Assignment 3

Reinforcement Learning Programming - CSCN8020

NOVEMBER 12, 2025

PAULA MARCELA RAMIREZ

ID 8963215

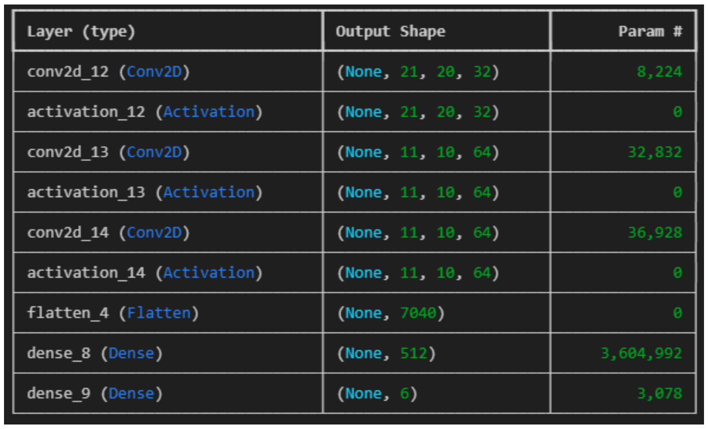
In this Report

REPOSITORY: https://github.com/paulamrz-c/CSCN8020_A3_PR

Final Network Architecture

For training the agent in the ALE/Pong-v5 Atari environment, a Deep Q-Network (DQN) algorithm was implemented to allow the model to learn optimal actions over time.

The architecture used was a Convolutional Neural Network (CNN) designed to process four consecutive frames as input, with the following layers:



```
Layer (type)                    Output Shape         Param #
conv2d_12 (Conv2D)              (None, 21, 20, 32)     8,224
activation_12 (Activation)      (None, 21, 20, 32)         0
conv2d_13 (Conv2D)              (None, 11, 10, 64)    32,832
activation_13 (Activation)      (None, 11, 10, 64)         0
conv2d_14 (Conv2D)              (None, 11, 10, 64)    36,928
activation_14 (Activation)      (None, 11, 10, 64)         0
flatten_4 (Flatten)             (None, 7040)               0
dense_8 (Dense)                 (None, 512)        3,604,992
dense_9 (Dense)                 (None, 6)              3,078
```

1st Convolutional Layer + ReLU

```
model = Sequential()
model.add(Conv2D(32, (8, 8), strides=4, padding='same', input_shape=self.state_size))
model.add(Activation('relu'))
```

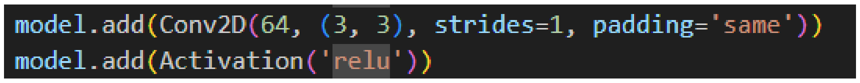
- ***Input Shape:*** The input shape was 84x80x4, because the agent used 4 stacked grayscale frames. This allowed the network to capture movement and temporal information.
- ***Filters:*** 32 filters of size 8x8 were used. This helps detect basic visual features from the game screen.
- ***Stride:*** A stride of 4 was applied, moving the filter 4 pixels at a time to downsample the image and reduce computation.
- ***Padding:*** 'same' padding was used so that relevant features near the edges were not lost.
- ***Activation:*** ReLU (Rectified Linear Unit) was applied to introduce non-linearity and help the model learn complex spatial patterns in the frames.

2nd Convolutional Layer + ReLU

```
model.add(Conv2D(64, (4, 4), strides=2, padding='same'))
model.add(Activation('relu'))
```

- ***Filters: 64 filters of size 4x4:*** Increasing the number of filters allows the model to capture more complex and higher-level. The smaller 4x4 kernel helps focus on finer visual details, such as the ball's movement or small paddle adjustments.
- ***Stride:*** The stride was set to **2**, which reduces the spatial dimensions while keeping enough resolution to track relevant motion in the game.
- ***Padding:*** 'same' padding was maintained to avoid losing information along the .
- ***Activation:*** ReLU, It was applied again to maintain non-linearity and ensure efficient gradient propagation during training.

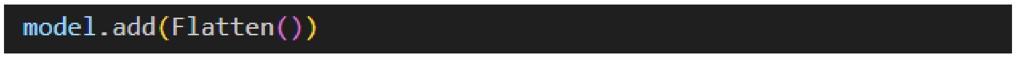
3rd Convolutional Layer + ReLU



```
model.add(Conv2D(64, (3, 3), strides=1, padding='same'))
model.add(Activation('relu'))
```

- ***Filters: 64 filters of size 3x3:*** We keep the same number of filters but now 3x3 to help the model see smaller details.
- ***Stride:*** A stride of 1 helps the network learn very small movements or positions.
- ***Padding:*** 'same' keeps the output size the same as the input.
- ***Activation:*** ReLU.

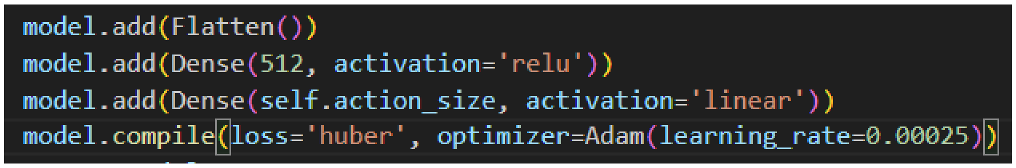
5th Flatten Layer



```
model.add(Flatten())
```

That means that after the last convolution, the three-dimensional outputs (11, 10, 64) are "flattened" into a one-dimensional vector of size 7040.

4th Two Fully Connected Layers (Dense Layers) - 512 and 6 neurons



```
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(self.action_size, activation='linear'))
model.compile(loss='huber', optimizer=Adam(learning_rate=0.00025))
```

- ***Dense Layer (512 neurons):*** This fully connected layer helps the network understand the main patterns found by the previous layers and learn complex connections between them.
- ***Dense (Output) Layer***: Has 6 outputs, corresponding to the six possible actions in the Pong environment (NOOP, FIRE, UP, DOWN, UP + FIRE, DOWN + FIRE). Each output gives a Q-value used to decide what action to take. A linear activation is used because Q-values can be any number, positive or negative.

