Applying Deep Reinforcement Learning towards NPCs in Pacman

Vidya Preetha Anandamurali, Harshil Prajapati, Aditya Singh , Swarnim Sinha, Nidhi Tiwari
{vidyaam, harshil, aditya28, swarnims, nidhit}@bu.edu

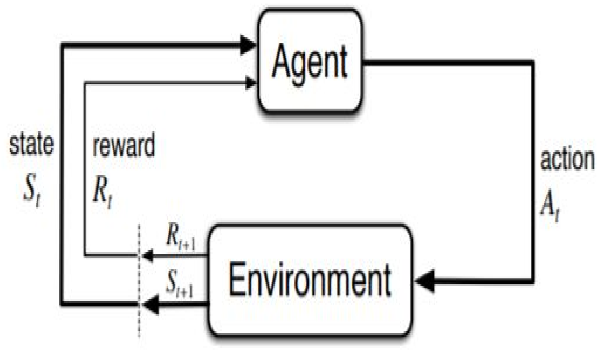

Figure 1. The agent–environment interaction in a Markov decision process [18].

Abstract

The application of deep learning models to accurately learn the control policies of the player in modern computer games is one that has been trained and tested on many computer games with high performance. However, deep reinforcement learning can also be used to train the Non Player Characters or NPCs to make the bots in a game more competent and difficult to defeat. We will attempt to apply techniques that have been traditionally used to provide expertise to the player characters, to provide similar expertise to the NPC enabling it to learn and improve with the players movements using a reward-penalty system implemented via a Deep Q-network and provide an in depth analysis of the change in behaviour of the NPCs.

1. Task

Reinforcement Learning, formulated as a Markov Decision Process, refers to goal oriented algorithms which move toward learning a complex objective with a system of incentives and penalties. The task we have outlined for ourselves is to give the Non-Player Characters (NPCs) in a modern computer game smarter abilities of movement and decision making by applying Deep Reinforcement learning so that they are not easily defeated by practised manoeuvres [15]. The NPCs will be able to learn more creative strategies than those hard coded by a human and the game can become a more challenging and enjoyable experience for the player.

2. Related Work

Google's DeepMind's Deep Q-Learning algorithm that learnt to expertly play dozens of Atari games based on raw pixel data was the first significant step in the direction of using Deep Reinforcement Learning for unsupervised learning problems [11][12]. Post this, there has been an expansion of applying such techniques to train the players in a myriad of modern computer games. Most recently OpenAI implemented reinforcement principles on Dota 2 bots to give impressive results but was defeated by human players [7].

Training the NPCs of a game has been done using reinforcement learning using a combination of Q-learning and fuzzy logic without formulating a deep neural network for the training [2]. Specifically this was done for Battle Tanks. We aim to combine the methods normally used to train player characters with the application of training NPCs without the requirement of fuzzy logic.

Evolutionary Artificial Neural Networks (EANNs) have been effectively used to train NPCs in video games using Temporal Difference Learning. The difference between the one state and a later state were calculated. By refreshing each value and result, the best policy to reach the goal was attained [8].

A different architectural approach to Deep Reinforcement learning was the development of Duelling Networks that separates the two streams: the state value and the (state dependent) action advantage which is combined via a aggregating layer to estimate the Q-function value [See Section 3]. This network architecture has outperformed the state-of-the-art in the Atari 2600 domain [19].

There has been a lot of enabling work in terms of environments and software framework where such reinforcement learning algorithms can be implemented and tested. OpenAI Gym is one unique toolkit which has been developed for developing and comparing reinforcement algorithms [13]. The gym library is a collection of environments that makes no assumptions about the structure of your agent and is compatible with Tensorflow, greatly easing the training and testing of deep reinforcement learning algorithms [14].

3. Approach

Our approach will be centered around applying Deep Reinforcement Learning to the movements of the Non Player characters or ghosts in the Pacman game. For this purpose, we will implement the Q-learning algorithm which uses the Q-function used to approximate the reward based on a state [16]. This approach is a good solution for problems in an unknown environment. The following section outlines the approach. For more information on the experimental environment, refer to Section 4.

We define the loss function in terms of the Q function ($Q(s,a)$) which calculates the expected future value from state s and action a. The loss function we want to optimise is:

$$loss = (r + \gamma \cdot max_{\widehat{a}} \widehat{Q}(s, \widehat{a}) - Q(s,a))^2$$

Action a is carried out, the reward r is observed and the resultant state is s [1][20].

