Intelligent Transportation Systems Final Report

Duncan Waugh, Alex Cavaco-Donia, James Szymanski

251246391, 251241758, 251237012

Dr. Zaki Hussein

CEE 4412

April 26th, 2024

Abstract

This paper presents the development and evaluation of a Deep Q-network (DQN) for autonomous driving tasks within the "highway-env" simulation. The network initially used the original Stable Baselines framework to train the vehicle, however, it was later changed to implement a new PyTorch script to enhance adaptability and performance tuning. The project's objective was to explore the efficacy of reinforcement learning in dynamic and complex driving scenarios, where traditional algorithms may be inadequate. The DQN model was trained to optimize driving strategies, focusing on lane-keeping and overtaking maneuvers, while prioritizing safety and efficiency. Experimental results indicate that the DQN, particularly in its PyTorch incarnation, effectively learned diverse driving policies, significantly outperforming baseline models in safety and operational metrics. This study emphasizes the potential of deep reinforcement learning, implemented via advanced frameworks, to improve autonomous driving technologies.

Motivation

As the autonomous vehicle industry progresses, the complexity of driving scenarios that vehicles must navigate also increases. While effective in predictable environments, traditional algorithmic approaches often fall short in the highly dynamic and variable conditions typical of real-world driving. Reinforcement learning (RL) offers a promising alternative by enabling systems to learn optimal behaviours directly from environmental interaction, adapting to new and unforeseen situations without explicit programming for each possible scenario. The Deep Q-Network (DQN), a robust model within the reinforcement learning paradigm, has demonstrated success in various domains, including gaming and robotics. This research aims to extend its application to

autonomous driving, particularly using the "highway-env" simulation, a tool designed to test and develop driving policies in everyday driving scenarios.

Scope

This paper details the application of DQN models to the highway-env simulation, focusing on their development and optimization for autonomous driving tasks such as lane keeping, safety and efficiency. Stable Baselines were initially used to establish a baseline performance for the reliable set of implementations for RL algorithms. Subsequently, a DQN model was implemented in PyTorch to enhance customization and improve performance. This dual approach allows for a comparative analysis of the frameworks regarding flexibility, efficiency, and effectiveness in learning complex driving behaviours.

Objectives

This research aims to assess and compare the efficiency of two distinct implementations of DQNs for optimizing autonomous driving behaviours within the open-source "highway-env" simulation provided by Farama-Foundation. The study utilized the Stable Baselines framework to establish a performance benchmark and ensure the model adheres to proven reinforcement learning methods. This was then changed to a custom-built DQN using PyTorch to explore advanced customization possibilities and performance enhancements. This transition allows for customization of the environments which permits more specific training using custom scenarios that mimic the real world. This comparative analysis assesses the advantages of deep reinforcement learning in improving safety and efficiency in autonomous driving scenarios. Moreover, this study is intended to contribute significantly to the field by showcasing how

flexibly and adaptively the deep learning frameworks can be integrated into real-time decision-making processes in autonomous vehicles, thus setting a foundation for future research and practical implementations in the industry.

Code Architecture StableBaselines with Optuna

System Architecture and Workflow Overview

This document delineates the implementation of a reinforcement learning system using the Stable Baselines3 framework and OpenAI Gym, optimized for highway_env, a simulated driving environment. The focus is on environment setup, hyperparameter optimization using Optuna, and model evaluation.

Environment Configuration and Hyperparameter Optimization

The script initializes by setting up necessary libraries for simulation and model evaluation. The optimize_dqn function is critical for configuring logging for outputs and defining variable hyperparameters such as learning rate, network architecture (net_arch), discount factor (gamma), and batch size. These parameters are optimized through Optuna to find the most effective settings that maximize the operational reward.

Model Training and Evaluation

A simulation environment (eg. highway-fast-v0) is created and enhanced with a Monitor for performance tracking. A DQN model is configured with these hyperparameters and connected to the environment. An evaluation callback periodically assesses the model, saving the best iteration based on performance metrics.

Post-training, the best model is evaluated in a newly set up environment to compute the average reward over several episodes, providing a measure of the model's effectiveness.

Execution and Performance Visualization

Controlled by the TRAIN flag, the script either runs hyperparameter optimization and records model details or loads a pre-trained model for extensive testing. The evaluation process includes predicting actions, accumulating rewards, and assessing overall performance. Rewards obtained across episodes are visualized using Matplotlib to display the model's consistency and performance across trials.

Code Architecture for Pytorch Deep Q Implementation

System Configuration and Neural Network Design

The initialization phase loads necessary Python libraries and sets up the simulation environment through the gymnasium package. Computational efficiency is ensured by configuring operations to run on CUDA, if available, with a fallback to the CPU otherwise. The neural architecture at the heart of this system is a DQN, which comprises three linear layers. This network is tasked with approximating the optimal action-value function; this is essential for making informed actions within the environment. Additionally, a ReLU activation function is used to help introduce non-linearity to the model which is crucial for learning complex patterns in high-dimensional spaces.

Memory and Action Selection Mechanism

The implementation employs an experience replay mechanism, wherein interactions with the environment are stored in a replay buffer and sampled randomly to update the network. This approach mitigates the correlations in the observation sequence and smooths out learning over

previous experiences. Action selection is governed by an epsilon-greedy algorithm, which ensures that the agent explores the environment effectively by varying the level of policy randomness according to a predefined decay schedule.

Training Dynamics and Optimization

Training proceeds over several episodes, each limited to a set number of steps. In each episode, the agent repeatedly interacts with the environment to collect data, selects actions based on the current policy, and records the outcomes to update the model. The collected data is then stored in the replay buffer. The optimization step involves calculating the temporal difference error between estimated and actual Q-values using a smooth L1 loss. This error is then back propagated through the network to adjust the weights, with the Adam optimizer facilitating efficient convergence.

Performance Visualization and Monitoring

The progress is monitored through real-time visualizations that plot the cumulative rewards obtained per episode, providing immediate feedback on the agent's learning and performance. These plots are crucial for debugging and tweaking the learning process, ensuring that the agent's performance improves consistently over time. Logging within the training loop offers additional insights by detailing rewards and episode durations, which helps in fine-tuning the training parameters and the model architecture.

Results

The comprehensive analysis of the experimental results from the "highway-env" simulations offers quantitative insights into the performance and effectiveness of different DQN models, as detailed in the appendix figures. Initially, the use of the Stable Baselines framework in the

highway environment provided a baseline for performance, with moderate rewards and relatively high loss rates as shown in Figures 1.0 and 2.0. The subsequent adoption of the PyTorch model marked a significant improvement, where enhancements in mean rewards and reductions in the loss were evident, especially after optimizing memory values as depicted in Figure 14.0.

Further refinements in the PyTorch model parameters, as shown in Figures 16.0, 17.0, 18.0, and 20.0, culminated in optimal settings that achieved the highest performance metrics. The adaptation to more complex scenarios like merge and roundabout environments demonstrated the versatility of the PyTorch framework. In the merged environment, adjustments in the GAMMA value from 0.9 to 0.8 significantly enhanced learning rates and stability, achieving greater reward consistency as detailed in Figures 12.0 and 19.0. Similarly, in the roundabout environment, Figures 5.0, 6.0, 23.0, and 25.0 highlight the impact of tuning EPS_DECAY and LEARNING_RATE, which substantially increased the model's ability to manage continuous and complex traffic flows efficiently.

The use of Optuna for systematic hyperparameter optimization was particularly effective, as evidenced by Figures 9.0 and 11.0. These figures show the best hyperparameters identified for the merge and roundabout environments, leading to noticeable improvements in later simulations. This methodical optimization process facilitated significant advancements in model stability and performance metrics.

The comparative analysis between Stable Baselines and PyTorch further underscores the advantages of transitioning to a more advanced framework. The PyTorch models not only improved upon its own earlier versions but allowed for much more customization which proved to be vital when dealing with different complex environments. This transition validates the

potential of sophisticated DQN implementations to enhance the operational efficacy and adaptability of autonomous driving systems, especially in dynamic and complex scenarios.

Conclusion

This study underscores the efficacy of Deep Q-networks (DQNs) in advancing autonomous driving technologies, particularly through the "highway-env" simulation platform. The initial application of the Stable Baselines framework provided a robust baseline for performance metrics. Subsequent integration of the PyTorch framework facilitated enhanced adaptability and refined performance tuning, particularly in complex driving scenarios such as merges and roundabouts.

The results of this research illustrate that advanced machine learning frameworks like PyTorch significantly enhance the adaptability and operational efficiency of autonomous vehicles. Through systematic hyperparameter optimization, notably employing the Optuna framework, the PyTorch models demonstrated superior performance metrics, showcasing the potential of reinforcement learning to surpass traditional algorithmic approaches in autonomous driving applications.

Looking to the future, the integration of deep learning into autonomous vehicle systems is expected to drive significant technological advancements, enhancing both safety and operational efficiency. This research establishes a foundational base for further studies, aiming to refine these models and potentially revolutionize autonomous driving technologies with a focus on real-world application and scalability.