Default Hyperparameters

The hyperparameters established for this model were the following:

Hyperparameter	Value	Description
Discount factor (γ)	0.95	Balances short-term and long-term rewards.
Exploration rate (ε initial)	1	Starts with full exploration using random actions.
Exploration decay (δ)	0.995	Gradually reduces exploration after each update.
Minimum exploration (ε_min)	0.05	Keeps a small amount of exploration at the end of training. (epsilon-greedy)
Mini-batch size	8 *(default)* / 16 *(in experiments)*	Number of samples used for each network update.
Target network update rate	10 *(default)* / 3 *(in experiments)*	Number of episodes before syncing with the target network.
Learning rate (α)	0.00025	Step size for the Adam optimizer.

Metrics

Below are the metrics that are recorded and analyzed in TensorBoard and the printing instructions during training.

Metric	Description
Score per episode	Total game score obtained by the agent in each episode. How well the agent performed.
Average reward (last 5 episodes)	It helps visualize the agent's recent performance and stability over time.
Loss	The training loss of the Q-network, showing how the predicted Q-values differ from the target.

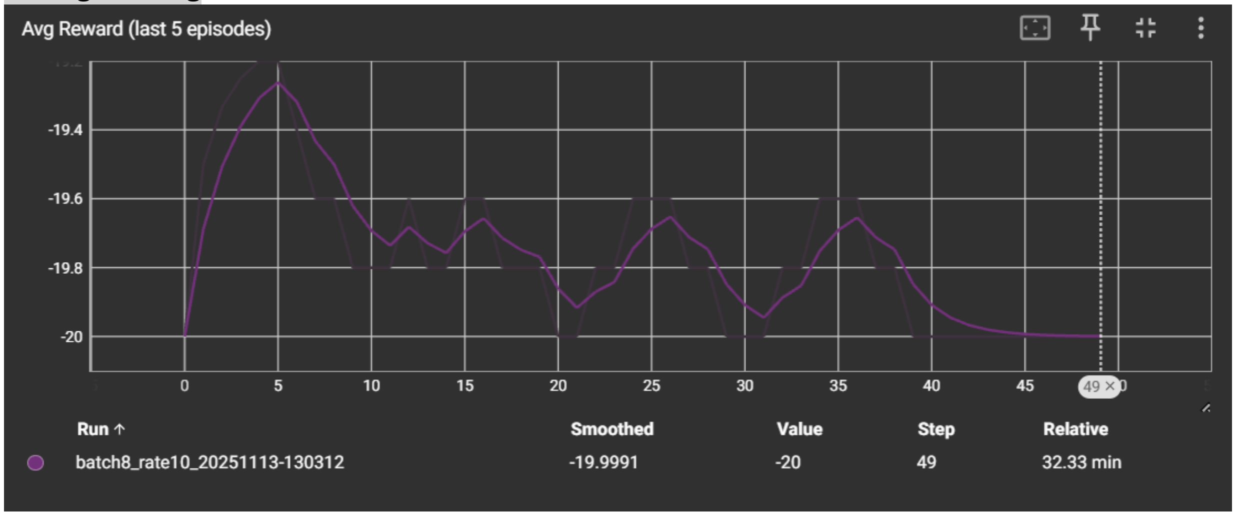
Hyperparameter Tuning Experiments - 50 Episodes

Note: The experiments were limited to 50 episodes due to a technical issue when trying to activate the GNU environment, which prevented running longer training sessions.

mini-batch size: 8 (default) - update rate of the target network: 10 (default)

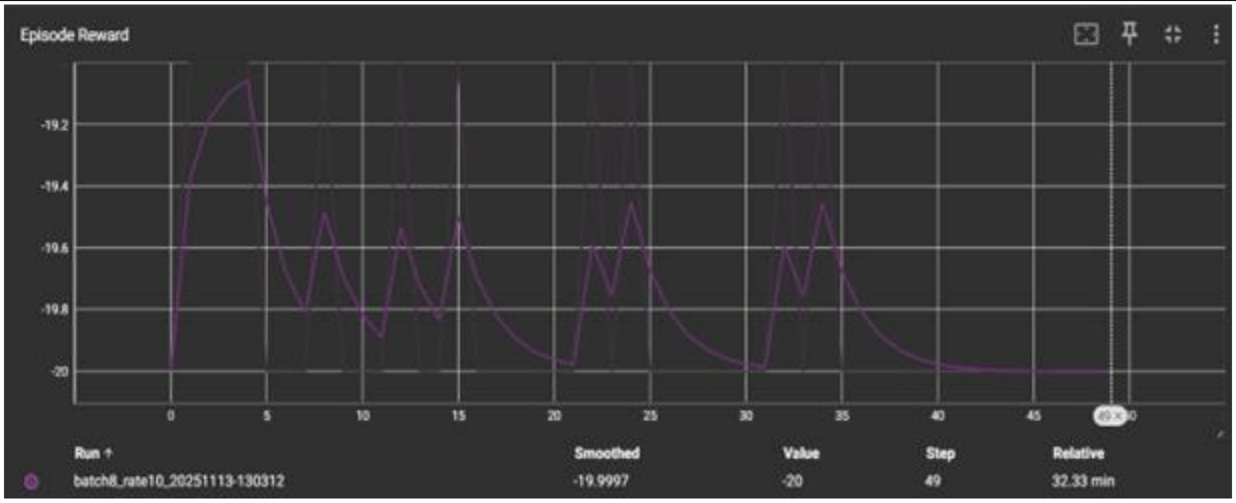
Leaving the defaults; batch size to 8 and the update rate to 10 steps. It can be noticed that:

This metric stays around -20, showing that the agent is still losing most games. The line fluctuates slightly at the beginning and then stabilizes, indicating little improvement during training.



