CPSC 8810: Final Project
Rainbow Road: Autonomous Driving using DQN Variants

Sebastián Gracia Guzmán

June 17, 2025

1 Abstract

The reinforcement learning algorithm DQN has many different variants which improve performance. As described in [3], combining different variants achieves superior performance across different benchmarks. The aim of this project was to implement Rainbow DQN in a simple autonomous driving environment [4] and compare results obtained to results obtained to the paper. Reading the literature from which the variants

2 Introduction

2.1 Environment

The environment in which the following work was implemented was the highway-v0 environment. The task of the environment is to maneuver a car between lanes and avoid colliding with other cars. The agent receives positive rewards for going faster. At any time step, the agent, or ego car, receives an observation consisting of a 5 by 5 array. This includes an occupancy grid, velocity, and other variables which describe the state.

The actions space for the agent are the discrete actions of *Merge Left, No Action, Merge Right, Faster, Slower*. This section will briefly go over the different variants of DQN and the intuition behind them.

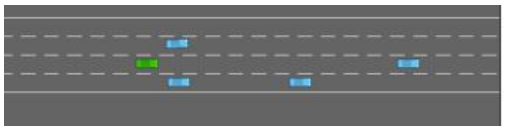


Figure 1: Highway-v0 environment.

2.2 Vanilla

The original DQN algorithm was published by Minh 2015. The main algorithm uses a neural network to approximate the action value function and sampling experiences from a memory buffer. The gradient is updated using the following loss function:

$$L(\theta_t) = (R_{t+1} + \gamma \max Q_{\theta^-}(S_{t+1}, a) - Q_\theta(s, a))^2$$

2.3 Double DQN

First introduced by [6], double DQN seeks to reduce the overestimation of actions inherent to DQN by implementing a different loss function.

$$L(\theta_t) = (R_{t+1} + \gamma Q_{\theta^-}(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q_\theta(S_{t+1}, a)) - Q_\theta(s, a))^2.$$

Where p_t is the priority of experience at time step t and ω is a hyperparameter. A priority is assigned to each experience then a probability distribution which formulated from the priorities.

2.4 Dueling DQN

Introduced in [7], by rewriting the q-function as the sum of the value function and advantage function we can obtain better policy evaluations.

$$Q(s, a) = V(s) + A(s, a)$$

The original authors demonstrated that better approximation of state values improved training performance. When implementing this variant, only the neural network architecture needs to be modified.

$$Q_\theta(s, a) = V_\phi(s) + (A_\eta(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A_\eta(s, a'))$$

Here η and ϕ are the parameters of the specific neural network layers corresponding to their respective function approximation. θ are the parameters of the entire neural network which encompass η and ϕ.

2.5 Prioritized Experience Replay

Prioritized experience replay [5] introduces the notion of sampling experiences with a preference towards useful experiences instead of uniform randomly. The preference is given in proportion to magnitude of TD-error. This intuition is based on experiences in which the agent made large errors offer more to learn from than other experiences. This is given by:

$$p_t \propto \left|(R_{t+1} + \gamma \max Q_{\theta^-}(S_{t+1}, a) - Q_\theta(s, a))^2\right|^\omega$$

2.6 Noisy Nets

Presented first in [2]. In lieu of fully connected linear layers, noisy layers were used. Noisy layers take the form of:

$$y = (\mu^w \sigma^w \odot \epsilon^x)x + \mu^b + \sigma^b \odot \epsilon^b.$$

Here $\odot$ signifies element wise multiplication and b, w are hyperparameters.

2.7 Distributional Loss

Generally in reinforcement learning we are attempting to maximize expected rewards. From [1], a different strategy is taken in which we approximate the return distribution instead. The approach involves the formulation of a distribution using a support vector z for a probability mass function.

$$d'_t = (R_{t+1} + \gamma \mathbf{z}, \mathbf{p}_{\bar{\theta}}(S_{t+1}, \bar{a}^*_{t+1}))$$

This is defined to be the Kullbeck-Leibler divergence. This is comparing the computed reward distribution, from the predicted distribution from the neural network with parameter θ. ironment Due to time constraints, this variant was not implemented.

2.8 Multi-step Learning

In Q-learning, a single step TD-error is calculated. This simple variant involves using an $n-$step discounted future rewards mentioned in CITE. Therefore the rewards would be calculated in the following way:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^k R_{t+k+1}.$$

Then using the resulting n-step rewards in the loss function. In the original Rainbow paper, 3 step rewards was used.