Tensorflow is used to implement a Q-network to evaluate the Q-function corresponding to the five possible actions at each state: moving North, South, East, West, Stop [5]. We will use the methods of $Remember()$ and $Replay()$.

- $Remember()$ helps us maintain memory experiences so we can re-train the model with previous experiences.

- $Replay()$ is the method that enables the training using experiences from memory using mini-batches[1][9]. To perform experience replay we store the agent's experiences:
$$e = (s_t, a_t, r_t, s_{t+1})$$

This enables efficient use of previous experiences, by learning with it multiple times. This is the key method to use when gaining real-world experience is costly. The Q-learning updates are incremental and do not converge quickly, so multiple passes with the same data is beneficial, especially when there is low variance in immediate outcomes (reward, next state) given the same state, action pair.

The agent (NPC or ghost) will first randomly select its action based on a value called the '*exploration rate*' or '*epsilon*' so that it gets to try all kinds of actions before it starts to see patterns [17].

For the first step of our implementation, we limited ourselves to the training of the Pacman using a single ghost. We trained the Pacman using a 4-layer Deep Q-Network where the reward system will directly relate to the score of the game as follows:

- The reward for eating food is 10 score points.
- The reward for eating a ghost is 50 score points.
- The penalty for number of moves taken in taking the above actions is 1 score point.
- The reward for eating the last food without getting eaten by the ghost is 500 score points.
- The penalty for getting eaten by a ghost is kept high at 500 score points leading to the end of the game and returning 'True' or 'False' indicating the final game state.

3.1. Implementation

We worked with a low complexity neural network architecture to implement the concept of Q-Learning.
Q-network Architecture:

- Layer 1: Convolutional (size-3, channels-6, filters-16, stride-1)
- Layer 2: Convolutional (size-3, channels-16, filters-32, stride-1)
- Layer 3: Fully Connected (hidden - 256)
- Layer 4: Fully Connected (hidden - 4)
- Cost calculation based on the Q-function with epsilon 0.1 to 1. We ran the network for different values of discount and its effects are discussed in the results.
- Cost Optimizer - Adam

4. Dataset and Metric

We will be using a simulator of Pacman in python as the environment in which we will carry out the training and testing of the Deep Q-Learning approach or DQN on the NPCs. We chose the following implementation of Pacman:

- Pacman implementation by UC Berkeley - A python based implementation of Pac-man with options to customize the controls and the behavior of the game [24].

- Deep Reinforcement Learning in Pac-man - An implementation of DQN on UC Berkeley's Pac-man game [25].

In the UC Berkeley version of Python, there are two options of controlling the bots - random and directional (where the bots move in the direction of the Pacman - using the way which would minimize the Manhattan distance between them).

We aim to simultaneously train our reinforced ghosts against a Pacman using two Deep Q-networks [21] with different reward system. We attempt to show that our expert version of the bot behaves smartly.

For training our Deep Q-network, we are sending the state of the game as a Matrix with coordinates delimiting the positions of the agents of the system and not the screen shots of the game itself.

5. Preliminary Analysis

- We successfully implemented the Deep Q Learning Algorithm by training it on the Pacman first to understand the reward system using a single ghost. We used the implementation to save the trained model with the aim of using the same to train our Ghost.
- In Q-learning, the function Q_Θ(s,a), parameterised by θ, is defined as the expected discounted future reward in a state s given an action a. The function returns the Q-value of a certain state-action pair. The Q-value can be calculated with the function Q_Θ(s,a) and represents the "quality" of the action 'a' from state 's' [22].
  - A small negative reward is given over time, in order to advocate quick solutions.
  - The Q-network and the target network consist of two convolutional layers followed by one fully connected layer. The convolutional layers are valuable because the game states in the Pac-man game imply the existence of local-connectivity. As a result, CNNs can be extremely useful to improve the classification of features in the game grid. The last layer (fourth) gives the outputs used to optimise the Q function.
  - Conventionally, Q learning uses a data table that stores the state and action pair of each set in the table. This is a highly complicated approach with almost no optimization. Deep Q learning involves the Q values that are obtained from the convolutional neural network instead of the data table and hence, the learning process is smarter in this case.

6. Reward System Hypothesis:

According to our rewards system, we are rewarding the NPCs (ghosts) for successful moves against the Pacman and penalties if the Pacman succeeds.

