# Mastering Pac-Man using Reinforcement Learning


| Aman Kansal | Kritin Garg | Ansh Khurana | Kushagra Juneja |
|---|---|---|---|
| 170050027 | 170050028 | 170050035 | 170050041 |


## 1. Introduction

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning has wide applications. Throughout the course, we have learnt about Machine Learning techniques in supervised, unsupervised and reinforcement learning. For reinforcement learning, we learnt how to form MDP for various problems and how to solve them using the Value Iteration algorithm. In this project, we further explore reinforcement learning by using approaches like Q-learning, Approximate Q-Learning and Deep-Q-Learning on Pac-Man. Further, we demonstrate a new paradigm, Joint-Training of Pac-Man and the Ghost.


Figure 1. Reinforcement Learning Paradigm

## 2. Method

### 2.1. Q Learning

Q-learning is a model-free reinforcement learning algorithm. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. The agent explores the environment and learns by trial-and-error from interactions with the environment. The agent during its course of learning experience various different situations in the environment it is in. These are called states ($s$). The agent while being in that state may choose from a set of allowable actions ($a_i \in A$) which may fetch different rewards ($R$). The learning agent overtime learns to maximize these rewards so as to behave optimally at any given state it is in. The Q-Learning algorithm is called "model-free" since it does not try to capture the transition function and the reward. Contrary to how Value Iteration algorithm models the value function (the expected reward) and then chooses the action. The following are the update equations for the Q value (in episode n), capturing the essence of the algorithm:

$$Q_n(s,a) = (1-\alpha_n)\, Q_{n-1}(s,a) + \alpha_n \left[r_n + \gamma V_{n-1}\left(y_n\right)\right]$$

### 2.2. Approximate Q-Learning

In the Approximate Q-Learning approach, the model the agent learns weights for features of states, where many states might share the same features. Function approximation learns the weights of different features during training. Using function approximation for parameterizing the problem of Q-Learning significantly reducing training time. It is important to recognize significant features on which the expected gain for the state will depend. For example, for approximating the Q-function for the Pac-Man game would depend upon features like Distance to closest ghost or dot, number of ghosts nearby, proximity to walls etc. Considering a linear model in the features, the update equations for the weights is given by:

$$\text{Q(s, a)} = \sum_{i=1}^{n} f_i(s,a) w_i$$
$$\text{difference}=(r + \gamma \max_{a'} Q\left(s', a'\right)) - Q(s,a)$$
$$\text{w}_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

### 2.3. Deep Q-Learning

Deep Learning does away with the need to handcraft features for the states. In the Deep Q-Learning based approach, the state is captured by an image of the current situation of the game. Deep Q-Learning is an extension of function approximation for Q-Learning. Deep Q-Learning uses a deep neural network to approximate the Q values for every action, given the current state. Thus, the output layer of the Deep Q-Network has dimensions = $|A|$ (The number of actions). The best action after learning the network can be taken by calculating an argmax over the output layer.


Figure 2. A generic Deep Q-Network

## 3. Joint Training of MDP for Pac-Man and Ghost

### 3.1. MDP Formulation

An MDP is defined by set of S, A, T, R where
S: State Space i.e. set of all possible states 's'
A: Action Space i.e. set of all possible states 'a'
R: Reward for being in a state
T: Transition dynamics/ Probability transition Matrix
In our current formulation of MDP a state is a function of Pac-Man position, ghost position and dots position. Each state is assigned a number between 0 to NumStates-1 so that it uniquely determines the value of all three functions. Action Space 'A' is East, West, North, South. Reward values are defined such that the maximum positive reward is obtained on eating the dots and negative reward on getting eaten by ghost and idle movements. The only valid transitions end up in a non-blocking grid location with probability with due to only one final state possible for each action.

### 3.2. Joint Training

According to the concept of joint training for Pac-Man and ghost we increase the level of Pac-Man by training it on the most intelligent ghost. Similarly, we train the next level using previously trained Pac-Man. So level n+1 of Pac-Man is obtained by training on MDP obtained using ghost policy of level n. Similarly level n of ghost is obtained by training on MDP obtained using Pac-Man policy of level n. We apply value iteration algorithm to obtain Pac-Man policy for MdpPac-Man until it convergers.

## 4. Experiments

### 4.1. Q-Learning and Approximate Q-Learning

Using a python2 based game environment, we implement agents that learn by interacting with the environment. The parameters for the game are shown in Table 1. The average scores obtained by different agents, across different game sizes have been shown in Table 2.

| Situation | Reward |
|---|---|
| Win (finish food) | +500 |
| Lose (eaten by ghost) | -500 |
| Eat food | +10 |
| Consume ghost | +200 |
| Idle/no food | -1 |

Table 1. Game parameters for Q-Learning

The features chosen for training the Approximate Q-Learning Network are: bias, number_of_ghosts_1_step_away, distance_to_closest_food

| Approach | Grid size | Avg Score | #Epochs | Time (train) |
|---|---|---|---|---|
| AQL | mediumClassic | 1202.7 | 50 | 5m 45s |
| AQL | mediumGrid | 526.4 | 50 | 32s |
| QL | smallGrid | 499.5 | 2000 | 1m 24s |