Appendix

Table 1.0: Work assignments for final report

Name	Work Assignments
Alex Cavaco-Donia	Meeting minutes, objectives, scope, appendix
James Szymanski	Abstract, motivation, conclusion, appendix
Duncan Waugh	Code architecture, results, appendix

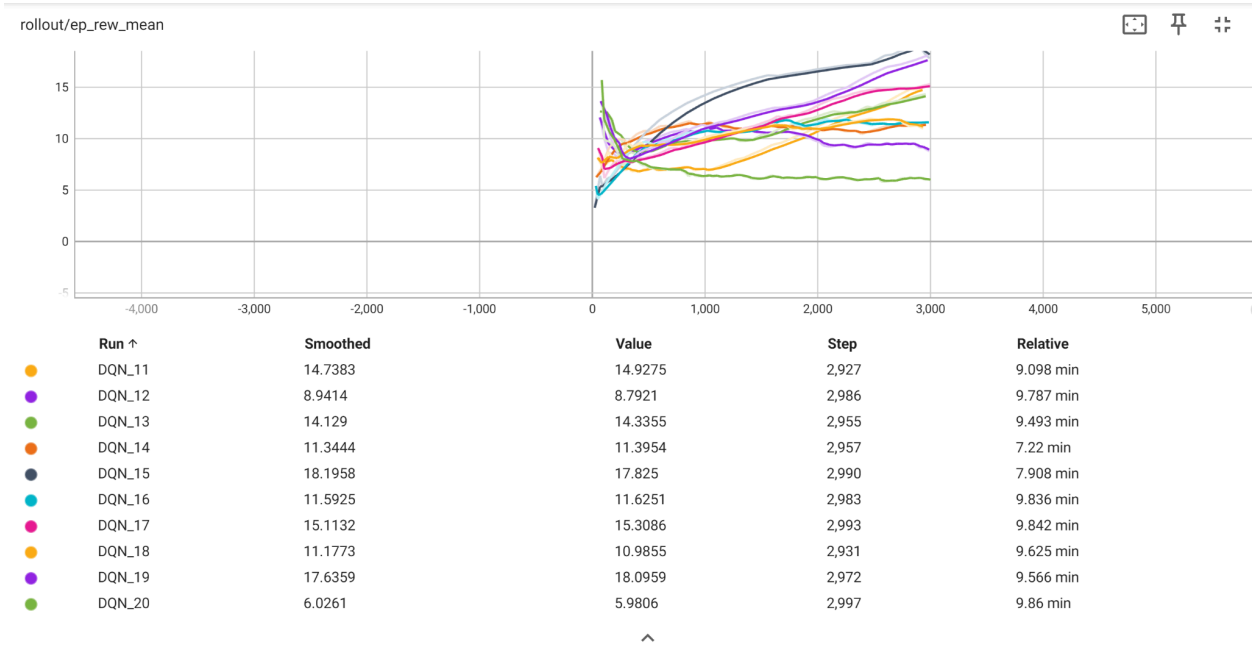
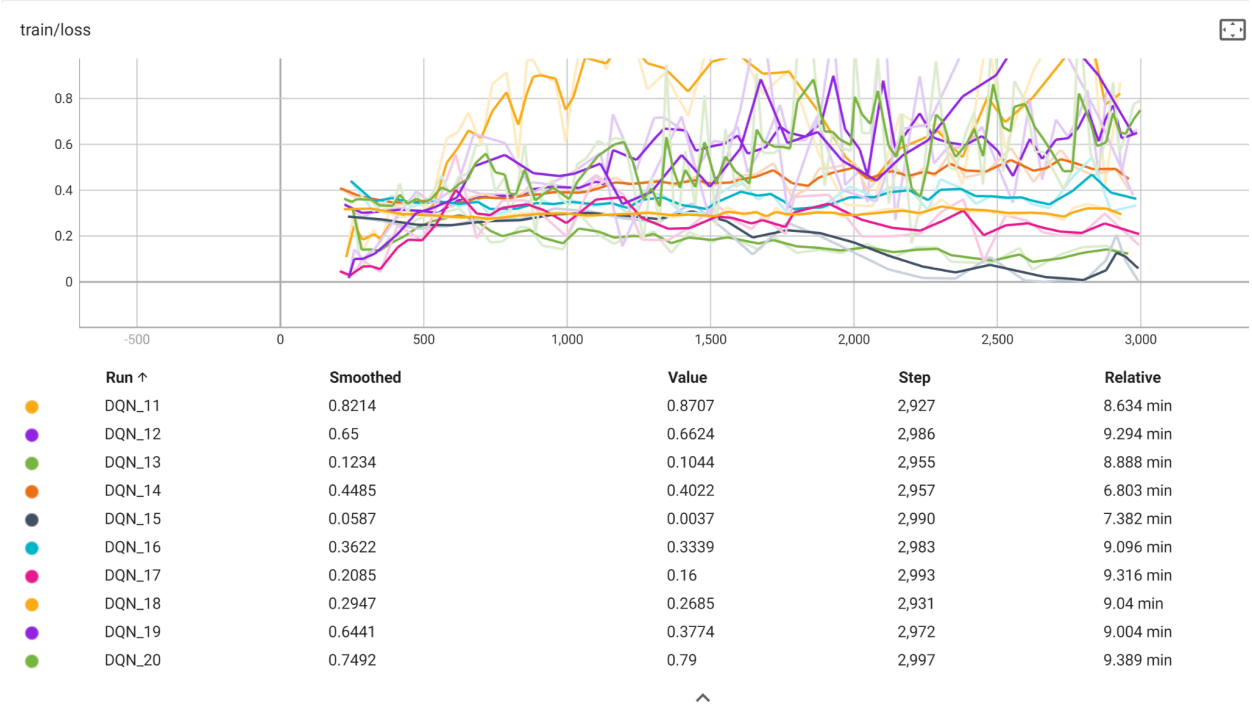


Figure 1.0: Highway-fast-V0 Tensorboard for the mean reward with 10 different Optuna trials

.

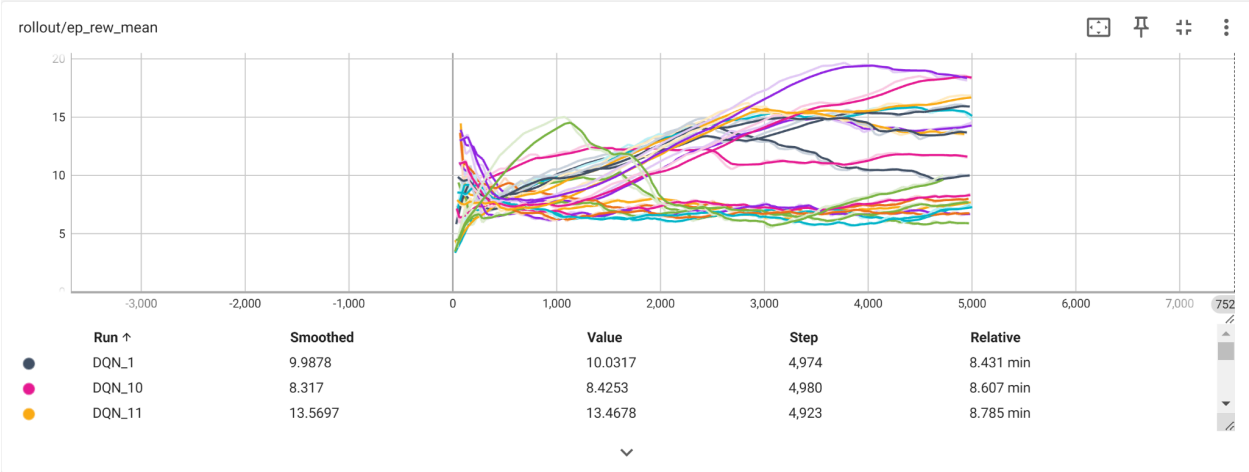


Run	Smoothed	Value	Step	Relative
DQN_11	0.8214	0.8707	2,927	8.634 min
DQN_12	0.65	0.6624	2,986	9.294 min
DQN_13	0.1234	0.1044	2,955	8.888 min
DQN_14	0.4485	0.4022	2,957	6.803 min
DQN_15	0.0587	0.0037	2,990	7.382 min
DQN_16	0.3622	0.3339	2,983	9.096 min
DQN_17	0.2085	0.16	2,993	9.316 min
DQN_18	0.2947	0.2685	2,931	9.04 min
DQN_19	0.6441	0.3774	2,972	9.004 min
DQN_20	0.7492	0.79	2,997	9.389 min

Figure 2.0: Highway-fast-V0 Tensorboard for loss with 10 different Optuna trials

```
[I 2024-04-01 23:25:07,964] Trial 9 finished with value: 5.1704497 and parame
 64, 'gamma': 0.8568800347472593}. Best is trial 8 with value: 22.0630751.
Number of finished trials: 10
Best trial:
 Value: 22.0630751
 Params:
    learning_rate: 0.0022612406819308988
    net_arch: [128, 128]
    batch_size: 64
    gamma: 0.9755349680416233
PS C:\4412_labs\progress_report>
```

