REINFORCEMENT LEARNING FOR GAME PLAYING

Akshay Kumar (AXK180196) Deepanshu Sharma (DXS190018)
Abhinav Thirumoorthy(AXT190007) Theja Kunam(TXK190012)

INTRODUCTION

Artificial intelligence includes various machine learning algorithms which uses lot of computing power in order to make valid predictions based on a lot of data. Machine learning is basically a field of Artificial Intelligence where the machines that are usually computer programs learn and adapt to new and bulk data without the help of humans. There are generally three types of machine learning which includes Supervised, Unsupervised and Reinforcement Learning. Even though, most of the data scientists and engineers use supervised and unsupervised learning, Reinforcement learning has become popular these days and is found to be very efficient and powerful for complex problems. Reinforcement Learning is a type of training that is built on rewarding desired outcomes and punishing the undesired outcomes. The learner is not informed about the action that it is going to to take, but it must discover the actions that would yield the highest award and perform them. In this paper, we are going to look at how reinforcement learning is used and applied for game playing.

ABSTRACT

Reinforcement learning is a trial and error searching method that can be done using many algorithms but we are using DQL (Deep Q-learning) to train the Pacman agent to outrun the ghost and eat food and scared ghosts which would result in a higher score. The agent is trained in such a way that it looks for a long term as well as maximum reward possible and thus obtain an optimal solution. Every learning needs an efficient algorithm to work on and give the best result. Since DQL uses Q- network which is a convolution neural network (Deep Q-network) has a Q-function that works best with few games like Pacman and sometimes outperforms the humans as well. DQN's use convolutional layers and dense classifiers to detect the abstract features and map them to output layers respectively. Since the pacman game has lot of state spaces, it is impossible to use Q-learning since the amount of memory that is required as well as the total count of iterations for convergence cannot be done. To overcome the above mentioned problem, we use DQN which interpolates Q-values of state action pairs using a convolution network rather than a data table which was previously used in Q-learning.

BACKGROUND WORK

Reinforcement Learning (RL) is very much efficient and useful in cases where there are situations in which the probabilities and rewards are unknown. It is quite different from supervised learning because in supervised learning, patterns are learned according to the labeled examples. In Reinforcement learning, there is the communication between the algorithm and the environment which is a cycle that lets the system learn on its own.

Q-learning has been the most favoured algorithm used in RL and it uses a Q function Q_θ (s, a) and is defined as the expected discounted future reward in a state 's' given an action 'a'. The Q function returns the Q value of the respective state-action pair. The Q-value can be calculated with the function $Q_\theta(s, a)$

$Q_\theta(st, at) \leftarrow Q_\theta(st, at) + \alpha * (rt+1 + \gamma * \max a\ Q_{\theta^-}(st+1, a) - Q_\theta(st, at))$

where

$Q_\theta(st, at)$ → Q function

s→ state

α →learning rate

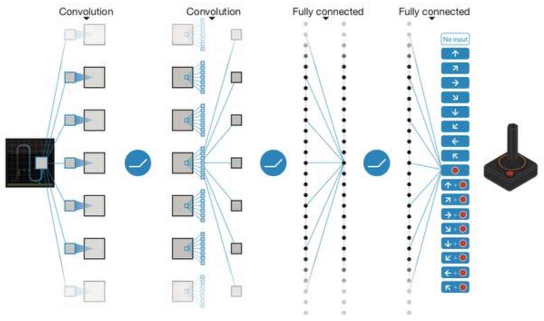
a→ action

rt+1→ reward with the current state transition.

Thus we know that Q-learning uses a data table that contains the Q values of all state-action pairs.

THEORETICAL AND CONCEPTUAL STUDY

In this project we implement DQN to the RL agent is trained to play the Pac-man game. Deep Q-Learning uses a neural network in place of a table. This makes it possible to apply reinforcement learning to complex domains with large state-action spaces which makes it more efficient than Q-learning. [1]



The DQNs make use of the screen with the help of CNNs which are a series of convolutional layers and they learn the important game information from the mess of pixel values. We process the raw input by downsizing from original size to a more manageable small grid size.