The reward system for a single ghost and Pac-man will initially be the opposite of the reward system for the Pacman DQN outlined in the approach Section 3.

6.1. Reward system for one ghost and Pacman:

We wanted to incentivise the ghost to move closer to the Pac-man and and track its movements to increase the odds of eating the Pacman. The basic reward system of the ghosts is formulated thus:

- Punish 150 points if the Pac-man eats the ghost (Pac-man can eat the ghost only in the Classical layout)
- Penalize 10 points if the Pac-man eats food
- If the ghost eats the Pac-man, grant it 500 points and set game status to win.
- For each movement made, if the ghost moves farther away from the Pac-man (the Manhattan distance increases), penalize with 1 point.

6.2. Reward system for two or more ghosts and Pacman:

Apart from the reward system, described in Section 6.1, we wanted multiple ghosts to work together. For this, we kept the reward system for one of the ghosts (leader) the same as the basic reward system. For the other ghosts (responders), we wanted them to work together with the leader and attack the Pac-man from different angles.

We added the following rewards for the responders:

- If the leader, Pac-man and the ghost are on the same X or Y axis and the leader and the responder mirror each other's movements, we gave a reward of +10 points.
- If they (leader and responder) were on the same axis but weren't moving in opposite directions, we penalized 1 point.
- If they weren't on the same axis but were mirroring each other's movements, we penalized 5 points.
- If they weren't on the same axis and weren't mirroring their moves, we rewarded 1 point.

7. Results

The training that was carried out on three different configurations of grids.

(i) Small Grid: 7 by 7 pixels

(ii) Medium Grid: 7 by 8 pixels

(iii) Medium Classic Grid: 20 by 11 pixels

We implemented the Small grid and Medium grid with a single Pacman and a single ghost. For two ghosts we trained the model with on medium Grid. For three ghosts and single Pacman, we utilised the medium Classic layout. While implementing two ghosts, we trained them with a reward system independently and with teamwork, for three ghosts, we relied purely on team work to train the model. In formulating the reward system for more than one ghost, one ghost was assigned an independent status (which we refer to as leader), while the other ghosts are assigned responder status, implying their reward systems are dependent on the movements of the independent ghost.

While analysing the results, we decided to analyse the behaviour of the player and non-player characters of the game, and not only based on the win/loss rate. We analysed the Q values, the steps taken to finish a game, how these values varies based on the change in the epsilon values and the discount values. We also analysed the behaviour of the ghosts in a way to judge their response to the reward system that we implemented. A discussion of the same is found below.

We analysed the variation of the Q values with respect to the number of iterations that we trained the game for. The following results are for the Q values for 2 ghosts and Pacman in a medium grid with teamwork.

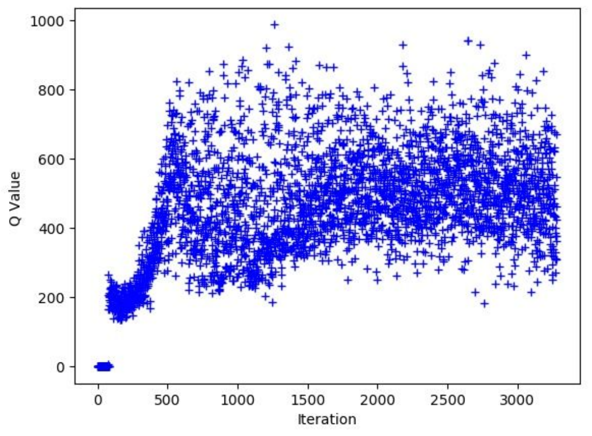

Figure 2. Ghost (2) Q values vs. number of iterations

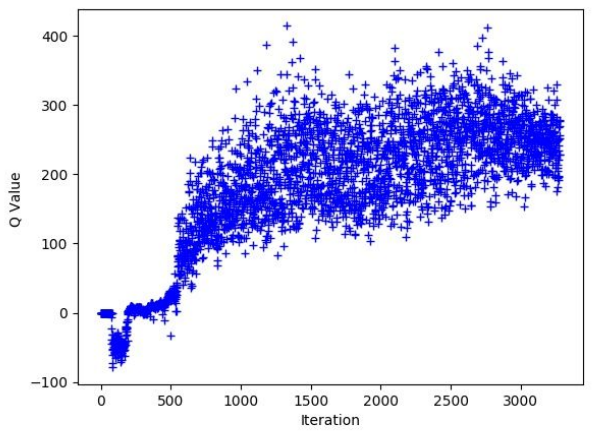

Figure 3. Pac-Man Q values vs. number of iterations

For the first ghost, the Q values do not increase initially as the training has only begun. However, as the training proceeds and the ghost begins to learn from the Pacman movements and the reward system and the probability of it winning increase with increase in the Q - value.

