Introduction to AI & IOT

**Reinforcement learning for robot control**

Deep Reinforcement learning in MsPacman

horizontal line

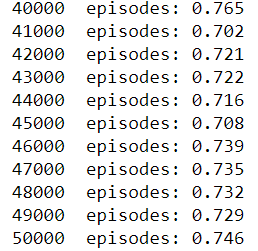
# 

# INTRODUCTION

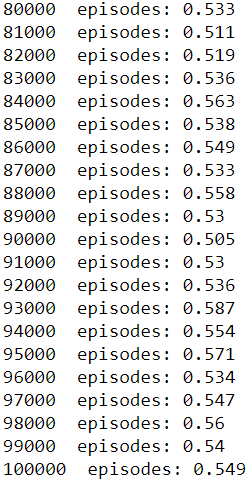
In 2013, DeepMind published the first version of its Deep Q-Network (DQN), a computer program capable of human-level performance on a number of classic Atari 2600 games. Just like a human, the algorithm played based on its vision of the screen.DQN, and similar algorithms like AlphaGo and TRPO, fall under the category of *reinforcement learning* (RL), a subset of machine learning.In reinforcement learning,the agent gets a random state from an environment and the agent takes an appropriate action that could maximise some kind of reward.Then,it gets a next state and settles on another action until the environment terminates.Hence,our aim is to train our agent in a completely unknown environment from scratch.We would apply basic Q-learning with epsilon greedy algorithm to train our agent in [FrozenLake](https://gym.openai.com/envs/FrozenLake-v0) environment and then,move on to deep Q-learning along with epsilon-greedy algorithm to train our agent in an Atari game [MsPacman](https://gym.openai.com/envs/MsPacman-v0/).

**SUMMARY**

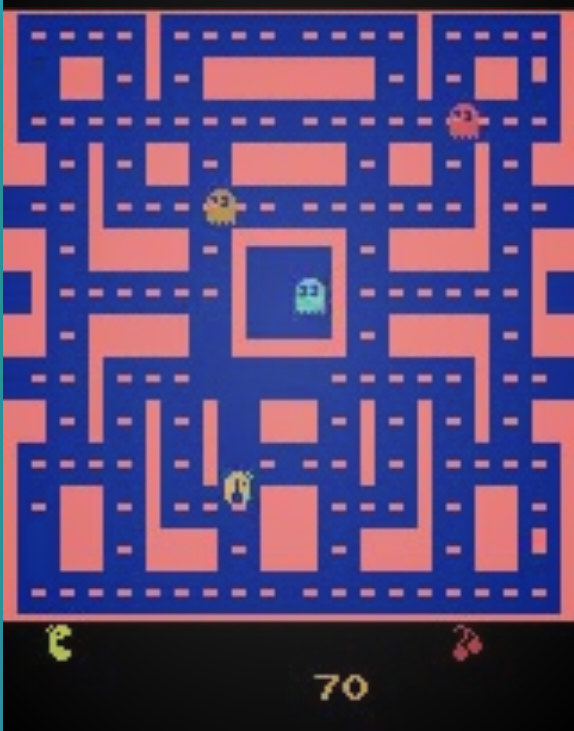
We implemented Q-learning with epsilon-greedy algorithm to train our agent to play in FrozenLake environment which had 16 states and 4 actions.After training,our agent was able to reach the goal 72% of the time but didn’t improved further(after training for 50000 episodes).



Henceforth,we tried increasing the states from 16 to 64,that is,changing environment to [FrozenLake8x8](https://gym.openai.com/envs/FrozenLake8x8-v0).Now,our agent took very long to train due to computational ineffieciency and was able to reach the goal about 54% on average.

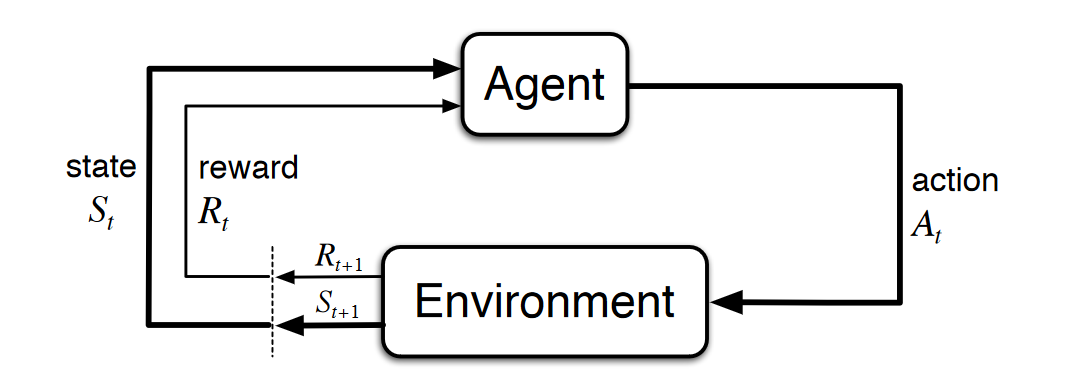


To improve performance of our agent and cater with the increased number of states,we resorted to what [DeepMind](https://deepmind.com/) did in 2013 of using a function approximator to get our optimal Q-value function.Deep Neural networks are used as function approximator which when blended with Q-learning,results in Deep Q-learning.We have implemented this on MsPacman environment.



