California State University, Fresno

Final Report

A comparison between DQN & Double-DQN

Miguel A. Ibarra-Gallardo
CSCI 166
Professor Ruby
11/24/2025

The Setup

The setup for this project was simple, though there were a few features I wished to implement. The original code with the addition of the Double-DQN loss calculation was sufficient for the experiment, but I wanted to automate the tracking of variables and displays, making it easier for myself in the long run. I located and implemented libraries that enabled me to display changes in variables in tables and record tables containing information from training. This required changing how the variables were accessed, yet it was worth the extra effort.

Beyond this, I found it necessary to set up the environment in a way that allowed for subsequent testing of both Single-DQN and Double-DQN without too much supervision for long training periods. This was achieved by simply having a toggle that switched between loss calculations and switching it before restarting the entire process. This meant that once done, we would have a table from those mentioned before that displayed all relevant data regarding the two calculations, allowing for easy comparison.

About the Domains

For this experiment, I had decided to try something different from what was inherently required for this project. I had decided to test both loss calculations against two different games to really grasp the benefits of each and to see if one's benefit can really only be tied to certain games or just learning in general. Due to this, I picked one simple game (e.g., Pong-v5) and one more complex game (e.g., Breakout-v5) for the comparison.

About the Model Variants

The model used in this experiment is the standard DQN given to us. It contains two variants: the single DQN, which came standard, and the Double-DQN, which was chosen by me. As stated previously, this experiment focuses on comparing these two model variants to find strengths, weaknesses, use scenarios, and even the effects hyperparameters have on them. This is done in the same architecture, allowing for fair testing. This is important as single-DQN is known to overestimate Q values, leading to unstable learning, while Double-DQN circumvents this issue by using two networks for action selection and action evaluation. This means differences in parameters can really benefit one

or harm the other in certain cases, making it an important topic to study.

The Hyperparameters

In this section, I will discuss the effect of hyperparameter manipulation with regard to single and Double-DQN learning. In the initial baseline test, it was observed that, of the two, Double-DQN benefited the most from the default state of the hyperparameters, experiencing better, more consistent, and rapid growth after 150,000 frames compared to the single-DQN variant. This is similarly seen with the Breakout training, though it is notable that past the 190,000th frame, learning came to crawl as the epsilon bottomed out. This happened even earlier in the Double-DQN at around 150,000 frames.

With this in mind, minor and controlled alterations were made with the intent of improving the model's speed, learning stability, and total reward. These changes included the modification of batch size, learning rate, gamma/discount factor, target network sync frequency, and epsilon schedule parameters. This will reveal what parameters will not only help but also harm performance while exhibiting how sensitive these models can be to these changes.

The initial change made to the hyperparameters involved doubling the batch size allotted, which was in an effort to give stability to the data by reducing variance in Q values, while assisting computation efficiency, meaning that with a larger batch size, the GPU can process more information every update while reducing per-sample overhead. This had the effect of improving the AI's ability to learn substantially for the single DQN variant while having almost negligible effects on the Double-DQN, leading to a difference of 0.59 between baseline and configuration 1.

Following this, changes were made to the Learning Rate, which had a massive effect. To begin, I doubled the learning rate of the model to 2e-4(0.0002) from the baseline 1e-4(0.0001) in an attempt to increase the rate at which the model made changes. I wanted to accelerate the rate at which the model updated parameters and subsequently learned, as learning took long periods of time between small improvements. Ultimately, this led to the AI learning quicker and getting more confident for the single-DQN, though it had the unintended effect of completely preventing the Double-DQN from learning at all. The single-DQN reached the minimum epsilon at 151,861 frames, while in the previous base, this was reached at 152,931, and reached the lowest reward at that point at -11.32 before time was

up. As mentioned before, Double-DQN did not learn and ended with the reward of -21 and epsilon of 0.998453. This is likely due to the learning rate making it so that the online network updates faster than the target network, which could lead to useless information.

Resulting from this, I decided to simply lower it instead, dropping the learning rate to 7e-5 (0.00007) from the baseline, leading to an interesting outcome. It was observed that single-DQN improved once again by almost 4 points, less than the test before, yet it had the benefit of reducing the damage done to the Double-DQN learning ability. Though it must be noted that it did experience a decrease of at least 13 points. This leads to the conclusion that both aggressive and stable learning work well for single-DQN models, while Double-DQN doesn't benefit from the learning rate being too much of the other. This is once again because Double-DQN already has the stabilisation built into its calculations, and the aggressive learning provides too rapid data updates to make any sense of it.

