TERM PAPER DRAFT

Topic: ”Automating Pac-man with Deep Q Learning”

Name: Prajwal Mani

UCID: pbm6

Email: pbm6@njit.edu

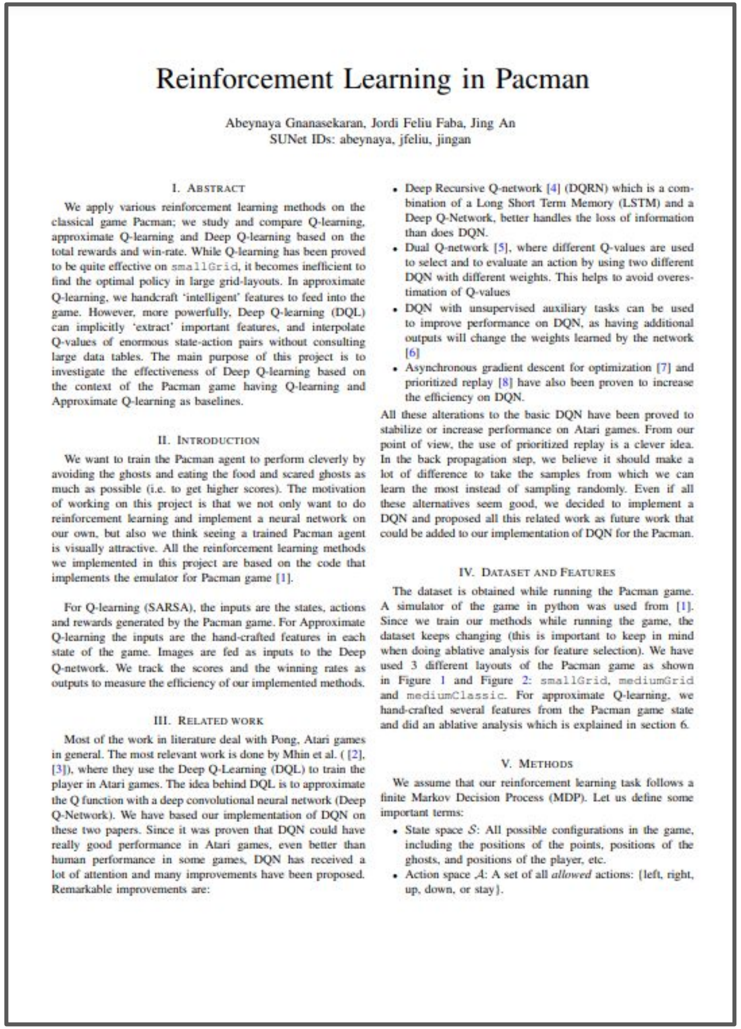
Subject: CS 634 104

Contents

Base paper

Paper link: Reinforcement Learning in Pacman

The base paper is a course paper under stanford university written by Abeynaya, Jordi and Jing. The paper talks about the process of combining Q Learning, Deep Learning, and the game itself with all the possible strategies to get the best outcome of the entire project.



Reinforcement Learning in Pacman

Abeynaya Gnanasekaran, Jordi Feliu Faba, Jing An
SUNet IDs: abeynaya, jfeliu, jingan

I. Abstract

We apply various reinforcement learning methods on the classical game Pacman; we study and compare Q-learning, approximate Q-learning and Deep Q-learning based on the total rewards and win-rate. While Q-learning has been proved to be quite effective on `smallGrid`, it becomes inefficient to find the optimal policy in large grid-layouts. In approximate Q-learning, we handcraft 'intelligent' features to feed into the game. However, more powerfully, Deep Q-learning (DQL) can implicitly 'extract' important features, and interpolate Q-values of enormous state-action pairs without consulting large data tables. The main purpose of this project is to investigate the effectiveness of Deep Q-learning based on the context of the Pacman game having Q-learning and Approximate Q-learning as baselines.

II. Introduction

We want to train the Pacman agent to perform cleverly by avoiding the ghosts and eating the food and scared ghosts as much as possible (i.e. to get higher scores). The motivation of working on this project is that we not only want to do reinforcement learning and implement a neural network on our own, but also we think seeing a trained Pacman agent is visually attractive. All the reinforcement learning methods we implemented in this project are based on the code that implements the emulator for Pacman game [1].

