Implementing and Analyzing Reinforcement Learning on Atari Freeway

Daniel Finelli
finelli.d@husky.neu.edu

Mittie Elaine Parr
parr.mi@husky.neu.edu

1 Goals

Our main and initial goal for this project was to implement a reinforcement learning agent. After starting, we added the objective of utilizing neural network programming libraries in our implementation. Finally, we wanted to explore and implement new theoretical concepts not learned in class like convolutional neural networks.

2 Introduction

OpenAI offers many environments to test and implement reinforcement learning. We both enjoyed the PacMan programming assignments, so we thought it would be fun to apply concepts we've learned to a different video game. We chose the Atari game, Freeway, which has the main objective to move a chicken across a freeway. We chose Freeway because it is similar to the video game, Frogger, which both of us have experience playing. This saved us time because we knew the basic rules of the game. The second reason we chose Freeway is because of its simple action space.

3 Problem Formulation

Freeway's objective is to score as many points as possible by moving the chicken to cross the road as many times as possible, while dodging cars, within the time limit. There are two players, but we reduced the scope of our problem to just one player (the left chicken). The state space consists of RGB pixels which is stored as an array of 210x160x3 values. The origin of the screen is the top left corner, so columns numbers ascend from left to right, and row numbers descend from top to bottom. The chicken starts somewhere around (191,48), and the goal row is 17. The chicken's possible actions are to stay, move up, or move down. Each time the chicken crosses the road, it is rewarded +1 point. The input to a learning model is the state space of pixels and the output would be an action-value pair.

Because of the problem formulation, we decided reinforcement learning would have the best tools to provide a high performing solution. Our initial hypothesis was: For Atari Freeway, a model implementation using a Q learning algorithm would produce a high-scoring agent.

4 Methodology & Algorithms

Q Learning

We developed an Approximate Q-Learning algorithm to (attempt to) train the agent to make the chicken cross the road. We chose Approximate Q-Learning because of the huge state space: every single frame of a game of Freeway could potentially be unique (since the cars all move at different rates and the chicken moves around as well), so it was important for the agent to extract significant features from its environment (such as a nearby car) rather than memorize states.

We used linear function approximation with the formula:
TD error: difference = $[r + \gamma \max_{a'} Q(s', a')] - Q(s,a)$
For each weight w_i corresponding to feature f_i: $w_i \leftarrow w_i * \alpha[\text{difference}] * f_i(s,a)$

What we saved in space by skipping the Q-table, we paid for in computational time each time we updated our weights. To run a single episode (one whole game, about 2730 frames) one step at a time, it took just under 3 minutes. To save time, we followed DeepMind's lead (Mnih et al, 2015) and had the agent choose an action and update corresponding weights every 4th frame, and repeat the previous action for the frames it skipped. This reduced our training time to about 40 seconds per episode. Although more manageable, this still significantly limited the number of episodes we could train the agent. We ran numerous variations and tweaked multiple components with little change in results, so here we will report only on 3 different approaches we compared: a naive Q learning strategy, a Q learning strategy with shaping, and a "head start" strategy.

For all 3 approaches, to make them comparable, we chose a discount factor of 0.99, a learning rate of 0.8, and a decaying epsilon that started at 0.2 and decreased by 0.01 each update (unless it reached 0, where it stopped). Epsilon was reset to 0.2 for each episode. We trained each agent for 100 episodes (plus 10 episodes for the "head start" agent), comprised of 700 iterations of the 4-frame action/update process described earlier (or until the game timed out).

Our naive Q-agent was to learn completely from scratch, choosing its actions according to their Q-values based on the weights and the features for that state, action pair. Our shaping strategy was exactly the same, except we also provided a "mini" reward for getting closer to the actual goal; we chose the chicken's current lane number / 22 as the miniature reward, hoping it was enough to prompt the agent to move forward into higher lane numbers, without actually outweighing the reward of 1 for crossing the road. For our "head start" strategy, we ran the Q-agent through 10 episodes where its only action was "move forward", and updated the feature

weights accordingly as it went. Then it ran 100 episodes exactly like the naive agent, but already having the weights from its 10 previous episodes.

