

---

# Reinforcement Learning for Road Crossing Games

---

Marlon Facey, Abraham Hussain, Jeffrey Pak, Kevin Tat, Michelle Zhao

## 1 Introduction

Road Crossing games such as the Atari games, *Freeway* and *Frostbite* have many opportunities for machine learning research. Atari games were a region of interest for DeepMind Technologies. DeepMind Technologies created a variant of Q-Learning called Deep Q-learning that takes in a representation of the game as raw pixels and returns a value functions that is used to estimate the reward of future actions [1]. Due to these Reinforcement Learning (RL) techniques, this model was able to outperform any previously used model in about 6 Atari games as well as outperform expert human players in 3 of those games. Some of the initial problems with using the Deep Learning modification were that Deep Learning models require there to be an abundance in trained label that are independent. We can obviously see that Atari games specifically work by having states that are not independent but instead depend on heavily on previous states that could possibly have long chains of states.

The way that DeepMind Technologies fixed this problem was using an experience replay mechanism, which allows the agent to learn from earlier memories can speed up learning and break undesirable temporal correlations [2]. This was beneficial because the model converges slower and learns from multiple passes as well as having a better convergence.

The most current technique at the moment are called Schema Networks which is a physics simulator that is capable of disentangling multiple causes of events and reasoning backward through causes to achieve goals [3].

During this paper we will talk more about the previous work done to to solve Atari games and what what research questions we had specifically for road crossing type of games. We will then cover our approach, results and any further direction that we could have potentially taken.

## 2 Previous Work

Learning to control agents directly from high-dimensional game spaces is one of long-standing challenges of reinforcement learning. Recent advances in RL have included utilizing neural network architectures, including convolutional neural networks, recurrent neural networks, multilayer perceptrons, and restricted Boltzmann machines [4]. However, reinforcement learning on high dimensional inputs still face several challenges from a deep learning perspective. Most deep learning applications require large amounts of hand-labelled training data. RL algorithms must be able to learn from a scalar reward signal that is often sparse, noisy and delayed. Deep

learning also generally expects training input to be independent, but in RL for game states, the data distribution changes as the algorithm learns new behaviors, resulting in training input that is continuous and highly correlated.

Recent progress in the application of RL with deep neural networks to Atari games has led to *DeepMind*'s deep Q-learner network (DQN), which can autonomously learn how to play various Atari games, at or above the level of humans. Deep Q Learning (DQN) overcomes unstable learning on high-dimensional Atari games by using the techniques: experience replay, target network, clipping rewards, and skipping frames [4]. Experience relay stores experiences including state transitions, actions, and rewards, and makes mini-batches to update neural networks. This reduces the correlation between experiences in updating the deep network, which reduces overfitting, and increasing learning speed with mini-batches. The target network technique fixes parameters of target function and replaces them with the latest network every couple thousand steps, which also reduces overfitting and prevents too frequent changes of the target function. Clipping rewards normalizes or reduces rewards to be within a range. Skipping frames is exactly that, because some frames are very similar to its neighbor frames that no new information is gathered from the frame, so the number of frames can be scaled down for increased speed.

Previous research in DQN reinforcement learning has created the technique called *distilling*, which is essentially an ensemble method applied to neural networks [5]. Using the distilling approach, DQN agents are first trained on two individual games and then fused together to form one agent which applies to both games. The distilling technique shows that it's possible to compress knowledge learned from two models in an ensemble into a single model that is easier to deploy using different compression techniques.

Additionally, recent work has shown that when training an agent to learn one task, and second sequentially, a DQN will suffer catastrophic forgetting. Catastrophic forgetting occurs when new tasks are introduced to a network, and typical deep neural networks are prone to forgetting previous tasks and destroying the network weights associated with the first task. Greater catastrophic forgetting occurs when the two tasks are differ extremely from each other. One method of reducing this issue with sequential learning is using elastic weight consolidation, where weights that are important for the first task are protected from large changes when learning the second task. Our work focuses on applying similar models to similar games. By observing performance of DQN models on semi-similar games: *Freeway* and *Frostbite*, we can then extend these models by creating an ensemble model with both, or applying these models to sequential learning.