For Q-learning (SARSA), the inputs are the states, actions and rewards generated by the Pacman game. For Approximate Q-learning the inputs are the hand-crafted features in each state of the game. Images are fed as inputs to the Deep Q-network. We track the scores and the winning rates as outputs to measure the efficiency of our implemented methods.

III. Related work

Most of the work in literature deal with Pong, Atari games in general. The most relevant work is done by Mhin et al. ([2], [3]), where they use the Deep Q-Learning (DQL) to train the player in Atari games. The idea behind DQL is to approximate the Q function with a deep convolutional neural network (Deep Q-Network). We have based our implementation of DQN on these two papers. Since it was proven that DQN could have really good performance in Atari games, even better than human performance in some games, DQN has received a lot of attention and many improvements have been proposed. Remarkable improvements are:

- Deep Recursive Q-network [4] (DQRN) which is a combination of a Long Short Term Memory (LSTM) and a Deep Q-Network, better handles the loss of information than does DQN.
- Dual Q-network [5], where different Q-values are used to select and to evaluate an action by using two different DQN with different weights. This helps to avoid overestimation of Q-values
- DQN with unsupervised auxiliary tasks can be used to improve performance on DQN, as having additional outputs will change the weights learned by the network [6]
- Asynchronous gradient descent for optimization [7] and prioritized replay [8] have also been proven to increase the efficiency on DQN.

All these alterations to the basic DQN have been proved to stabilize or increase performance on Atari games. From our point of view, the use of prioritized replay is a clever idea. In the back propagation step, we believe it should make a lot of difference to take the samples from which we can learn the most instead of sampling randomly. Even if all these alternatives seem good, we decided to implement a DQN and proposed all this related work as future work that could be added to our implementation of DQN for the Pacman.

IV. Dataset and Features

The dataset is obtained while running the Pacman game. A simulator of the game in python was used from [1]. Since we train our methods while running the game, the dataset keeps changing (this is important to keep in mind when doing ablative analysis for feature selection). We have used 3 different layouts of the Pacman game as shown in Figure 1 and Figure 2: `smallGrid`, `mediumGrid` and `mediumClassic`. For approximate Q-learning, we hand-crafted several features from the Pacman game state and did an ablative analysis which is explained in section 6.

V. Methods

We assume that our reinforcement learning task follows a finite Markov Decision Process (MDP). Let us define some important terms:

- State space $\mathcal{S}$: All possible configurations in the game, including the positions of the points, positions of the ghosts, and positions of the player, etc.
- Action space $\mathcal{A}$: A set of all *allowed* actions: {left, right, up, down, or stay}.

Abstract

In this paper the authors talk about different Reinforcement Learning methods to make the agent play the classical pacman game. The three methods which they compare are Q Learning, Q Learning approximate and Deep Q Learning based on win rates and total rewards. Most of the time the paper talks about how to make Deep Q Learning method more effective while keeping Q Learning and Q Learning approximate as baselines.

Introduction

The agent is trained to play cleverly by avoiding the ghosts, eating food and scared ghosts as much as possible. The entire game was emulated. The authors goals were not only to train the agent but also to make the agent visually attractive

Then the paper talks about what kind of inputs are passed for different methods

For **Q Learning** the inputs are states, actions and rewards generated by the game.

For **Q Learning approximate**, the inputs are preprocessed features in each state of the game.

For **Deep Q Learning**, the images are feed as the inputs.

To measure the efficiency they used win rates and scores of each method.