Features

We extracted features based on the area around the location of the chicken (indicated by "yellow" pixels in a particular column), and the location of the chicken itself, because those are the most salient aspects of the environment. We used gray pixels as an indication of the lack of cars (so we didn't have to look for all the different colors of the cars in the array) immediately around the chicken. In greater detail, our features were as follows:

Feature 1: "Gray ahead"	Number of gray pixels in the chicken's column, from the chicken's row to 16 rows ahead of the chicken
Feature 2: "Gray behind"	Number of gray pixels in the chicken's column, from the chicken's row to 16 rows behind the chicken
Feature 3: "Horizontal Gray"	Number of gray pixels in the chicken's current row from 16 columns to the left of the chicken to 16 columns to the right
Feature 4: "Current Lane"	The current lane the chicken is in (out of 12; 1 before the road starts, 10 on the road itself, and 1 after the road ends)

We initially separated out the directions because it was important for the agent to know if a car was in a neighboring lane in the direction the chicken would move, and the cars in different lanes come from different directions. However, since the direction the car is coming from depends on which lane the chicken is in, we combined the left and right features.

We attempted to use lives as a feature until we found out the game didn't track lives at all. We also tried to use the distance of the chicken from the goal as a feature, but had difficulty balancing the large pixel numbers (that also were in reverse order) and chose the lane number instead.

After a lack of success with our Approximate Q-Learning agents (explained in Results), we decided to move to a deep learning approach.

Deep Q Learning (DQN) and Double Deep Q Learning (DDQN)

The Deep Mind Research paper, Playing Atari with Deep Reinforcement Learning, explains that their algorithm, DQN, uses Convolutional Neural Networks (CNNs) which learn features from an input of pixels (Vlad Mnih, Koray Kavukcuoglu, et al, 2013). Two years later, DeepMind

developed an improved deep learning algorithm, called Double Deep Q (DDQN) Learning (van Hasselt, Guez, Silver 2015). DDQN chooses an action using two models, the "Q-Network model" to choose an action and the "Target Q-Network model" to evaluate an action. In DQN the action is both chosen and evaluated in the same model which consequently causes the agent to prefer overestimated values because it is choosing an action according to maximum q value.

Initially we tried building DQN models from scratch but were perplexed by the complexity of CNNs and the associated neural network libraries. We used an implementation of DDQN that worked for Atari Space Invaders and provided useful guidance (Sagar, 2019). We tested the existing implementation for DDQN and reengineered it to function with Freeway. The DDQN implementation has some key features. First, it saves one state as a sequence of 3 frames. This gives the agent an understanding of motion. Instead of seeing a static frame of pixels, the agent can see 3 frames understand a car moving. The implementation uses an experience buffer to store a limited number of state experiences. The states are converted to grayscale pixels and dimensions are reduced to 84x84. There are two models, the Q Network model and the Target Q Network model. Both are made of the same CNN structure based on DeepMind and use Keras. The preprocessed sequence of states are fed into the first hidden layer which has 16 8x8 filters and stride 4 using a relu activation function.The second includes 32 4x4 filters with stride 2 using a relu activation function. Theres is a fully-connected hidden layer at the end. The output is a Q value for all 3 actions on a fully connected flattened layer. Finally, the Adam algorithm is used as the optimizer.