### 3 Research Question

The primary goal of this research project is to compare the performance of different models on similar games. Specifically, we want to analyze the efficacy of training AI on simpler games prior to training on the complex game compared to training purely on the complex game. In this

project the simple game will be *Freeway* for the Atari 2600, a simple road crossing game where players control a chicken that must traverse a road while avoiding collisions with cars. The only controls in this game allow the chicken to go either up or down. The complex road crossing game is *Frostbite*. In *Frostbite* a player can move up, down, left, or right, and jump on moving platforms that move the player horizontally across the screen.

Questions and research goals we are attempting to answer include:

1. How transferable is knowledge gained by simple models to complex models?
2. Will models heavily trained on the simple model become too inflexible to adapt to the new game states and actions of the complex game?
3. What are the benefits of training large numbers of simple agents and utilizing the best agents as beginning models for complex games? Are there any cons?
4. Compare the performances of agents trained for different number of generations in the simple game to one another and an agent trained directly on the complex game.
5. If there is time, can we go the other direction and examine the performance of agents trained on complex games on simple games. Will they perform better than agents trained only on the simple game?
6. How does a model trained on a simple game with a smaller action space perform on a more complex games with a larger action space?
7. Which models are best for games with large state spaces and how do these models perform on other similar games?
8. How much do small changes in the reward function affect the overall performance of the model?
9. Can we improve the performance of the model by increasing the penalty for time spent playing the game?
10. What is the effect of exploration on the performance of the model (For example taking an action that our model would not predict to have the highest reward occasionally)?

## 4 Approach

Our goal is to develop an agent that learns and plays *Freeway*, a video game for the Atari 2600 console. We hypothesize that this agent can be trained to play this game using reinforcement learning. However to test this hypothesis, we must first analyze the game we are playing and find a suitable state-action space for the agent and then figure out how to best design a Q-learning model to train the agent.

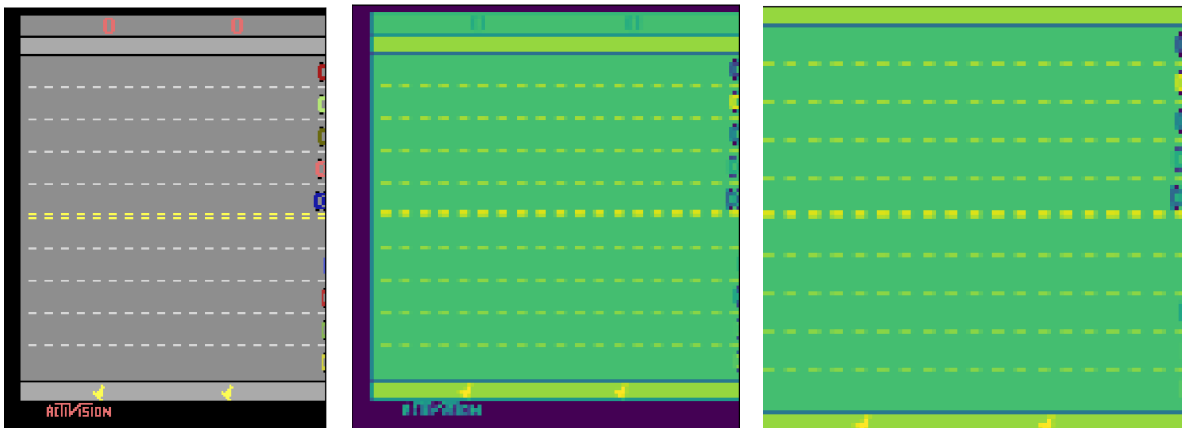
### 4.1 State-Action Space

Before we begin learning, we must first develop the state-action space for our problem. The action space for *Freeway* is quite small, with three operations (No Op, Move Forward, and Move Backward). However, its state space is not as simple. A majority of Atari games in OpenAI Gym are represented as 210x160x3 pixel cubes, where the last dimension represents RGB values and 210x160 is the area of the game screen. We see that completely capturing the state space requires  $210 \times 160 \times 3 = 100800$  elements. In standard Q-learning, a Q-values table is kept where each index corresponds to the Q-value of a state and action pair. It is clear that we cannot use vanilla Q-learning to train our agent.

