There are several possible interpretations. One would be to sample 350 transitions in total, by first sampling one full episode, then sampling a chunk of another episode until 350 transitions had been achieved. Another possible interpretation would be to sample 350 episodes. We ultimately decided that the authors most likely meant that we should sample one entire episode (per their original pseudocode), and use a minibatch of 350 for just the vanilla DQN architecture.
2. Similarly, while the authors specified a memory buffer of size 30000, they did not specify how to account for *episodic memory*. While it was evident that we needed to store 30000 transitions for the vanilla DQN, for EBU and N-Step, we were unsure how this applied. In the end, we chose a memory size of 170 episodes for both EBU and $N-Step$. We decided on this number by dividing 30000 (the number of transitions listed in paper) by the average length of a vanilla DQN episode, which was 175.
   Specific guidelines on how to initialize and manage episodic memory would have been aided our reproducibility endeavour. Such information is critical, because there is evidence in the literature that the different memory schemes change performance [3].
3. Lastly, the authors do not specify how they aggregated the values of their different trials across the x-axis. Some level of interpolation must occur, because episodes are of varying step lengths. We chose to aggregate our data by partitioning the episode lengths by steps of each trial into tranches of size 2000, and taking the mean across these steps.

Ultimately, we able able to produce results that were roughly similar to those cited in the paper (see the **Results** section for output graphs). However, there were some notable differences, including the markedly higher performance of our One-step DQN implementation.

4.3. Figure 4

The authors describe running experiments on the Atari 2600 suite of games. Due to limited time and computational resources, we only aimed to replicate the Figure 4 results with respect to the Atari game Breakout. The authors don't explicitly state which version of Breakout is used for their tests. In Breakout-v0, after your agent selects a particular action at time t, denoted a_t, that action is repeated for the next k timesteps, where k is samples uniformly at random from $\{2, 3, 4\}$. In BreakoutDeterministic-v4, we don't have "sticky actions", and the action chosen at time a_t, is executed for that timestep only. While we ran trials of EBU on both versions of Breakout, ultimately due to time constraints, we ran complete training only on BreakoutDeterministic-v4. [4].

In their paper, the authors train 11 parallel learners with different diffusion factors $\beta = 0.0, 0.1, ..., 1.0$, and they sync all of their learners at the end of each epoch (0.25M frames). This method is referred to as adaptive EBU. Due to the lack of computational resources, we were unable to do end to end training using 11 parallel learners, and have reported values only for EBU DQN $\beta = 0.5$ and $\beta = 1.0$.

In their implementation, they were able to accomplish end to end training of Nature DQN with 8 random seeds and EBU DQN with 4 random seeds. Once again, due to resource constraints, we were only able to do end to end training of Nature DQN and EBU DQN with one random seed.

4.4. Figure 5

The difficulties in reproducing Figure 4 also followed with Figure 5, as they both required completion of the same task. In addition to the previously discussed difficulties with Breakout, it was also unclear how the minibatch of size 32 specified in Appendix D applied to this task. Specifically, it was unclear if it applied to the number of transitions or episodes, and if the former what the cutoff criterion was. As in the 2D MNIST Maze environment, for EBU, we decided the most reasonable course of action was to sample all the transitions in a single episode.

Another difficulty we faced was with replicating the compute environment that the authors used. The authors of the paper used a NVIDIA Titan Xp, whereas we were limited to a Tesla P100-12GB with 40GB of CPU RAM. Our setup increased our end-to-end training time by more than 10x the authors'. Because changes to our implementations took more than 24 hours to train 10M frames in Breakout with our hardware, our development cycle was relatively slow since it was difficult to "fail fast" since this figure was reliant upon results for full end-to-end training. Running robust experimentation on our implementation took several weeks.

In addition, when implementing and running EBU DQN, because the authors used entire episodes (i.e. 18,000 frames) as batch sizes, we were unable to do use the same batch size given our memory constraints. Consequently, we had to split each episode into even smaller batches, which extended training times significantly.