Table 2. Some simulations

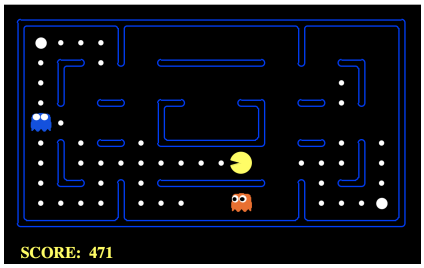


Figure 3. Pac-Man during Training

### 4.2. Deep Q-Learning

We use OpenAI's MsPacman-v0 environment to train a DQN based agent. Here, we use a neural network to approximately calculate the Q values corresponding to a given state which will be given as an input image.
The first three layers of the architecture are convolutional layers. The first layer has 32 filters of size 8*8 with stride 4. The second layer has 64 filters of size 4*4 with stride 2. The third layer has 64 filters of size 3*3 with stride 1. The fourth layer is fully connected 512 hidden nodes. The last layer is fully connected too with 9 output nodes corresponding to each of the actions possible.
We have used concepts like Replay memory and Epsilon greedy to help our model train better.

### 4.3. Joint Training of Pac-Man and Ghosts in MDP

We have trained 4 levels of both Pac-Man and ghost on two different grids. Refer to table 4 for the various types of

| Situation | Reward Values |
|---|---|
| Eat any dot | +200 |
| Eat last dot | 10000 |
| Eaten by ghost | -10000 |
| Normal Movement | -1 |
| Idle | -1 |

Table 3. Game parameters for Joint MDP

| Grid | Dimensions | No. of Dots |
|---|---|---|
| Small | 7X7 | 2 |
| Large | 14X14 | 8 |

Table 4. Game parameters for Joint MDP

grids used and table 3 contains the different rewards values corresponding to each action.

Level 0 policy of both ghost and Pac-Man is generated randomly. Level 1 of Pac-Man is generated by training on level 0 policy of ghost. Level 1 Pac-Man on running with level 0 ghost eats all the dots because ghost is moving randomly. Now we train level 1 ghost using level 1 Pac-Man, which eats the Pac-Man now before game ends because it was trained on Pac-Man Policy. Now we train Pac-Man on ghost level 1 which allows it to find a new path using ghost policy to collect all the dots. Similarly on training ghost level 2 it becomes even smarter than Pac-Man level2 and eats it before game ends. We train both of them till level 3, where ghost level 3 always eats Pac-Man irrespective of Pac-Man policy, and Pac-Man level 3 always eats all dots irrespective of ghost level.

## 5. Discussion & Conclusion

In this project, we tested a wide range of common reinforcement learning techniques. The Deep Q-Learning based model, the network learns important features from game-state images and thus, approximates Q values by experience. Given enough training time and model complexity, the network can learn and improve upon the game. However, computational cost for the Deep Neural Network is a major concern. As compared to Approximate Q-Learning, Q-Learning takes a much larger number of epochs to gain similar performance. Thus, Q-Learning is not scale-able for large grids. We find that approximate Q-Learning outperforms the Q-Learning based agent and also trains faster than both the above approaches. Thus, it may be the ideal approach for medium load reinforcement learning tasks like learning how to play Pac-Man on a medium sized grid.

For the joint training of Pac-Man and Ghost we tested on 2 grids training both the ghost and Pac-Man on various levels. It is a novel idea using MDP which works well on small and medium size grids with few number of dots. As the number of dots increases the number of states in the resulting MDP rises exponentially hence making the policy determination slower, however with few dots and small grid the learning and convergence is quite fast. The MDP policy is different from the other algorithms in terms of its complete deterministic nature and dependence on ghost policy. Hence, we can conclude that it is a good approach for high score in smaller grids as well as it serves as a method to train better ghosts.

## References


[1] https://www.geeksforgeeks.org/deep-q-learning/

[2] https://www.cs.swarthmore.edu/ bryce/cs63/s16/slides/3-25approximateQ-learning.pdf main ref for QL:

[3] http://www.gatsby.ucl.ac.uk/ dayan/papers/cjch.pdf

[4] https://courses.cs.washington.edu/courses/cse473/16au/slides-16au/18-approx-rl2.pdf

[5] https://www.intel.ai/demystifying-deep-reinforcement-learning/#gs.hxyudi

[6] https://pdfs.semanticscholar.org/2d7e/7d809af7d68c.pdf

[7] https://github.com/moduIo/Deep-Q-network

[8] https://towardsdatascience.com/advanced-dqns-playing-pac-man-with-deep-reinforcement-learning-3ffbd99e0814

[9] https://lilianweng.github.io/lil-log/2018/05/05/implementing-deep-reinforcement-learning-models.html

[10] https://gym.openai.com/envs/MsPacman-v0/

[11] https://github.com/gauravmittal1995/Pyman

[12] https://pythonprogramming.net/pygame-python-3-part-1-intro/

[13] https://oregonstate.instructure.com