Instead, we propose the use of a convolutional neural network, trained with a variant of Q-learning. As input, the model takes in pixels that represent the game state and outputs a value function that is used to estimate future rewards.

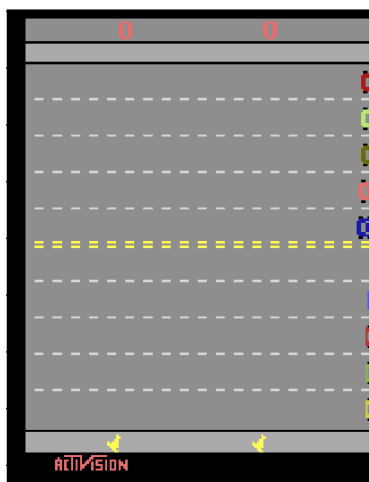
## 4.2 State Space Preprocessing

As mentioned in section 4.1, the state space of *Freeway* is quite large and tracking all raw pixels of the game would require 100800 elements in our neural network. While tracking all raw pixels may provide a marginal benefit in model performance, we preprocess our game states into two dimensional 100x100 pixel representations to reduce the number of nodes in our neural network required to represent the game. We first convert the raw image of the game state to grayscale and then down-sample to create a 108x118 image. This image is then cropped to 100x100, removing pixels on the screen that are not actually part of the gameplay (the score at the top, black bar to the left, and *Activision* logo at the bottom).



To provide a sense of time for our model, we then stack the last four frames of the game on top of each other to create a 100x100x4 image. This provides information about movement of the player and obstacles in the game.

## 4.3 Convolutional Neural Network (DQN)



The neural network itself is quite simple. It receives as input 100x100x4 image frames and the first hidden layer performs a convolution using 32 8x8 filters with stride 4. The second hidden layer convolves 64 4x4 filters with stride 2 and the final hidden layer convolves 64 3x3 filters with stride 1. The final hidden layer is fully-connected and our output layer is a fully-connected linear layer with a single output for each valid action.

## 4.4 Control

Our control was a simple agent that only had one action type, which was moving up.

## 4.5 Agent 1

Our first agent was trained with a naive reward function which only gave positive award (+1) for reaching the other side of the road.

## 4.6 Agent 2

Our second agent was trained by first observing one action type (moving up) over N frames in order to encourage the agent to first move upwards and successfully navigate to a positive reward state.

## 4.7 Agent 3

Our third agent sought to minimize the amount of time spent to score a point by penalizing on the time spent to traverse the road once. In theory, this would teach the agent to avoid collisions with obstacles, which knock it backwards and thus increase the time for the agent to cross the road.

# 5 Experiments and Results

## 5.1 Control

The control performed well, scoring about an average of 18 points. As expected, it continuously ran into cars, averaging about 4 collisions per point scored. This led to the control being bumped back several times. Nevertheless, it was a decent agent.

## 5.2 Agent 1

The first agent, with its naive reward function, didn't perform well at all, scoring about an average of 6 points. One major flaw was in that the agent staggered backwards continuously, leading it to run into cars both in front and behind of it, averaging about 8 collisions per point scored. This led to the agent being bumped back multiple times, making it a very ineffective model.

## 5.3 Agent 2

After observing and training on N frames of solely moving upwards, the second agent scored an average of 15 points. It began to stagger backwards on occasion. Rather than oscillating between lanes continuously like agent 1, it showed a sense of direction towards the end goal and generally moved upwards. However, the staggering did not seem to help the agent, as it did not look like it

was considering collisions at all. The agent was expected to stagger backwards when a car is about to hit it. Instead, it staggered at random intervals whether or not there was an incoming car. The agent averaged about 5 collisions per point. Though it was an improvement from the first agent, it still performed worse than the control.