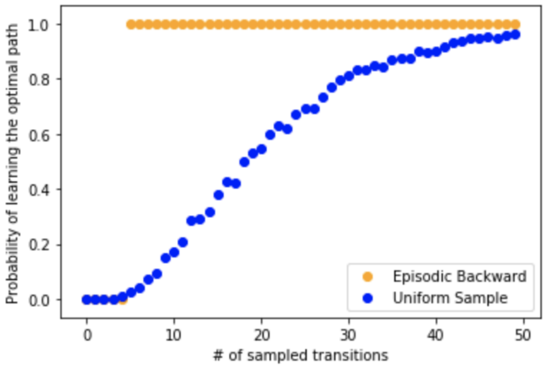
As in the 2D MNIST Maze environment, the authors did not specify the number of episodes to be stored in the memory buffer. We experimented with an informal grid-search across different split sizes for the batches and across different episodic replay memory sizes in an attempt to identify the best parameters. However, this also significantly increased the amount of time that it took to train our EBU DQNs. Furthermore, the authors did not specify what library versions they used. We used Tensorflow version 1.14, but believe that there is a possibility that we would get different results with a different version of Tensorflow.

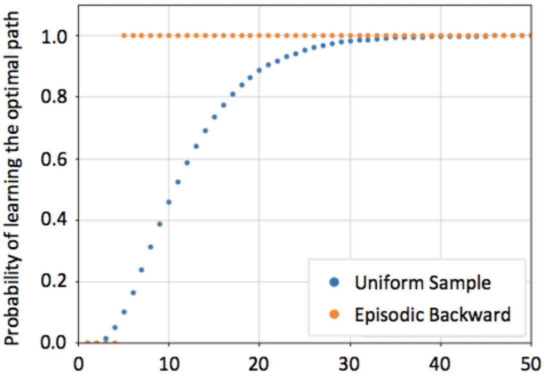
5. Results and Contributions

5.1. Figure 1

To reproduce Figure 1, we simply implemented the provided skeleton EBU algorithm, as well as an algorithm that performed uniform sampling. We were able to reproduce their results for EBU on this toy example.

We can see the reproduced results on the left, and the original results from the paper on the right, below.





Note that our uniform sampling result is slightly lower performing than the author's implementation, which may have to do with implementation details such as sample size (we used 1000 samples) and random libraries. Nonetheless, we were able to generally reproduce the graph in terms of the performance of EBU relative to uniform random sampling in the context of this toy MDP.

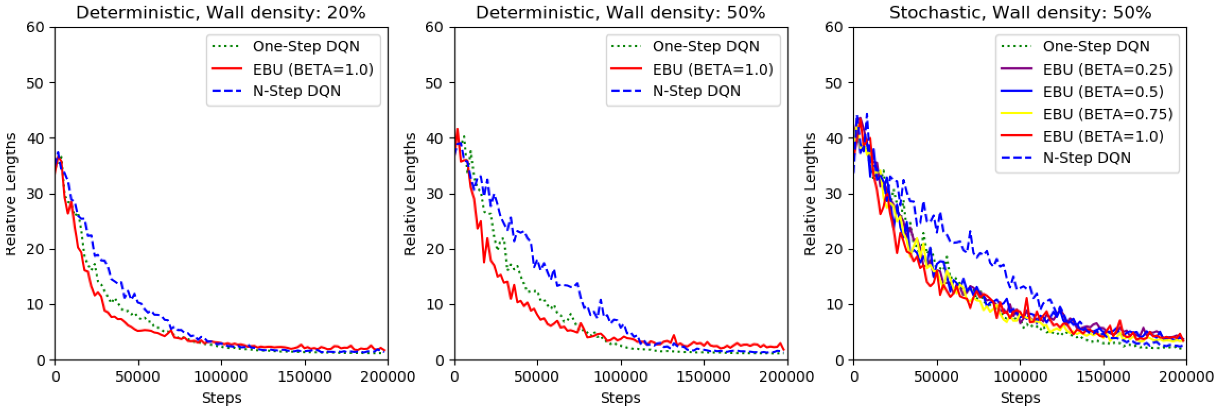
5.2. Figure 2

To reproduce Figure 2, we implemented the provided skeleton EBU algorithm for DQN, as well as one-step and n-step DQN. We utilized experience replay for both of these algorithms. We implemented MNIST Maze generation with randomly placed walls. In order to avoid situations with unsolvable mazes, we implemented and used the A^* algorithm to first solve a maze and ensure it had a solution before using it for training.