Afterwards, I decided to try and change how rewards were handled by modifying the discount factor for the games. I lowered the gamma/discount factor by 0.02, lowering it from 0.99 to 0.97, meaning that the AI would focus more on immediate rewards vs. future, which had a significant benefit for the single-DQN model. It was able to score approximately 5.00 more points compared to the baseline configuration, though once more, the Double-DQN suffered roughly as much loss as the single-DQN gained. This is because a reduction in gamma means a reduction of preference for planning for the future. Normally, this would not be a problem, but due to single-DQN's problem with overestimation, focusing on the future creates much more noise, creating unstable Q-values. Due to this, that model variant benefited greatly from the lower gamma value, while Double-DQN can handle planning for the future without the overestimation, so reducing its ability to do so would lower its performance.

Moving forward, the next hyperparameter chosen was the sync target frames, which were both lowered and raised to observe the differences in training. When lowered from 1000 to 500, the single-DQN model benefited by roughly 1 point; meanwhile, the Double-DQN suffered, majorly falling behind the baseline by approximately 6 points. Similarly, when raised from 1000 to 1500, single-DQN gained roughly 8 more points compared to the baseline, a massive improvement, while

Double-DQN once more fell behind by around 1 point. This once again can be explained by Double-DQN's ability to stabilise by decreasing the sync target frames; data is updated too quickly, meaning that it's often too similar to the previous, while, when increased by a large extent, the data becomes outdated. Single DQN benefits from both due to its innate instability, meaning low target frames stabilise and ensure that the data is dated, while higher values ensure that the transitions between values are smoother as the target network experiences less violent shifts.

For the final hyperparameter, I decided to experiment with epsilon factors, mainly the final epsilon value. Originally, I lowered the value from 0.01 to 0.005 to reduce randomness and help the single-DQN model learn more efficiently by following the best values more closely. This increased by approximately 11 points the biggest improvement yet in all these modifications, which of course came with the caveat of reducing the performance of the Double-DQN. For the initial reduction of the epsilon final value, it allowed the model to commit to the behaviours that score it better points rather than making more random actions, and because of this reduces random noise that can make Q values unstable. Comparatively, Double-DQN, being already really stable, can find itself stuck in sub-optimal policies that lead to early convergence. When increased, the randomness helps single-DQN avoid stagnation and learn more, while for Double-DQN, the random actions generated are injected into an increasingly near-optimal policy, leading to mild score reductions.

On the Learning Curves

Upon review, the curves tell a very interesting story of how the model learned throughout the training cycle. On baseline configurations, the Double-DQN variant began learning at an accelerated state once it passed the 150,00 mark, before which both models were relatively equal. Though this, it is evident that with most changes, single-DQN benefited much more than Double-DQN; this could be attributed to DQN's sensitivity to these changes. This variant is well known for its tendency to overestimate q values and thus is very unstable, so these changes, which were selected mainly to assist in stabilisation, would understandably improve that variant much more than they would for Double-DQN, which already stabilises itself.

It is also of note that the graph regarding Breakout is more subdued than that of Pong due to

the difference in scoring. This means when reviewing the Breakout graphs against the pong, normalisation may need to get a perfect comparison. Though that being said, it's not wholly necessary, as one can draw good insights from them.

The Results

Across both Pong and Breakout, the experiments revealed clear differences in how the single-DQN and Double-DQN models learn and how they respond to changes in hyperparameters. The two domains served as useful contrasts. Pong offered dense, rapid feedback with a clear win/loss structure, while Breakout had sparser, more prolonged scoring patterns. Despite these differences, both environments showed a consistent trend. The single-DQN variant is highly sensitive to hyperparameter changes, while Double-DQN is far more stable but far less responsive to tuning.

Throughout the experiments, adjustments such as increased batch size, modified learning rate, reduced discount factor, altered sync frequency, and changes to the epsilon schedule often produced large improvements for single-DQN, sometimes boosting its performance by five to eleven points. These same changes frequently harmed Double-DQN, which already stabilises its targets internally. Whenever the tuning reduced stability or limited exploration too early, Double-DQN's performance fell noticeably. This contrast demonstrates that Double-DQN performs strongly under default configurations but degrades quickly when pushed outside its optimal stability range.

Breakout results were more muted overall, largely due to its reward structure and slower progression. However, the same relationships between algorithms were still present. Even with limited training time, both models plateaued near 150,000 frames regardless of hyperparameter improvements. With more time or more efficient throughput, Breakout could likely be driven to convergence, but within the project's constraints, its curves consistently flattened earlier than Pong's. This suggests a broader point about DQN-based methods: while they can learn competent policies in some domains with moderate tuning, their scalability and sample efficiency vary dramatically by environment.