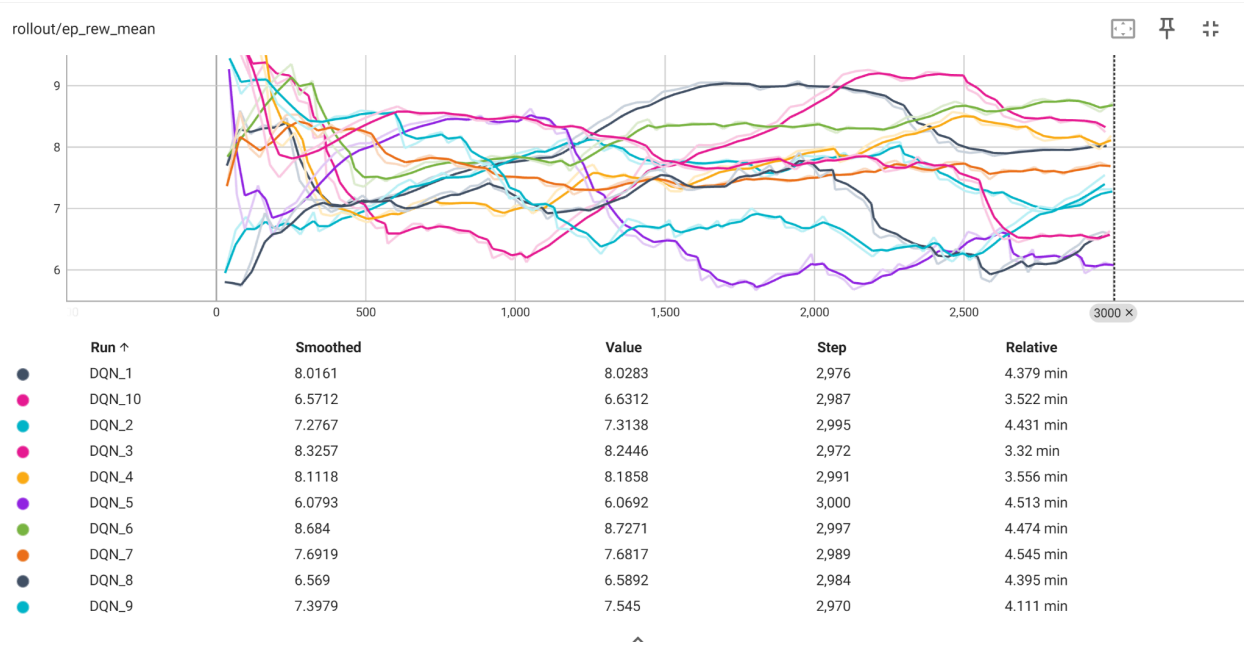
Figure 3.0: Best hyperparameters for Highway-fast-V0 found after 10 Optuna trials




```
Best trial:
  Value:  21.143043
  Params:
    learning_rate: 0.00062386089814659 91
    net_arch: (128, 128)
    gamma: 0.928778435411863
    batch_size: 32
```

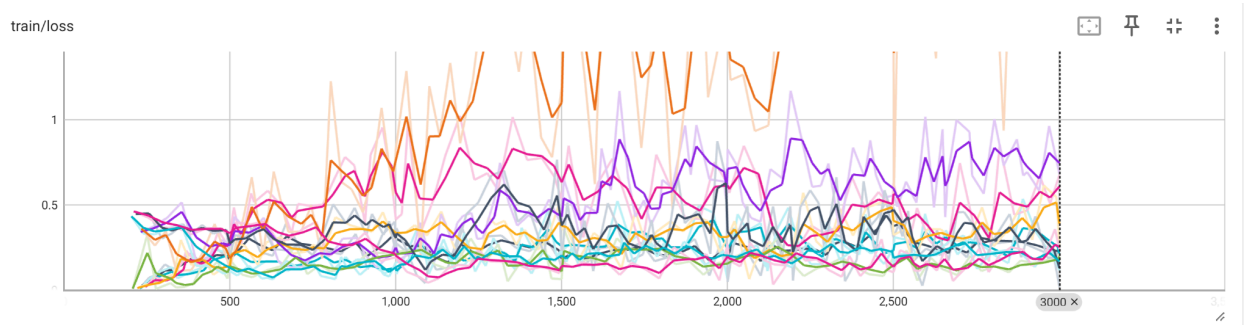
Figure 4.0: Highway Env after 20 trials, 5000 timesteps

.



Run	Smoothed	Value	Step	Relative
DQN_1	8.0161	8.0283	2,976	4.379 min
DQN_10	6.5712	6.6312	2,987	3.522 min
DQN_2	7.2767	7.3138	2,995	4.431 min
DQN_3	8.3257	8.2446	2,972	3.32 min
DQN_4	8.1118	8.1858	2,991	3.556 min
DQN_5	6.0793	6.0692	3,000	4.513 min
DQN_6	8.684	8.7271	2,997	4.474 min
DQN_7	7.6919	7.6817	2,989	4.545 min
DQN_8	6.569	6.5892	2,984	4.395 min
DQN_9	7.3979	7.545	2,970	4.111 min

Figure 5.0: Roundabout-V0 Tensorboard for the mean reward with 10 different Optuna trials



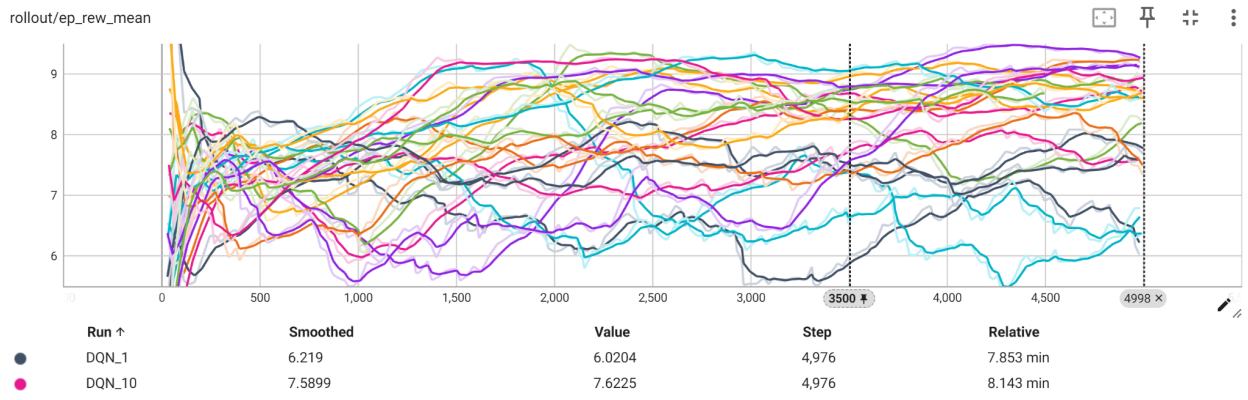
Run	Smoothed	Value	Step	Relative
DQN_1	0.2278	0.2197	3,000	4.296 min
DQN_10	0.2532	0.2378	3,000	3.408 min
DQN_2	0.205	0.1527	3,000	4.285 min
DQN_3	0.6085	0.6757	3,000	3.203 min
DQN_4	0.3903	0.2605	3,000	3.455 min
DQN_5	0.7457	0.6824	3,000	4.32 min
DQN_6	0.1976	0.2136	3,000	4.377 min
DQN_7	2.8162	2.9081	3,000	4.411 min
DQN_8	0.1296	0.0199	3,000	4.27 min
DQN_9	0.1691	0.1161	3,000	4.04 min

Figure 6.0: Roundabout-V0 Tensorboard for loss with 10 different Optuna trials

.

```
Number of finished trials:  10
Best trial:
  Value:  9.8875001
  Params:
    learning_rate: 0.0021092228284437122
    net_arch: (64,)
    gamma: 0.9335358104367866
    batch_size: 128
PS C:\4412_labs\progress_report>
```

