

Project Final Report

Team 27 - Section 01

Andrew Lau (1004022336)

Hanke Liu (1004004365)

Peter Maksymowsky (1004401756)

Chris Lam (1003958531)

April 9, 2020

Word Count: 2495 (including figures) | Penalty: 0%

Link to **Google Drive**: <https://drive.google.com/drive/folders/1ICyl33TK7xL2Ug5md2wsTrAn73LV5PIs?usp=sharing>

Link to **Github Repository**: <https://github.com/andrewlaugit/reinforcement-learning-project>

Introduction

For our project, we implemented a reinforcement learning (RL) algorithm that learns how to play the OpenAI Gym Car Racing game. Our goal was to outperform our baseline, and also have a model capable of completing a lap of the track within the specified timeframe. This problem provides a very good opportunity for us to try out RL, and could have strong implications for real world scenarios depending on its performance. This includes autonomous driving, better AI games, more realistic robots and chip design. The project could serve as a first step of developing a plan for bringing reinforcement learning algorithms to the real-world domain once its performance is proven in the simulated environments of video games. The system would then be brought into real world applications once its architecture is improved and more stringent tests or simulations are applied.

Machine learning should be used because driving has countless numbers of conditions and state-spaces that cannot be coded in easily. As a direct consequence, driving games, which are simulations of real-world domains, should be approached with machine learning as well. Similarly, because of the vast number of possible states in the game, it would be impractical to use supervised learning methods so reinforcement learning methods must be used.

Background and Related Work

The breakthrough in Deep Reinforcement Learning is the introduction of Deep Q Networks (DQN) in DeepMind's paper *Human-level control through deep reinforcement learning* [1], where computer scientists trained DQNs to play around 30 Atari games based solely on visual inputs with varying levels of success. Our car racing task is similar to Atari gameplay, and thus we expected a DQN to perform well on Car Racing.

To solve the car racing task, we started our work based on Pytorch's *Reinforcement Learning (DQN) Tutorial* [2] as it is a simpler task and has a more straightforward tutorial to follow. The goal of the tutorial was to balance a pole on a cart in the OpenAI Gym environment. After modifying the tutorial's code to solve the cartpole task, we gradually adapted its architecture and modified it to solve the car racing task.

Data and Data Processing

The data and data processing techniques we used for our model are described below.