A Reflection

When deciding how to begin this project, I had many games in mind that I would have loved to try. I knew originally I wanted to start with pong, then compare the results of the hyperparameter

change to games such as Mario, bowling, or even Battlezone. Saying this, the changes required to make those games work would've been against the nature of this examination, too many changes would have needed to be made to the hyperparameters, and to test if these two variants react similarly would be futile. Due to this, the two games were selected so that the models could face a notable increase in challenge and change in environment.

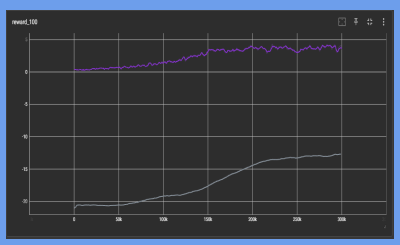
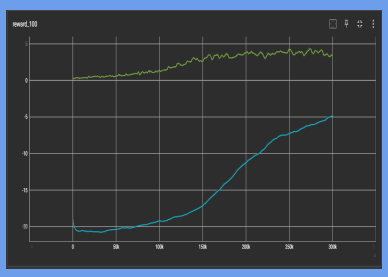
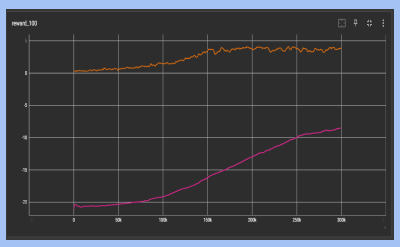
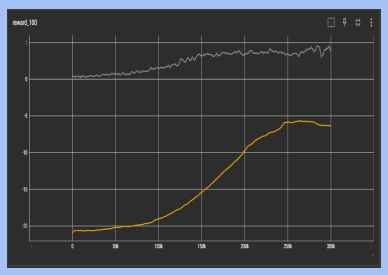
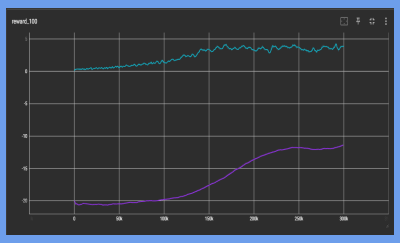
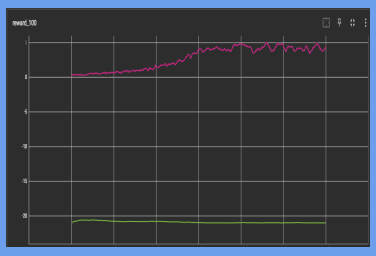
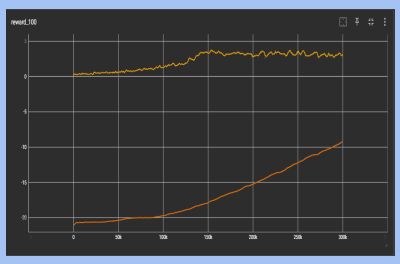
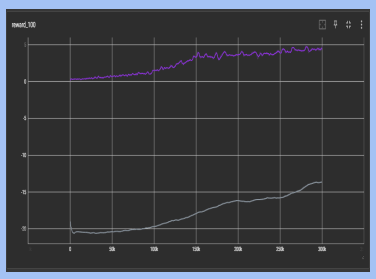
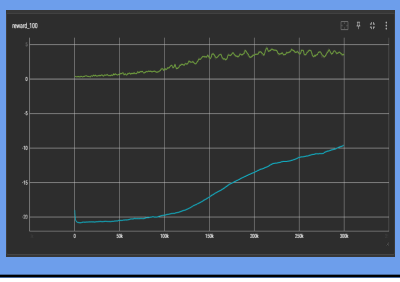
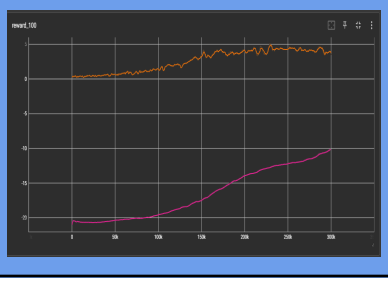
Though in testing, even with these two games, challenges were encountered. Issues with the model reaching a premature convergence left my original tests useless, as after even 10-12 hours of training, nothing was learned. In addition to the model seeming to be basically CPU-bound led to the decision that I set 300,000 frames to be the limit for each run, still netting us good data while not relying on convergences of the model. In the future, I would like to try Prioritized Replay due to those issues mentioned before. This addition would help the model hopefully learn better from its mistakes and escape/prevent early convergence by putting focus on said mistakes.