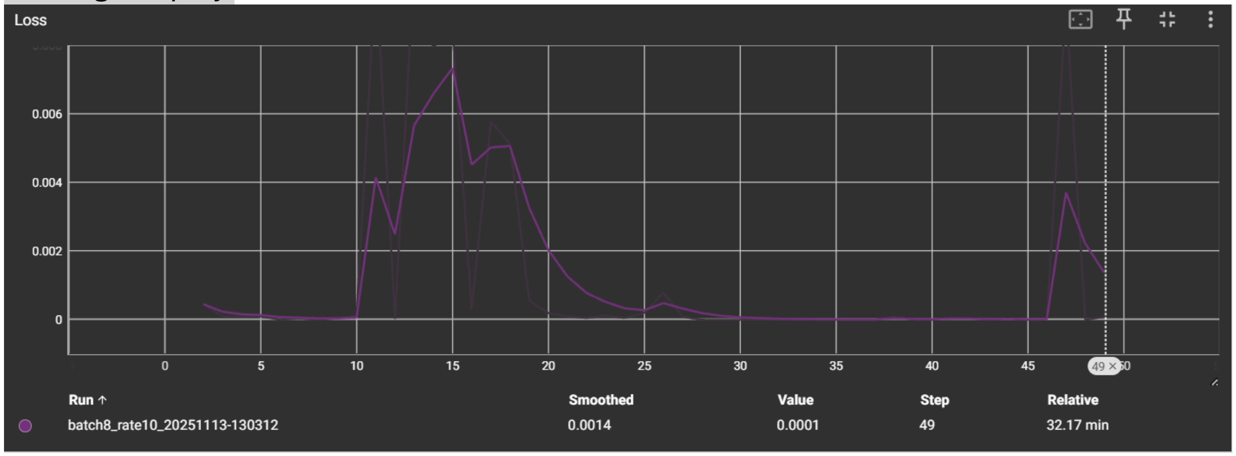
Run ↑	Smoothed	Value	Step	Relative
● batch8_rate10_20251113-130312	-19.9991	-20	49	32.33 min

The episode reward follows a similar pattern, staying close to -20. The agent's performance varies at first but quickly returns to a constant low score, showing that it has not yet learned an effective strategy.



Run ↑	Smoothed	Value	Step	Relative
● batch8_rate10_20251113-130312	-19.9997	-20	49	32.33 min

The loss decreases early in training and remains low for most episodes. This means the model updates are small and stable, although this stability does not yet translate into better gameplay.



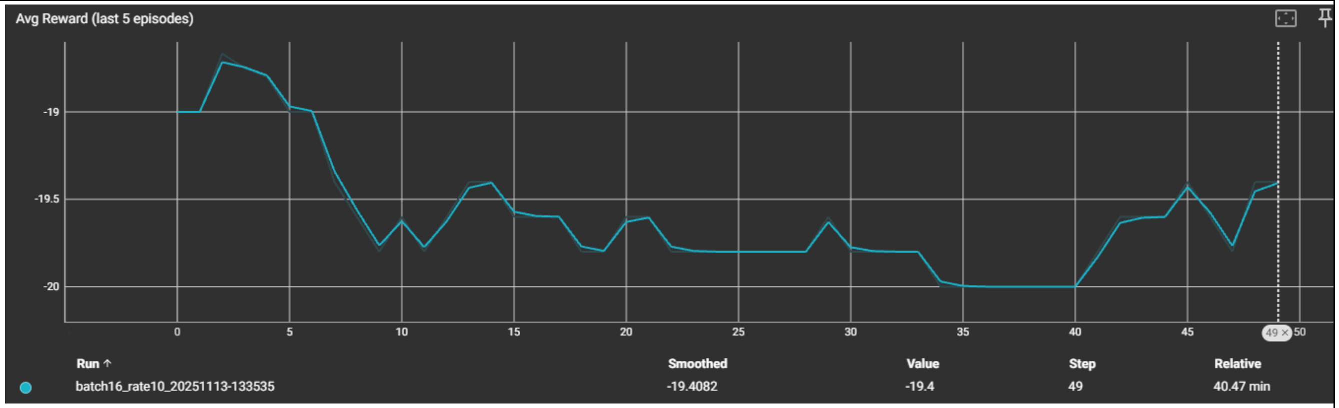
Run ↑	Smoothed	Value	Step	Relative
● batch8_rate10_20251113-130312	0.0014	0.0001	49	32.17 min

The agent remains at a beginner stage, with stable but poor performance. It is learning to predict Q-values consistently, but more episodes, higher replay memory, or parameter tuning would be needed for the agent to start improving and win in the Pong environment.

mini-batch size: 16 - update rate of the target network: 10 (default)

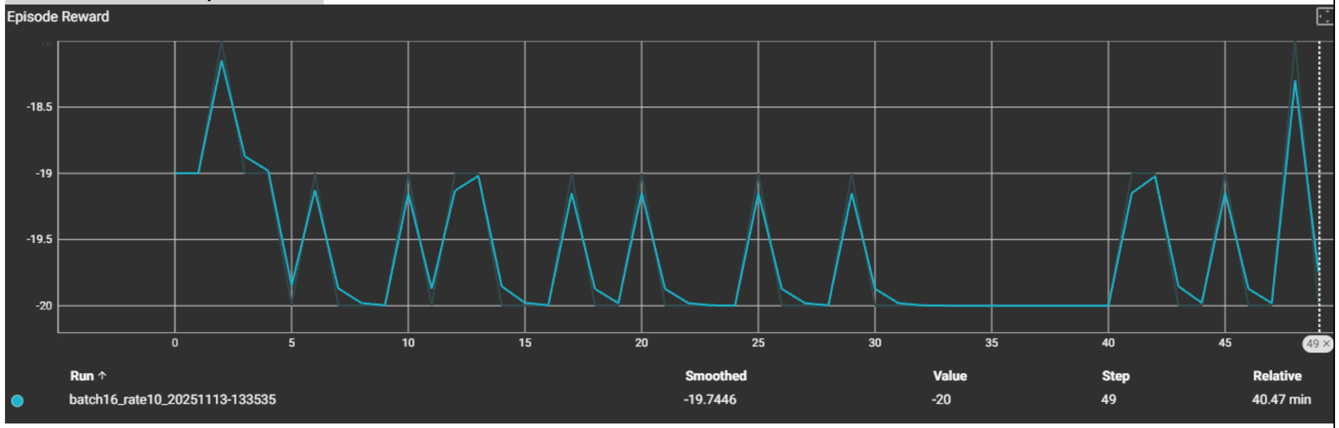
When changing the batch size to 16, it can be noticed that:

The average reward stays close to -19.5, showing slight ups and downs but no real improvement in the agent's performance.



