Exploring Deep Q Networks in Ms. Pac-Man

Elijah Tai

Abstract

Deep Q-Networks have been applied to a variety of games for the Atari 2600 with varying levels of success. Some games perform particularly well, and some games see little improvement as compared to previous algorithms. Ms. Pac-Man is one of the games that sees very little improvement, and part of the reason could be because the power-up pellet mechanic could be particularly hard for the network to learn. In this paper, I explore this notion, attempting to improve upon a baseline Deep Q-Network by altering how the algorithm responds to power-up pellets. I train different variants for 2 million to 4 million frames, finding that withholding bonuses from eating ghosts improves the average score compared to the regular baseline Deep Q-Network.

1. Introduction

1.1. Background

In the area of reinforcement learning, deep Q networks proposed by Mnih et al. have become the standard. Deep Q-Networks (DQNs) use deep neural networks to approximate the Q-value function. Q-values represent the expected reward an agent can achieve by taking a certain action in a given state. The neural network takes a state as input and outputs a vector of Q-values for each possible action. DQNs are trained using experience replay, where past experiences are stored in a replay buffer and randomly sampled to train the network. DQNs have been successfully applied to various tasks, including playing Atari games and controlling robots in simulated environments (Mnih et al., 2013). They are a powerful tool in the field of machine learning and continue to be an area of active research.

Deep Q-Networks can be applied to Ms. Pac-Man, a classic arcade video game that was released in 1981 as an enhancement to the original Pac-Man game. In Ms. Pac-Man, the player controls the character as she navigates through various mazes, eating pellets and fruits while avoiding four ghosts: Blinky, Pinky, Inky, and Clyde. The ghosts will chase Ms. Pac-Man around the maze in an attempt to catch her, but she can eat power pellets to temporarily turn the tables on them and make them vulnerable, allowing her to eat them for extra points. The game becomes progressively more challenging as the player advances through the levels, with the ghosts becoming faster and the mazes more complex.

The game has become a popular subject for artificial intelligence research, with many researchers using it as a benchmark for testing their algorithms. Deep Q- Networks take in raw game pixels as input and output predicted optimal actions. Through trial and error and reward-based learning, DQNs can have internal networks approximate the Q-function and use these values to choose the right action and navigate the maze, defeating the ghosts and achieving higher scores in the game.

1.2. Related Work

”Playing Atari with Deep Reinforcement Learning” by Google DeepMind introduces a novel deep reinforcement learning algorithm called Deep Q-Network (DQN) and applies it to play classic Atari games. The paper demonstrates that the DQN algorithm outperforms all previous reinforcement learning algorithms on a range of Atari games and achieves human-level performance on several games (Mnih et al., 2013). In a later paper, these ideas are formalized, introducing several enhancements to the basic algorithm, including experience replay and a mechanism for adjusting the learning rate. The researchers also apply their algorithm to more games, including Ms. Pac-Man. They report only a modest 13% improvement over linear learners (Mnih et al., 2015).

The original algorithm has since been improved in a variety of ways. Prioritized Experience Replay (PER) is a technique used in reinforcement learning to improve the efficiency of the learning process. In traditional experience replay, the agent stores all of its experiences in a replay buffer and randomly samples from this buffer during learning. However, not all experiences are equally important for learning. PER assigns a priority value to each experience in the buffer based on how much the agent can learn from it. During learning, experiences with high priority values are sampled more frequently, allowing the agent to focus on the most informative experiences (Schaul et al., 2016).

The Average-DQN paper trains multiple Q-networks in par-

allel using different replay buffers, and takes the average of their Q-values as the final estimate. This helps to reduce the overestimation of Q-values, which is a common issue in the standard DQN. The paper presents experimental results showing that Average-DQN outperforms standard DQN on several Atari games (Anschel et al., 2017). This is the inspiration for the switching DQN experiment.

Finally, researchers use Double Q-learning to address the overestimation problem. The overestimation problem occurs when Q-learning is used with function approximators because the maximum Q-value is often overestimated due to correlations between state-action pairs. Double Q-learning solves this problem by using two separate Q-functions to decouple the maximization step from the value estimation, resulting in more accurate Q-value estimates. The paper applies this to several Atari games (van Hasselt et al., 2015). This paper forms the basis for my experiments.

