CIS 520 Fall 2019, Project Report

Reinforcement Learning in Ms. Pacman

Team Members:

- Anthony Colangeli; Email: `colange@seas.upenn.edu`
- Michal Porubcin; Email: `michal19@seas.upenn.edu`
- Yeshas Thadimari; Email: `yeshastv@seas.upenn.edu`

Abstract

Recent research in reinforcement learning has used game environments to experiment with different learning algorithms. With breakthrough methods like Deep Q Learning, it is now possible to learn end-to-end behaviors straight from the pixels of a videogaame. In this report we compare two architectures, Deep Q Learning (DQL) and a variant called Clipped Double Deep Q Learning (CDDQL), on Ms. Pacman, simulated in OpenAI's Atari Learning Environment. We examine various degenerate behaviors learned by the agent, and explain various solutions. Finally, we find CDDQL consistently outperforms DQL as expected, yet is affected little by any number of tweaks in hyperparameters, preprocessing steps, or even raw training length.

1 Motivation

The ability of Reinforcement Learning (RL) methods to interpret and understand various environments in order to shape an agent's behavior is very powerful and opens the door to solving many difficult problems that can't be solved through traditional supervised learning methods. This is due to the ability to drop an agent in an environment and allow it to learn by trial and error, similarly to how humans learn. This allows for determining actions in complex environments where we have limited knowledge of what actions would be successful prior to beginning. The most tractable problems for RL to solve are currently in simple games. The most famous of these is Deepmind's AlphaGo. AlphaGo's success in beating professional Go players without handicaps generated a lot of popularity for Reinforcement Learning and was a landmark example in showing the power and potential of reinforcement learning [10]. Games provide a structured, simplified environment for developing learning algorithms, and the belief is that RL algorithms developed for game-playing would eventually translate to real-world applications such as robotics and autonomous driving [2].

The current issue with RL is that it requires large amounts of data. A typical solution is to gather data from simulation environments where an unlimited amount can be easily generated. While not always feasible for real-world problems due to imperfect simulations or complex dynamics, many game playing problems specifically rely on a pseudo-simulation environment. For our project we hope to compare different RL methods by making use of the Ms. Pacman environment on the open-source OpenAI Gym [1].

2 Related Work

Since the inception of Reinforcement Learning, there has been a lot of literature in it's applications to Pong and Atari games. In the realm of model-free RL, one of the first and most relevant works is from DeepMind in [7]. In this, Ming et al. propose the Deep Q-Learning method to train Atari 2600 games on the Atari Learning environment. The model proposed is a convolutional neural network, trained with a variant of Q-learning. The model was also unchanged and applied to six different Atari games and went on to outperform all previous approaches. In [8], we see results of above human-level control in various games trained by a DQN-agent. We also note that the Ms Pacman game is one of the hardest games to train the DQN-agent on. The paper reports that it is far below human-level and performs at only 13% human-level. The DQN-agent we used in this project is based off these two papers' implementation of DQN.

Building directly off of the success that has been found with DQN, an extension of this method, called Deep Double Q-Learning, has been developed. Deep Double Q-Learning was also developed by DeepMind and was detailed in [11]. This algorithm attempts to account for the overestimation of Q-values that is present in DQN, especially in Atari games. This method proposes the use of two separate Q-learning networks, where action selection and action evaluation are decoupled between these two networks. The results found that improvement in terms of both value accuracy and policy quality.

In [4], the authors propose a Deep Recurrent Q-Network (DRQN) and test the effects of adding recurrence in the form of a recurrent LSTM in place of the first fully-conected layer after convolution. The results show that DRQN's performance scales as a function of observability and the performance degrades less than DQN's and performs better when there is a loss of information. In [6], the authors present asynchronous variants of several standard reinforcement methods and found it to perform exceedingly well and surpass the state-of-the-art methods on the Atari Learning Environment. There have also been methods that make use of auxiliary tasks that basically makes the agent maximize many reward functions at the same time during the learning stage.

3 Data Set

3.1 OpenAI Gym

OpenAI Gym is an open source toolkit that can be used for comparing and developing reinforcement learning algorithms. There are various problems and environments that can be experimented with on this platform, ranging from simple control problems such as the inverted pendulum to game play environments such as the Atari Learning Environment (ALE). As mentioned, we propose to compare different relevant RL methods by making use of the popular Ms. Pacman game.