3 Implementation

Implementing the variants had varying degrees of difficulty and success. The double DQN and dueling DQN were both straight forward to implement and immediately showed improvements over vanilla DQN. Other variants were more difficult to implement and did not show improvements. Prioritized experience replay and noisy nets did not learn adequately. Prioritized experience replay most likely had an error in implementation while noisy nets implementation came from an existing implementation by [8]. Thus it is more likely that for the highway environment, noisy nets is not helpful. It is also possible that due to the relatively training steps taken, the effects would not be seen until later in training. Despite not having all variants in working condition, working on the implementation proved useful in understanding the variants.

3.1 Training

Training was done using Google Colab's cloud computing. Training time for each variant was around 7 hours. The neural network architecture was 5 layers deep with 512, 128, 64 in between the input state size and output action size. A Relu activation function was used in between each fully connected layer.

3.1.1 Hyperparameters

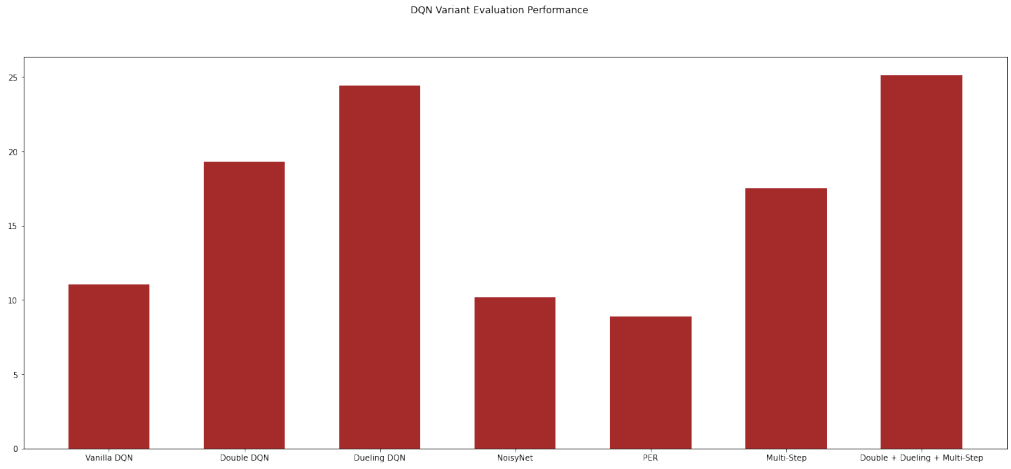
Hyperparameters were taken from tuned hyperparameters used from DQN frogger environment implementation. For the economy of time, hyperparameters were not further tuned. Below are hyperparameters used for training across all variants (when applicable).

```
batch_size = 128 # Number of samples from replay buffer
lr = 5e-4
gamma = 1 # .99 for n-step returns
max_episodes = 1000
mem_capacity = 10000 # Size of replay buffer
target_update_freq = 500 # Frequency to update Q target network
learning_freq = 5 # Frequency to perform gradient step on Q network
# For multi-step learning
n = 3
# For Prioritized Experience Replay
priority_scale = .7
beta = .5
```

4 Results

Variant	Mean Reward (μ)	Standard Deviation (σ)
Vanilla DQN	11.0564	$\pm$ 5.9109
Double DQN	19.3297	$\pm$ 7.5077
Dueling DQN	24.4364	$\pm$ 6.4404
Noisy Net	10.1840	$\pm$ 5.0195
Prioritized Replay	8.9022	$\pm$ 5.5704
Multi-step Learning	17.5464	$\pm$ 9.70300
Double + Dueling + Multi-Step	25.1386	$\pm$ 5.0862
Distributional RL	–	–
Rainbow*	-	-

Figure 2: *Since not all variants were included, the true *Rainbow* algorithm couldn't be implemented.



4.1 Analysis of Results

From results we observe that the single most impactful variant was the dueling DQN architecture. Double DQN and Multi-step learning also made significant improvements. Implementing noisy linear layers did not seem to have a significant effect on performance. Prioritized experience replay had a detrimental effect on performance. After combining all improving variants that improved performance we obtain an overall a marginally improved result. Hyperparameter training could have a large impact across variant performance and thus results obtained could be quite different with tuned hyperparameters. The full learning curve results for each variant can be found in Appendix A.