Related work

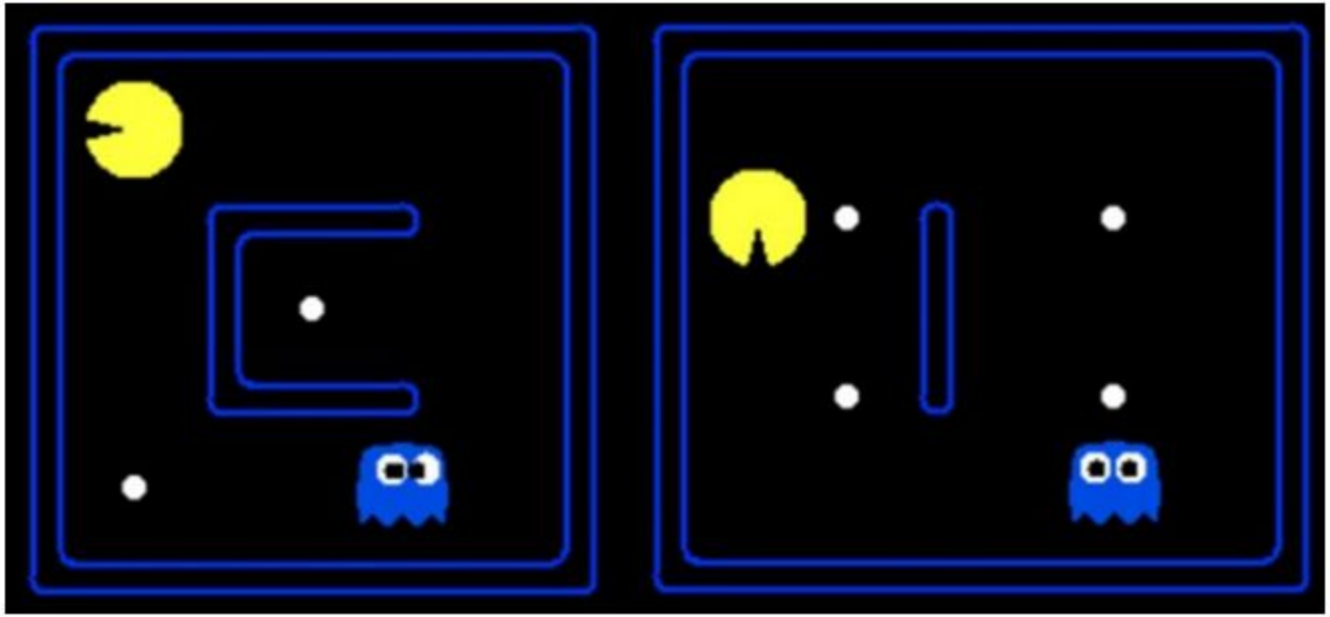
- Most of the papers they referred to deal with ping pong(Atari games)
- The most relevant work was done by Mhin et al which was a combination of Deep Q Learning and Q Learning approximate.
- Deep Recursive Q-Network(DQRN) which is a combination of Long Short Term Memory(LSTM) and a Deep Q Network which handles loss better than DQN
- Dual Q Network where different Q values are used to select and to evaluate an action by using two different DQN this method helped to avoid over estimation of Q values.
- To increase the efficiency of the agent we can use asynchronous gradient descent for optimization and prioritized replay
- After checking out all these methods, the authors used DQN as a final method to implement.

Dataset

The entire dataset is obtained while running the pacman game using the simulator which is available in python. Since the data is captured while running the model the data will not be static based on how the agent plays the game. The pacman game comes with different layouts compare to the classic game they have used three of the popular ones for Reinforcement Learning viz smallGrid, mediumGrid and mediumClassic.