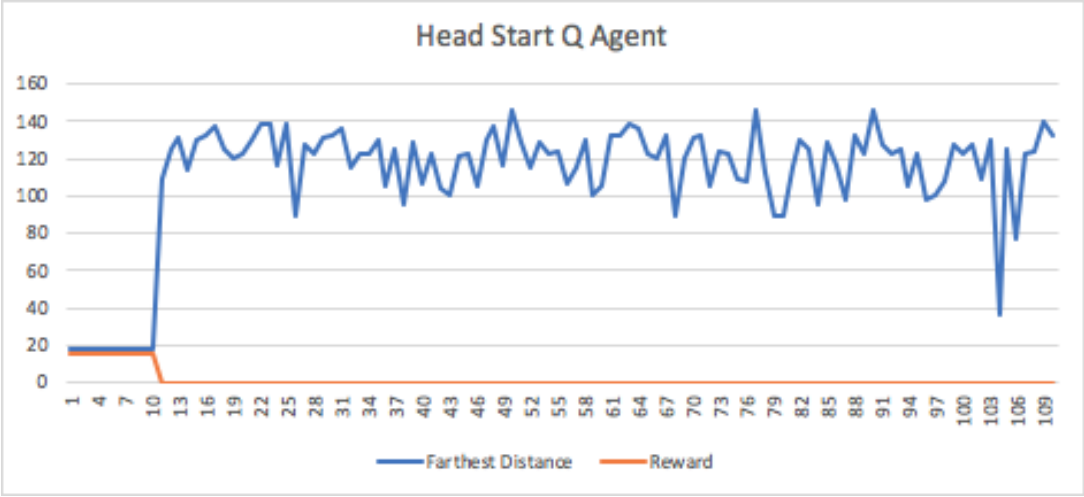
5 Results and Analysis

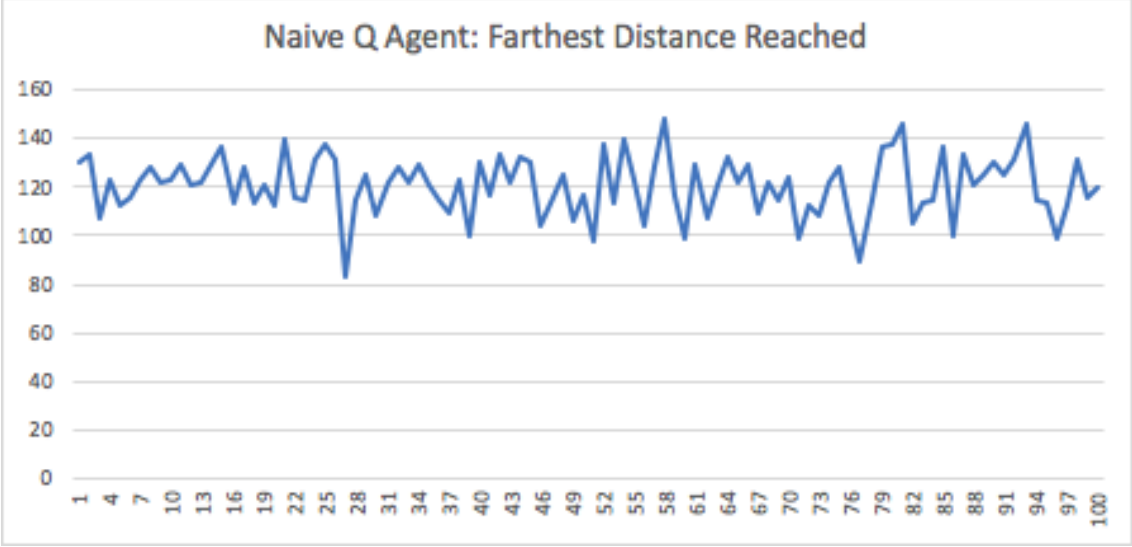
Q Learning

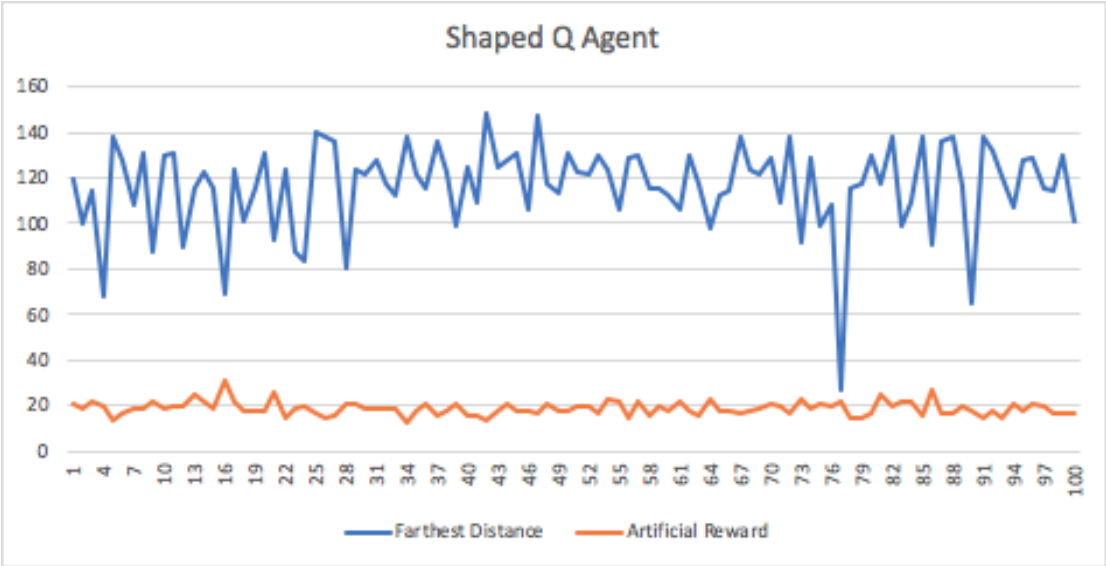
None of our variants were successful at getting the agent to learn to cross the road at all, and none of them received any reward (as defined by the game) for any episode in their training runs (minus the 10 "move forward" trials for the head start agent). The shaping we implemented wasn't very effective, and the head start agent jumped right back to naive performance after its 10 "forward" trials.