1.3. Research Question

My research question investigates alterations surrounding the power-up pellets to see if changing how these are handled can increase algorithm performance. There are 3 explorations performed in this project:

1. How does removing the score bonus from eating ghosts affect performance?
2. How does reversing Ms. Pac-Man's movements upon eating a power-up pellet affect performance?
3. How does having two separate Q-Networks (one for when Ms. Pac-Man is powered up and one for when Ms. Pac-Man is not powered up) affect performance?
4. How do these compare to a baseline Deep Q-Network without any alterations?

2. Methods

2.1. Initial Code Base

I expand on the Double Deep Q-Network implementation repository found here. This repository gives the overall structure for my project. Meanwhile, my altered codebase to include the new experiments and new visualization scripts and code can be found here. See Figure 1 for the general structure of the DQN used in this repository. This structure is pretty common, but may be slightly different from some other papers. See the caption for more details.

2.2. Preprocessing

See Figure 2 for the preprocessing step. First, the board is greyscaled and cropped. Arguably, cropping loses out information, for example which life Ms. Pac-Man is on, the

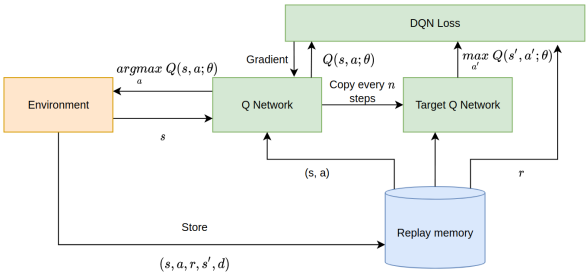

Figure 1. **DQN**: Structure of the DQNs used in each experiment. Note that the Policy Q-Network and the Target Q-Network are updated to match every $n = 8000$ steps, unlike some other implementations such as the original implementation by Mnih et al. or the Polyak updating method. The replay mechanism does not implement PER.

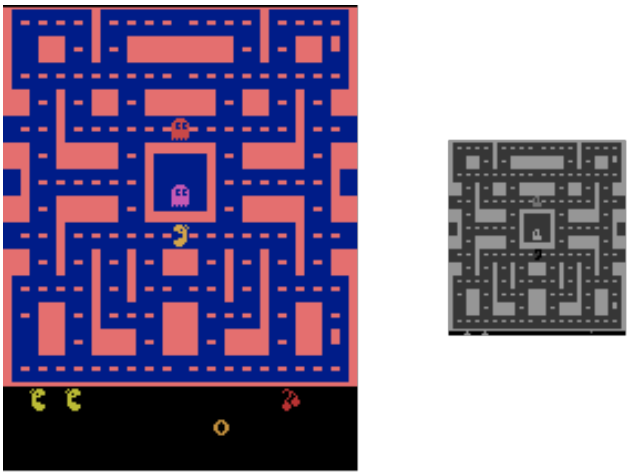

Figure 2. **Preprocessing**: The preprocessing grey-scales the pixels and crops them to cover only the board. The input is 160×192 RGB pixels, and the output is cropped to 160×170 RGB pixels, greyscaled, then down-sampled every four pixels to be 40×44. The image above is to scale. This process is applied in the same way across all experiments.

current score, and the fruits that Ms. Pac-Man has eaten. However, the image is cropped to allow for faster learning, focusing on the map and the positions of the pellets a ghosts. This makes the assumption that the strategy should be the same regardless of the current score or the remaining number of lives. While this simplification may not be completely valid, it seems generally reasonable because Ms. Pac-Man wants to maximize the score regardless of the current score or the current left-over lives.

Finally, the input is down sampled every four pixels. See the caption for more details.

2.3. Rewards

The rewards are issued on a log-based scale of the raw Ms.Pac-Man scores. The reasoning behind this relates to dealing with orders of magnitudes of rewards, much in the spirit of the DeepMind paper which shows the importance of normalization (van Hasselt et al., 2016). Negative rewards are also incurred for losing a life.

2.4. Hyper-parameters

The hyper-parameters were taken from the original code base. We are using a standard epsilon greedy exploration/exploitation scheme here with decay. See the codebase for more details on these parameters.

Hyperparameters	
Name	Value
Batch-size	128
Discount	0.99
Target Update	8000
Replay Memory Size	18000
Optimizer	SGD + momentum
Learning Rate	$2.5 \cdot 10^{-4}$
Momentum	0.95

2.5. CNN Architecture

The architecture of the network in Figure 3 includes two convolutional layers with 4 and 32 filters respectively, followed by two fully connected layers with 512 and 128 neurons respectively. The kernel sizes and strides are 4x4 and 2x2 respectively for the first and second convolutional layers. The padding is set to 2 for the first layer and 0 for the second layer. The batch normalization is applied after the convolutional layers, and the activation function used is ReLU. Finally, the output layer returns the Q-values for the possible actions.