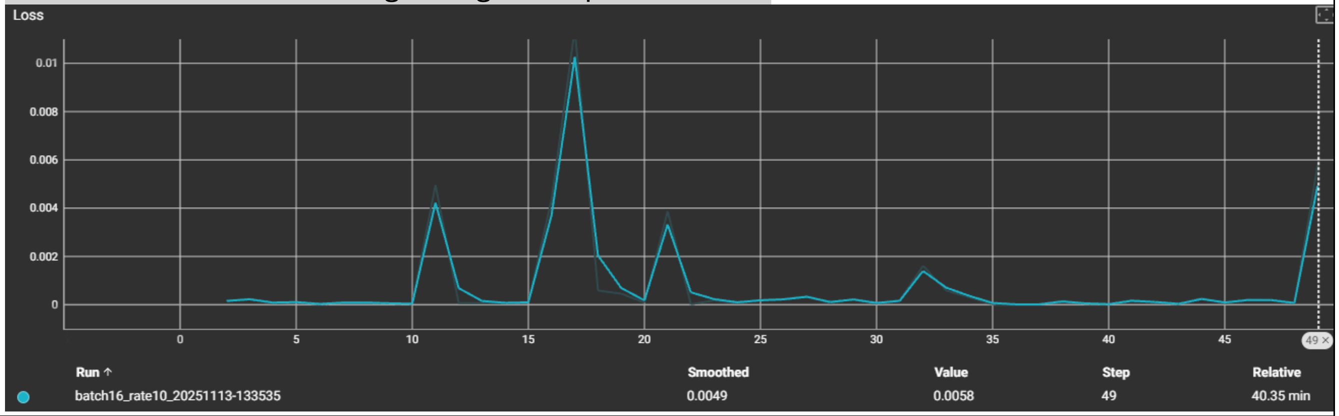
Run ↑	Smoothed	Value	Step	Relative
● batch16_rate10_20251113-133535	-19.4082	-19.4	49	40.47 min

The episode reward also remains around -20, meaning the agent continues to lose most games even after 50 episodes.



Run ↑	Smoothed	Value	Step	Relative
● batch16_rate10_20251113-133535	-19.7446	-20	49	40.47 min

The loss values are small and stable after some early spikes, indicating that learning updates are consistent but not strong enough to improve results.



Run ↑	Smoothed	Value	Step	Relative
● batch16_rate10_20251113-133535	0.0049	0.0058	49	40.35 min

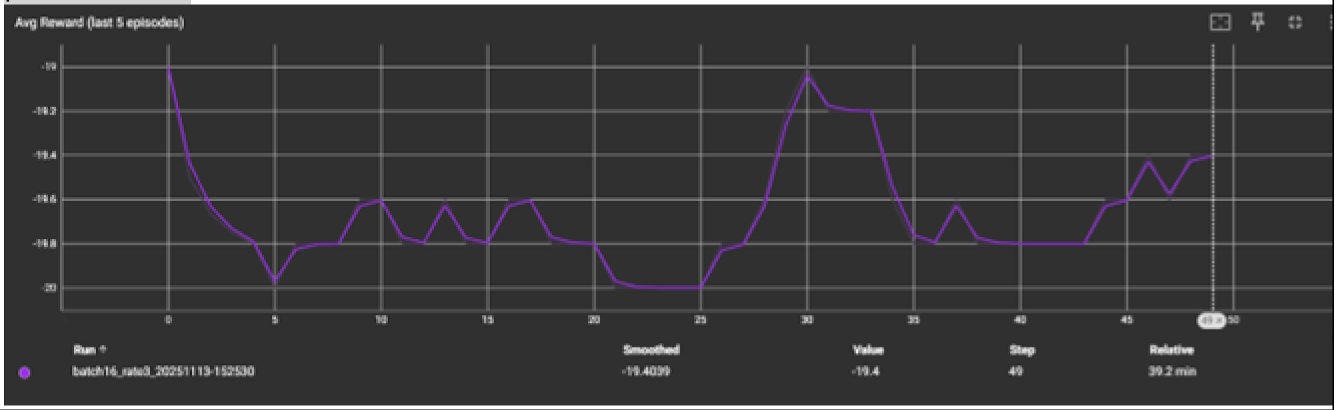
The model with batch size 16 shows similar behavior to batch 8, they are stable but with poor performance.

Also, the agent has not yet learned effective strategies and needs more training or further tuning of hyperparameters.

mini-batch size: 16 - update rate of the target network: 3 steps

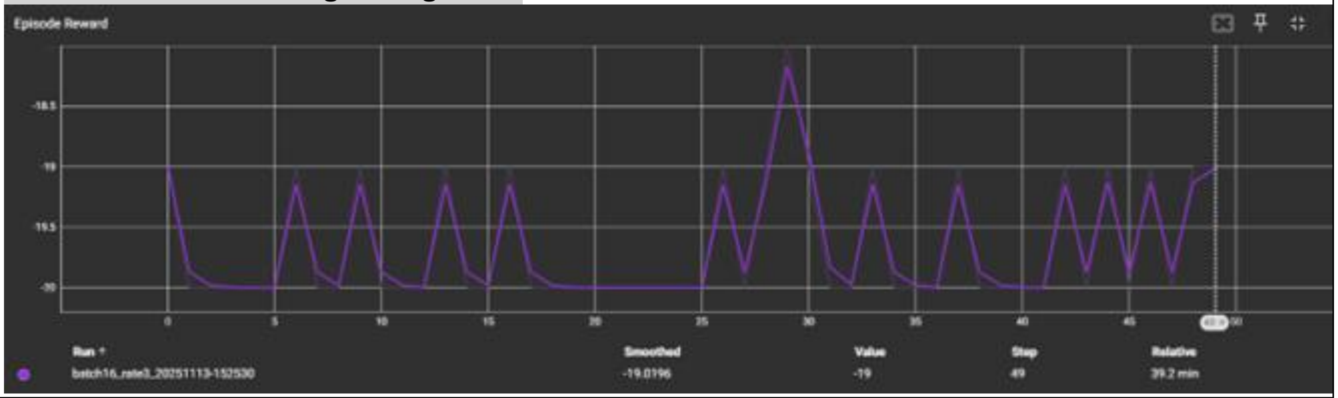
When reducing the update rate to 3 steps, the same pattern was observed; The reward obtained per episode was -20 more frequently. Considering the above, this behaviour is related to the fact that the batch size is smaller, which means fewer samples per update.