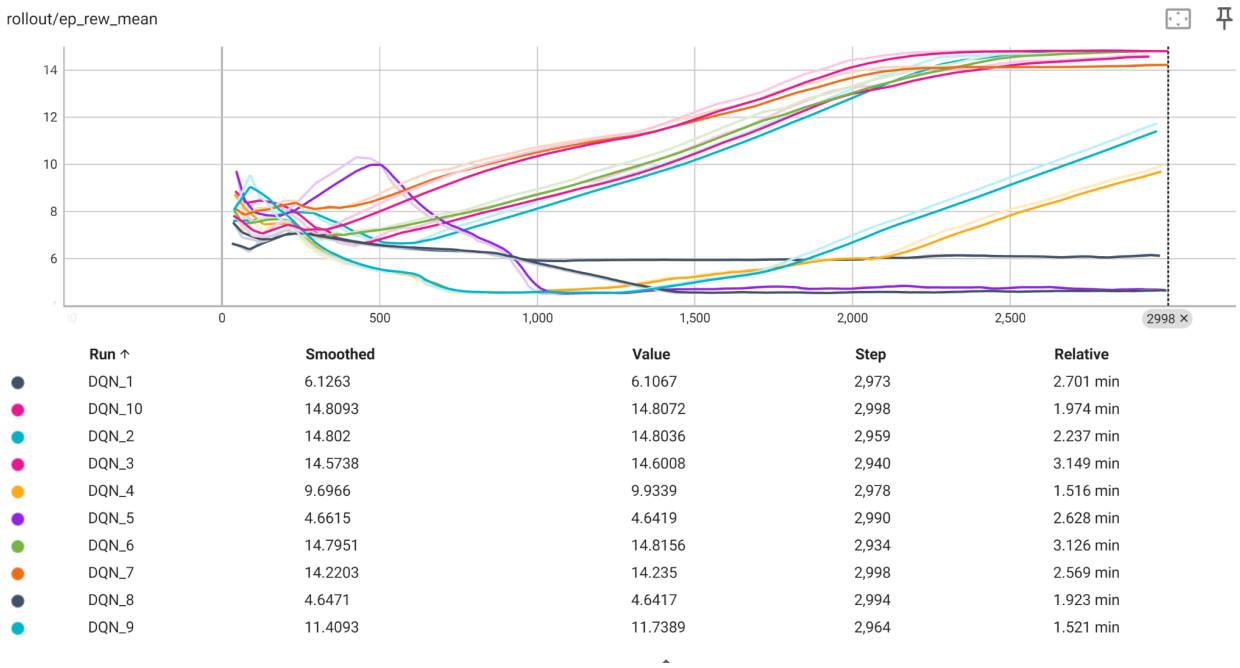
Figure 7.0: Best hyperparameters for Roundabout-V0 found after 10 Optuna trials




```
Best trial:
  Value:  10.1125
  Params:
    learning_rate: 0.0020961519397430257
    net_arch: (256, 256)
    gamma: 0.8733220101691884
    batch_size: 32
```

Figure 8.0: Roundabout-V0 after 20 trials, 5000 timesteps

.



Run	Smoothed	Value	Step	Relative
DQN_1	6.1263	6.1067	2,973	2.701 min
DQN_10	14.8093	14.8072	2,998	1.974 min
DQN_2	14.802	14.8036	2,959	2.237 min
DQN_3	14.5738	14.6008	2,940	3.149 min
DQN_4	9.6966	9.9339	2,978	1.516 min
DQN_5	4.6615	4.6419	2,990	2.628 min
DQN_6	14.7951	14.8156	2,934	3.126 min
DQN_7	14.2203	14.235	2,998	2.569 min
DQN_8	4.6471	4.6417	2,994	1.923 min
DQN_9	11.4093	11.7389	2,964	1.521 min

Figure 9.0: Merge-V0 Tensorboard for mean reward with 10 different Optuna trials

.

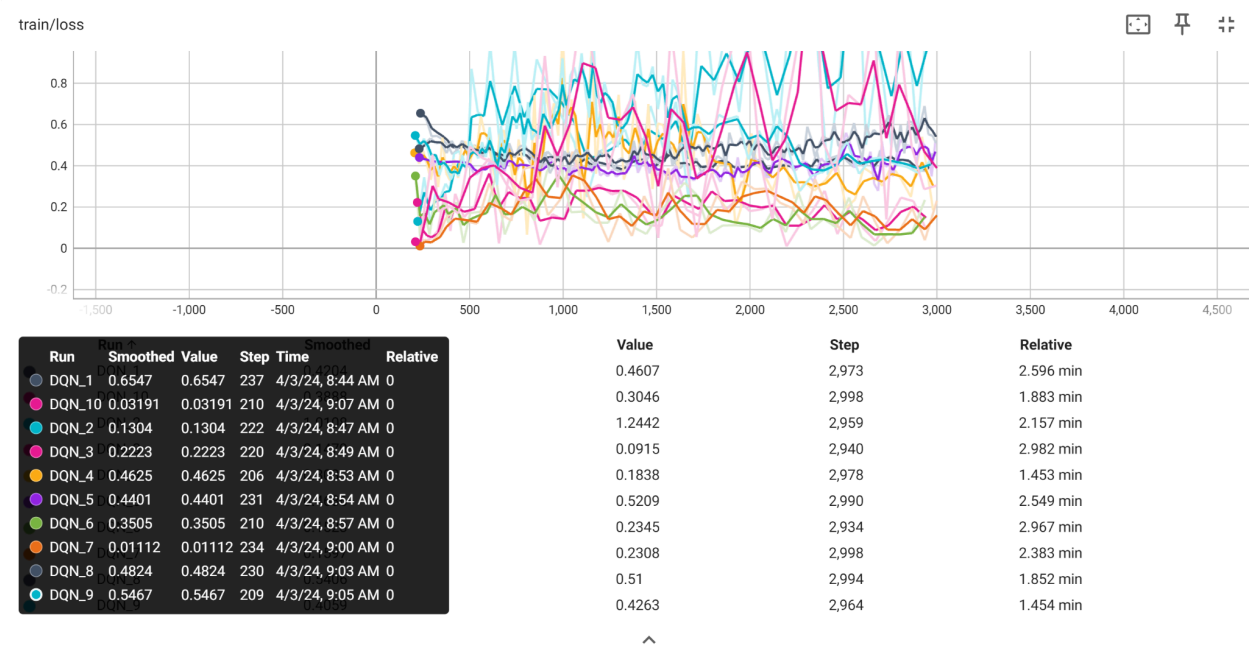
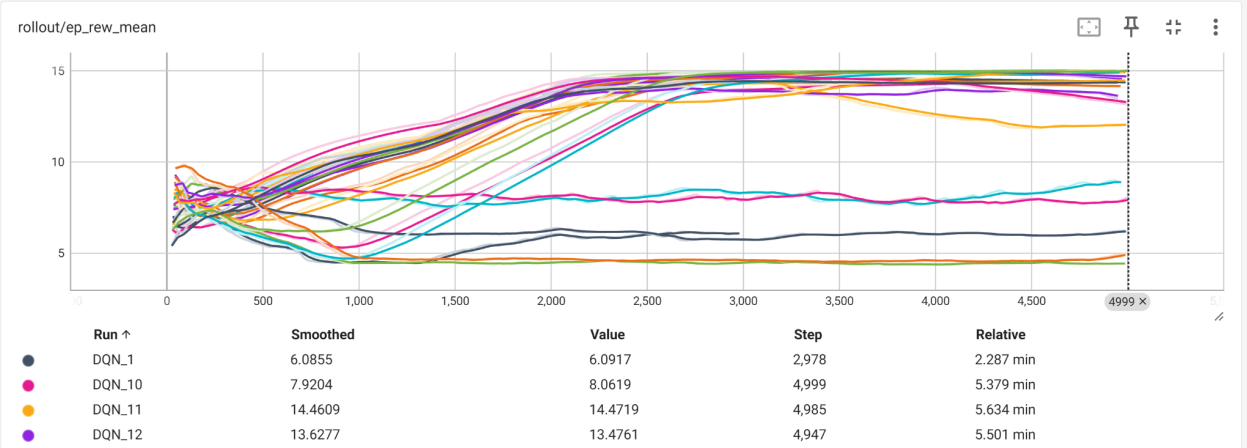


Figure 10.0: Merge-V0 Tensorboard for loss with 10 different Optuna trials

```
Number of finished trials: 10
Best trial:
 Value: 15.027778000000001
 Params:
    learning_rate: 0.0004893614496129762
    net_arch: [128, 128]
    batch_size: 32
    gamma: 0.8336759433904127
PS C:\4412_labs\progress_report>
```

Figure 11.0: Best hyperparameters for Merge-V0 found after 10 Optuna trials

.




```
Best trial:
  Value:  15.222221999999999
  Params:
    learning_rate: 0.00036649046677617795
    net_arch: (64,)
    gamma: 0.8761684336713984
    batch_size: 128
```

Figure 12.0: Merge Env after 20 trials, 5000 timesteps

.