3.2 Ms. Pacman

Ms. Pacman is a 2D single player game, where the user controls a yellow character (Ms. Pacman) around a maze. Ms Pacman can eat pellets scattered throughout the maze to earn points, and if she collects all of them, the user progresses to the next level. Ghosts spawn from a center cell and move autonomously around the maze. They will kill Ms. Pacman upon contact. Ms. Pacman has three lives to score as high as possible. The score differential at each epoch corresponds exactly to the rewards given by ALE, and are described in Table 1.

Were we to maximize another behavior, the scoring system would seem to be a kind of intricate reward shaping. We are to maximize the score itself, however, which presents a few problems. For example, normal pellets are scattered throughout almost the entire traversable space in the maze. The regularity of the normal pellets as well as their measly value makes them very insignificant in the Q update. But eating all of them is the sole condition for progressing to higher levels, which spawn much higher-valued fruit. Consider also the ability to eat ghosts; the score for eating multiple ghosts far outweighs any other rewards in the early game. When the agent is in its first iterations ($i < 500$), this tends to lead to degenerate behaviors, which will be discussed in 7.1 Interpreting Failures.

There are two available Ms. Pacman environments on the openai gym: MsPacman-v0 gives a $(210, 160, 3)$-shape RGB image as the observation, as shown in Figure 1, and MsPacman-v0-ram gives a 128-byte array representing the RAM of the Atari machine. One more value, number of lives remaining, can be obtained from the debug information provided by ALE at each step. We chose to engage both these environments to explore the different methodologies and approaches needed for these problems.

4 Problem Formulation

We use the reinforcement learning framework to train our agent. That is, an agent in state s_t can take an action a_t with probability $\pi(a_t|s_t)$ and move to state s_{t+1} with reward r with probability $P(s_{t+1}, r|s_t, a_t)$, where $s \in \mathcal{S}$, $a \in \mathcal{A}$, $r \in \mathbb{R}$. $\pi(a_t|s_t)$ is also called the policy and $P(s_{t+1}, r|s_t, a_t)$ describes the environment dynamics. The goal is to learn a policy which maximizes expected future rewards, i.e. the game score.

Event	Value
Normal pellet	10
Power pellet	50
Eating a ghost	$2^n * 100$ for the nth consecutive ghost
Cherry	100
Strawberry	300
Apricot	500
Pretzel	700
Apple	1000
Pear	2000
Banana	5000

Table 1: Scores/Rewards

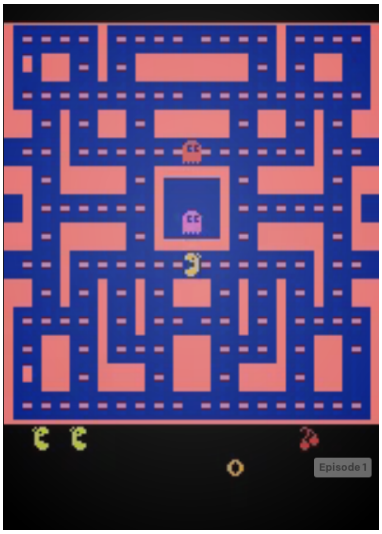


Figure 1: Ms Pacman environment on Openai Gym

Our environment is MsPacman-v0 running on ALE, in which the state is either the RGB or RAM array as described above, the actions are $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ representing eight directions and a stop action, and the rewards are the score differentials as discussed above. We developed modifications to all three aspects of the default environment. First, we reduced the action space to the four diagonals, $\mathcal{A} = \{5, 6, 7, 8\}$. After manually testing each action, we discovered two of the first seven actions failed to have any effect. Whether or not this is a bug in ALE, we decided the diagonals give the agent full ability to traverse the maze, while halving the action space. We also greatly reduce the state space, and developed a reward shaping function to enhance the learning ability of the agent [9]. Both of these are discussed in section 5.1 Preprocessing.

We can define an action-value function for policy π as $q_\pi(s, a) := \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t, a_t]$, which denotes the expected return starting from s, taking action a, and thereafter following policy π. Traditionally, maximizing qi over π results in the Bellman optimality equation $q_*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a_{t+1}} q_*(s_{t+1}, a_{t+1}) | s_t, a_t]$, which can be maximized with dynamic programming. Unfortunately, this method is prohibitively expensive.