It stays around -19.4, with small peaks and drops. The trend shows stable but still weak performance.



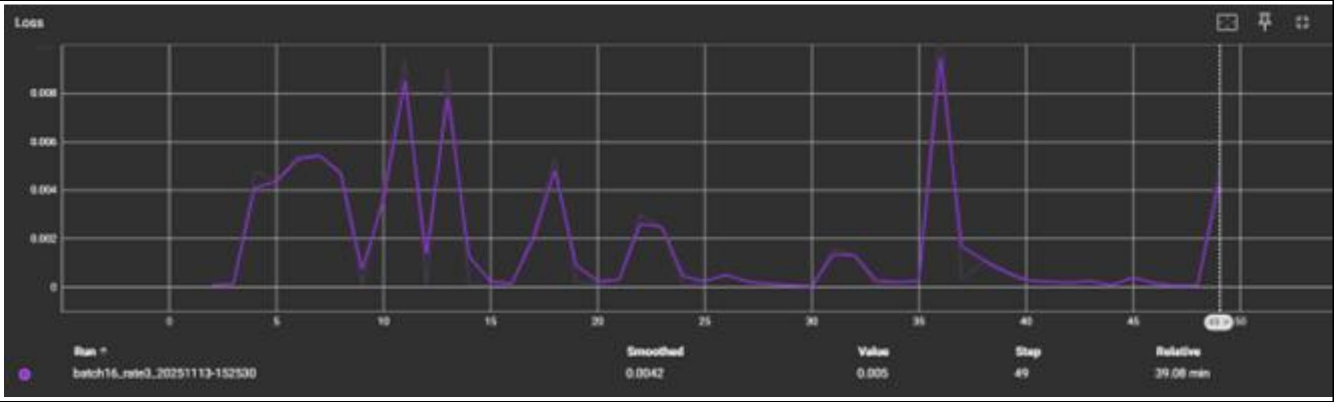
Run ↑	Smoothed	Value	Step	Relative
batch16_rate3_20251113-152530	-19.4039	-19.4	49	39.2 min

Episode rewards vary between -20 and -19, meaning the agent performs slightly better than random but is still losing most games.



Run ↑	Smoothed	Value	Step	Relative
batch16_rate3_20251113-152530	-19.0196	-19	49	39.2 min

The loss shows small oscillations and a few spikes early in training, then stays low. It seems the network updates are stable.



Run ↑ | Smoothed | Value | Step | Relative
batch16_rate3_20251113-152530 | 0.0042 | 0.005 | 49 | 39.08 min

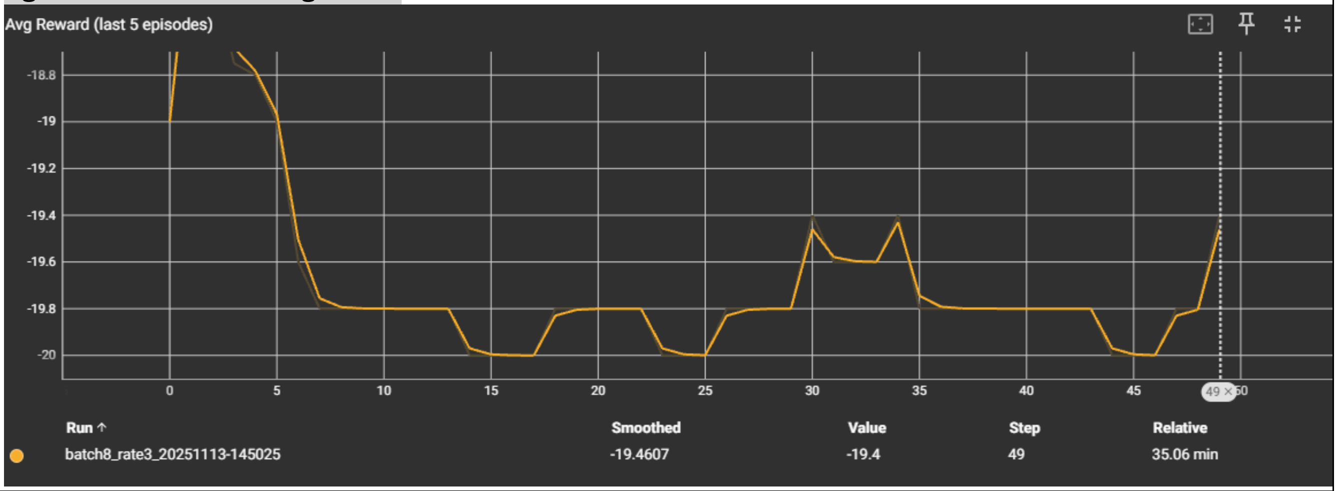
With batch 16 and update rate 3, the model remains stable and consistent but still does not show real improvement.