For the pac-man, the q-values initially does not increase as the training hasn’t started. After training starts, the Q-Value becomes negative for some iterations as the Pac-Man is “dumb” and the game is heavily favored towards the ghosts, it isn’t able to collect the food and just wastes time by making random/useless moves. After a few iterations, as the Pac-Man learns the game, the Q-value gradually increases.

7.1 Small Grid - Pacman & One ghost:

Since the number of iterations required to successfully train the Pacman and the ghost depends on the grid size, we initially tested our training on the smallest optimal grid configuration. The reward system implemented was the most basic with an added condition involving the manhattan distance as explained in Section 6. However, as the grid size is really small, the movements of the ghost is restricted as there weren’t many decisions for the ghost to make. We decided to not use the Small Grid for further training.

7.2 Medium Grid - Pacman & One ghost

The ghost responds to the negative penalty for Pacman eating the food and the reward for reducing the Manhattan distance between itself and Pacman and eating the Pacman. The ghost now attempts to guard the food from the Pacman and responds more strongly to tracking the Pacman's movements rather than eating the Pacman. Since, the ghost cannot change its direction by 180 degrees, it kept forming a 2x2 square where one of the corners was food. Due to close proximity with the Pacman, a random movement of the ghost results in killing of the Pacman and this increases the win rate for the ghost.

We believe that the ghost was behaving in such a way because if it kept chasing the Pac-man, the ghost would have never been able to catch it as their speeds are the same and the Pac-man would have been able to eat all the food - winning the game. So, the ghost learnt to guard the food particles while being close to the Pac-man and adding random "attacking" moves in between.

7.3 Medium Grid - Pacman & Two ghosts

Independently:

Here, the reward system is applied to both the ghosts independently. Both ghosts independently attempt to reduce the distance with the Pacman and block it from eating the food. While doing so they track and limit the Pacman's movements and consequently increase the win rate for the ghost. In many iterations, one ghost guards the food while the other follows the Pacman; however, this is not a case of teamwork but two reward systems manifesting differently for the two ghosts as they kept switching their roles during the same game.

Teamwork:

For implementing teamwork between the two ghosts against the Pacman, the reward system was as described in Section 6.2. This changed the ghost movements significantly. With the responder rewarded to work with the independent ghost to try and attack the Pacman and limit the food it can eat, when the opportunity arises to trap the Pacman between the two ghosts, the responder is incentivised to mirror the movements of the leader and restrict the movements of the Pac-man thus increasing the win rate.

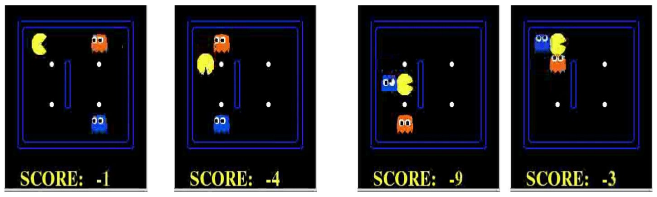

Figure 4. Teamwork between ghosts for Medium Grid

As evident from the above image, the two ghosts move closer to the Pac-man and then attack from different angles - restricting its movement. For more descriptive GIFs, please visit our GitHub page.

7.4 Medium Classic Grid - Pacman & Three ghosts

We again trained the ghosts using the reward system described in Section 6.2 but with one leader and two responders. The attempt was to enable all the ghosts to reduce their manhattan distance with the Pacman and trap it by working in tandem. We observed that one of the ghosts kept chasing the Pac-man while others explored the map, guarded the food and attacked from different angles. Since, there were 'bigger' food blobs in this version of the game, which "scared" the ghosts, they became edible by the Pac-man, the Pac-man learnt to first target the big food blobs in the game in order to slow down the ghost movements and during that time quickly eat the food.

In future, we wish to penalise eating of the bigger food blobs more than the normal food blobs to make sure that the ghosts prioritize covering the bigger blobs over covering the normal ones.