Pytorch highway env:

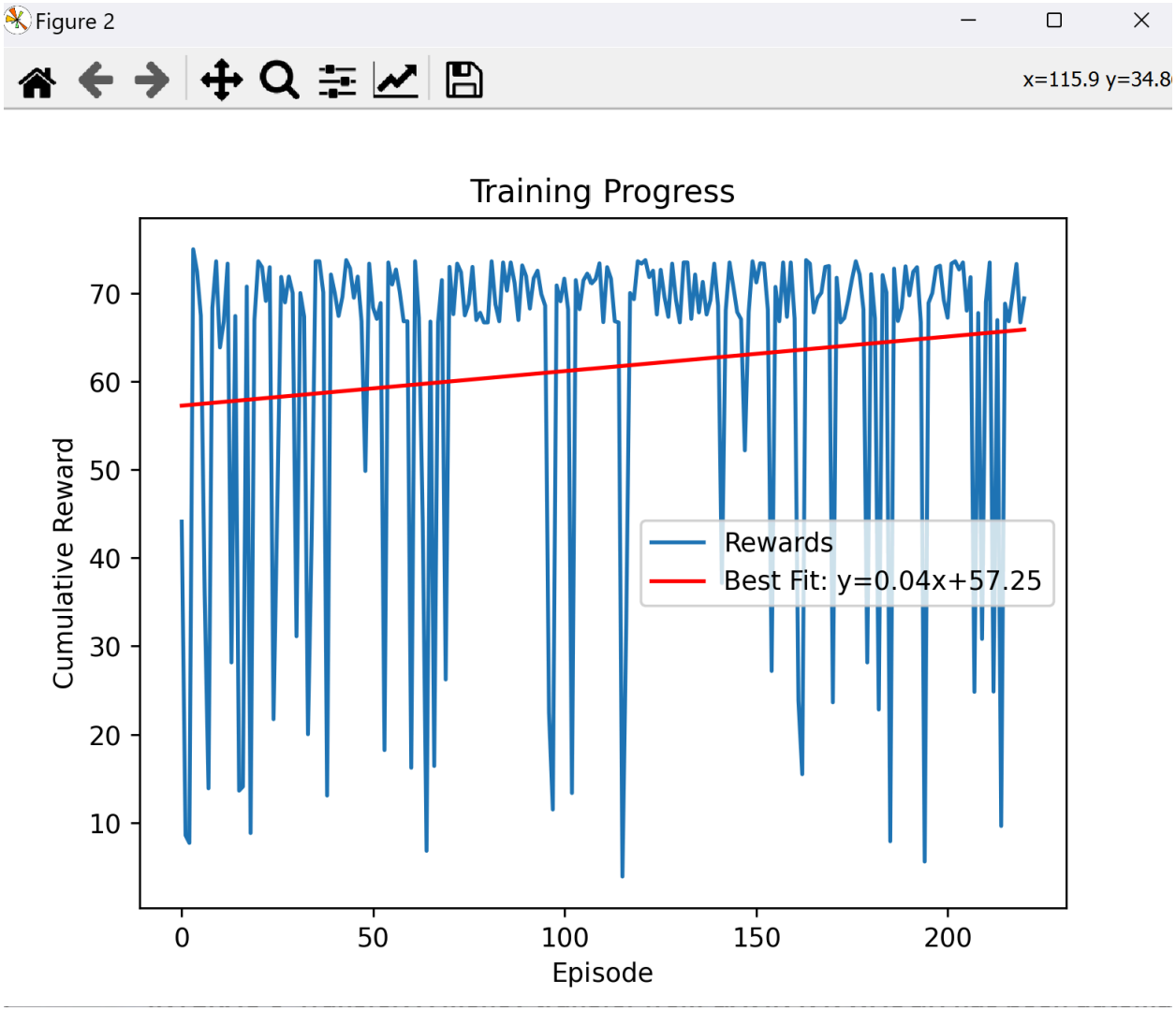


Figure 13.0: Initial run with the PyTorch highway model

.

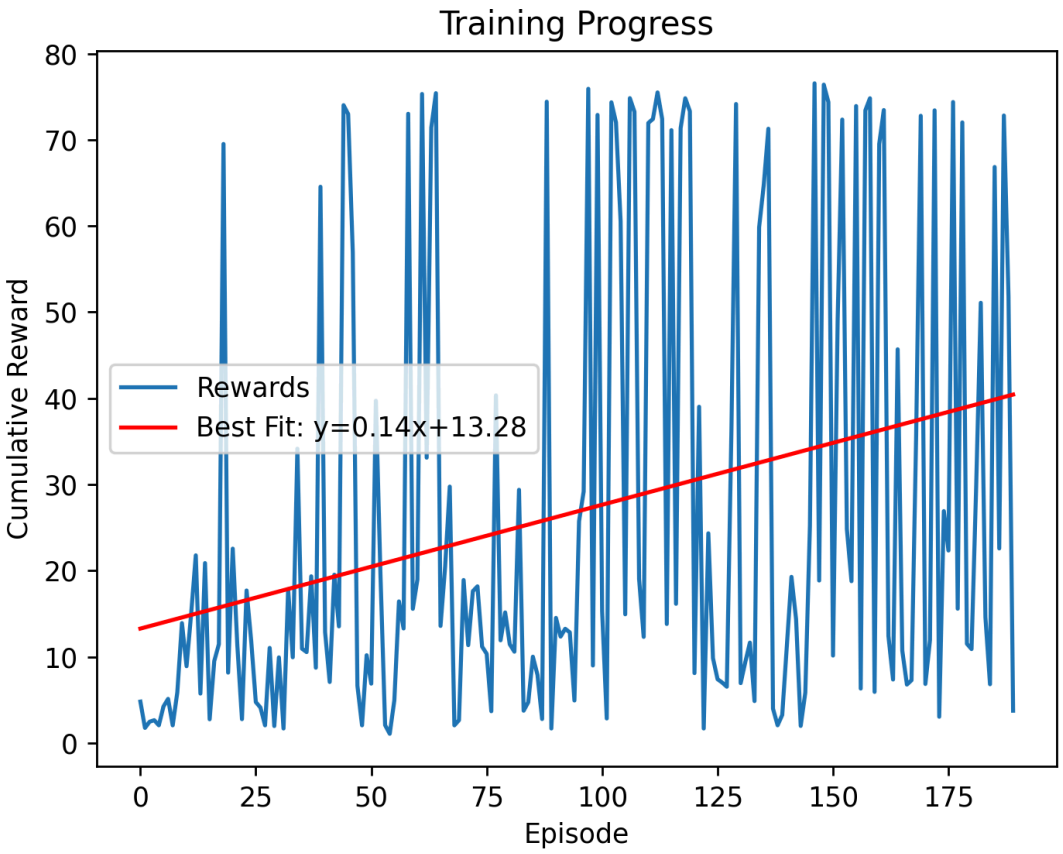
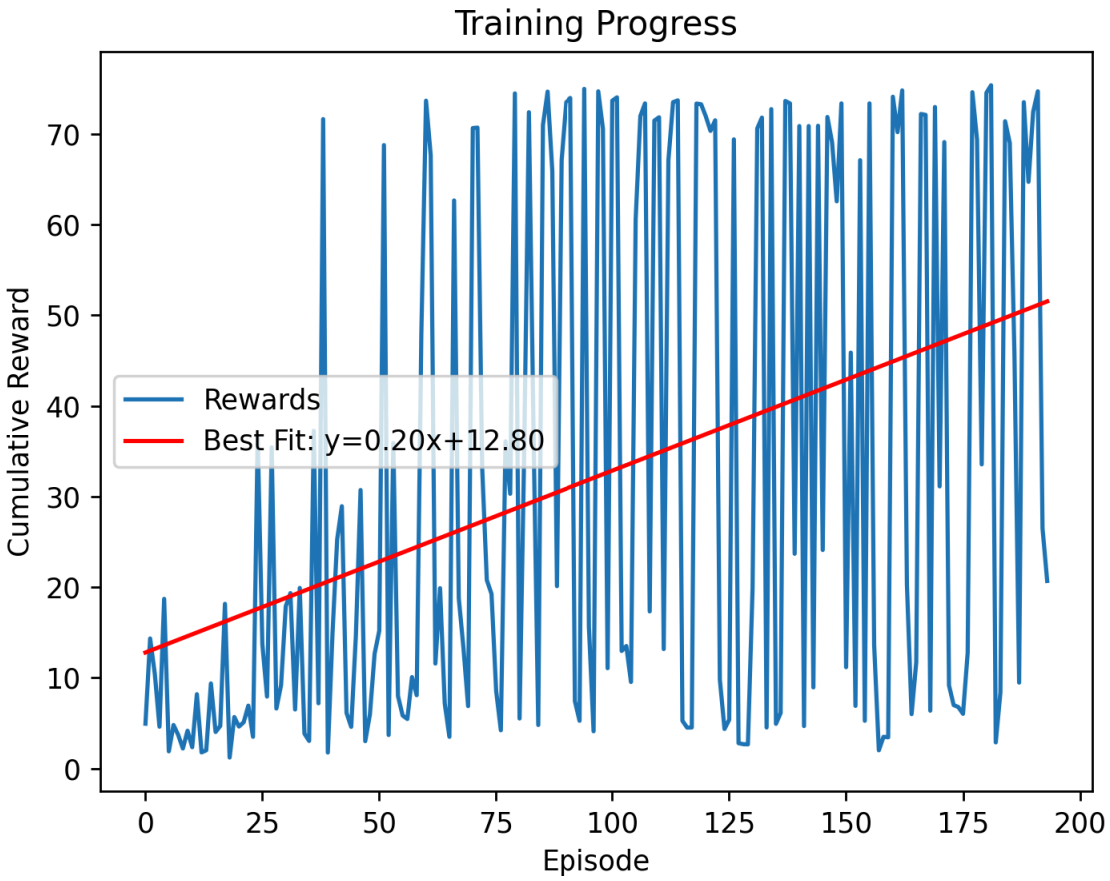