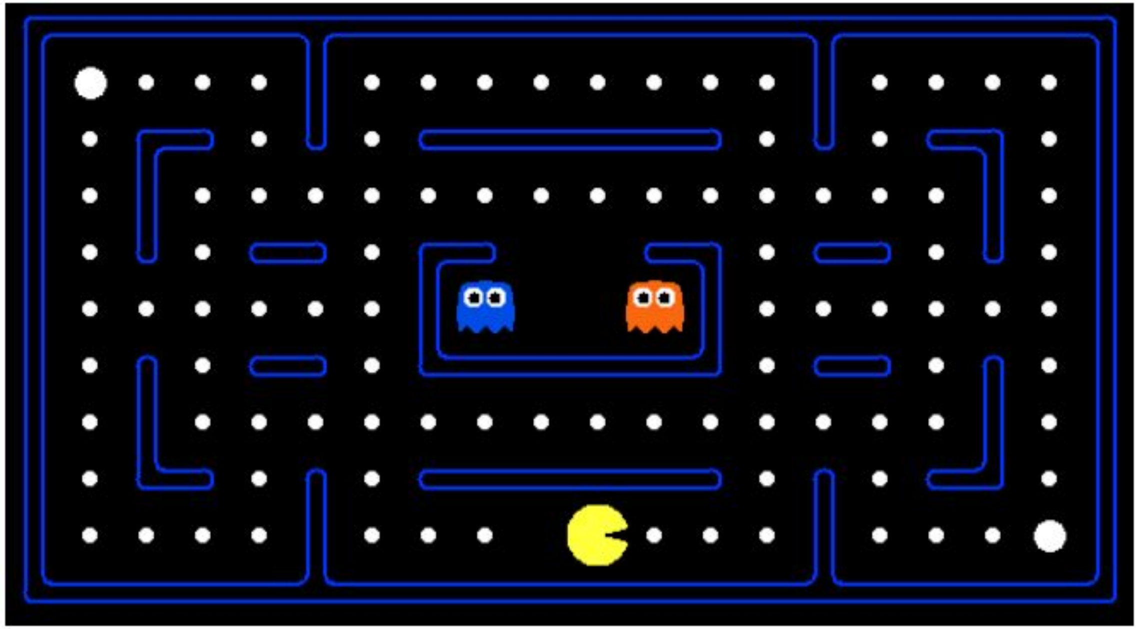
SmallGrid and MediumGrid

The below picture shows a snapshot of the smallGrid and mediumGrid from the simulation



MediumClassic

As the previous slide showed us 2 of the layouts, here the mediumClassic layout is shown.

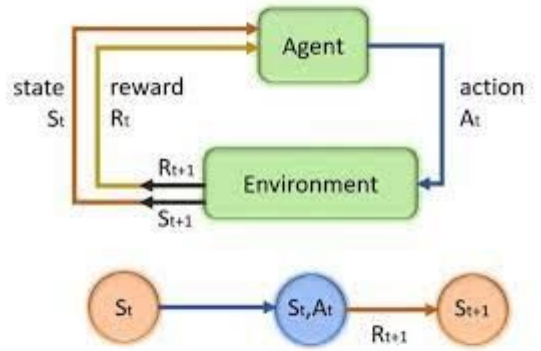


Method

The Reinforcement Learning tasks follows finite Markov Decision Process(MDP)

Some of the basic terms used in paper are:

- State space **S**: This is all the possible configuration possible in the game like the position of all the elements in game
- Action Space **A**: A set of all the actions allowed in the game. For eg: left, right, up etc
- Policy **π:S->a**: A mapping function from state to action space



The main goal here is to make the agent to take optimal actions to maximize the total reward or we can tell this as the total point earned after playing the game.

Q function:

$$Q(s_{t+1}|s_t, a) = r(s_t, a) + \mathrm{E}(\sum_{k=1}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}))$$

r is the reward for the state s_t with action

gamma is the discount rate
E is the epsilon greedy policy

The above equation defines the Q functions for s_{t+1}. We need to choose a value so it maximes the Q value for s_{t+1}.

The optimal pol

$$\hat{\pi}(s_t) = a_t = \arg \max_{a \in \mathcal{A}(s_t)} Q(s_{t+1} | s_t, a)$$

Since the calculation of finding the optimal Q is expensive, this is where the Neural Networks come into the place.

Loss function:

This function is used to find the error in the model or the Q Learning so that we can minimize the loss to make the model better.

$$L(\theta) = \frac{1}{T}\sum_{i=1}^{T}(r_i + \gamma \max_{a'_i} Q(s'_i, a'_i; \theta) - Q(s_i, a_i; \theta))^2$$

Q Learning

The Q Learning is based on the SARSA which is an algorithm for learning a Markov Decision Process policy where Q values are updated according to the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Approximate Q - Learning

The Approximate Q - Learning is nothing but Q Learning with an approximate function.

$$Q(s,a;w) = \sum_{i=1}^{n} f_i(s,a) w_i$$

The Q(s, a) is obtained from a linear combination of features fi(s, a) (including the bias term)

Here the agent learns the weights from the features extracted from the game starts and the bias term is 1. The weights are updates based on the below rule:

$$w_i \leftarrow w_i + \alpha \cdot (r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \cdot f_i(s,a)$$

Deep Q - Learning

Handcrafting features is not that efficient since there is high chances that we may miss out the important features for this reason. The Deep Q-Learning was introduced in this paper here. Convolutional Neural Network(CNN) is used to extract the features and outputs different Q values for all the possible actions in the state. Most the time the CNN has architecture where the first few layers is full of Conv2d layers and then fully connected layers.