Maintaining exploration is crucial in reinforcement learning, and there are typically two approaches: on-policy and off-policy control. On-policy methods aim to maintain exploration on the current decision-making policy, while off-policy methods use separate policies for data gathering and decision making. A very popular off-policy algorithm for estimating q_* is called Q-learning [12]. Since the state and actions spaces are discrete, we could technically maintain a table of Q values for state-action pairs in Ms. Pacman, but the dimensions are far too large for this to be practical. Instead we approximate Q values with a neural network in an architecture called Deep Q Learning (DQN) developed by Deepmind [11].

Q-learning has a problem with maximization bias, due to the max operation in the update step. Double

Q Learning addresses this using two networks to update each other. This does not translate perfectly to DQN, since DQN already has the separate online and target networks. Fujimoto proposes using two networks to calculate maximum Q values and taking the minimum at each training step, in a method called Clipped Double DQN (CDDQN) [3]. This method leads to a preference for low variance value estimates. Though CDDQN was used in an actor-critic setting, we found it was an improvement over DQN without needing a critic.

5 Methods

5.1 Method overview

We ultimately decided to test multiple network implementations in order to compare results due to there being no clear-cut best solution to this problem in the literature. These different methods can be categorized into either image-based or RAM-based methods. The image-based methods make use of Convolutional Neural Networks(CNN) to extract information from the would-be Atari game-screen; however, the RAM based methods are done with standard artificial neural networks(ANN) and make use of the game RAM readings to extract information. Within the image-based methods we have two separate implementations using similar but different preprocessing strategies and network architectures for both DQN and CDDQN. We will refer to these two implementation methods as **image-based-a** and **image-based-b** for clarity.

5.2 Preprocessing

Our main goal for preprocessing was to decrease computational load, since we aimed to compare several models, and training a reinforcement learner can be expensive. We settled upon the following two final preprocessing scheme for MsPacman-v0 **image-based-a**:

1. Convert image to grayscale
2. Clip the bottom 20 pixels
3. Downscale by two and minor cropping down to shape (85, 77)
4. Copy any visible power pellets from the previous frame.
5. Increase contrast
6. Recolor the ghosts (all ghosts are the same color, increase visibility of scared ghosts)
7. Rescale pixel values to $(-.5, .5)$

Step four is similar in spirit to a technique Deepmind used in their DQN models to keep track of flashing sprites. We observed power pellets to alternate between visible and invisible for two to four frames at a time. Due to our frame skipping technique, it was possible to miss a power pellet on on or more of its visibility cycles. By extending the visibility of the power pellet for one frame, we guarantee it is present in a training frame if it has not been eaten. We expected this technique to give minimal improvement. It would have been much more helpful to deal with flashing ghosts, but we could not apply the same technique because the ghosts' flashing period is much longer, and they are non-stationary.

We also implemented reward shaping. At every time step we add a reward of -1 to discourage the agent from idling. For each pellet we add a reward equal to the number of pellets eaten so far, to incentivize eating more pellets. At the beginning of an episode the agent is guaranteed to eat a few pellets, so we hoped increasing the pellet value over time would offset their low perceived importance at the start.

For **image-based-b** we used the following preprocessing strategy:

1. Re-color ghosts and pacman
2. Clip the bottom 34 pixels
3. Increase Contrast
4. Significantly downscale to 88x80 shape
5. Rescale pixel values to $(-1, 1)$

Additionally, no reward shaping was used here, we simply used the in-game score as a reward function.

As for MsPacman-v0-ram, to our knowledge, there is no available information on the actual meaning of each of the bytes and actually understanding and interpreting the meaning of each byte is a very tedious process. There is no need for us to do any feature engineering or data pre-processing.

5.3 Models

For this project we used a Deep Q Network and some variants. **MsPacman-v0-Q (a)** is a DQN corresponding to **image-based-a**. Hyperparameters are described in Table 2. **MsPacman-v0-DQ (a)** is a CDDQN corresponding to **image-based-a**. Hyperparameters are described in Table 3.