2.6. Removing Ghost Bonus

Removing ghost bonuses was one of the tests performed. In this test, the bonus points from eating ghosts is set to zero

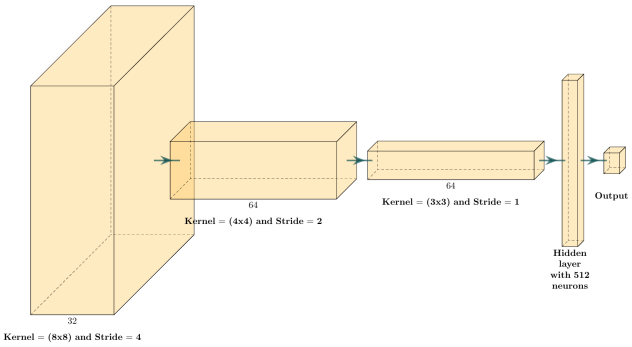

Figure 3. **CNN**: Visualization of the Convolutional Neural Network Architecture used in the DQNs.

while training the neural network. So, the information stored inside the experience replay is actually fictitious. However, when gauging the reward of the episode, we do tally up the points gained from eating ghosts so that we have a fair comparison across all alterations. In a sense, we are merely with-holding information from the neural network.

The reasoning behind this change is that the ghost will be more focused on learning how to survive for a longer and rather than dangerously learning to chase ghosts for temporary rewards. Upon inspection, I have observed the agent 'pellet camping', or idly waiting by a pellet for a ghost to approach; this is commonly not seen as a good strategy by experts in the game. This phenomenon may encourage bad behavior in the agent, perhaps even leading to sub-optimal local minima of the loss function. Perhaps removing ghost bonuses can help alleviate this.

2.7. Power-up Movement Reversal

Flipping the movement of Ms. Pac-Man when she is powered up was another of the tests performed. The idea is that Ms. Pac-Man wants to flee the ghost normally, but chase the ghost when powered up. Perhaps this alteration can take some advantage of this inherent symmetry in the game to allow the agent to learn the correct movements faster.

Specifically, there are 8 possible actions for Ms. Pac-Man which are up, down, left, right, up-left, down-left, up-right, down-right. I map up to down, left to right, up-left to down-right, and down-left to up-right. After the neural network has evaluated the position, and an action is chosen based on this evaluation, (after the epsilon exploring and exploiting), that action is reversed. In short, the neural network can learn to 'flee the ghost', even when powered up, and this behavior actually will chase the ghost and presumably result in additional bonuses.

2.8. Separate Q-Networks

Another experiment is implementing two separate neural networks, one to use when Ms. Pac-Man is powered up, and one to use when Ms. Pac-Man is not powered up. The reasoning is largely the same as before.

The specific implementation initializes two copies of the network. It is important to note that each has its own replay mechanism that is not shared. Each target network is still updated to resemble the policy network when that network reaches 8000 steps; each network is keeping a separate counter of when this should happen. The reward is taken over the different actions over these two Deep Q-Networks.

2.9. Training

The algorithm that ignores the ghost-related bonus was run for $2 \cdot 10^6$ step frames. For reference, each epoch is a complete game with three lives, and each epoch takes roughly 400 frames depending on the survivability of the algorithm. The ghost-reversal algorithm was also run for the same amount of steps. Meanwhile, the regular Deep Q-Network and the twin Deep Q-Network pair were each run for $4 \cdot 10^6$ steps to get a better sense of the long term behavior. The training process took many days running non-stop on a M2 MacBook Air.

3. Results and Analysis

3.1. Aggregate Results

See Figure 4 for the graphed results. Here we see that removing ghost bonuses led to the best rewards. Meanwhile reversing Ms. Pac-Man's movements performed about as well as the default Deep Q-Network. Meanwhile, having two Deep Q-Networks performed the worst, even when training for double the number of frames.

3.2. Removing Ghost Bonus

See Figure 5 and Figure 6 to see the rewards and the Q-value. As expected, as the Q-values increase, the rewards also increase.

3.3. Power-up Movement Reversal

See Figure 7 and Figure 8 to see the rewards and the Q-value. As expected, as the Q-values increase, the rewards also increase.