In this project $Q(s, a) = Q(s, a; w)$ this is the approximate function, where w is the weights of the network and bias.

Result

The methods were implemented on top of UC Berkley’s CS188: Introduction to Artificial Intelligence course(CS 188: Introduction to Artificial Intelligence)

Q Learning:

For smallGrid the agent was able to give positive reward and 100% win rate for 100 games. But the agent takes a long time to train, seeing this the hypothesis was defined that the model was not able to learn for largeGrid which was true since the agent was able to give 77% for 1100 games.

For this we can come to a conclusion that the SARSA update is not scalable

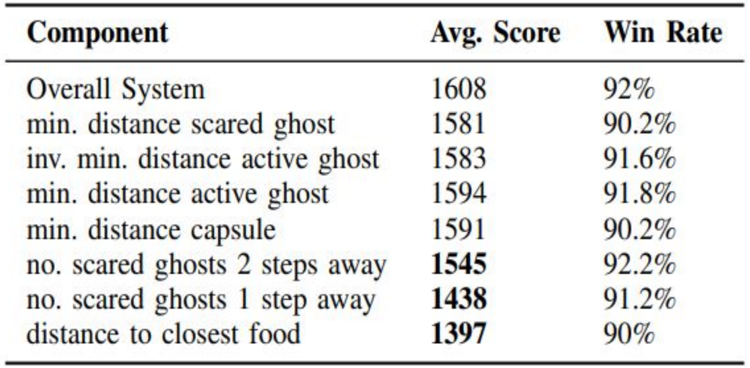
Approximate Q - Learning

The features were picked based on assumptions, based on the game logic and which is near based on distance between the ghost food etc.

The have performed the ablative analysis to select the important features in terms of the average score and the win rate of 500 training games and testing games

After removing a feature one at a time there was not much drop rate until the scared ghosts "1-step away" and "scared ghosts 2 steps away" features since eating the scared ghost helps boost the rewards significantly

Because of this, the distance from the agent to the food was important



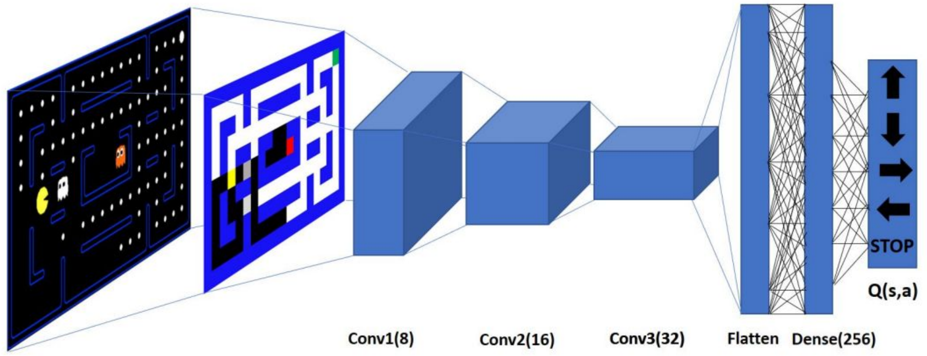
Component	**Avg. Score**	**Win Rate**
Overall System	1608	92%
min. distance scared ghost	1581	90.2%
inv. min. distance active ghost	1583	91.6%
min. distance active ghost	1594	91.8%
min. distance capsule	1591	90.2%
no. scared ghosts 2 steps away	**1545**	92.2%
no. scared ghosts 1 step away	**1438**	91.2%
distance to closest food	**1397**	90%

After the entire iteration based on the features, the analysis is shown above. The conclusion of the best features are the number of scared, active ghosts, one and two steps away, eating food if there are no active ghosts nearby, and the distance to the closest food.

The agent was trained on mediumClassic and mediumGrid with a very short training phase of 50 episodes. The agent was able to hit a win rate of

Deep Q Network:

Use of tensorflow to implement CNN. It is composed of three convolutional layers (3x3x8, 3x3x16 and 4x4x32), a flattening layer and a fully connected layer with 256 neurons. All the layers use a Relu activation function.