Parameter	Value
ϵ (Epsilon)	.9
ϵ Decay Rate	0.998
Min ϵ	0.2
Discount	.95
Loss	MSE
Optimizer	Adam
Learning Rate	.0005
Batch Size	32
Training Start Step	32
Replay Memory Size	4000
Scheduler Step Size	1
Scheduler Gamma	.9

Table 2: Hyperparameters for **MsPacman-v0-Q (a)**

Parameter	Value
ϵ (Epsilon)	.5
ϵ Decay Rate	0.9994
Min ϵ	0.2
Discount	.98
Loss	MSE
Optimizer	Adam
Learning Rate	.0001
Batch Size	32
Training Start Step	5000
Replay Memory Size	50000
Scheduler Step Size	10
Scheduler Gamma	.99

Table 3: Hyperparameters for **MsPacman-v0-DQ (a)**

Both **image-based-a** methods use the same neural network architecture for its Q networks, illustrated in Figure 3.

- Convolution {Depth: 32, Kernel Size = 5×5, Stride=2, Padding=2}, ReLU
- Convolution {Depth: 64, Kernel Size = 5×5, Stride=2, Padding=2}, ReLU
- Convolution {Depth: 64, Kernel Size = 3×3, Stride=1, Padding=1}, ReLU
- Max Pool {Size = 2×2, Stride=2}
- Fully connected {Size = 512}, ReLU
- Fully connected (Output) {Size = 4}

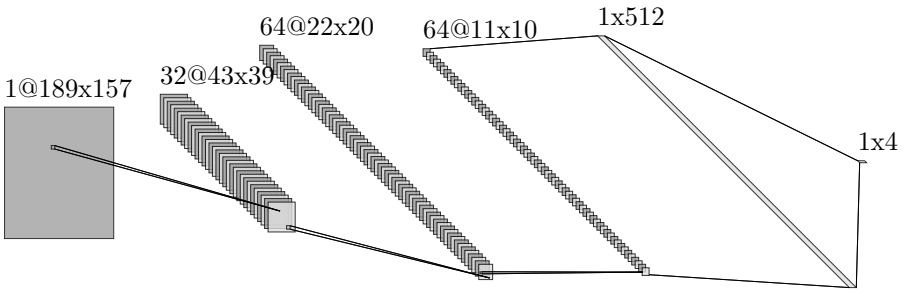


Figure 2: **MsPacman-v0-Q (a)** and **MsPacman-v0-DQ (a)** architecture

MsPacman-v0-Q (b) is a DQN corresponding to **image-based-b**. **MsPacman-v0-DQ (b)** is a CDDQN correspoding to **image-based-b**. They use the same hyperparameters, as described in Table 4.

Parameter	Value
ϵ (Epsilon)	.9
ϵ Decay Rate	0.99
Min ϵ	0.2
Discount	.99
Loss	MSE Loss
Optimizer	Adam
Learning Rate	.001
Batch Size	64
Training Start Step	10000
Replay Memory Size	500000
Scheduler Step Size	1
Scheduler Gamma	.9

Table 4: Hyperparameters for **MsPacman-v0-Q (b)** and **MsPacman-v0-DQ (b)**

Parameter	Value
ϵ (Epsilon)	.9
ϵ Decay Rate	0.99
Min ϵ	0.2
Discount	.99
Loss	MSE Loss
Optimizer	Adam
Learning Rate	.001
Batch Size	64
Training Start Step	10000
Replay Memory Size	50000
Scheduler Step Size	1
Scheduler Gamma	.9

Table 5: Hyperparameters for **MsPacman-v0-ram**

Both image-based-b methods use the same neural network architecture for its Q networks, illustrated in Figure 3.

- Convolution {Depth: 32, Kernel Size = 8×8, Stride=4, Padding=3}, ReLU
- Convolution {Depth: 64, Kernel Size = 4×4, Stride=2, Padding=1}, ReLU
- Convolution {Depth: 64, Kernel Size = 3×3, Stride=1, Padding=1}, ReLU
- Fully connected {Size = 512}, ReLU
- Fully connected (Output) {Size = 4}