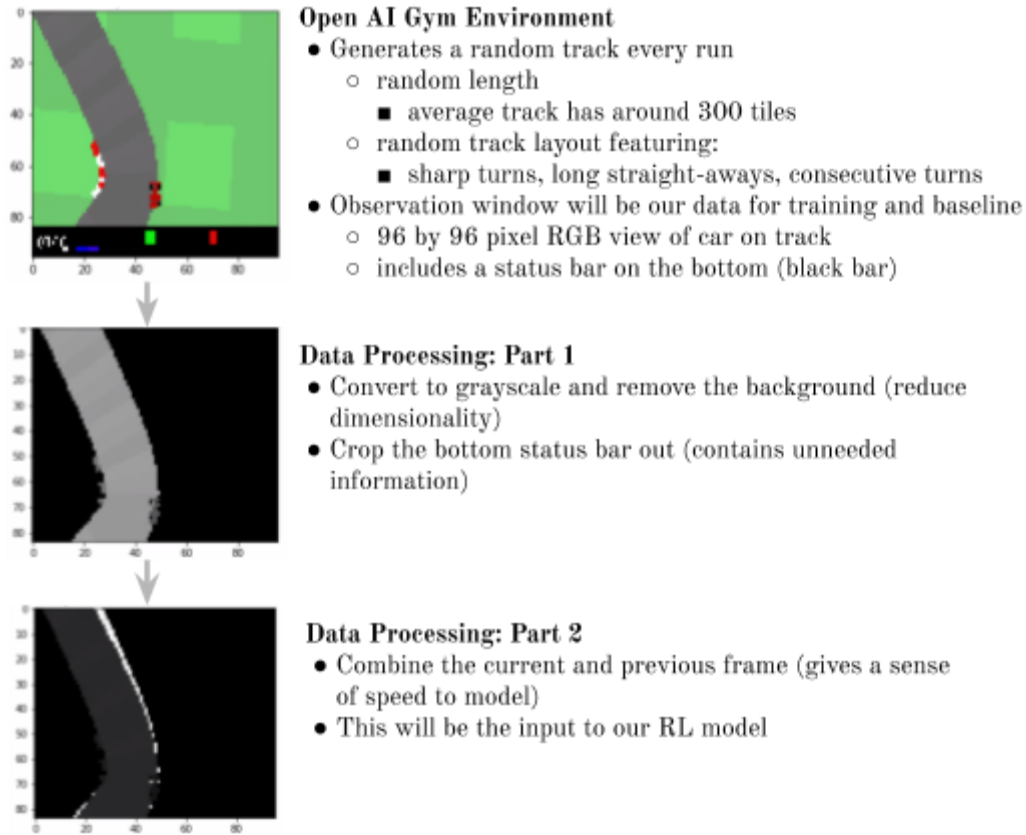


Figure 1: Data Processing Techniques for Car Racing

To train the RL model, a reward structure was needed. Fortunately, OpenAI Gym already includes the following reward structure: -0.1 for every frame played and $+1000/(\text{number of track tiles})$ for every track tile crossed. From training, we decided to tweak this reward structure slightly to stop an episode early if the car deviates too far off the track, giving a penalty proportional to the remaining frames in the run to speed up training.

To get a better sense of the data, there are a few more images from different parts of a run below.

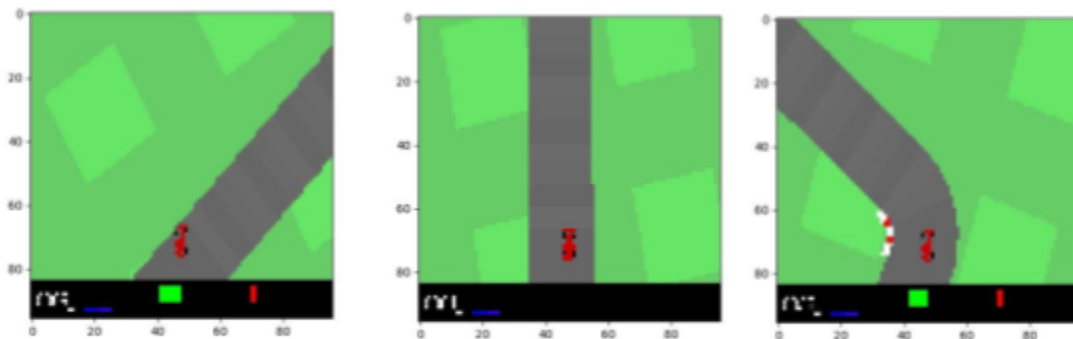


Figure 2: More data samples from the Car Racing environment

Architecture

For Deep Q learning, we need two identical CNNs: a Policy Net and a Value Net. The pipeline for our training loop is as follows:

1. Select action from Policy Net or a random action, depending on epsilon-greedy policy
2. Sample the environment states and save it as a transition in the Replay Memory
3. Take a random batch from the Replay Memory, compute the gradient with respect to the Value Net, and update the Policy Net
4. Every few episodes, copy the Policy Net parameters to the Value Net

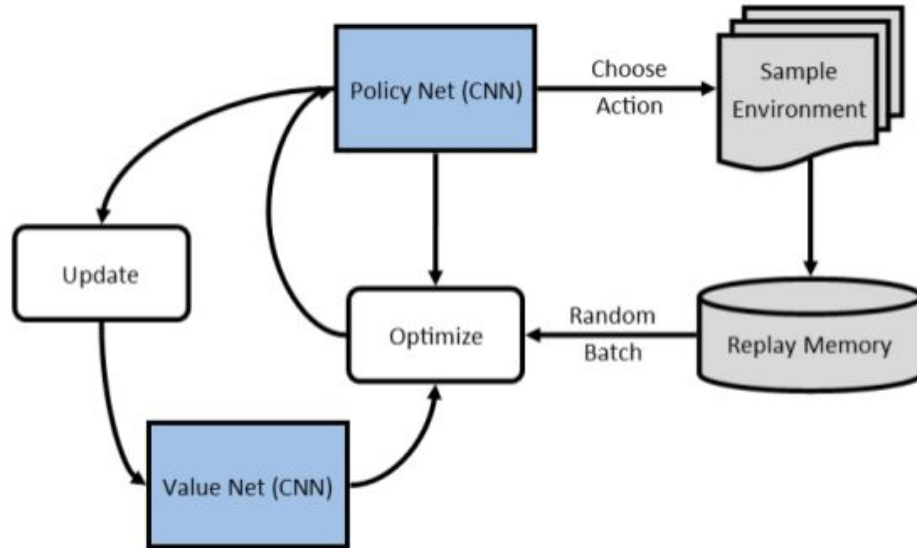


Figure 3: Training flow for Deep-Q learning

Now looking at the CNNs in more detail, we found that as we grew the network, the better it performed. The architecture of our CNNs is as follows:

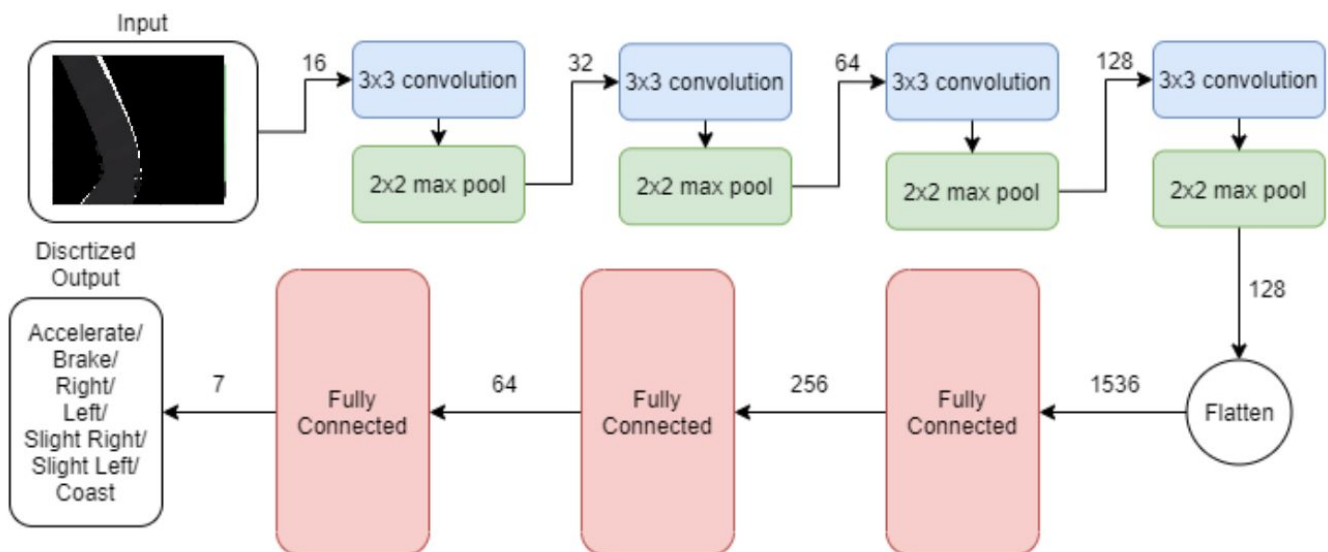


Figure 4: CNN Architecture for Policy and Value Net

Car Racing has a continuous action space, which leads to great complexity. To simplify the problem, we discretized the actions to the seven actions described below, developed through experimentation.

Table 1: Action Space Discretization

Action	Value (turn, acceleration, braking)
Accelerate	(0.0, 1.0, 0.0)
Brake	(0.0, 0.0, 0.3)
Right	(1.0, 0.05, 0.0)
Left	(-1.0, 0.05, 0.0)
Slight Right	(0.3, 0.15, 0.0)
Slight Left	(-0.3, 0.15, 0.0)
Coast	(0.0, 0.1, 0.0)

Originally, we found that the model would stop too often so we reduced braking intensity and added a little bit of acceleration to coasting. Then, we noticed that if the model was stopped, it would try to turn in place, so we added slight acceleration to turning, further encouraging forward movement. Finally, we noticed that the model would turn really sharply on straights so we added the option to make slight turns.

The other hyperparameters we used are summarised in the table below.

Table 2: Hyperparameter Summary

Hyperparameter	Value
Batch size	32
Gamma	0.98
Epsilon start	0.95
Epsilon end	0.05
Epsilon decay (per frame)	1e5
Target Update (every x episodes)	5
Adam Learning rate	0.0007
Adam weight_decay	1e-6

Baseline Model

For our baseline model, we used a similar data processing technique as described above. We converted the observation image to grayscale, removed the background, and cropped to an area of interest shown in red below. Using this processed image, we determine in which direction the wheels should turn. The speed of the car is maintained at 10% of max speed by using alternation between acceleration and braking. Maintaining a constant speed simplifies the model and prevents spin outs of the car.