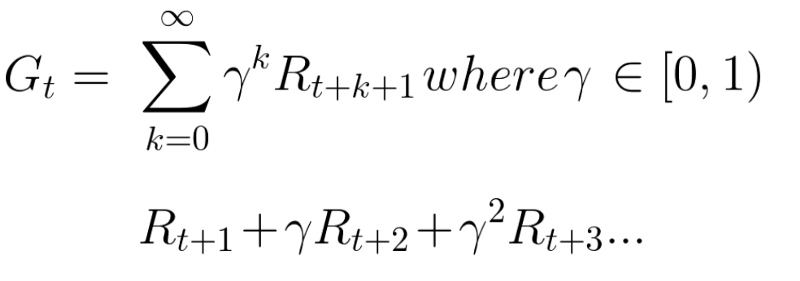
### Mathematical Insight

The correspondence between action and reward indicates markov property and the decision making process based on it is called Markov decision process.



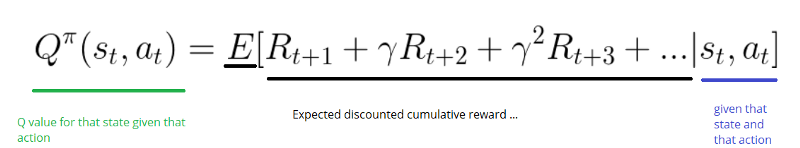
A ‘policy’ decides what action a(t) to take given a state s(t).

Objective:To find an optimal policy that could maximise cumulative discounted reward shown below:-

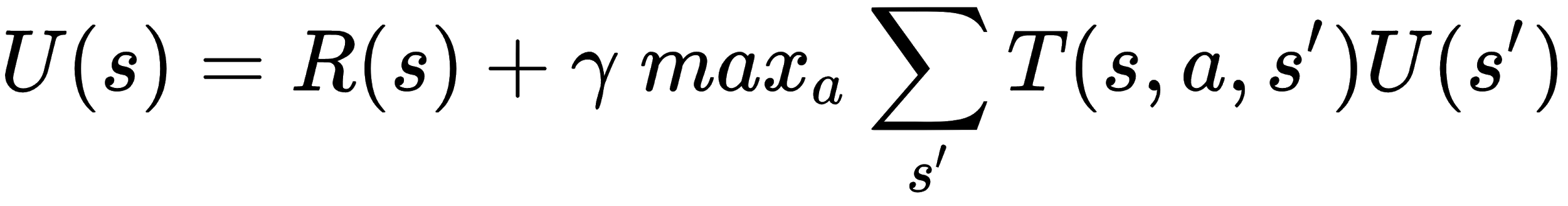


Following a policy produces sample trajectories(or paths):S0,A0,R0,S1,A1,R1,S2,A2,R2,.......

**Q-value function:-**



**Bellman equation:-**



Firstly,we try to iteratively converge Q-value for a (state,action) pair to optimal Q-value derived from Bellman equation.Also,we follow an epsilon-greedy algorithm which sets a trade-off between exploration and exploitation.As epsilon decreases,agent tends to exploit more than explore the environment(epsilon=1 indicates 100% exploration).

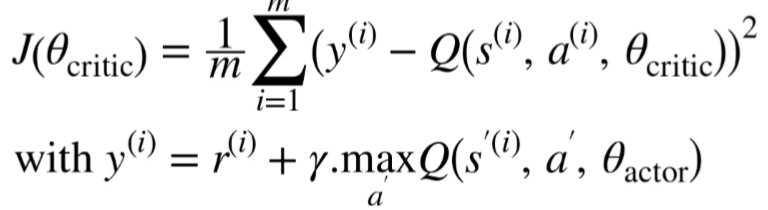
**Drawback:**The iterative Q algorithm or tabular Q-learning is computationally infeasible for the entire state space when states are represented in the form of images.The main problem with Q-Learning is that it does not scale well to large (or even medium) MDPs with many states and actions. Consider trying to use Q-Learning to train an agent to play Ms. Pac-Man.The states equates to 2^250 and therefore,there’s no way to keep track of every Q-value.

**Solution:**Use function approximator to get an optimal Q-value.

**Deep Q-learning**

Q-learning using Deep neural network as function approximator.

The training algorithm we will use requires two DQNs with the same architecture (but different parameters): one will be used to drive Ms. PacMan during training *(the actor*), and the other will watch the actor and learn from its trials and errors (*the critic*). At regular intervals we will copy the critic to the actor.



*Where,*

J[Theta(critic)]:cost function for critic DQN

[s(i),a(i),r(i),s’(i)]:[state,action,reward,next state] of ith memory sampled from experience replay.

M:no. of batches sampled from experience replay

Theta(critic),Theta(actor):parameters of critic DQN and actor DQN

gamma:Discount factor

**Experience replay and replay memory**

With deep Q-networks, we often utilize this technique called *experience replay* during training. With experience replay, we store the agent’s experiences at each time step in a data set called the *replay memory*. We represent the agent’s experience at time t as e(t).

e(t)=(s(t),a(t),r(t+1),s(t+1))

The critic DQN will try to make its Q-Value predictions match the Q-Values estimated by the actor through its experience of the game. Specifically, we will let the actor play for a while, storing all its experiences in a replay memory.Then,we will batch out experiences from replay memory randomly and train critic DQN to converge to Q-values given out with the help of actor DQN.

**CONCLUSION**

Applying deep Q-networks(*the actor and the critic)*  helped us cater with the large number of states and improved the performance of our agent to the extent of achieving superhuman actions.Though the computational time increased by a considerable amount yet the performance of our agent had a remarkable improvement.