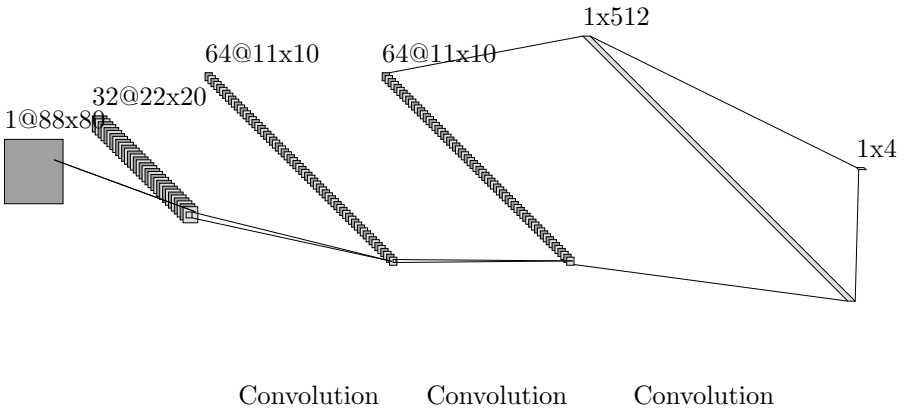


Figure 3: **MsPacman-v0-Q (b)** and **MsPacman-v0-DQ (b)** architecture

For the **MsPacman-v0-ram** we use a Deep Q Network with fully-connected neural network structure for learning. The model input is a 128×1 array which feeds into 2 256×1 hidden layers which are outputted into a 128×1 final layer that output a 9×1 array which represents the possible actions of the Ms. Pacman agent. We chose ReLUs for the activation functions due to several major benefits they hold over sigmoidal activations including better computational efficiency. It has been shown to also converge faster than other methods [5].

5.4 Training

Following from work by Deepmind, we implement a replay buffer to store $(S_t, A_t, R_t, S_{t+1}, terminal)$ tuples. At training, we once again randomly sample a mini-batch of tuples from the buffer and run it forward in one or more neural networks. This process breaks correlations among experiences within a trajectory, increasing sample efficiency and stability during training. It also decouples learning from experience-gathering, allowing us greater control over the learning rate. We also use separate target and online networks, to offset moving targets for value updates. All our networks train on mini-batch updates from a replay buffer.

MsPacman-v0-Q (a) was trained for 1400 epochs on an NVIDIA RTX 2060 GPU. We skip two out of every three frames, train every five frames, and copy parameters from the target to online network every 50 frames.

MsPacman-v0-DQ (a) was trained for 3000 epochs on an NVIDIA RTX 2060 GPU. We skip two out of every three frames and train every four frames.

MsPacman-v0-Q (b) and **MsPacman-v0-DQ (b)** were both trained for 1000 epochs on Google Colab. We also use separate target and online networks for the traditional Deep Q Learning and copy parameters every 10000 steps. Additionally, training is only done on every fourth frame, with the first 90 frames of each new game not used for training due to the game not allowing for Ms. Pacman to move during this period.

MsPacman–v0-ram was trained for 10 hours for over 2000 epochs on Google Colab. Apart from the network structure and state sizes, there is no difference between the DQN implementation of the fully-connected neural network and the CNN for MsPacman-v0 (a).

6 Experiments and Results

We initially trained the DQN agent for both the **MsPacman-v0-ram** model and both image-based Q-learning models but soon found the image-based models to continuously outperform the RAM version. The below figure shows the best results achieved after training and testing multiple times with various hyperparameters for the fully-connected network. The average score was around 450.5 points which did not

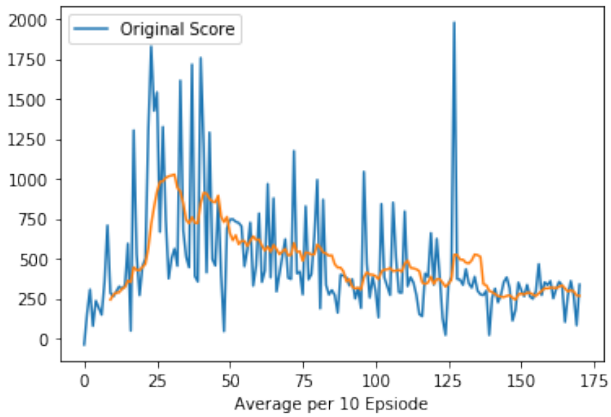


Figure 4: Results for DQN using a FCN for **Ms-Pacman-v0-ram**

seem to match up with the scores trained by the DQN for the image based methods. We believe that this is due to the RAM bytes poorly representing the features of the system. At this point we believed that it was more worthwhile for our project to focus on the DQN and DDQNs on the image version of Ms-Pacman-v0.