More training or stronger parameter adjustments would be needed for the agent to start winning games.

mini-batch size: 8 (default) - update rate of the target network: 3 steps

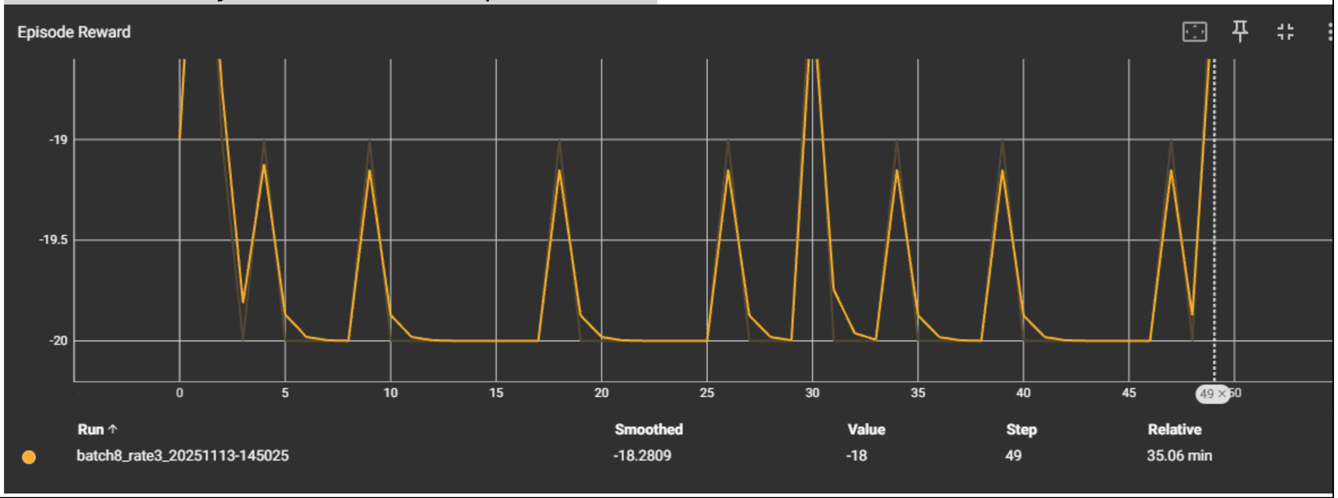
When changing the batch size to 8 and the update rate to 3 steps, it can be noticed that the rewards per episode are slightly better.

The average reward slightly improves at first but then stabilizes around -19.5, showing that the agent still loses most games.



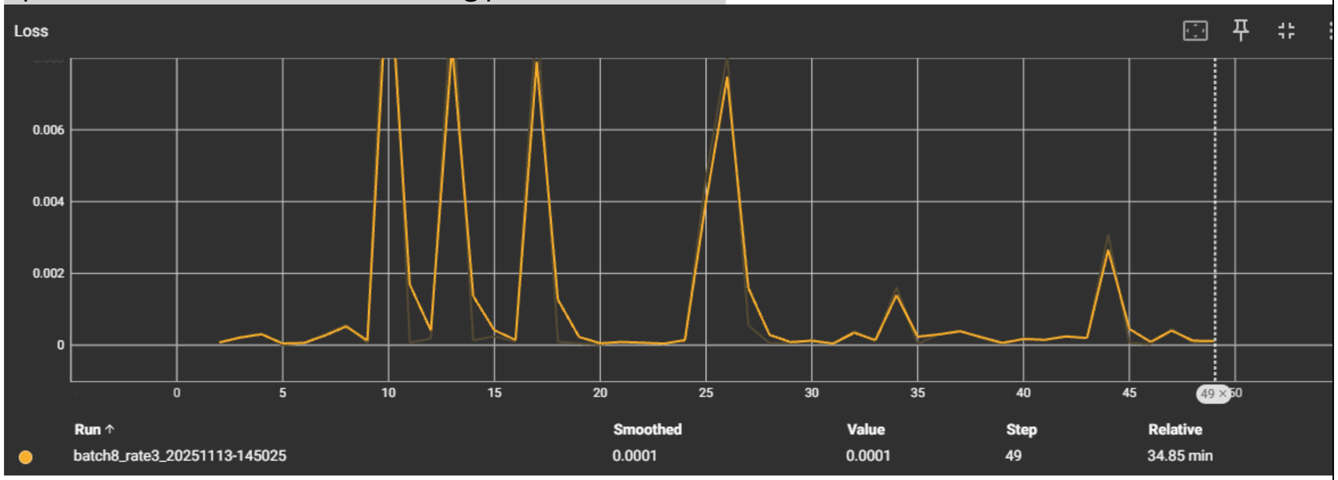
Run ↑ | Smoothed | Value | Step | Relative
batch8_rate3_20251113-145025 | -19.4607 | -19.4 | 49 | 35.06 min

The episode reward fluctuates between -18 and -20, meaning the agent performs inconsistently but occasionally achieves small improvements.



Run ↑	Smoothed	Value	Step	Relative
● batch8_rate3_20251113-145025	-18.2809	-18	49	35.06 min

The loss shows several spikes early in training but remains very low overall, suggesting that updates are small and the learning process is stable.



Run ↑	Smoothed	Value	Step	Relative
● batch8_rate3_20251113-145025	0.0001	0.0001	49	34.85 min

The model with batch size 8 and update rate 3 shows minor improvements compared to other tests but still stays close to -20.

The agent has not yet learned effective strategies and would need more training episodes to improve performance.