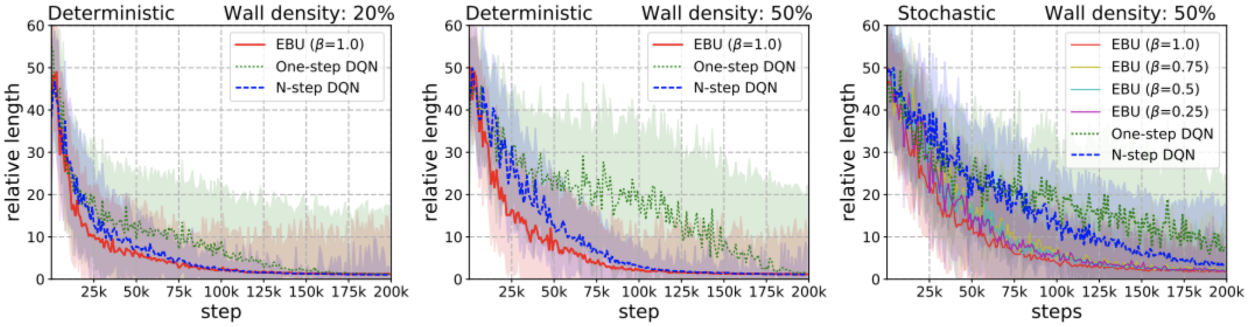
Beyond the assumptions discussed in the **Reproducibility** section on this report, we utilized the same network structure as that which was used in standard DQN, with small adaptations to fit the given task.

Below, we present our graphs, followed by the original graphs from the paper.

5.2.1. Reproduced Results



5.2.2. Original Results



We were fairly successful in reproducing the author's original results, as can be visualized by the parity across the two figures, and the similiar results across Table 1 and Table 2. However, there are a few critical differences. First, our one-step DQN implementation converged much more quickly than than the n-Step model across all experiments (In fact, they look reversed on the original graph). After $100,000$ steps, it even performed better than *EBU*.

Of note, the other columns in the table are comparable; the means and medians of EBU and n-Step are relatively close to the results the authors reported. This may indicate that the authors' table just understates the ability of a one-step DQN, and with it the importance of One-step DQN's independent uniform sampling of transitions.

Wall Density	EBU ($\beta = 1.0$)	One-step DQN	N-step DQN
20%	(3.00,5.69)	**(2.37,5.10)**	(2.93,5.20)
50%	(3.69,7.20)	**(2.51,6.12)**	(5.78,10.42)

Table 1: Reproducibility Challenge: Comparing Medians and Means after 100k Steps

However, earlier on in the episode, *EBU* appears to first indeed converge faster than one step DQN across the trials. These results are included in Table 3, and indicate that, at least initially, EBU converges more quickly.

Wall Density	EBU ($\beta = 1.0$)	One-step DQN	N-step DQN
20%	(2.42,5.44)	(9.25,14.40)	**(2.24,3.26)**
50%	**(2.34,5.51)**	(16.62,23.36)	(3.12,11.32)

Table 2: Su Young Lee et al.: Comparing Medians and Means after 100k Steps

Wall Density	EBU ($\beta = 1.0$)	One-step DQN	N-step DQN
20%	**(5.21,9.3)**	(7.66,10.14)	(10.3,13.42)
50%	**(8.10,10.05)**	(12.42,14.91)	(18.09,21.9)

Table 3: Reproducibility Challenge: Comparing Medians and Means after 50k Steps

As stated in the previous section, there are several possibilities for why these differences occur. For example, because the authors did not specify memory buffer behavior, it is possible our implementations and, thus results diverge. In fact, we found preliminary results that the N-Step DQN can outperform One Step DQN when it uses an episode memory of 1000. However, we did not have the resources to test multiple buffer sizes. In addition, that would not have affected the performance of the One-step DQN model.

In addition, it is possible that our 2D MNIST Maze implementations were sufficiently dissimilar, given the ambiguity about the environment we discussed in the previous section. However, our results are evidence that the one-step DQN model's transitions are highly effective, even in an environment with sparse rewards. Thus, in the 2D MNIST Maze, EBU shows promise with its early convergence properties. However, future work needs to further establish its performance relative to one-Step DQN approaches. Our empirical results seem to indicate that in the long run (after 200K steps), EBU DQN is outperformed by Nature DQN. This gives weight to the argument that the Q values generated via episodic backwards update have some nontrivial drawbacks, including potentially increasing convergence time from a near optimal policy to an optimal policy. This observation could be attributed to the overestimated Q values caused by the recursive max operation in updating correlated samples in a sequence and the increased likelihood in network divergence as we increase our value of β.