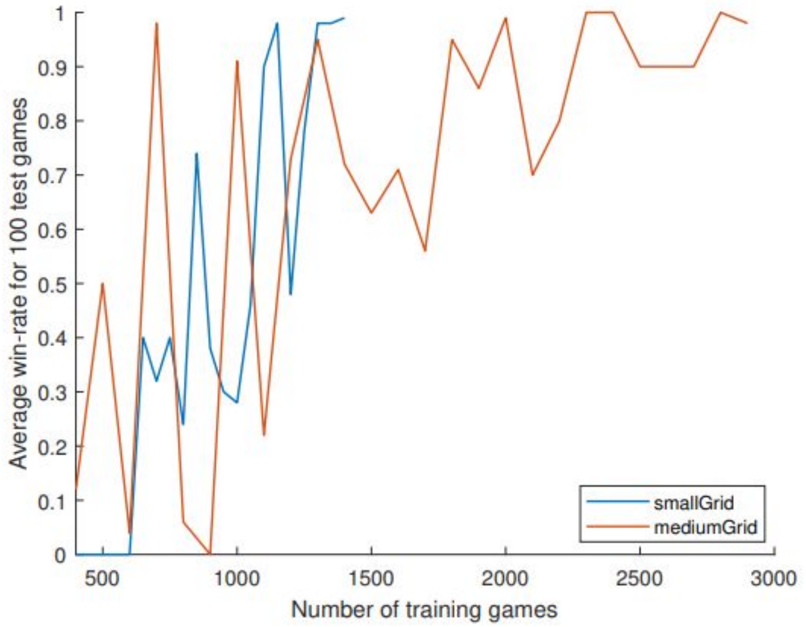


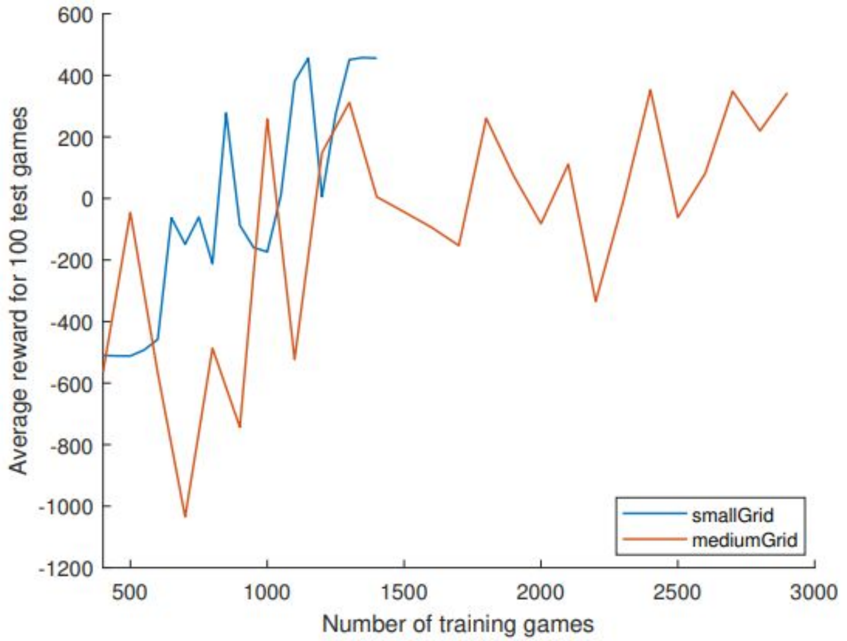
1) Create the equivalent images of frames with each pixel representing the object of pacman. This reduces the data by 100 without any loss.
2) Use replace memory which acts like a buffer to store the last 100000 experiences including the states, rewards and policies.
3) Use mini batch size as 32, this helps to avoid correlation.
4) Use an additional target network (Qˆ-Network) to generate the target values for Q-Network.

$$L = \frac{1}{T}\sum_{i=1}^{T}(r_i + \gamma \max_{a'_i} \hat{Q}(s'_i, a'_i) - Q(s_i, a_i))^2$$