Figure 14.0: PyTorch highway model results after the memory value was doubled

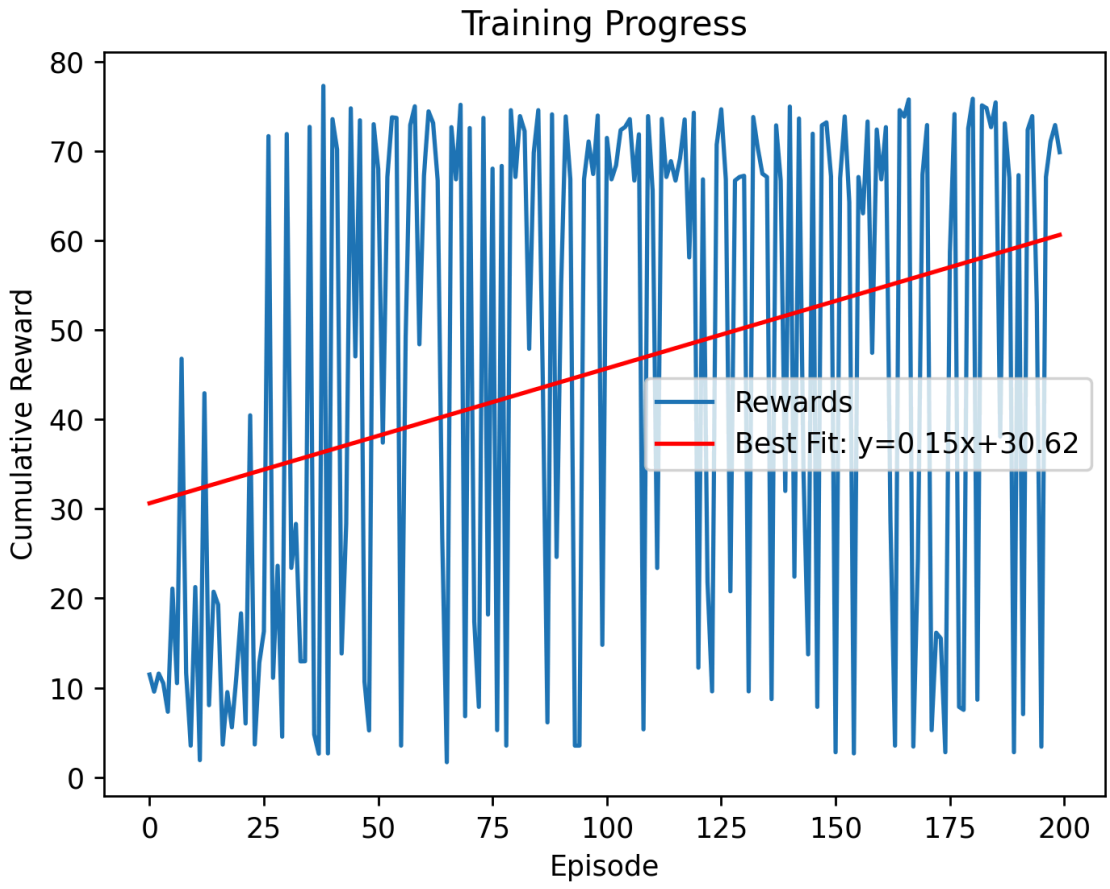
.




```
BATCH_SIZE = 1024

GAMMA = 0.9 #0.9

EPS_START = 0.999

EPS_END = 0.05

EPS_DECAY = 500

LEARNING_RATE = 0.000002679

NUM_EPISODES = 500

MAX_STEPS_PER_EPISODE = 100

MEMORY_REPLAY = 20500
```

Figure 15.0: PyTorch highway model results with the specified parameters

.




```
BATCH_SIZE = 2048

GAMMA = 0.9 #0.9

EPS_START = 0.999

EPS_END = 0.05

EPS_DECAY = 500

LEARNING_RATE = 0.00002679

NUM_EPISODES = 200

MAX_STEPS_PER_EPISODE = 100

MEMORY_REPLAY = 20500
```

Figure 16.0: Results from the PyTorch highway with the new specified parameters

.

```
BATCH_SIZE = 1024

GAMMA = 0.9 #0.9

EPS_START = 0.99999

EPS_END = 0.05

EPS_DECAY = 400

LEARNING_RATE = 0.00002679

NUM_EPISODES = 200

MAX_STEPS_PER_EPISODE = 100

MEMORY_REPLAY = 20500
```

Figure 17.0: Most optimal parameters for PyTorch Highway Environment

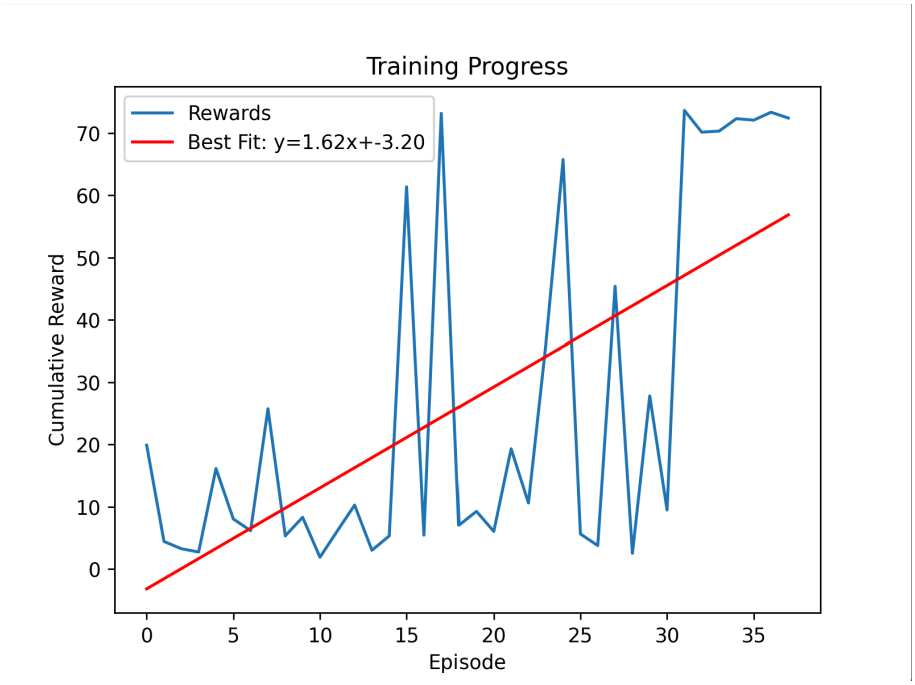


Figure 18.0: Initial training using the most optimal parameters for PyTorch Highway Environment

.