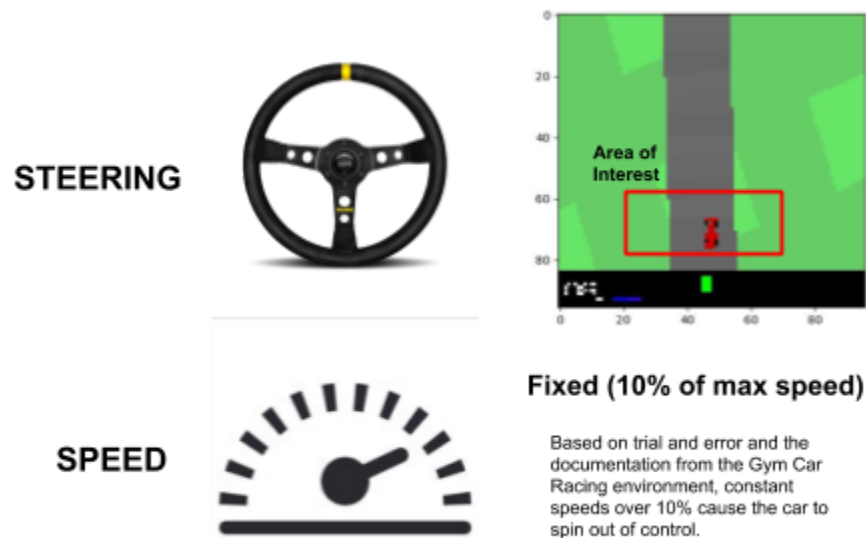


Figure 5: Baseline Model Description

From the area of interest, we can focus on either side of the car, and sum the total track area to the left and to the right. If more than 25% of the pixels to the left of the car represent the track, then the baseline will turn the car's wheels left. If more than 25% of the pixels to the right of the car represent the track, then the model will turn the car's wheels right.

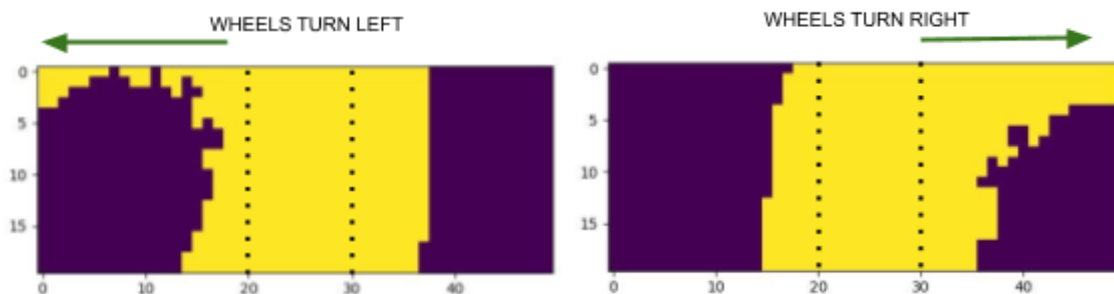


Figure 6: Examples of Data Prompting Left and Right Turns

Using the model described above, we took the results from 100 episodes of playing the Gym Car Racing game, and the results are below.

Table 3: Baseline Model Car Racing Results (aggregated over 100 episodes)

	Mean Reward	Minimum Reward	Maximum Reward	Number of Completed Runs
Baseline	170	123	234	0

Quantitative Results

To quantify the success of our model, we will use 3 measurements. These results will be gathered from 100 runs using the model, and are described below. The length of the track and the track layout differ each time, so using the results from 100 games gives a more complete picture of the models performance.

Quantitative Measurements of Success

1) Average Score

- shows how well or poorly a model performs overall
- with the random tracks, some tracks are 'easier' than others
- the higher, the better

2) Minimum and Maximum Score

- shows the model's worst and best case performances
- if range between the maximum and minimum is small, the model is consistent.
- If both the maximum and minimum scores are large, the model is performing well

3) Number of Completed Runs

- shows "how close to solving - problem" our model is
- if able to complete a large number of runs, we know the model is close to "perfect"