5 Conclusion

Overall the suite of variants all generally provide improvements compared to the vanilla DQN compared to each individually. When combined, the power of the variants provide a significant improvement to the standalone DQN algorithm. Each of these variants are not difficult to combine and implement. The difficulty arose in implementing them independently to obtain baseline results for each standalone variant. This could have been alleviated using independent implementations of each variant instead of having one implementation where each variant can be toggled to be used or not.

5.1 Future Work

Obviously, it is left to finish implementing all the variants and troubleshoot all bugs. Due to the very slow rendering of the environment, training was limited to 1000 episodes, or about 140000 time steps, per variant. The true effect of the rainbow algorithm could be further expressed with more training iterations.

Since training was limited to 1000 episodes for the economy of time, the true possible effects of the variants is unknown. For future work implementing parallel training or training on the Palmetto Cluster for more timesteps would be looked at. Further work in a simpler environment would allow more timesteps to be collected and more insights on variants. Furthermore, comparing the Rainbow results with other state of the art algorithms such as PPO or DDPG would be interesting. This comparison was missing from the original paper and would be a direction of further study.

References

[1] Will Dabney et al. *Distributional Reinforcement Learning with Quantile Regression*. 2017. arXiv: 1710.10044 [cs.AI].

[2] Meire Fortunato et al. *Noisy Networks for Exploration*. 2019. arXiv: 1706.10295 [cs.LG].

[3] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: 1710.02298 [cs.AI].

[4] Edouard Leurent. *An Environment for Autonomous Driving Decision-Making*. https://github.com/eleurent/highway-env. 2018.

[5] Tom Schaul et al. "Prioritized Experience Replay". In: *CoRR* abs/1511.05952 (2016).

[6] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: (Sept. 2015).

[7] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: 1511.06581 [cs.LG].

[8] He Wen. *DQfD-PyTorch*. https://github.com/hw9603/DQfD-PyTorch/tree/master/DQNwithNoisyNet. 2019.

A Training Results

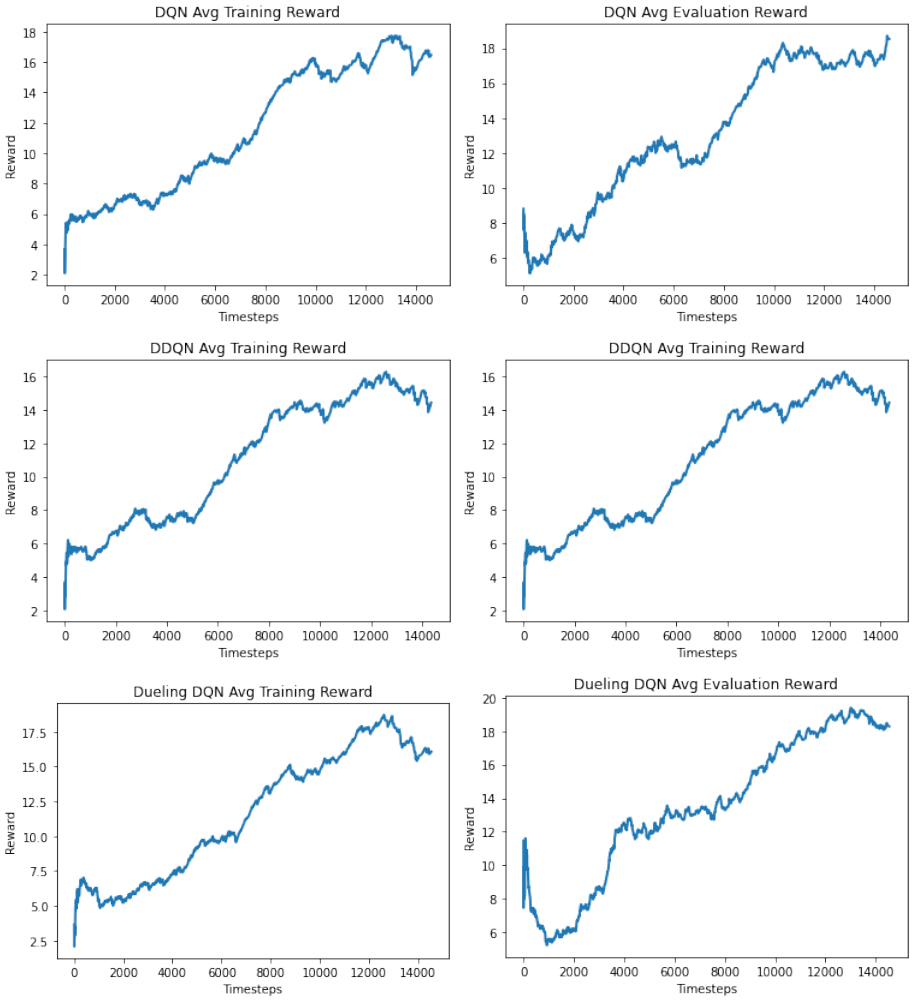


Figure 3: Learning curve for DQN variants.