5.3. Figure 4

Our results differ dramatically from the authors' results. The authors cite that their adaptive EBU DQN was able to attain a score $7.882x$ the baseline score from Nature DQN when trained for ten million frames. For our replication, we defined test episode score as the **highest** score attained during an evaluation period. We were able to replicate the authors' baseline score from Nature DQN, but we were not able to replicate their results for EBU DQN. After ten million frames, our implementation of Nature DQN was able to attain a score of 137.17, and our implementation of EBU DQN was able to attain a score of 5.21 for a relative score of $0.038x$.

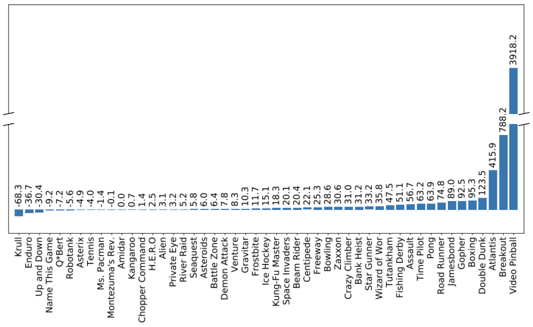
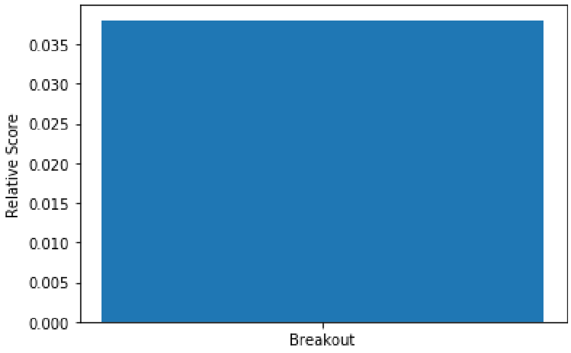


Figure 4: Relative score of adaptive EBU (4 random seeds) compared to Nature DQN (8 random seeds) in percents (%) both trained for 10M frames.

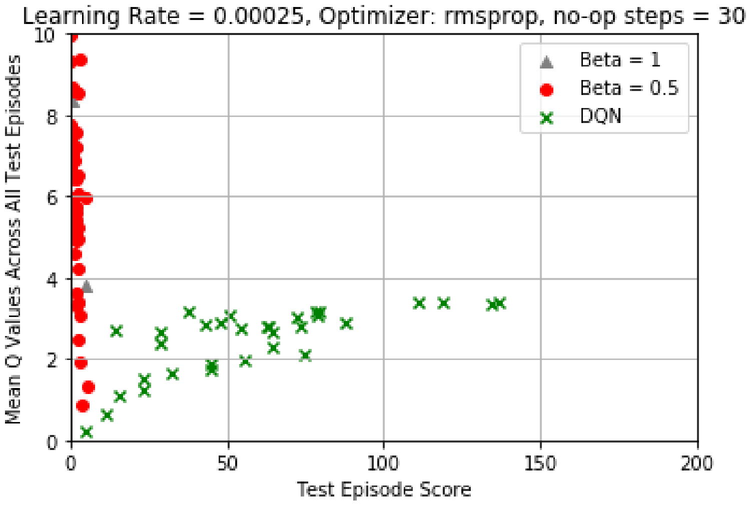


5.4. Figure 5

With the Episodic Backwards Update method, we are using a rich function approximator and performing updates using correlated states in a sequence. In order to mitigate the influence of overestimated values being compounded through recursive max operations, the authors introduce a parameter β, which they state can be considered as a learning rate for the temporary Q table or ”degree of backwardness” for updates. In Figure 5, observe how we were able to replicate the way EBU DQN causes an overestimation in values due to the sequential update of correlated states. As expected, this overestimation is more pronounced when $\beta = 1$ as opposed to $\beta = 0.5$, since when $\beta = 1$ we have less attenuation of overestimated values in our reward update vector y_k (see Algorithm 1 pseudocode). Although we did replicate this overestimation bias, our implementatation of EBU DQN was not performant to degree reported in the original paper. Instead of attaining a test episode score of 200 and 100 for EBU DQN with $\beta = 0.5$ and $\beta = 1$ respectivally, we attained a test episode score of 10 and 170. Unlike their implementation, we didn't observe an EBU DQN that could simultaneously have overestimated Q values and still be performant.