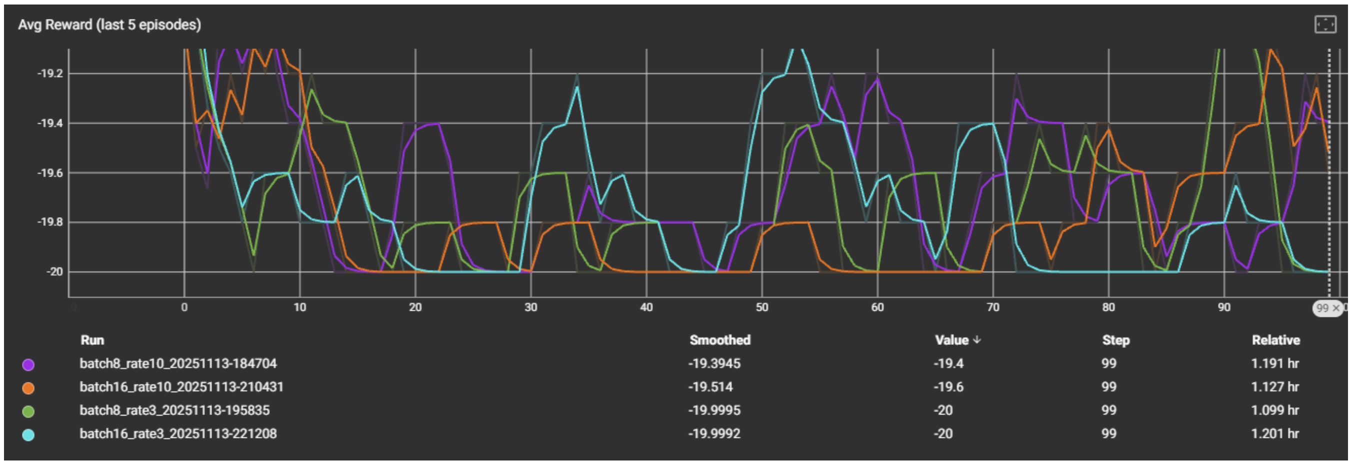
Hyperparameter Tuning Experiments - 100 Episodes

Since the previous runs with 50 episodes were not enough for the agent to show any learning progress, I decided to increase the number of episodes to 100.

The idea was to give the model more time to explore the environment, adjust the Q-values, and reduce the randomness caused by the high exploration rate (ε = 1.0 at the beginning).

With 100 episodes, the agent could perform longer training cycles, allowing a better comparison of how each hyperparameter combination affects the learning behavior.

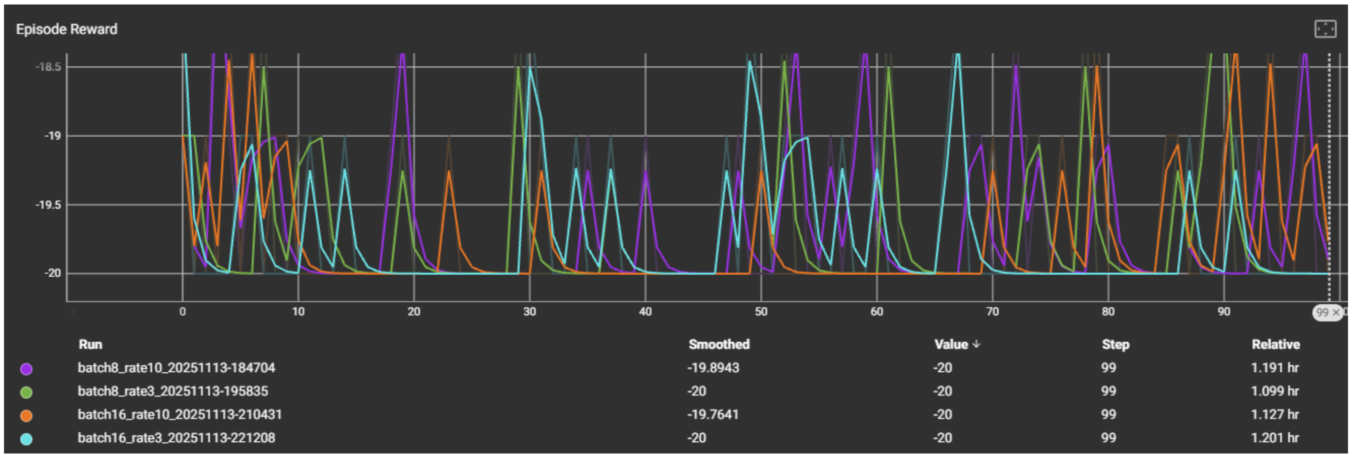
Avg Reward (last 5 episodes)



	Run	Smoothed	Value ↓	Step	Relative
●	batch8_rate10_20251113-184704	-19.3945	-19.4	99	1.191 hr
●	batch16_rate10_20251113-210431	-19.514	-19.6	99	1.127 hr
●	batch8_rate3_20251113-195835	-19.9995	-20	99	1.099 hr
●	batch16_rate3_20251113-221208	-19.9992	-20	99	1.201 hr

When comparing all four configurations, the average reward remains between -20 and -19, showing that none of the setups produced significant improvement. However, the configuration with batch size 16 (orange and blue lines) appears slightly more stable and shows fewer drops, indicating a smoother and more consistent learning process.

Episode Reward

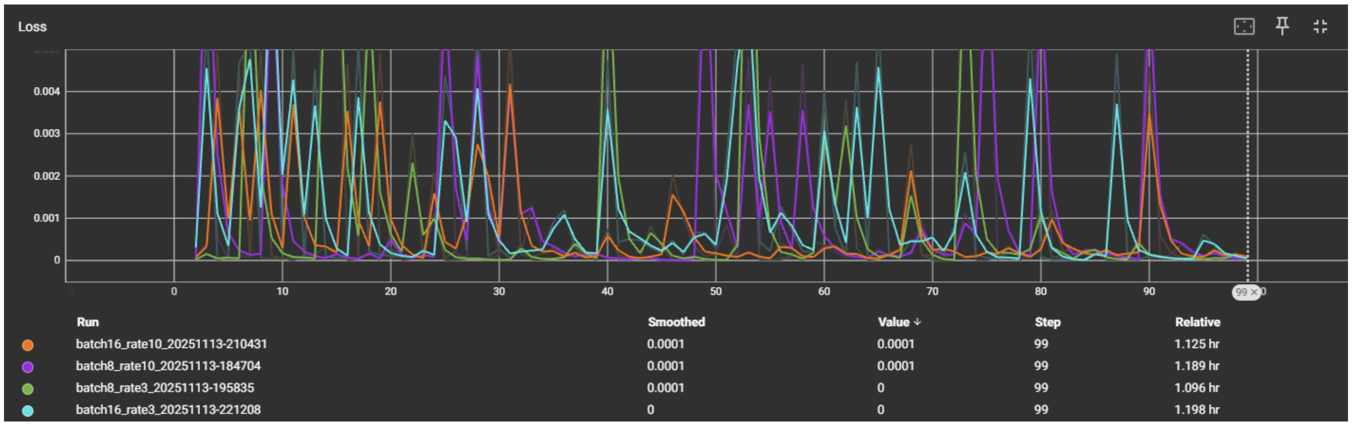


	Run	Smoothed	Value ↓	Step	Relative
●	batch8_rate10_20251113-184704	-19.8943	-20	99	1.191 hr
●	batch8_rate3_20251113-195835	-20	-20	99	1.099 hr
●	batch16_rate10_20251113-210431	-19.7641	-20	99	1.127 hr
●	batch16_rate3_20251113-221208	-20	-20	99	1.201 hr