8. Detailed Timeline and Roles

Task	File names	Lead
Reward system design in DQN implementation & result discussion	ghostDQN_ Agents.py, ghosts.py	Swarnim
DQN implementation & Result plotting	extract_logs .py	Aditya
Reward system design in DQN implementation & result discussion	ghostDQN_ Agents.py, ghosts.py	Nidhi
DQN implementation & Result plotting	extract_logs .py	Vidya
DQN implementation & Result plotting	extract_logs .py	Harshil
Prepare report and presentation	Final Report	all

9. Code Repository

https://github.com/adityachamp/DL_RL_CollisionAvoidance

References:

1) Brandon Brown, Outlace. (2015, Oct 19). "Reinforcement Learning". Retrieved from http://outlace.com/rlpart1.html

2) Fang, Yung-Ping, and I-Hsien Ting. "Applying Reinforcement Learning for the AI in a Tank-Battle Game." JSW 5.12 (2010): 1327-1333.
3) Grant Jenks. "Free Python Games". Retrieved form https://github.com/grantjenks/free-python-games/blob/master/freegames/pacman.py
4) Hans-Jürgen Pokmann (2012. Oct 31). _"Pacman in Python with PyGame"_. Retrieved from https://github.com/hbokmann/Pacman
5) Jabrils. (2018, Jan 14). "FINISHING MY FIRST MACHINE LEARNING GAME". Retrieved from https://www.youtube.com/watch?v=GDy45vT1xlA
6) Jake Grigsby. (2018, June 29). "Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning". Retrieved from https://towardsdatascience.com/advanced-dqns-playing-pac-man-with-deep-reinforcement-learning-3ffbd99e0814
7) James Vincent. (2018, Aug 28). "OPENAI'S DOTA 2 DEFEAT IS STILL A WIN FOR ARTIFICIAL INTELLIGENCE". Retrieved from https://www.theverge.com/2018/8/28/17787610/openai-dota-2-bots-ai-lost-international-reinforcement-learning
8) Jin, Xiang Hua, Dong Heon Jang, and Tae Yong Kim. "Evolving Game NPCs Based on Concurrent Evolutionary Neural Networks." International Conference on Technologies for E-Learning and Digital Entertainment. Springer, Berlin, Heidelberg, 2008.
9) Keon. (2017, Feb 06). "Deep Q-Learning with Keras and Gym". Retrieved from https://keon.io/deep-q-learning/
10)Keras. "DQNAgent". Retrieved from https://keras-rl.readthedocs.io/en/latest/agents/dqn/
11) Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529
12) Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
13) OpenAI. (2018, June 25). "OpenAI Five". Retrieved from https://blog.openai.com/openai-five/
14) Open AI. "Getting Started with Gym". Retrieved from https://gym.openai.com/docs/
15) Robot Revolution. (2017, Sept. 09). "Machine Learning Is Making Video Game Characters Smarter And Robots More Competent". Retrieved from https://www.fastcompany.com/40469609/machine-learning-is-making-video-game-characters-smarter-and-robots-more-competent
16) Skymind. "A Beginner's Guide to Deep Reinforcement Learning". Retrieved from https://skymind.ai/wiki/deep-reinforcement-learning
17) Study Wolf. (2012, Nov 25). "REINFORCEMENT LEARNING PART 1: Q-LEARNING AND EXPLORATION". Retrieved from https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/
18) Sutton, Richard S., Andrew G. Barto, and Francis Bach. Reinforcement learning: An introduction. MIT press, 1998.
19) Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015)
20) Yash Patel. (2017, July 30). "Reinforcement Learning w/ Keras + OpenAI: DQNs". Retrieved from https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c
21) Abeynaya Gnanasekaran, Jordi Feliu Faba, Jing An. "Reinforcement Learning in Pacman.". Retrieved from http://cs229.stanford.edu/proj2017/final-reports/5241109.pdf
22) van der Ouderaa, Tycho. "Deep Reinforcement Learning in Pac-man." (2016) retrieved from https://esc.fnwi.uva.nl/thesis/centraal/files/f323981448.pdf
23) T. Kulkarni, Deep Q Learning in Tensorflow for ATARI, https://github.com/mrkulk/ deepQN_tensorflow, 2012.
24)UC Berkeley, Pacman implementation fhttp://ai.berkeley.edu/project_overview.html
25)Deep Reinforcement Learning in Python https://github.com/tychovdo/PacmanDQN/