In order to reproduce Graph 5, we adapted a Nature DQN implementation and modified the parameters to match those specified in the paper[3]. We changed the learning rate to 0.00025, set the optimizer to RMSProp, and increased the number of no-op steps to be 30. We also modified the training behavior policy to anneal linearly from 1 to 0.1 for the first 4M frames. As demonstrated below, we were able to replicate the results from the baseline Nature DQN, but were not able to replicate the EBU DQN results.

5.4.1. Reproduced Results



5.4.2. Original Results

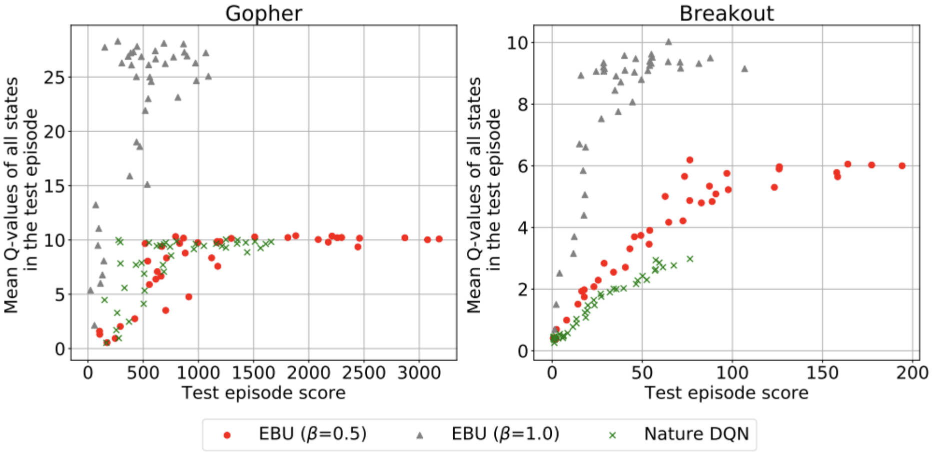


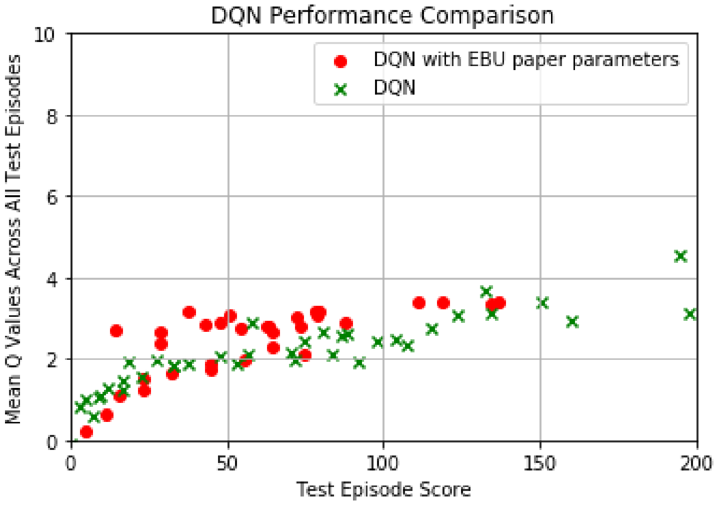
Figure 5: Episode scores and average Q-values of all state-action pairs in “Gopher” and “Breakout”.

Furthermore, we noticed that the author's baseline Nature DQN performed significantly worse than the scores reported in the original Nature DQN paper. The original Nature DQN paper reported an average test score of 168 on Breakout after 10M frames, whereas the author's results don't pass 100. We were not sure why the author's choose hyperparameters that significantly decreased the performance of the original baseline.

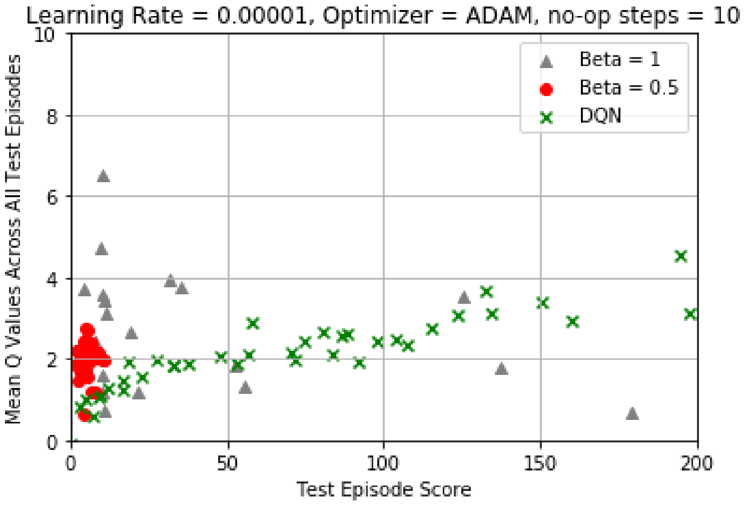
In addition, in the original Nature DQN paper, the behavior policy during training was ϵ-greedy with ϵ annealed from 1.0 to 0.1 for the first 1M frames and then was fixed at 0.1 after. However, in the author's paper, they annealed for the first 4M frames.