It is very challenging to compare this algorithm with the default algorithm. To attempt a comparison, I take data from the first 5000 epochs, performing a linear regression. The linear regression over the rewards suggests that the reversal algorithm is learning slightly faster, with a higher value of 314.63 vs 307.62. However, this difference could be subject to random change. To attempt to measure the 'variance', I measure the R^2 value with respect to the linear model. Again, the reversal algorithm appears have a higher R^2 value of 0.88 vs 0.83, indicating that it is less noisy as it learns the game. Again, the agent is very noisy, so these results do not hold statistical significance.

However, these results are drastically different when looking at the first 1000 epochs. Here, the linear regression of the movement reversal algorithm has higher coefficient of 99.61 vs 54.27, and the R^2 is 0.16 which is significantly better than 0.05.

This indicates that in the early phase of learning (~ 1000 epochs), reversing the movement does lead to steadier, faster learning. However, over time (~ 5000 epochs), the default algorithm catches up (perhaps as it naturally learns the 'reversal' without being explicitly enforced). So when training over time, there is very little difference between the two.

3.4. Separate Q-Networks

See Figure 9 and Figure 10 to see the rewards and the Q-value. This algorithm, surprisingly, clearly performs the worst, even if allowed double the training time.

One idea why this did not improve performance is that the data and experience is split into two different networks and replays, so each network is fed less data. Ms. Pac-Man is seldom powered-up, so that network is very underdeveloped compared to the other.

3.5. Default Q-network

See Figure 11 and Figure 12 to see the rewards and the Q-value. As expected, as the Q-values increase, the rewards also increase. There is a slight dip at Epoch 7000 reflected in the predicted Q-values, again indicating that the Q-Network is truly learning Q-table.

4. Discussion

4.1. Inside Knowledge vs End-to-End Training

These experiments use insider knowledge to design new algorithms for Ms. Pac-Man. For example, when preprocessing, the board was cropped to only include the important information. Similarly, these experiments apply only to Ms. Pac-Man and not to other Atari 2600 games such as Space Invaders. This contrasts with end-to-end training techniques that can be widely applied to all games. The former, as shown, can allow us to obtain better, steadier algorithms. Meanwhile, the latter can easily learn in new, unfamiliar settings.

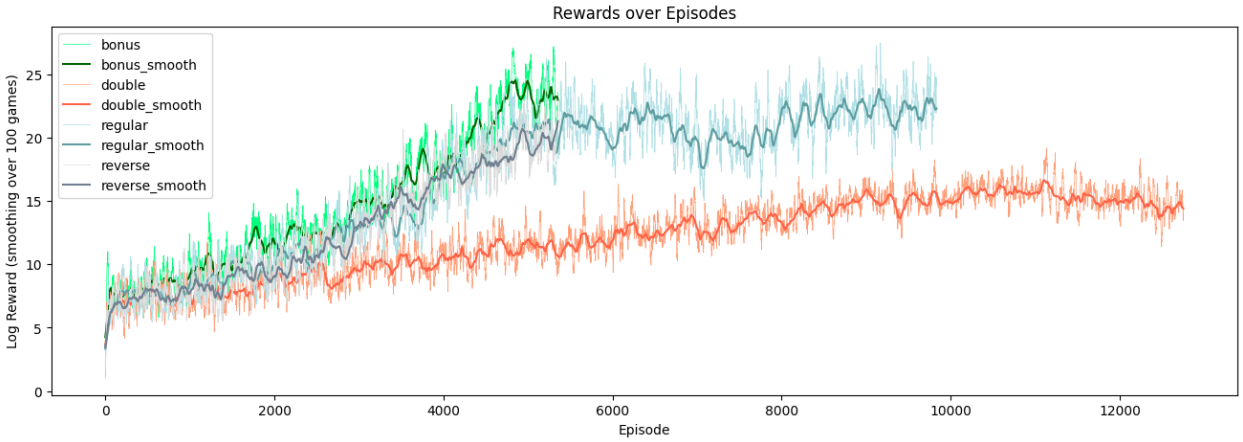

Figure 4. **Comparing the Algorithms**: The rewards for each algorithm over the Epochs trained are shown, and the darker line is smoothed as a rolling average centered over 100 epochs. (Each epoch is a game and each game has three lives.)