The results found for **Ms-Pacman-v0-Q (b)** and **Ms-Pacman-v0-DQ (b)** are shown below:

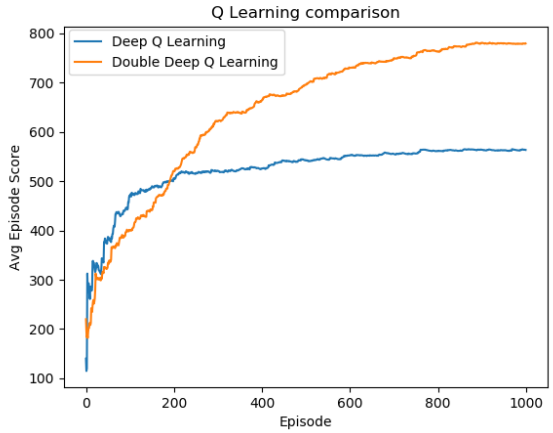
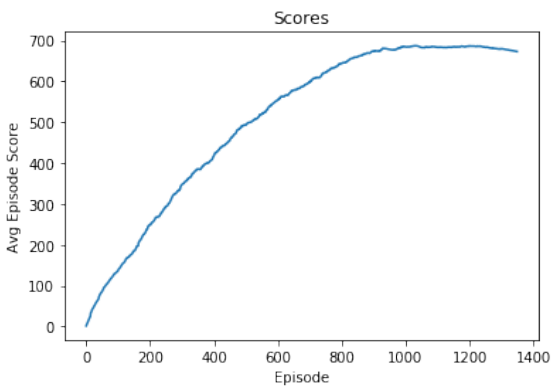


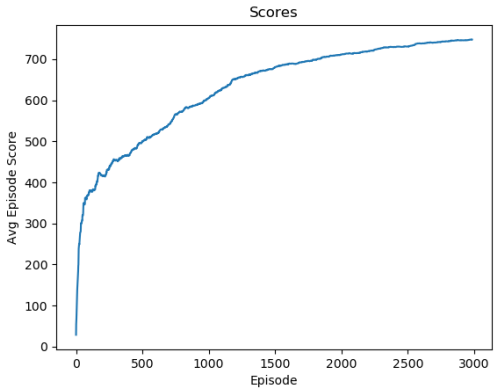
Figure 5: Results for DQN and CDDQN for **image-based-b**

This graph shows the average in-game score over all of the episodes thus far, over 1000 episodes of training. An episode denotes a round of Ms. Pacman, from beginning until running out of lives. Both Deep Q and Clipped Double Deep Q result in a significant improvement over a naive approach, but Double Deep Q has a significant improvement over Deep Q learning. This is to be expected as in the initial paper on Deep Double Q Learning by Deepmind, it was noted that this method is particularly important for Atari games as overestimation is common in them.

The results found for **Ms-Pacman-v0-Q (a)** and **Ms-Pacman-v0-DQ (a)** are shown below:



(a) Results for DQN for **image-based-a**



(b) Results for CDDQN for **image-based-a**

The CDDQN took longer to converge but this is likely due to a more gradual decrease in ϵ; the DQN reaches minimum ϵ at around 750 episodes, while CDDQN takes 2500 episodes, with the given parameters. Ultimately, CDDQN seems to approach a higher average score given enough time, while DQN seems unable to breach 700.

The DQN seems to have benefited from more elaborate preprocessing, as is apparent in comparison to the DQN in Figure 5 (**Ms-Pacman-v0-Q (b)**). We hoped with a longer exploration period, more iterations, and more preprocessing, **Ms-Pacman-v0-DQ (a)** would clearly outperform the other models, but it seems like these considerations were negligible compared to the learning algorithm and architecture.

7 Conclusion and Discussion

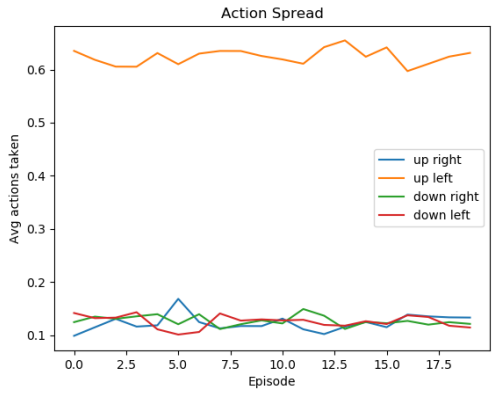
7.1 Interpreting Failures

Here we discuss several degenerate policies induced by the learned Q function, and some reasoning behind our methods.