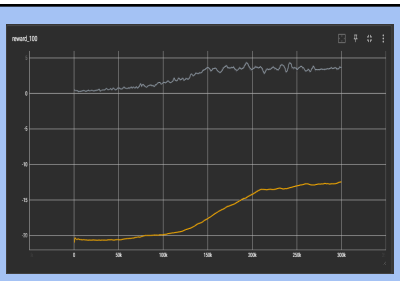
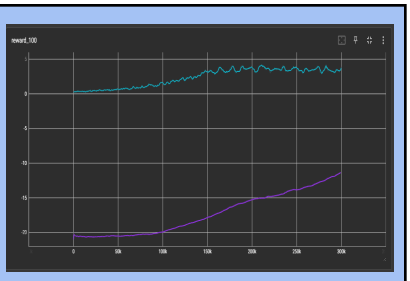
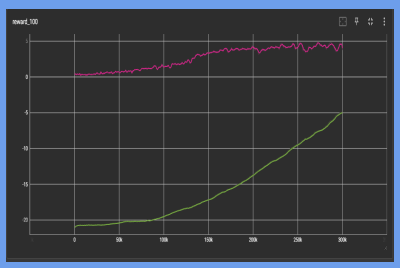
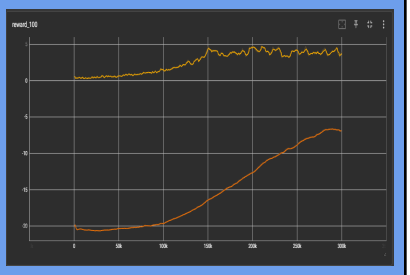
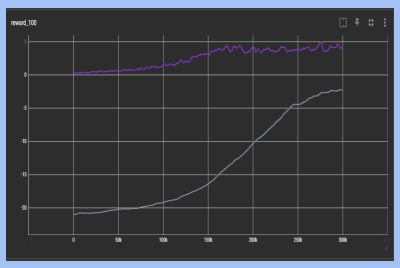
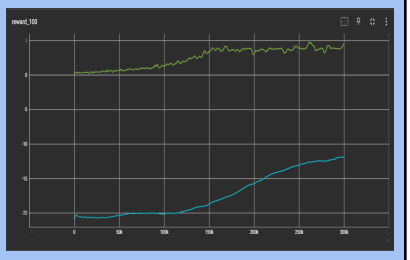
Appendix

Figure 1: Hyperparameters

Version	Timestamp	Mean Reward Bound	Gamma	Batch Size	Replay Size	Learning Rate	Sync Target Frames	Replay Start Size	Save Epsilon	Epsilon Decay Last Frame	Epsilon Start	Epsilon Final	Notes
0	2025-11-24 07:20:24	19	0.99	32	10000	0.0001	1000	10000	0.5	150000	1	0.01	Baseline run
1	2025-11-24 08:46:48	19	0.99	64	10000	0.0001	1000	10000	0.5	150000	1	0.01	Increase stability, try to help the GPU
2	2025-11-24 10:56:18	19	0.99	32	10000	0.0002	1000	10000	0.5	150000	1	0.01	Increased Learning rate to encourage more aggressive learning
3	2025-11-24 15:07:35	19	0.99	32	10000	7e-05	1000	10000	0.5	150000	1	0.01	Decreased Learning rate to encourage more stable learning
4	2025-11-24 21:33:35	19	0.97	32	10000	0.0001	1000	10000	0.5	150000	1	0.01	Decreased gamma to prioritise more immediate rewards
5	2025-11-24 22:59:08	19	0.99	32	10000	0.0001	500	10000	0.5	150000	1	0.01	Decreased sync target frames to encourage more frequent updates
6	2025-11-25 05:17:38	19	0.99	32	10000	0.0001	1500	10000	0.5	150000	1	0.01	Increased sync target frames to encourage less frequent and more stable updates
7	2025-11-25 09:21:56	19	0.99	32	10000	0.0001	1000	10000	0.5	150000	1	0.005	Decrease epsilon final to make the agent greedier and less random
8	2025-11-25 10:46:36	19	0.99	32	10000	0.0001	1000	10000	0.5	150000	1	0.02	Increase epsilon final to make the agent less greedy and more random

Figure 2: Log photos

Index	DQN	Double-DQN
0		
1		
2		
3		
4		

5		
6		
7		
8