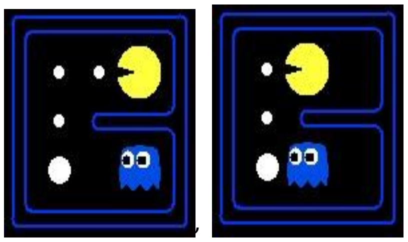
The DQN algorithm is done in python and it acts as the RL agent. The observations observed and noticed in each state and rewards are sent to the algorithm by the game which are designed and matched to the representation of a state. Information exchange regarding the reward, states and actions is performed between the environment and an agent in the usual RL backdrop. This exchange of information between the game and the algorithm is highly required for the implementation of Pac-man game. Since vast amount of iterations are needed, the usage of the DQN aids with the simulations on the Pac-man game which would make it run faster than real-time. We are using the Pac-man implementation code created by UC Berkeley which satisfies both the requirements and is suited for the RL backdrop [2]. The Pac-man implementation by UC Berkeley supports different maps and levels.

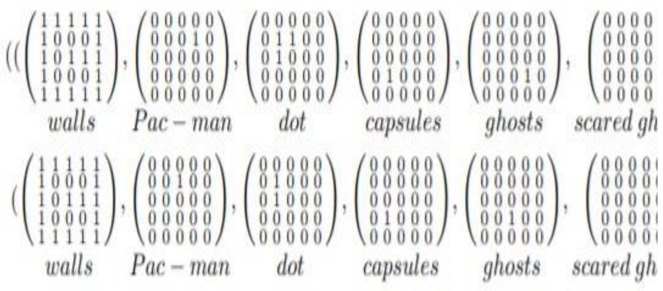
It is assumed that the learner follows a finite Markov Decision Process. Some of the important terms that are used are:

- State Space (S): It contains all the possible configurations present in the game that includes the point position, ghost position, player positions and so on.
- Action Space (A): It contains the set of all the possible allowed actions: move left, move right, move up, move down, stay put.
- Policy (π): S→A: Contains the mapping from the state space to the action space[4].

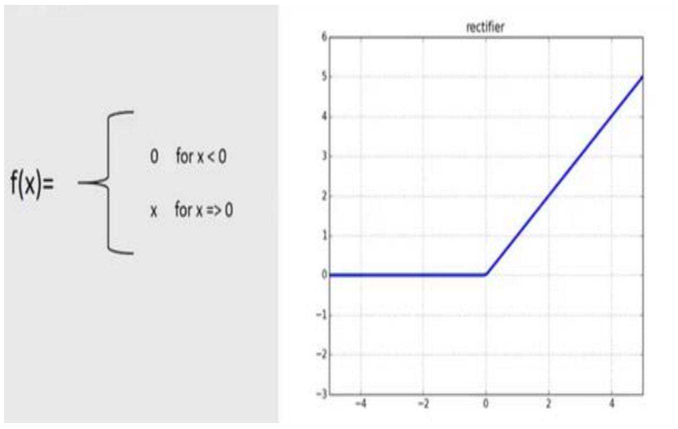
A tuple (s, a, r, s^1) is used during a state transition from s to s^1 of action 'a' and the attained reward 'r'. When the learning process is at the beginning, the Q-network gives back a random number. After many iterations, the algorithm changes the latest obtained Q value which results in the convergence to the expected discounted future reward. To calculate the Q-value of a certain state action pair (s, a), an intuitive approach is used to feed the (s, a) as an input to a neural network. By feeding the neural network of a state 's' and by defining the output of the network as a vector of Q functions for all possible actions from s, we can improve it. We generally use two neutral networks namely *target network and Q-network.* Back propagation method is used to update the Q-network and the update function during every iterations. Initially the Q-values are computed and is used by the target network which is eventually the previous copy of the Q-network.

The learning progress is improved by randomly sampling the experiences using Experience Replay. An experience tuple (s, a, r, s^1) is stored in deque by the algorithm at every iteration. Each game frame consists of a grid containing the main six elements comprising: Pac-man, dot, walls, capsules, scared-ghosts and ghosts. In each matrix, we represent it with 0 or 1 which basically defines whether the element exists or not on its corresponding matrix.





Small positive rewards are rewarded for swallowing dots and high positive rewards are rewarded when the scared ghosts are swallowed and the learner is winning. The difference in game score corresponds with the obtained reward by the RL agent. There are two convolutional layers and two fully connected layers in both the networks. The CNNs increase the classification of features in the game grid and they are very much valuable as the game states which are present in the game give the existence of local connection. Here, we use ReLu activation function with the CNNs which has proven to train the networks many times faster than any other activation functions. It is defined as f(x) = max (0, x).



PARAMETER ANALYSIS

Reinforcement Learning is very much sensitive to optimization methods. Changing the learning rate alone may not be good and dynamic enough to handle all the input changes during the part of training. There are various that are to be tuned or changed during the course of analysing and in the process of obtaining results. Some of the parameters that could be tuned are learning rate, epsilon, reward values and discount factor.