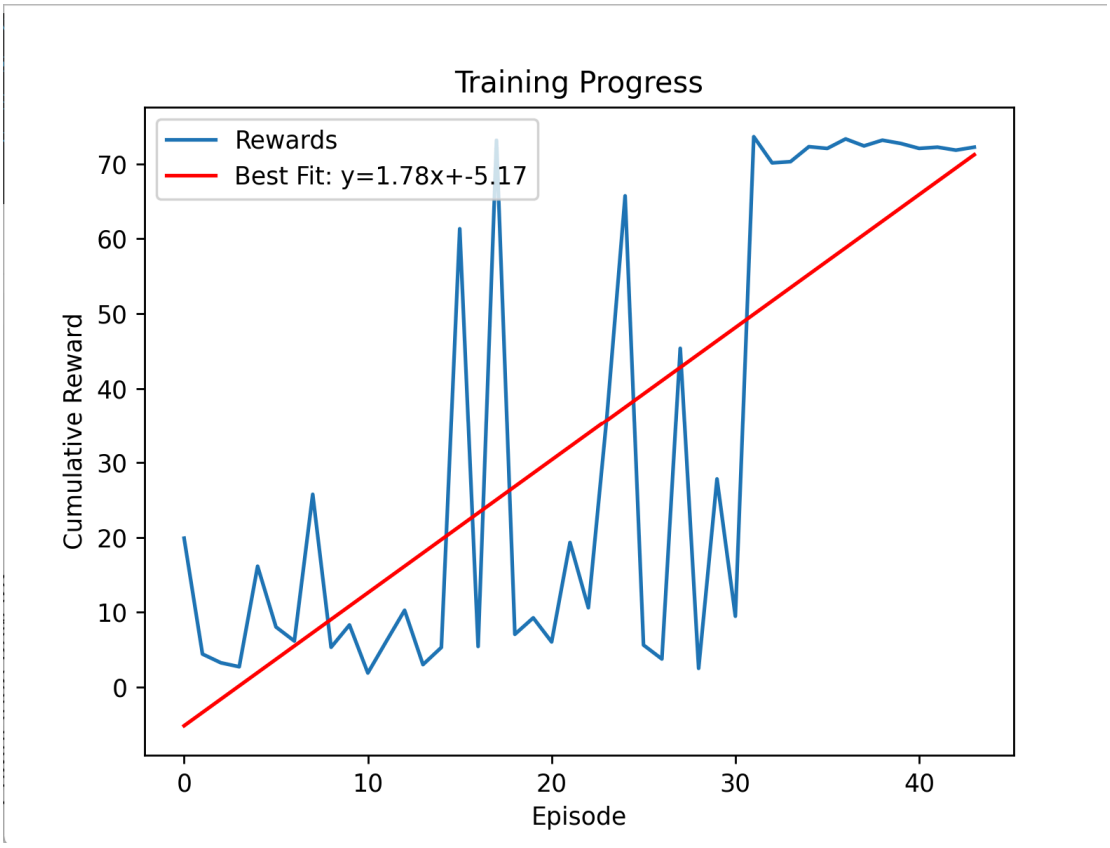
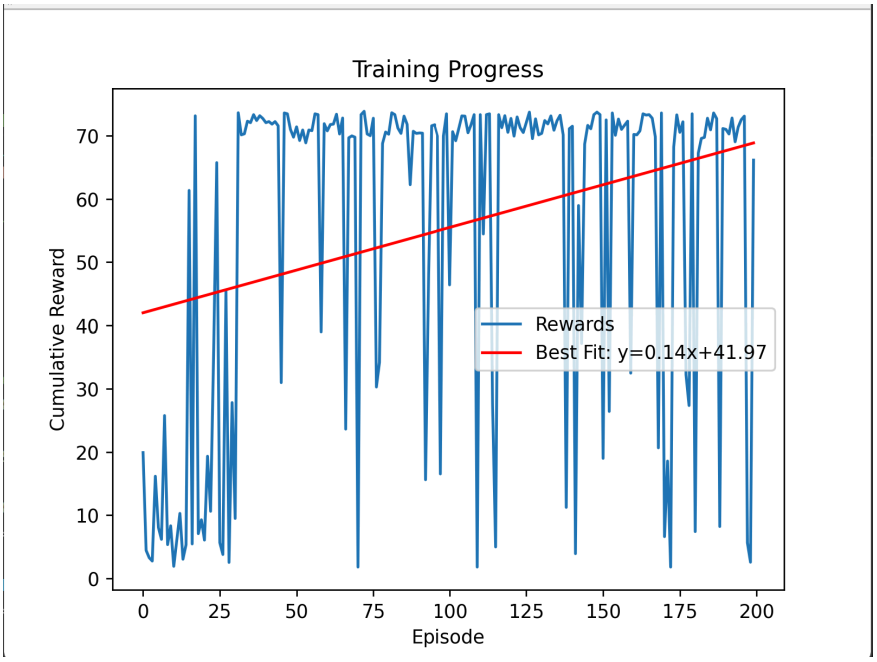


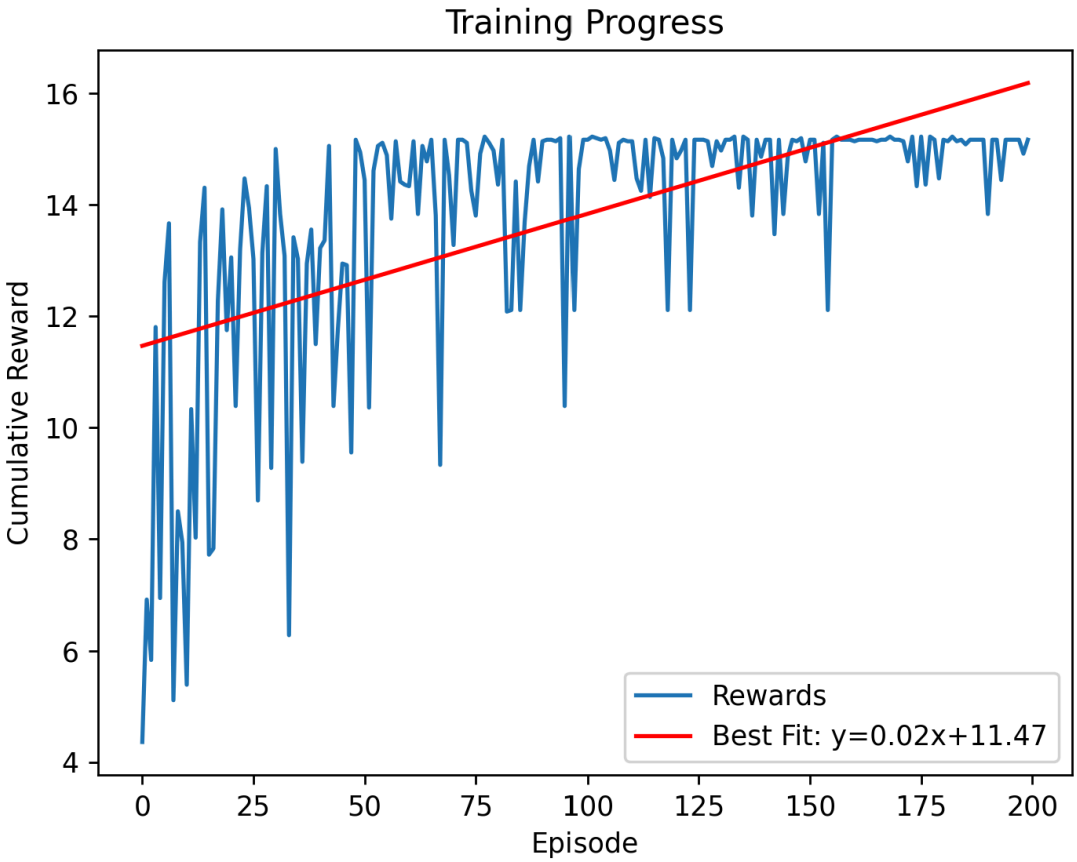
Figure 19.0: Continuation of optimal parameter training for PyTorch Highway Environment



.

Figure 20.0: Final results from the training for PyTorch Highway Environment

---

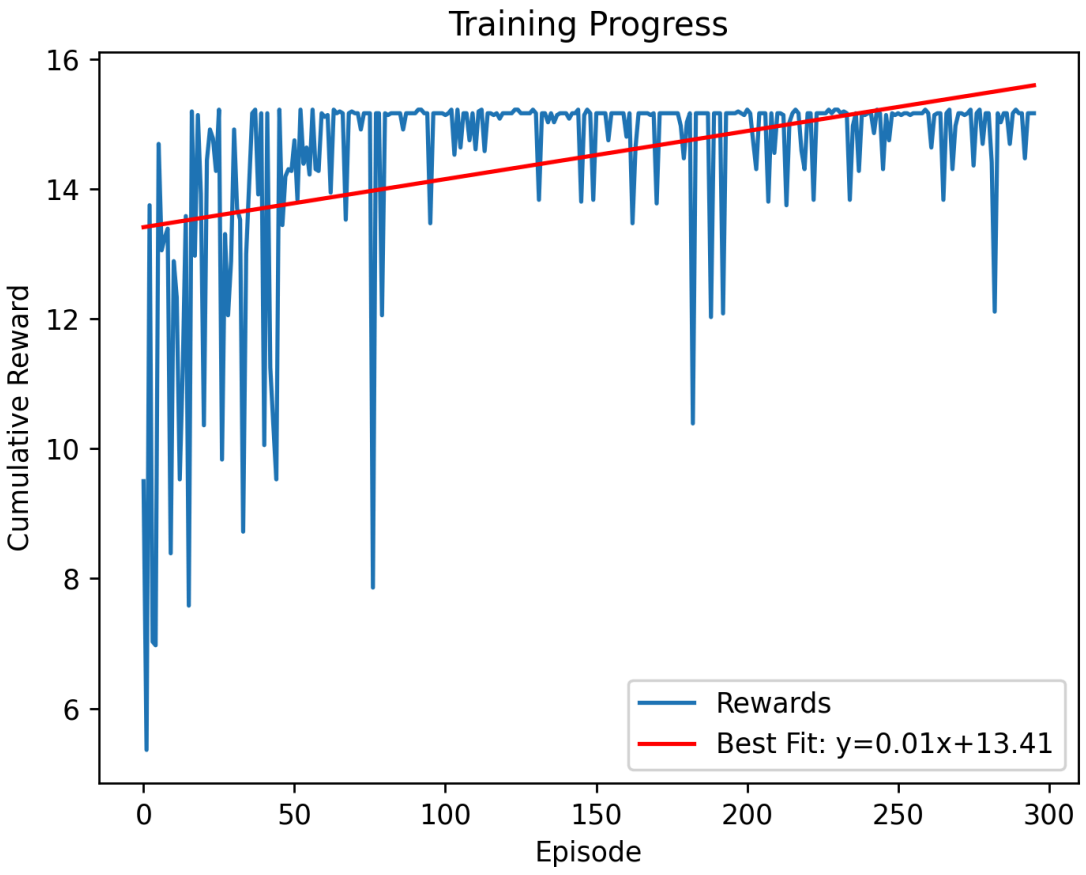



```
BATCH_SIZE = 1024

GAMMA = 0.8 #0.9 for highway, 0.8 for merge

EPS_START = 0.999

EPS_END = 0.05

EPS_DECAY = 500

LEARNING_RATE = 0.00002679

NUM_EPISODES = 200

MAX_STEPS_PER_EPISODE = 100
```