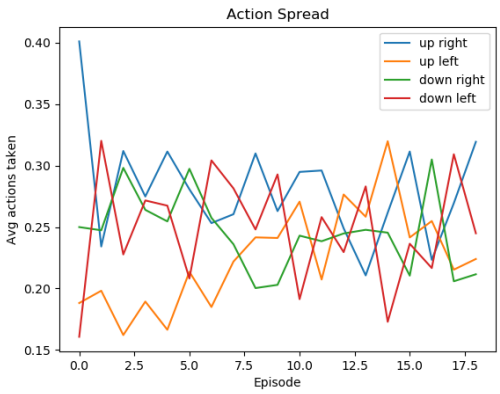
We observed that the agent often got stuck in the four corners after eating a power pellet. We first guessed the agent was attracted to the higher value of the power pellet, but the agent would continue to ram into the walls after eating it, and even after the effect wore off (the ghosts stopped being scared). We suspected that the agent was not actually "seeing" the pellet, and added further preprocessing to enhance the color and frame permanence of the pellet. Unfortunately this had little effect. While we could not rule out the agent's inability to see the pellet, we found a much likelier cause. Once the agent eats the power pellet, the ghosts, while initially moving away from the agent, would sometimes loop back around towards the agent and "feed" it. It was not uncommon to see two ghosts (600 points) eaten in this way, which understandably encourages the agent to camp the corner.

In fact, we found that many of our preprocessing attempts had only a marginal effect compared to hyperparameter tuning and architecture choices. For instance, we were concerned the normal pellets were too small to be "visible" to the agent, since after subsampling they are just one or two pixels in height. When we tried training on a full-sized grayscale image, however, the agent performed more or less the same, and the training time doubled even for the regular DQN – too costly for us. For another example, we return to the power pellets. Prior to eating a power pellet, we sometimes observed that the agent would consistently avoid them. In some rounds, the agent would not explore beyond the two vertical tunnels on either side of its initial position, avoiding the corners completely. This is in spite of many tweaks to the color of the pellets in the preprocessing stage. We suspect the agent was registering the shape more than the color, and confusing it with the ghosts.

Corner camping plagues our agent even after thousands of training rounds. But for some versions of our models, the agent would appear to learn corner camping from the first epoch. We were surprised that it would learn such a strategy so quickly, especially with high values of ϵ in the early rounds. We suspect that these particular models were susceptible to local optima of sorts, and by maximizing over the actions in a state, the agent will often favor whatever action was slightly better at the start. Random exploratory actions actually compound the issue, since they slow the agent, allowing ghosts to surround it. As a result, the agent will favor a single action, a diagonal in our case, and inadvertently camp a corner just by moving in a single (noisy) direction, even from the first training epoch (figure (a) below). Since we could not force the agent to explore with a higher ϵ, we found that increasing the randomness of batches from the replay buffer was key. That is, we waited longer for the buffer to fill before learning, up to about 5000 frames or 18 episodes, and increased the maximum size to 50000. In figure (b) below, after implementing this change we see the agent initially favor one action, but immediately lose the bias as training proceeds.



(a) Favoring a single action from the first epoch



(b) Reducing single action bias with greater mini-batch randomness

7.2 Next Steps

Clearly there is a lot of room for improvement given the still relatively low score compared to human performance. Simply due to computational and time constraints, the methods used were focused on shrinking data so that training would be quicker. It is possible that by keeping the data closer to it's original state would allow for better results as more could be learned from the more detailed environment. Additionally, more dense networks would be able to be used then and this could also help improve results.

The results above led us to conclude that the RAM-based method was inferior to the image-based. This is far from conclusively true given our findings, and a worthwhile avenue that would be pursued next is to combine the RAM-based and image-based methods. It is possible that linking the RAM-based methods with something more directly correlated with the in-game actions like the images can result in more useful information being extracted from the RAM. Together these could result in improvement on that found by the image-based methods alone.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving, 2019.

[3] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[4] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[9] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[10] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[11] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[12] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.