We were able to find other hyperparameters that replicated the original Nature DQN results: learning rate = 0.0001, optimizer = Adam, no-op steps = 10. We also modified the training behavior policy to be in line with the original Nature DQN paper. As seen in the graphs below, the DQN with the new hyperparameters outperforms the DQN with the hyperparameters from the paper. In addition, when we evaluated EBU with $\beta = 0.5$ and 1, we observed significantly worse results for $\beta = 0.5$ and see some learning for EBU with $\beta = 1$. We can see this reflected in the following two graphs, which we generated.

5.4.3. Hyperparameter Performance Comparison



5.4.4. Results with New Hyperparameters



6. Conclusion

In this work, we presented a reproducibility analysis of the NeurIPS 2019 paper "Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update" by Lee et al. While their paper includes thorough experimentation, the researchers did not explicitly include several specifications and training methodologies, which would be necessary for the complete reproduciblity of their presented results.

However, we were able to make reasonable assumptions based on the algorithm design and problem domains, and thus were able to achieve results similar to those presented in the original paper in the Tabular and 2D MNIST Maze environments.

In the 2D MNIST Maze task, we were able to show that, early on, Episodic Backward Update *does* learn more quickly than the established baselines of One step and N-step DQNs. However a striking contrast was that, our implementation of One-step DQN drastically outperformed the baseline included in this paper, and scored higher than EBU on all experiments after $100,000$ steps. This indicates that One-step DQN still performs very well in sparse-reward environments, even when compared to EBU.

In addition, we were not able to successfully reproduce the author's results for the Atari game Breakout. There are several reasons why this may have occurred, which we discussed in the previous section. They include ambiguity of Breakout version, batch size specification, and replay memory construction. Yet, while our implementation of EBU for Breakout under-performed, we observed many of the behaviors documented by the authors, included the relationship between the diffusion factor β and degree of overestimation of Q values.

In summary, EBU shows promise in environments with sparse rewards. However, in future work, we would like to see a more thorough comparison across a range of memory buffer sizes and batching parameters on the 2D MNIST Maze task. Also, in order to facilitate the reproduction of results on the Atari 2600 suite of games, future work should establish apt hyperparameter configurations for EBU-DQN. These specifications should include information about sampling, batching, memory buffers.

Resource constraints were a perennial struggle throughout this project. As powerful deep learning-based methodologies flourish in Reinforcement Learning, they will continue to extend that which is possible in the field. Yet simultaneously, the community must face the pitfalls: compute and time. These realities pose distinctly challenging problems for reproducibility.

7. Acknowledgements

We would like to thank our mentor TA, Neev Parikh, for providing help with the scope and plan for this reproducibility project, as well as providing debugging help and access to compute resources.

When we started observing results from our implementation of 2D MNIST Maze that differed from the authors' reported results, we reached out to Lee et al. and received their implementation of 2D MNIST Maze. After a conversation with our course teaching assistants, we elected to not look at or utilize the 2D MNIST Maze implementation provided by the authors.

8. Code

Our code is available here: https://github.com/vedasunkara/EBUReproducibilityChallenge.

We strongly advocate for and stand with the PapersWithCode movement, which encourages authors of academic papers in computer science to include a Gitub link to their implementation code with each publication in addition to saved weights of performant models.

References

[1] S. Y. Lee, S. Choi, and S. Chung, "Sample-efficient deep reinforcement learning via episodic backward update," *CoRR*, vol. abs/1805.12375, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[3] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," *CoRR*, vol. abs/1710.06574, 2017.

[4] F. M. Graetz, "How to match deepmind's deep q-learning score in breakout," Apr 2019.