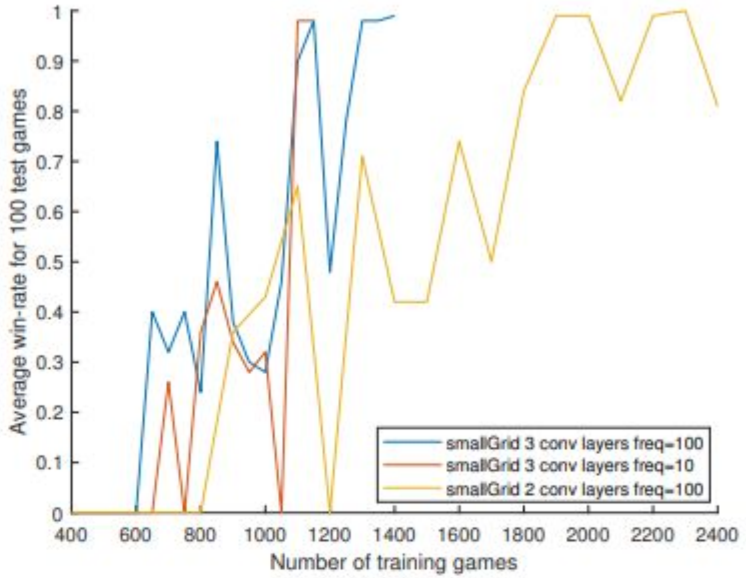
5) Used epsilon greedy strategy for exploration.
6) Learning rate is reduced from 0.0025 to 0.00005 as training goes and we train the DQN in every step of the game.

DQN Average win rate and score

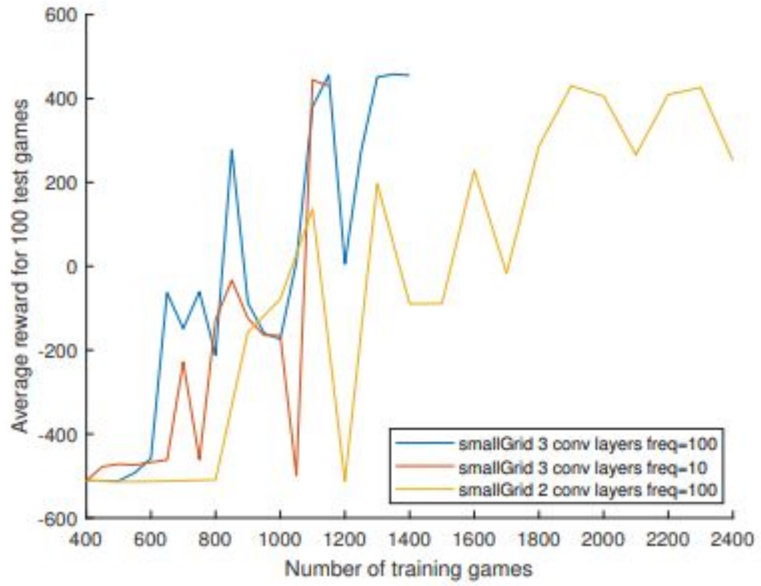




Average win rate and score for smallGrid



(a) Average win-rate for smallGrid



The DQN outperforms SARAS in smallGrid and mediumGrid but the down side of this method is that it takes a long time to train even on GPU.

The average reward remain negative even on 25000 games

Some of the other attempts:

How many layers? Most of the researchers use experience or twerk the hyper tuning until it works but here they followed Mhin et. al.paper.

The first method was to just give images in form of pixel but this was expensive so they chose equivalent images method and reduce their training time.

For replace memory the data structure used was numpy but later dequeue was used, which is faster.

The simple network used small regularization and gradient descent to update weights

Simple Neural Network was comparable to that of the approximate Q Learning

Conclusion

The Q Learning performed very badly

The approximate was way better than the SARAS update since there we were selecting features for both the smallGrid and the mediumGrid.

The approximate Q Learning had a chance of missing the important features so DQN was introduced which was effective and the cost was reasonable.

Future Work

The authors talks about what all they can do to increase the performance:

- Adding prioritized replay by choosing the samples from which the agent can learn the most
- Adding auxiliary rewards
- Adding a LSTM to our DQN in order to make the training more stable
- Implement double-DQN and compare performance with DQN to see if Q-values may be overestimated for Pacman game
- Train the agent on larger grids