The *discount factor* regulates the importance level of the future rewards and is usually a number between 0 and 1 and it has the ability to have an effect of valuing rewards that are received earlier than the rewards that are received later. The probability to succeed and get the maximum accuracy can be done mostly when the discount factor is as much as close to a value of 1. The *learning rate* or step size helps us to find to what extent the newly obtained information has overridden the old existing information. The *epsilon* gets a value based on whether the learner exploits or explores. Here, the agent interacts with environment by using the q-table and tries to look at all the possible actions for a given state. After that, the agent chooses its own action based on the maximum value obtained, it is called as exploiting. The other way is called as exploring when the learner has to take an action in a random manner which allows the agent to explore and discover new states that may not be found and explored in the exploitation process. By using epsilon (ε), we can manage both the exploit vs explore by setting the value accordingly. Also, the reward values can be changed accordingly to get the maximum accuracy.

RESULT AND ANALYSIS

In order to make this project viable, we need to find optimum values to the parameters. This is where we make use of parameter tuning. We varied the values of layers of DQN, Discount factor, learning rate and epsilon to find the best set of parameters. Below is the summary of observations made as part of parameter tuning.

Layers of DQN: What is the desired number of layers to be used? This question is not answered. Researchers choose parameters based on experience or by tweaking around till it fits the purpose. We worked with different number of layers to find out the best combination. By performing analysis on a layout and changing the number of layers, it is found that having 5 layers improves performance than having 3 convolutional layers.

CNN LAYER SIZE	ACCURACY
CNN layer 1 size= 5 CNN layer 2 size = 3	0.8
CNN layer 1 size= 5 CNN layer 2 size = 5	0.90

Simple Neural Network: Before choosing CNN as function approximators, we used neural networks. It is found that convolutional neural networks are better function approximators in this case given the large number of parameters.

Discount Factor Tuning: Keeping all the other parameters constants, we changed values of discount factor to understand its impact on accuracy. As per theory, discount factor means giving importance to future rewards. As the discount factor increases, the accuracy is seen to improve. This shows that the system focuses on getting rewards in longer run than immediate rewards.

DISCOUNT FACTOR	ACCURACY
0.85	0.85
0.95	0.90
0.99	0.95

Learning Rate: Keeping the other parameters, we tuned various values for learning rate. As learning rate increases, the accuracy is observed to increase. This parameter gives the system information about how much prior information is to be considered or if recently gained information is more valuable.

LEARNING RATE	ACCURACY
0.0002	0.20
0.0004	0.60
0.0009	0.65

Epsilon: This parameter allows the system to balance between exploration and exploitation. As we set different values to epsilon keeping other parameters constant, we observed that for smaller epsilon values the accuracy is higher than for higher epsilon values.

EPSILON FINAL	ACCURACY
0.1	0.95
0.2	0.70
0.5	0.05

CONCLUSION

From this project, we arrive at a few important conclusions. For smaller number of training iterations, the system will not be able to learn and hence the testing accuracy is 0. As the number of training iterations grows exponentially to 5000, the system gives better rewards and win rate. For complex layouts like mediumClassic, it takes longer to train and test. This is because of larger state space. In this project we experiment on smallGrid layout. Also, it is worthwhile to note that convolutional neural networks act as best function approximators for Q Learning. The value of discount factor lies between 0 and 1, the ideal value is closer to 1. A higher value is preferred for learning rate while a lower epsilon rate is preferred in this scenario.

FUTURE WORK

The Pac-man game is implemented in a small grid here and in the future it could be expanded and implemented for various grid sizes. Various new obstacles can be added and the type of rewards could be changed. Also the game implementation can be done using policy gradient which would possibly increase the training accuracy. Adding more obstacles, increasing the speed of the ghost or the Pac-man and changing the reward values can make the game more elaborate and more interesting. Since the project makes use of reinforcement learning to avoid obstacles and earn a reward by following a correct path, this concept can be extended to self-driving cars which makes use of the same idea. It can also be extended to automatic parking in which an AI learns to park using proximity sensors.

REFERENCES

1. https://towardsdatascience.com/advanced-dqns-playing-pac-man-with-deep-reinforcement-learning-3ffbd99e0814
2. http://ai.berkeley.edu/project_overview.html
3. D. Silver, RL Course by David Silver, https://www.youtube.com/watch?v=2pWv7GOvuf0& list=PL5X3mDkKaJrL42i_jhE4N-p6E2Ol62Ofa
4. http://cs229.stanford.edu/proj2017/final-reports/5241109.pdf