As measured by farthest reached point, (mean, median, and absolute) it appears that the shaping agent performed slightly better than the other two, but we did not run the analyses to determine statistical significance. It is possible that given many more training episodes these small differences would spread. Note that the pixels for the rows are numbered in descending order from bottom to top, so a lower farthest point reached is a better score.

	Farthest Point Reached	Average FPR	Median FPR
Naive Q	83	120.59	122
Shaped Q	27	116.6	120.5
Head Start Q	37	119.75	123







It also did not appear the agents improved their performance from beginning to end, as measured by the FPR means and medians of the first and last 10 episodes.

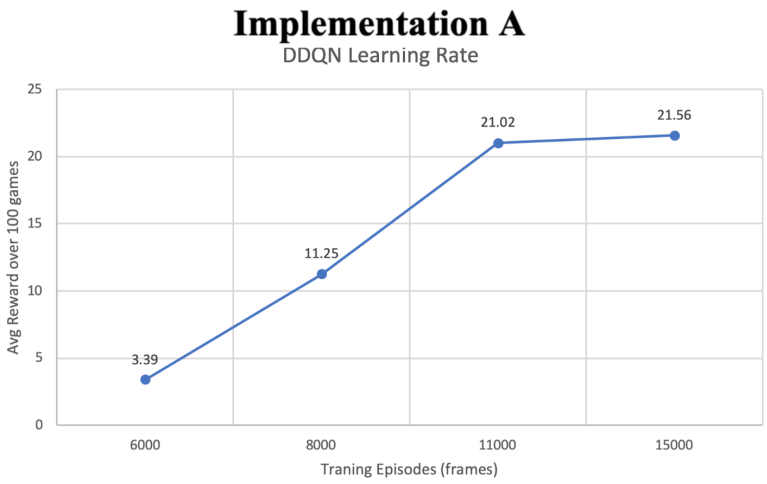
Agent	Mean First 10	Mean Last 10	Median First 10	Median Last 10
Naïve Q	121.7	121	123	118
Shaped Q	112.5	121.5	117	121
Head Start Q	124.7	112.4	125	124.5

Double Deep Q Learning (DDQN)
Our main objective with DDQN was to see how we changing different features of the implementation would impact the agent's learning ability. We have 4 implementations we tested. For **Implementation A**, the first round of training, our initial parameters were:

- Decaying epsilon greedy exploration policy
- Learning Rate of .01
- Discount rate .99

With this implementation we observed that the agent would easily get fixated on a policy once learning it was the "best" and would not be able to try other actions. Within the first few actions if the agent received a maximum Q value for going down or staying still, then the agent would not be able to learn beyond this. The agent exploited this policy and never found a reward because it is far away in Freeway. Sometimes serendipitously the agent fixated on going up and

found the reward which exacerbate the fixation. The agent would only repeatedly go up and maximize its overall reward. We hypothesized the agent would learn to choose between staying still, up, or down in order to dodge cars while crossing the road. In reality, the agent just learned to spam going up, down, or stay still. The following results are when the agent learned to go up. We omitted some results where game score averages = 0 because in these scenarios, the agent either stayed still or continuously went down at the starting line. It was most interesting to see how the number of training episodes did help the agent to perform better, despite the overall policy being so trivial. Also, it seems that the highest achievable score was around 21. This began to converge after 10,000 iterations. Intuitively it makes sense because repeatedly going up can only achieve so much in the allotted time. These results can be seen in the below graphs:



We hypothesize that with enough training time it is possible for the agent to learn combination of actions (for example: up, down, up, stay) to dodge cars. Since we are limited on time, we wanted to see if we could find ways to increase the learning rate by tweaking the implementation. For **Implementation B** we had the following parameters:

- "Up-only" initial exploration policy
- Learning Rate of .01
- Discount rate .99

The key to this implementation was to force the agent to move forward for the first 3000 frames. This would load the experience buffer with scenarios where the agent crosses the road by only going up. We hypothesized that the agent would find that the upward action is the best action quicker than the epsilon greedy strategy. This learning approach is analogous to a human telling a novice human Freeway player to continually going up to understand how the game works. We quickly saw that this reduced the required training time for the agent and confirmed our