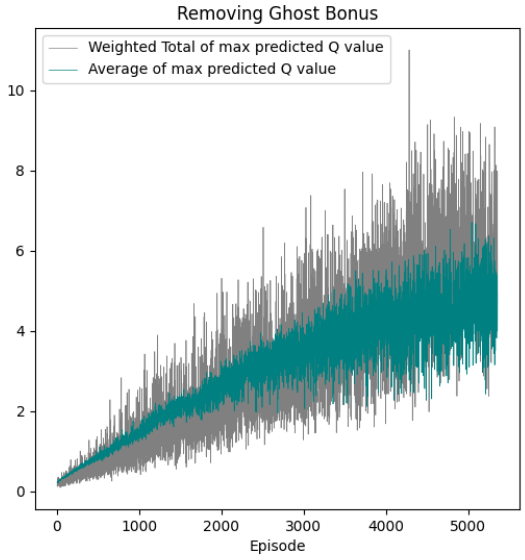

Figure 5. **Removing Ghost Bonus**: Predicted Q-values

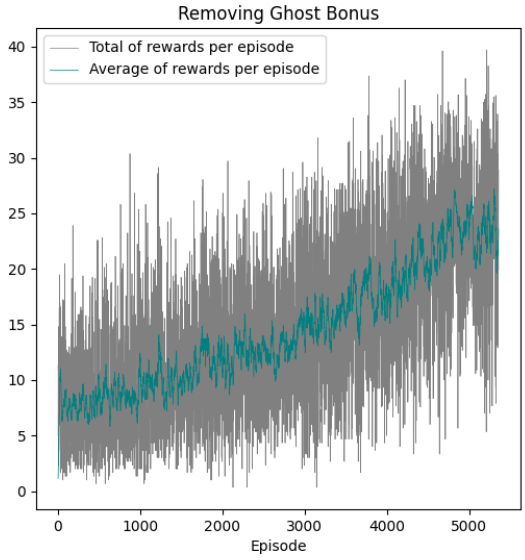

Figure 6. **Removing Ghost Bonus**: Rewards

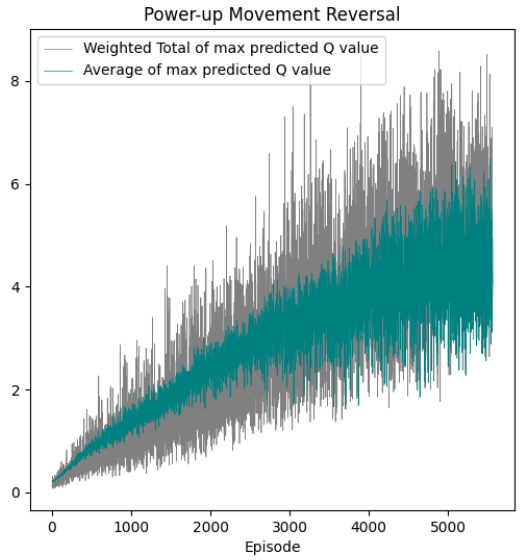

Figure 7. **Power-up Movement Reversal**: Predicted Q-values

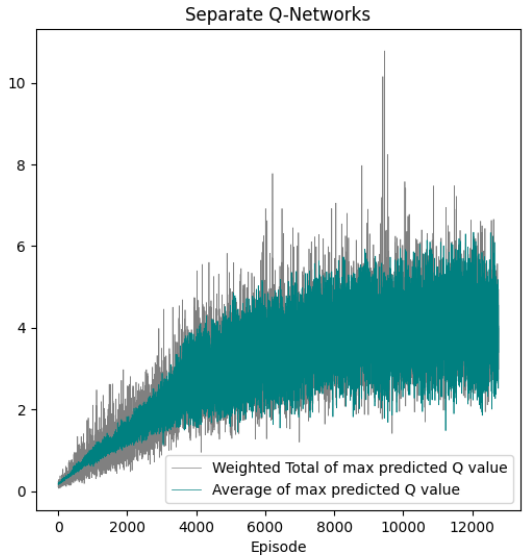

Figure 9. **Separate Q-Networks**: Predicted Q-values

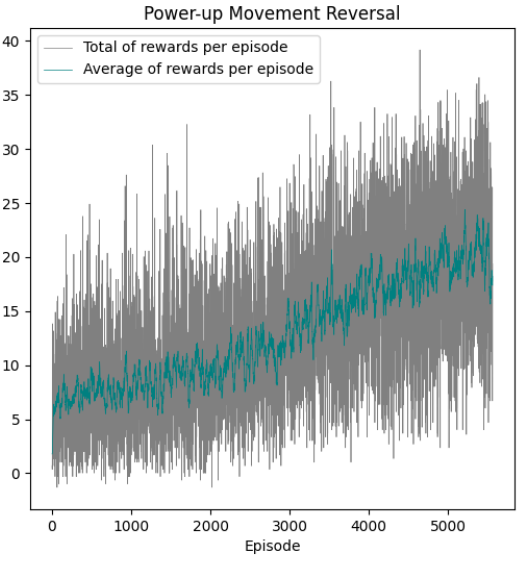

Figure 8. **Power-up Movement Reversal**: Rewards

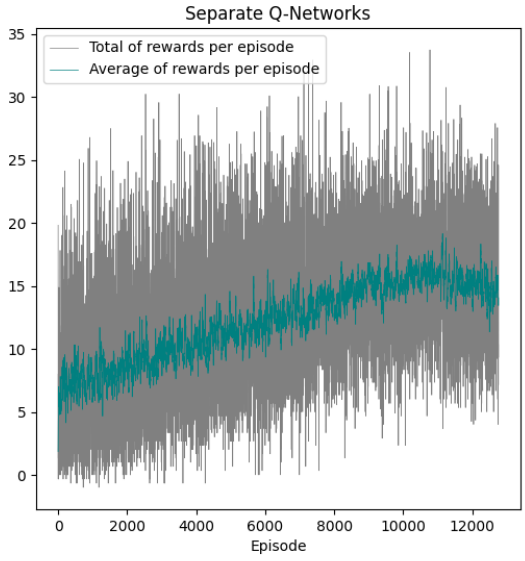

Figure 10. **Separate Q-Networks**: Rewards

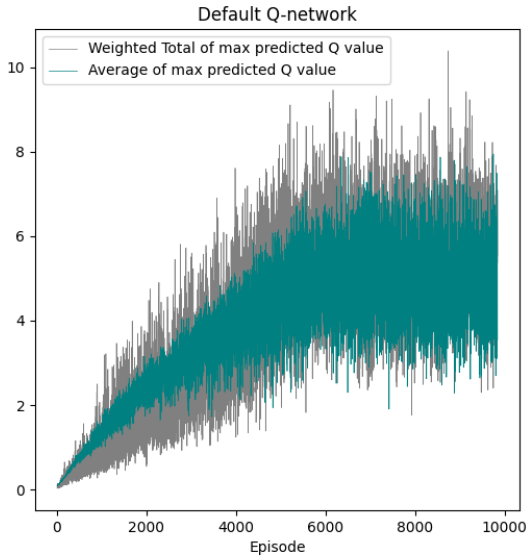

Figure 11. **Default Q-network**: Predicted Q-values

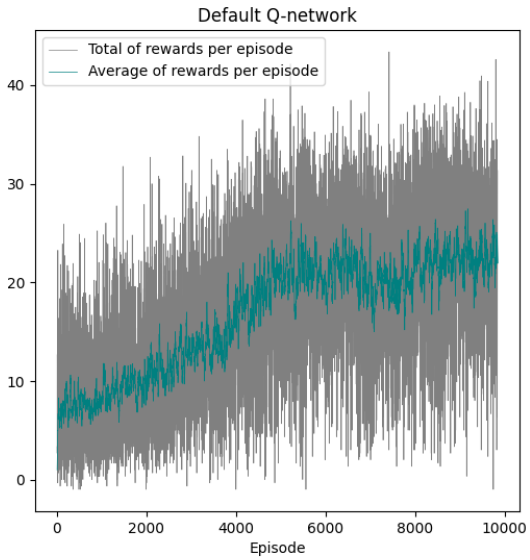

Figure 12. **Default Q-network**: Rewards

4.2. Importance

As noted above, the strategies discussed are specific to Ms. Pac-Man. However, perhaps some of the overarching ideas described (such as experimenting with symmetries, breaking these symmetries by removing rewards, trying separate networks) can be applied to other settings. However, this would again require some degree of leakage of information regarding the specifics of the problems being solved.

4.3. Limitations and Next Steps

Due to the noisy nature of the agents, perhaps performance varies by run. One major limitation was the computational power and the training time for the agents to play ~ 32000 games. It would be interesting to see if the results hold for more steps.

Another direction could be figuring out how to mathematically formalize or generalize the experiments in the context of the weights of the network. For example, perhaps there can be a mathematical definition for what it means to for a problem to have 'symmetry.'

References

Anschel, O., Baram, N., and Shimkin, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay, 2016.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning, 2015.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. Learning values across many orders of magnitude, 2016.