## 5.4 Agent 3

The third agent took into account the amount of time spent to score a point and deducted from the final reward. This was done to encourage the bot to find a faster path to the goal. However, against all expectations, this agent performed the worst with an average score of 1.3 and about 8.6 collisions per point. After the agent scored once or twice, it began to linger near the bottom of the track and constantly got hit by cars.

30000 iterations	Control	Agent 1	Agent 2	Agent 3
Average Score	18.3 points	6.1 points	15.4 points	1.3 points
Average # of Collisions	4.8 collisions	8.2 collisions	5.3 collisions	8.6 collisions

## 6 Frostbite Extension

The goal for the next part of the project is to develop an agent that learns and plays *Frostbite*, a video game for the Atari 2600 console using a similar DQN model. Frostbite is an Atari 2600 game in which the objective is to help the man build igloos by jumping on floating blocks of ice, while trying to avoid deadly hazards. The bottom two thirds of the screen are water with four rows of ice blocks floating horizontally. The player moves by jumping from one row to another while trying not to fall in the water and avoid the obstacles, including crabs and birds. On the top of the screen is the shore where the player must build the igloo. Each time the player jumps on a piece of ice in a row its color changes from white to blue and the player gets an ice block in the igloo on the shore. The player has the ability to change the direction in which the ice is flowing by pressing the fire button, but doing so costs a piece of the igloo. After the player has jumped on all pieces on the screen, they all turn from blue back to white, and the player can jump on them again. Once all 15 ice blocks required for building the igloo have been gathered, the player must get back to the shore and go inside the igloo, proceeding the next level. On every level, the enemies and ice blocks move slightly faster than before. Every second, the temperature drops from 45 to 0, so the player must finish each level in 45 seconds.

Although this is a much more complex game than Freeway, we wanted to extend our deep Q-learning model to train an agent to play this game using reinforcement learning. To test this hypothesis, we first analyzed the game we are playing to find a suitable state-action space for the agent and then figure out how to best design a Q-learning model to train the agent.

## 6.1 State-Action Space

The action space for *Frostbite* is much larger than for *Freeway*, with 18 operations, including Jump Down, Jump Up, Move Left, Move Right, Stop, and Fire. However, the state space is the same as for *Freeway*. *Frostbite* is represented as  $210 \times 160 \times 3$  pixel cubes, where the last dimension represents RGB values and  $210 \times 160$  is the area of the game screen. Completely capturing the state space requires  $210 \times 160 \times 3 = 100800$  elements. Similar to the training on *Freeway*, we will also use convolutional network DQN on this game. As input, the model takes in pixels that represent the game state and outputs a value function that is used to estimate future rewards.

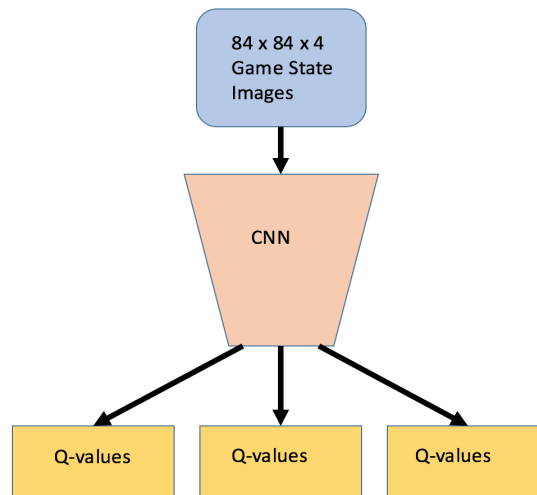
The player receives a reward of 10 when it jumps on a platform. Higher scores occur when the agent navigates to other stages, which generally requires exploration.