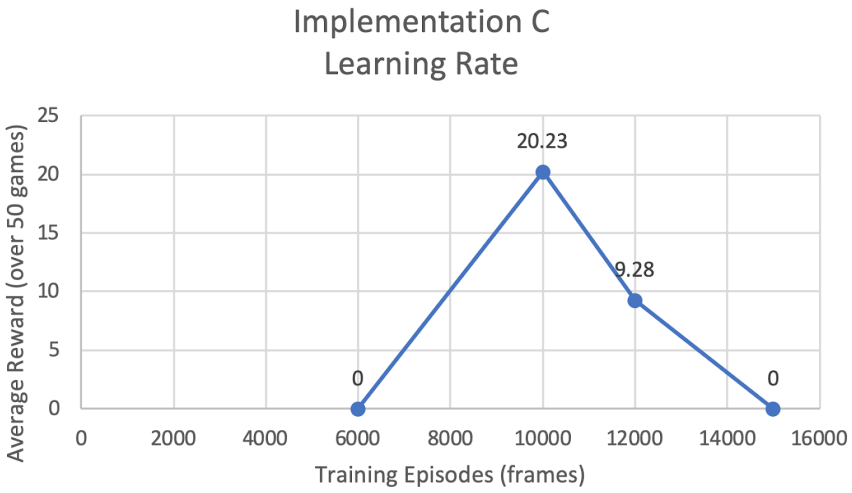
hypothesis. The below chart shows in just 3000 training episodes, the chicken has already learned the score that took 10,000 training episodes in Implementation A. Looking at the agent's movements we see that it is continually going up similar to Implementation A. By "artificially" loading the experience buffer, we taught the chicken this policy much quicker.

Implementation B Learning Rate	
Training Episodes	Avg Reward (over 50 games)
3000	20.22
6000	21.25

Next, in **Implementation C**, we wanted to try an entire new implementation from scratch. We used the following parameters:

- .3 Exploration policy of either going up or down (never choosing to stay still)
- Learning Rate .5
- Discount rate .5

We wanted to see if dramatic shifts in learning and discount rate would impact the learning agent. Also we force the agent to have a high degree of randomness, but only restricting its movements to up and down. Instead of definitively spamming up/down/stay, we hypothesized that these parameters would allow the agent to learn how to dodge cars by going up or down. Unfortunately we received poor results. Again we saw the agent getting stuck in some scenarios and would never reach an award which can be seen in the 16,000th training episode example. Sometimes the agent got lucky and exploited an up-only policy which can be seen in the 10,000th training episode. On the contrary, in the 12,0000th training episodes case, we observed the agent doing a combination of up and down movements. The Q values for this specific case were more even between all actions so the agent seemed to be choosing between them instead of only exploiting one. This implementation further shows the difficulties with limited training time. Also, illustrates that the results are convoluted and difficult to interpret with certainty.



We do not have the resources or time to train for long periods. Therefore in **Implementation D**, we provided an artificial reward for going up and down. We used the same setup in Implementation A, but added a +1 reward if the agent went up, +.5 if the agent went down, and -.02 if the agent stayed still. We multiplied the actual reward (for crossing the street) to ensure this would be a higher value compared to the step rewards. We are aware that these changes may be out of the scope of reinforcement because, but we were curious of the results. The result was more volatility of actions being chosen. This gave the guise of an agent dodging cars. In reality, the movements' origins were difficult to interpret. We believe the agent was making too many random actions which caused average scores varies. The artificial reward didn't make a because impact because in some cases the exploitation of one action was used. The data was convoluted and therefore we couldn't make strong analysis..

6 Difficulties

The most crippling difficulty we faced in training our agent was the delayed reward. The agent does not receive a reward at all until it has completely crossed the road. As a result, our Approximate Q-Agent never made it across the road at all, and the DDQN agent is a slow learner. The game timer also makes this difficult because the agent is cut off from reaching the reward early. We have tried to implement shaping to help our Approximate Q-Agent for travelling closer to the goal, and played around with different exploration and learning rates, but we haven't found a successful algorithm for getting the agent to cross the road even once.

Compounding this difficulty, the game does not track lives, so penalizing the agent for colliding with a car is tough to implement. We considered simulating the life loss by counting how many times the chicken suddenly moves back to the starting position from the previous state, however, this would be confounded by the fact that the chicken suddenly moves back to the start space when it reaches the goal as well, in which case we would be penalizing the agent for doing exactly what we want it to do!

While training the DDQN models, it took an extremely long time to understand what the agent was doing. After training for 10,000+ iterations, sometimes we would find out the agent was stuck in a policy. Therefore the time taken to try new implementations and debug code was significant.