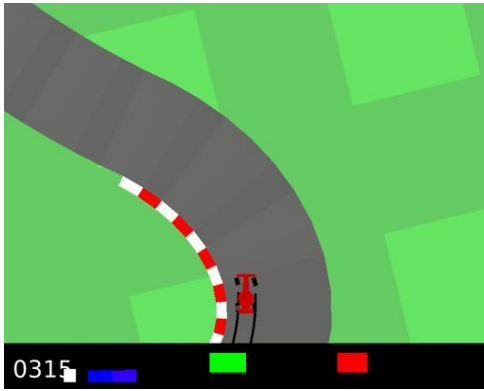
Figure 7: Quantitative Metrics for Car Racing

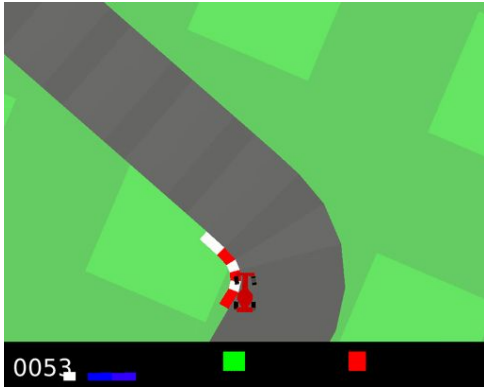
The scores mentioned above is the sum of all rewards during the run, and will start at 0, decrease by 0.1 for every frame played, and increase by $1000/(\text{number of track tiles})$ for every new track tile crossed. The minimum score possible is -100, and the maximum is 1000.

Qualitative Results

From our final trained RL model, we were able to analyze some completed runs:

Table 4: Analysis of Different Complete Runs

Run#	Screenshots/Links to Video	Notable Features
1	 <p>0904</p>	The car drives quickly, but constantly moves from the left to the right side of the track. At the very end, the car spins out at the finish due to this back-forth movement.
	Link to Video: https://drive.google.com/open?id=1cBAIjZvn-ycWz2Gg9RsO5tRSIFt1KEpC	
2	 <p>0399</p>	The car mostly sticks to one side of the track, and follows the "racing line" which allows it to make turns without braking and at speeds without spinning out.
	Link to Video: https://drive.google.com/open?id=16xJ0rZRGWi4EK7HQIkBZumuGyuuwlbQF	
3	 <p>0315</p>	The car is stable, and it knows how to make appropriate skid marks on the track, allowing it to drive smoother.
	Link to Video: https://drive.google.com/open?id=1SiTGjLjvylG4jddGgwDycjQg-TauchBl	

4		<p>One of the best runs recorded using our model, and although the RL agent doesn't drive perfectly and smoothly, it does not cut corners and is capable of capturing all the glowing tiles at the speed necessary to win the game.</p>
<p>Link to Video: https://drive.google.com/open?id=1Ls9VjT4QpNMDQP_oW5pB12tFUHxhHLcg</p>		

Generally, the agent performs well because the reward system was created in a way that forced the agent to quickly learn the appropriate weights by severely punishing the agent when it makes decisions that are fatal to the probability of winning the game. These fatal decisions include when the car is driven far away from the road, such that no track can be seen on the screen. It is also forced to have some minimal acceleration at the start of the game so it learns that it is necessary to drive faster early in the training session. The agent does not perform as well on maps with frequent and sharp turns that could cause the car to spin out due to the continuous back and forth movements.

Model Performance on New Data

In the Gym environment, tracks are generated randomly every episode, making data involved unique.

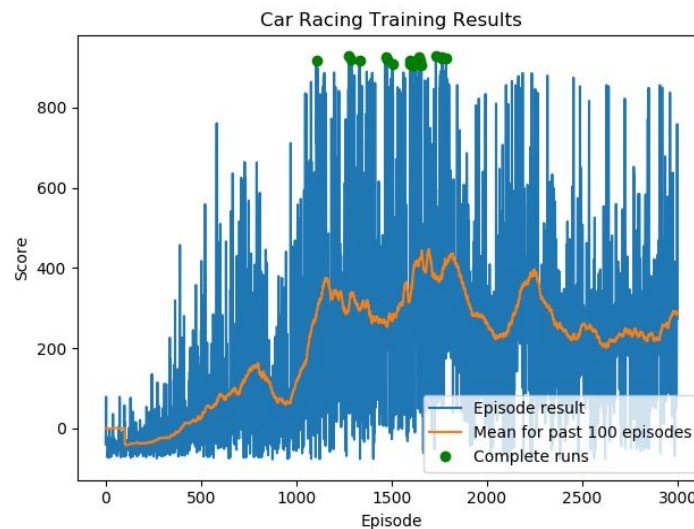


Figure 7: RL model's performance through 3000 episodes of training

As shown in the figure above, the training result does not necessarily improve as training progresses. We decided to perform an early stop and use the checkpoint at episode 1600 as our final model. Using the quantitative metrics described above, here is how our final model performs individually and relative to the baseline model:

Table 5: Comparison between our baseline model and RL model

Model	Mean Reward	Minimum Reward	Maximum Reward	Number of Completed Runs
Baseline	170	123	234	0
RL	447	-61	925	11

The baseline achieves a 100-run average score of 170, whereas the RL model is able to achieve around 2.6x that with an average score of 447, representing a significant improvement. The minimum and maximum scores widened relative to the baseline suggesting our final model is inconsistent. We are also able to get 11% of the runs to do a complete lap of the track with the RL model, whereas the baseline was only able to complete at most 1/4 of the track. Also, comparing the maximum scores, we can see the RL model was able to complete the track in only 75% of the allotted frames. We conclude that the RL model performs much better than the baseline model on new data. However, even with the relative success of our model, our model is not able to meet the Gym’s definition of solving the problem, which is defined to be when the model achieves a 100-game average score over 900.

Discussion

We were excited to see that the model was able to learn and improve over time. As shown in the results above, through training, we improved our mean score of -42 to 447 (for 100 episodes). Although we were not able to “solve” Car Racing (average score > 900), we are still happy with the model’s performance as it completed the game 11% of the time.

One unusual observation about our model was that after a certain point, our scores started to decline. From examining rendered runs, we learnt that the model would go too fast for its own good and spin out when trying to make a turn. This was the main cause for several of the troughs in our training curves as the model had to learn to slow down and approach turns better. However, in later stages of training, this was less noticeable as we saw worse performance overall.

We were surprised when tweaks to the model, which we believed would help it, greatly hurt its performance. For example, we attempted to feed more details for the model to learn (ie. remove the image processing) but the model struggled to learn and never showed promise while training. Another thing we tried was adjusting the epsilon curve for policy selection. We figured by making the curve decay slower and training the model longer, it would have more experience in different states and perform better, but this ended up degrading the training results. This was true even when we tried decreasing the learning rate.

We have learnt a lot from this project, specifically the pain, anguish, and struggles of Reinforcement Learning we were previously warned about (brutally long training times (5-24 hours per model), and the make-or-break aspect of the project). Most of our attempted models underperformed. Luckily for us, through tuning, our model was able to learn and improve over time. Another lesson learnt was our plan to start with something basic and gradually increasing the complexity turned out to be successful. We started with tutorials and solved an easier problem through OpenAI Gym’s CartPole environment, which

helped us grasp the concepts in RL before taking on the challenge of Car Racing. Despite the grueling weeks of training, we do not regret our choice of project because we learnt a lot along the way and felt like a proud parent when our model finally passed the road test and recorded perfect laps.

Ethical Considerations

The use of this system can create various ethical concerns that may vary in severity. Concerns that would be small in terms of controversy include offering this technology to gamers who may use it to gain an unfair advantage or modifying it into bots that overwhelm server performance. This will discourage other gamers and cause gaming companies to lose potential profits. More concerning ethical considerations include applying this system to real-life applications such as autonomous driving without sufficiently adapting or testing the system to ensure that it works effectively. Our agent was trained in a simulated environment with a smaller scope of factors relative to realistic scenarios so it would be dangerous and unethical to immediately assume it would work as effectively in self-driving cars. Even with extensive testing and modifications to adapt it for real-world scenarios, it will be difficult to determine if the RL agent has overfitted to the training environment. This could potentially result in loss of life and lawsuits to the designers of the system.

References

- [1] V. Mnih and K. Kavukcuoglu, “Human Level Control Through Deep Reinforcement Learning,” Deepmind, 25-Feb-2015. [Online]. Available: <https://deepmind.com/research/publications/human-level-control-through-deep-reinforcement-learning>. [Accessed: 20-Feb-2020].
- [2] A. Paszke, “Reinforcement Learning (DQN) Tutorial,” Reinforcement Learning (DQN) Tutorial - PyTorch Tutorials 1.4.0 documentation. [Online]. Available: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html. [Accessed: 20-Feb-2020].