## 6.2 State Space Preprocessing

For every state, preprocess the training image by getting rescaling the observation to  $84 \times 84$  pixels. Then convert the image to grayscale.

## 6.3 Convolutional Neural Network Structure

We construct a 3-layer convolutional neural network, passing a  $84 \times 84 \times 4$  tensor to the CNN. Forward propagation encodes the Q-values for a particular state. This model will have one output for each action which represents a scalar Q-value for each possible action.



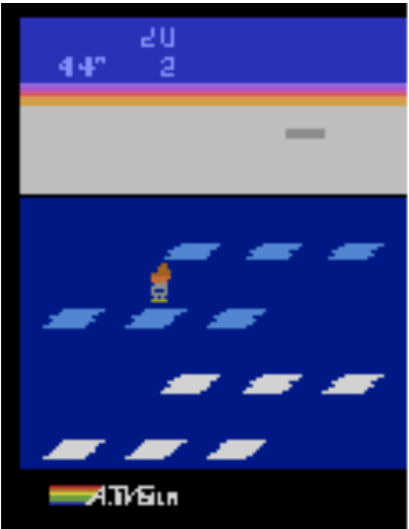

The convolutional neural network has the following structure:

Layer	Input	Filter Size	Stride	Num Filters	Activation	Output
conv1	84 x 84 x 8	8 x 8	4	32	ReLU	20 x 20 x 32
conv2	20 x 20 x 32	4 x 4	2	64	ReLU	9 x 9 x 64
conv3	9 x 9 x 64	3 x 3	1	64	ReLU	7 x 7 x 64
fc1	7 x 7 x 64			512	ReLU	512
fc2	512			18	Linear	18



## 6.4 Results

Training Iterations/Games	Average Reward Received
1000	17.20
5000	17.50
8000	20.10
10,000	22.70

Here is a look at the game states for multiple different runs of DQN on Frostbite for 150, 1000, 10,000, and 100,000 training iterations.

150 training iterations	1000 training iterations
 <p>Score: 20</p>	 <p>Score: 90</p>



10,000 training iterations	100,000 training iterations
 <p>Score: 160</p>	 <p>Score: 60</p>

## 6.5 Analysis

After 150 iterations of training, the agent learned how to jump from ice block to ice block. However, it did not perform well and 150 games was too few to train on. After 1000 games of training, the agent got an average score of 17.20. The agent improved its ability to jump from block to block, and it even reached scores as high as 90, as depicted in the game state above. The agent was able to reach high scores but still resulted in a low average score because there were many games in which the agent jumped into the water and lost, getting a score of 0, while high-scoring games were relatively rare. Training on 10,000 iterations allowed the agent to learn how to build the igloo, reaching a score of 160, but did not learn to do back into the igloo to move on to the level. The average score was 22.70 after 10,000 iterations. Training on 100,000 iterations did not perform as well as the agent trained on 10,000 games, as the agent was unable to build the igloo, suggesting some overfitting.

## 7 Discussion

The results gotten from our experimentation were extremely varied depending on how we trained our model. We saw that depending on the policy we saw that our model performed differently.

One of the main problems that we had when training the simpler game “Freeway”, was creating a model that would avoid getting hit by an obstacle. We first saw that our control agent performed very well which we were surprised considering it scored 18 points. We expected

the control agent to have one of the highest scores, as there is no large penalty for hitting cars (the game does not restart and the agent is only moved back two lanes).

We observed that agent 1 had a worse average score than the control agent, averaging a score of 6.1 per game. This was expected, as the model had no penalty on hitting cars and only received a positive reward upon crossing the street. But by just taking exploratory actions in the early stages, it is quite difficult to ever reach the other side of the road and attain a nonzero reward. In conclusion, agent 1 did not perform well as it did not punish the agent for never scoring (crossing the road). It was also not punishing for collisions, so the agent had no way of learning that collisions were to be avoided.