.

```
MEMORY_REPLAY = 20000
```

Figure 21.0: First successful run for training for PyTorch Merge Environment




```
# Hyperparameters

BATCH_SIZE = 1024

GAMMA = 0.823456 #0.9 for highway, 0.8 for merge

EPS_START = 0.9

EPS_END = 0.05

EPS_DECAY = 355

LEARNING_RATE = 0.00002679
```

.

```
NUM_EPISODES = 400

MAX_STEPS_PER_EPISODE = 100

MEMORY_REPLAY = 20500
```

Figure 22.0: Second run for training the PyTorch Merge Environment with tuned parameters

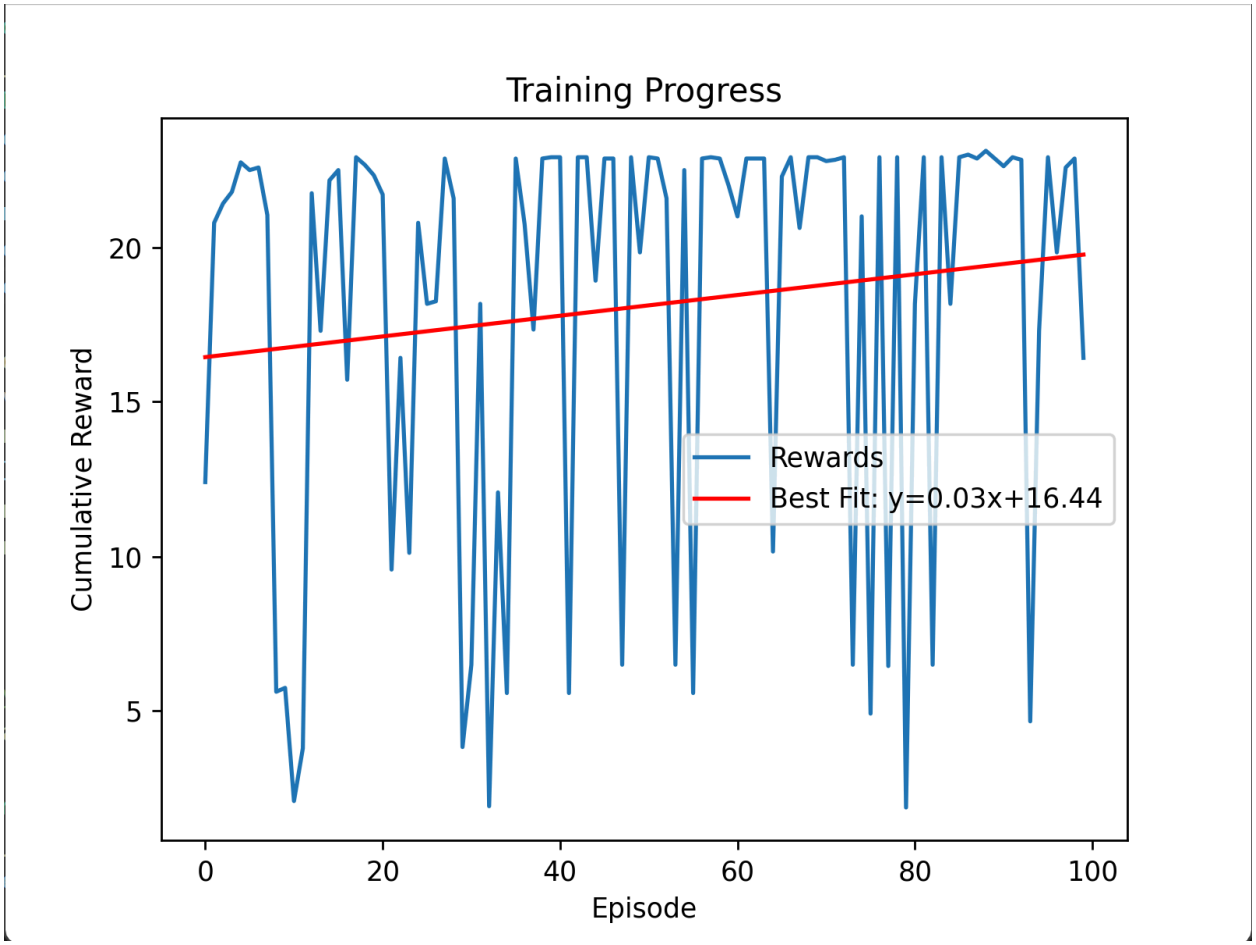


Figure 23.0: Initial run for training the PyTorch Roundabout Environment

.

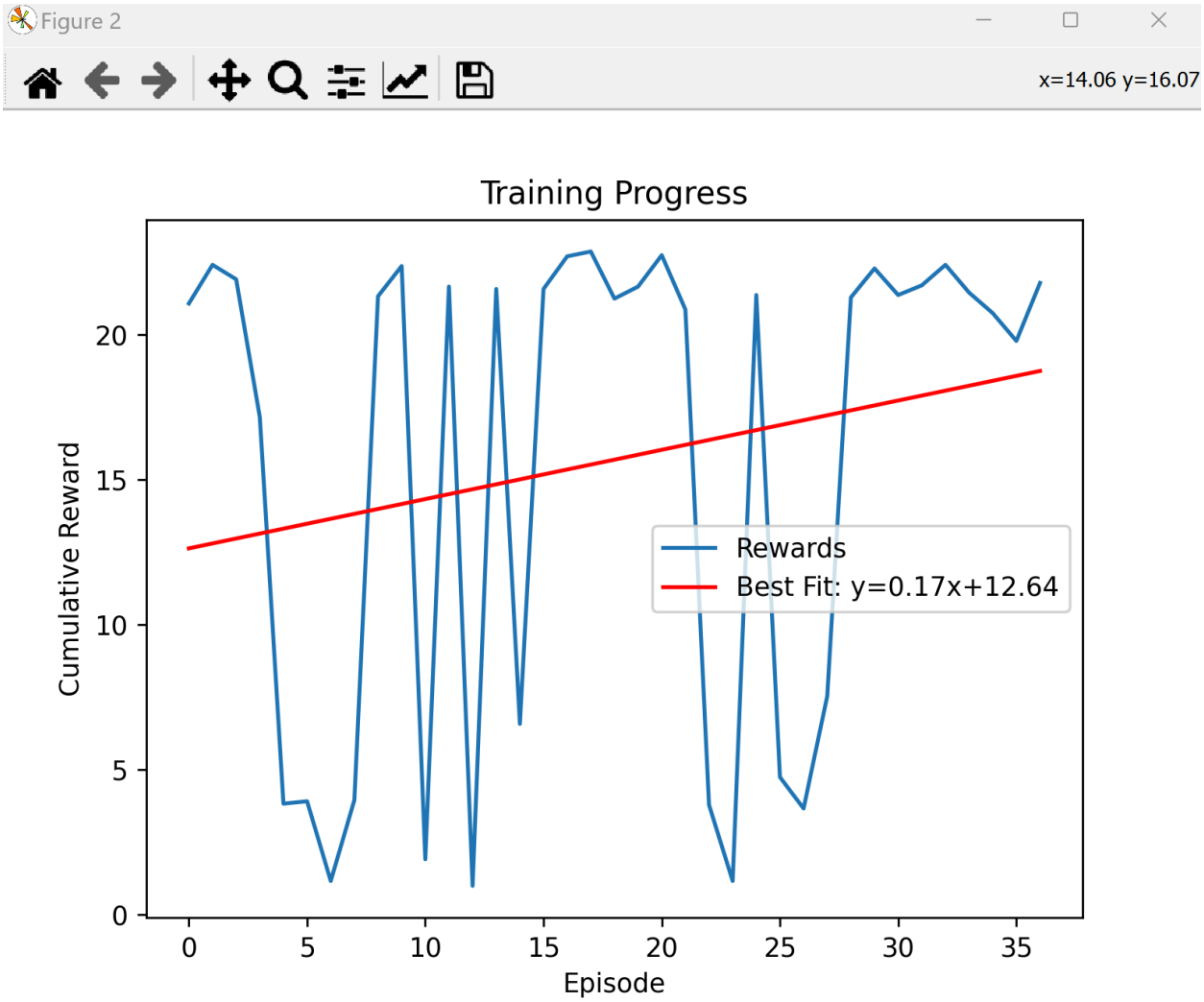


Figure 24.0: Second run for training the PyTorch Roundabout Environment with tuned parameters

Link to videos of running code:

https://drive.google.com/drive/folders/1A7vGhas9wXUwtRMuicu9_t4s6KUefa7y?usp=drive_link