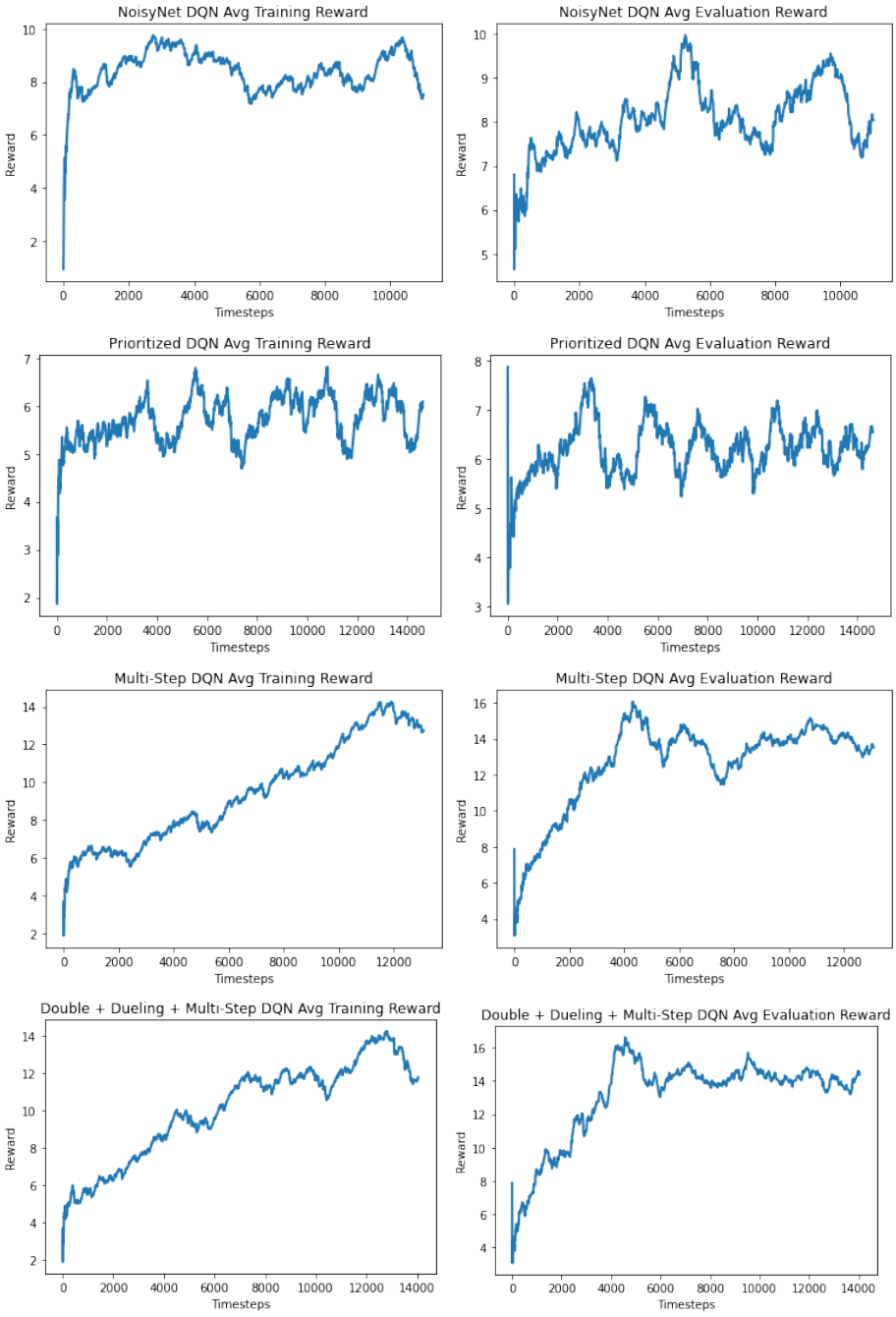


Figure 4: Learning curve for DQN variants.