Seeing that upward motion was rewarding, we developed agent 2 to observe some  $N$  frames of pure upwards movement. This would provide the agent a baseline policy that favored upward movement. After observing purely upward movement, the agent was allowed to explore the state space with the ability to take any action itself. While agent 2 scored less on average than the control, we did observe that it attempted to avoid collisions. Something that we could have tried and intend to try is to have the agent observe  $N$  frames of majority upward movement but not purely upward movement. This may the agent more likely to perform other actions than agent 2 currently does.

The most interesting agent that we saw during this experimentation would have to be agent 3. In this model, we introduced the idea of a time penalty such that the longer the agent takes to cross the road, the more negative of a reward it gets. Our logic for this was that since obstacles are constantly appearing, we thought it would be beneficial for the agent to move rapidly rather than stay in the same state where it could get hit by an obstacle as a higher chance. We also thought that this agent would be best for learning the more complicated game “Frostbite” since this game is also very time dependent. The results of this were surprising as well. We saw that agent 3 actually performed a lot worse than any of the models. The reason for this is that as time progressed and the agent could not get to the end of the road at a reasonable rate, its reward value was so negative that the agent decided it was no longer worth crossing the road for a single positive point. We initially penalized -0.05 per second and it ultimately got to a reward value of -18. Later, we changed to a normalized penalty of  $\frac{(\text{current\_time} - \text{previous\_time\_of\_reward})}{\text{current\_time}}$ . Though this was marginally better, it still had the same problems as the negative reward in that it discouraged the agent from reaching the goal. Some things that we could do to improve this model was run a grid search to optimize the penalty value for each second so timing is a priority.

In the extension of DQN to Frostbite, in general, the model did not learn the *Frostbite* game very well, as the agent was never able to progress past level 1. This is most likely because the exploration never covered entering the igloo. Frostbite is one of the most difficult classical Atari games for reinforcement learning algorithms to master, among the ranks of Montezuma’s Revenge and Gravitar. In 2015, Google compared their algorithm performance to human testers, and found that the human testers did 94 percent better than Google’s algorithm.

Overall, we found interesting results in the different models we tested.

## 8 Future Directions

During our research, we intended to test how a model that was trained using a simple crossing game versus a model that is learning directly from the more complex game. The more complicated game was fundamentally different in that it had a time limit in how long it could stay at a certain state (a single location on the road). If given more time, we would have performed this transfer learning by training the simple *Freeway* model with 18 action spaces but restricting it to its three possible actions. Then this model can be used as a baseline set of network weights to begin training on *Frostbite*. Comparing this model to a model trained purely on *Frostbite* is our main goal moving forwards.

Regarding the negative reward, we would like to figure out another method of implementing a penalty for hitting cars. Though time was a constraint that we should have considered, it did not properly teach the agents when a collision occurred. This is particularly important, as the models as of right now have no proper method of learning what it should be penalized for.

One other improvement that we could have made by modifying our policy was making it so that the reward was greater the faster it changed states. This modification would be a challenge since we would have to have some way to optimize the state placement and time management.

Aside from this, another direction we could have taken is comparing different the performance by using different model. One particular model that we considered was SARSA that has been used in previous papers that we researched.

## 9 Reference

- [1] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013). Harvard
- [2] Liu, Ruishan, and James Zou. “The Effects of Memory Replay in Reinforcement Learning.”
- [3] Kansy, Ken, et al. *Schema Networks: Zero-Shot Transfer with a Generative Causal Model of Intuitive Physics*. arxiv.org/abs/1706.04317.
- [4] McKenzie, Mark, et al. “Competitive Reinforcement Learning in Atari Games.” *AI 2017: Advances in Artificial Intelligence Lecture Notes in Computer Science*, 2017, pp. 14–26., doi:10.1007/978-3-319-63004-5\_2.
- [5] Chebotar, Yevgen, and Austin Waters. “Distilling Knowledge from Ensembles of Neural Networks for Speech Recognition.” *Interspeech 2016*, 2016, doi:10.21437/interspeech.2016-1190.