Across all configurations, the episode rewards stay between -20 and -19, showing that the agent still loses most games. However, runs with a **batch size of 16 (orange and blue lines)** appear slightly smoother, suggesting more stable behavior during training.

Loss Function

Across all configurations, the loss values remain very low, showing that the model updates are stable. Although small spikes appear throughout training, the loss quickly returns to near zero, meaning the Q-value predictions are consistent across all runs.



	Run	Smoothed	Value ↓	Step	Relative
●	batch16_rate10_20251113-210431	0.0001	0.0001	99	1.125 hr
●	batch8_rate10_20251113-184704	0.0001	0.0001	99	1.189 hr
●	batch8_rate3_20251113-195835	0.0001	0	99	1.096 hr
●	batch16_rate3_20251113-221208	0	0	99	1.198 hr

When training with 50 episodes, the agent showed almost no improvement, keeping rewards close to -20. After increasing to 100 episodes, the training became more stable, and small variations appeared in the average reward. Although the agent still loses most games, the longer training allows it to explore more and slightly improve its consistency.

Conclusion

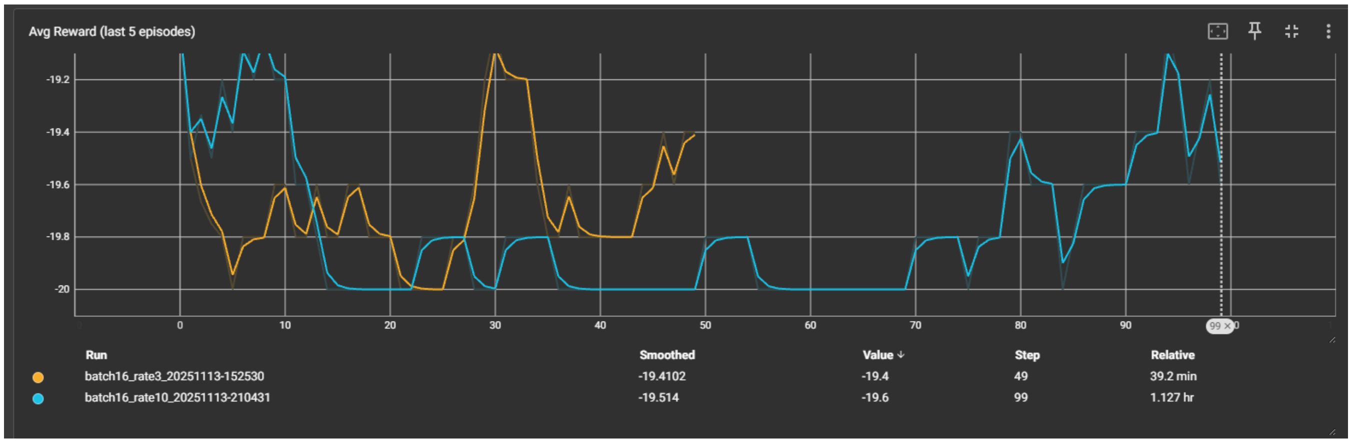
Comparing these plots, I can notice that keeping the same batch size and changing the rate at which the target model is updated resulted in similar behaviour, with the difference that with 3 episodes, the agent seems to get a -20 reward less often.

On the other hand, I can see that after 50 episodes, the reward of the model continues to be too low (between -18 and -20), which may reflect that the model is struggling to learn or that it will take more episodes to get better. However, if the Loss plots are analyzed, the differences between the Q-values and the target Q-values decrease, reflecting that the model is getting closer to the true values.

Configuration	Episodes	Batch size	Update rate	Average reward	Observations
1	**50**	8	10	≈ -20	Stable but with no improvement; the agent keeps losing all games.

2	**50**	16	10	≈ -19.4	Slightly more stable, with lower variation and smoother loss.
3	**50**	8	3	≈ -19.4	Small improvement at the beginning but remains close to -20; slow learning.
4	**50**	16	3	≈ -19.4	More stable, with small improvement peaks and lower loss. It is the most consistent configuration.
5	**100**	8	10	≈ -19.4	Stable but with no improvement; the agent keeps losing all games.
6	**100**	16	10	≈ -19.6	Slightly more stable, with lower variation and smoother loss.
7	**100**	8	3	≈ -20	Small improvement at the beginning but remains close to -20; slow learning.
8	**100**	16	3	≈ -20	More stable, with small improvement peaks and lower loss. It is the most consistent configuration.

In conclusion, the model performs better when using a batch size of 16, as this setting consistently provides smoother training and lower reward fluctuations. Among the tested configurations, the two best results were obtained with batch size 16 and update rate 3, and batch size 16 and update rate 10. The first one (16–3) reacts faster and shows short-term improvements, while the second (16–10) provides more stable learning over longer training. Larger batches help reduce the variance, and less frequent target updates prevent the model from making aggressive or unstable changes, leading to more consistent Q-value learning overall.



Run	Smoothed	Value ↓	Step	Relative
● batch16_rate3_20251113-152530	-19.4102	-19.4	49	39.2 min
● batch16_rate10_20251113-210431	-19.514	-19.6	99	1.127 hr

Both configurations confirm that batch size 16 provides better performance and stability compared to smaller batches, making it the most effective setting for this DQN implementation.