The unexpected challenge we faced was our own ambition. We chose a problem that was large, but should have started with a smaller problem and expanded. It took a significant time for us to learn about CNNs because we had to review linear algebra and calculus concepts to truly understand the mathematical underpinnings. Despite the time investment, we learned more about the architecture of neural networks which aligned with our goals.

7 Future Direction and Discussion

Q Learning

We were disappointed in the performance of our Q learning agent. Ultimately, we suspect our features were to blame for this poor performance. It is very difficult for us to think of the game solely in terms of the RGB values, and perhaps the ones we chose were not as helpful as we thought. One reason the features could be at fault is there are more cars towards the middle of the road in general, so perhaps the lighter density of gray pixels was discouraging the agent from moving into that area at all. Perhaps a more complex feature space could have covered more aspects of the environment and better represented the game for the agent to learn.

Another possibility is that each individual action matters more in Freeway than in other Atari games such as Space Invaders and Breakout, and repeating an action 4 times in a row doesn't work well in this environment. We did initially run dozens of training sessions taking one action at a time, though, and didn't see much difference (although in the case of Freeway, inches could be miles). We just didn't have the time to explore this possibility further.

Given more time, we would have liked to explore more shaping functions to find the reward "sweet spot" for the agent. We also would have liked to find a way to penalize the agent for getting hit by a car. However, we think working on the features would be most likely to improve the agent's performance, because the head start agent had 10 trials telling it almost exactly what to do and still performed poorly. The hardest part is figuring out which features would need to be added.

Lastly, it could just be that Approximate Q is a poor choice for this application, and DDQNs and other networks are better options.

DDQN

Some of mentioned Freeway's learning difficulties are shared by a subset of Atari games. There is research that has tried enhanced deep learning approaches that may help. For example, policy distillation with DQN was shown to slightly improve the score of Freeway (Rusu, et al, 2017). Also, research using "human demonstrations" like pre-training a network showed significant improvements for an agent learning Freeway (de la Cruz, et al, 2019). Our experiments tried some basic pre-training, but this would be an interesting area to expand upon.

It would be interesting to explore adding layers to the CNNs and altering the CNN parameters (stride, padding, and filter). We used the CNNs from DeepMind similar to a black box and didn't make changes to it. Deconstructing and altering the CNNs would be interesting to better understand max pooling and other CNN concepts we learned from online research (Jiwon, 2019).

We hypothesize that reducing the view of the CNNs to the column near the chicken's location would reduce computation and provide better feature detection because this space is more relevant to the chicken and its actions. It is similar to telling a human to focus their attention on the middle-left side of the screen.

We focused the majority of deep learning on altering DDQN implementations. Therefore, comparing DDQN to other algorithms, like DQN, should provide varying results because of the complexity of Freeway.

Goals

We were able to meet our core goals for this project and consequently learned a significant amount. We fulfilled our goal of implementing reinforcement learning models that train an agent in Freeway. Inadvertently we utilized neural network libraries and learned new theoretical concepts like convolutional neural networks.

In conclusion, we have failed to find support for our hypothesis that approximate Q learning is optimal at providing a high scoring learning agent in Freeway. Deep learning has proven to be more successful and produced tangible results which reflect topical research in reinforcement learning. With this evidence, there were some unknowns and volatility. Therefore there is still more validation and future work that can be done to efficiently and effectively teach a reinforcement learning agent to play Freeway.

References

de la Cruz, Jr., G., V., Du, Y., & Taylor, M. E. (2019) Pre-training Neural Networks with Human Demonstrations for Deep Reinforcement Learning. *Cambridge University Press*.

van Hasselt, H., Guez, A., & Silver, D. (December 2015) Deep Reinforcement Learning with Double Q-learning. *NIPS*.

Jeong, J. (January 2019) The Most Intuitive and Easiest Guide for Convolutional Neural Network. accessed from https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (December 2013) Playing Atari with Deep Reinforcement Learning. *NIPS*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G. et al. (2015). Human-level control through Deep Reinforcement Learning. *Nature, 518,* 529-533.

Rusu, A., A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., & Hadsell, R. (2017). Policy Distillation. *ICLR*.

Sagar, A. (July 2019). Deep Reinforcement Learning Tutorial with Open AI Gym. retrieved from https://towardsdatascience.com/deep-reinforcement-learning-tutorial-with-open-ai-gym-c0de4471f368

Freeway Image, www.atarimania.com.