Deep Q-Learning

Recap MDPs

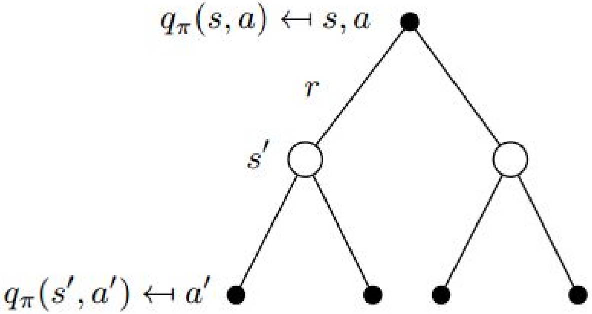
MDP:

- A MDP is a Markov reward process with decisions. It is an environment in which all states are Markov.
- A Markov Decision Process is a tuple <S, A,P, R, γ>
- A policy π is a distribution over actions given states: $\pi(a|s) = P[A_t = a \mid S_t = s]$

Action - Value Function(Q(s,a)):

- The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π
- $q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$
- The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

Bellman Optimality Equation

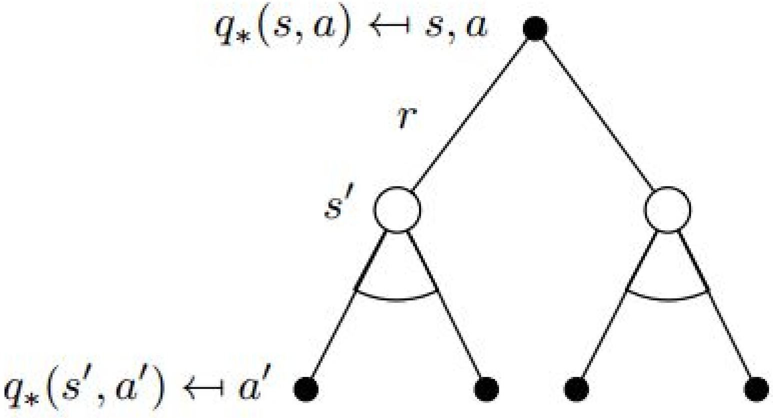


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum \mathcal{P}_{ss'}^a \sum \pi(a'|s') q_\pi(s',a')$$

An optimal policy can be found by maximising over $q_*(s,a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

Bellman Optimality Equation



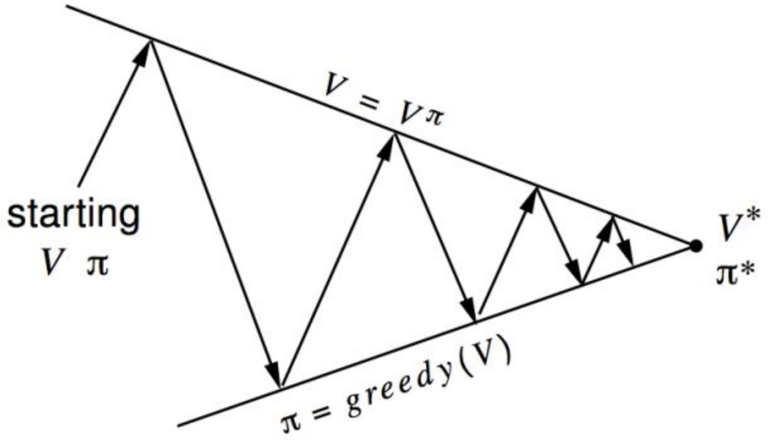
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

Dynamic Programming Approach

- Bellman equation gives recursive decomposition
- Value function stores and reuses solutions

Policy Evaluation:

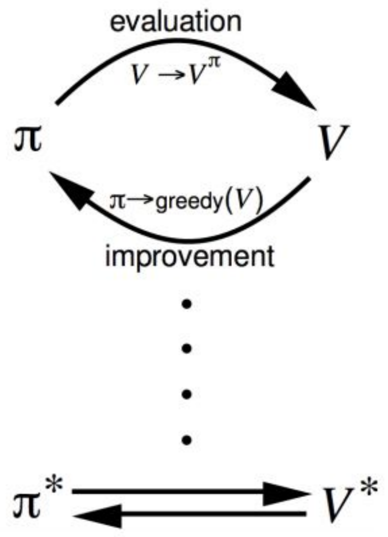
- Estimate qπ(s,a) given a policy using bellman equation
- Start with any random policy



Dynamic Programming Approach

Policy Iteration:

- Generate π >= π’ by acting greedily according to the bellman optimality equation.
- Guaranteed to converge to the optimal Policy (Proof by Contraction-Mapping)



Cons

Cannot work with Continuous variables,since the state space is large.

Difficult to work when MDPs are not specified,which is in most cases

Model based and Model Free approach

Model-Based:

- Explore environment & learn model, $T=P(s'|s,a)$ and $R(s,a)$ everywhere.
- Use policy-evaluation and policy-iteration on the MDP learnt.
- Not feasible in Large state spaces

Model-Free:

- Rather than learning a model for the environment learn actual state value or action value functions

Q Learning

- Utility-Sum of discounted rewards in the future
- For all q-states, s,a Compute Qi+1(s,a) from Qi by Bellman backup at s,a. Until max s,a |Qi+1(s,a) – Qi (s,a)| < ϵ
- Bellman Equation

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V_i(s')\right]$$

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

Q Learning with Tables

- The reward on entering the goal state is +1,and -1 on entering a hole.
- Episode is terminated on entering either of the two states
- Reward is zero in intermediate states

```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

Exploration vs Exploitation

- To decide upon what action is to be performed, a set probability value(epsilon) is used
- A randomly generated number is used to determine whether the agent will take random action(Explore) or take the best greedy action(exploit)
- If the random number is above the probability threshold, the optimal action yielding the highest q-value is selected (exploitation).
- Otherwise, a random action is selected (exploration)

Exponential Moving Average

- Makes recent samples more important

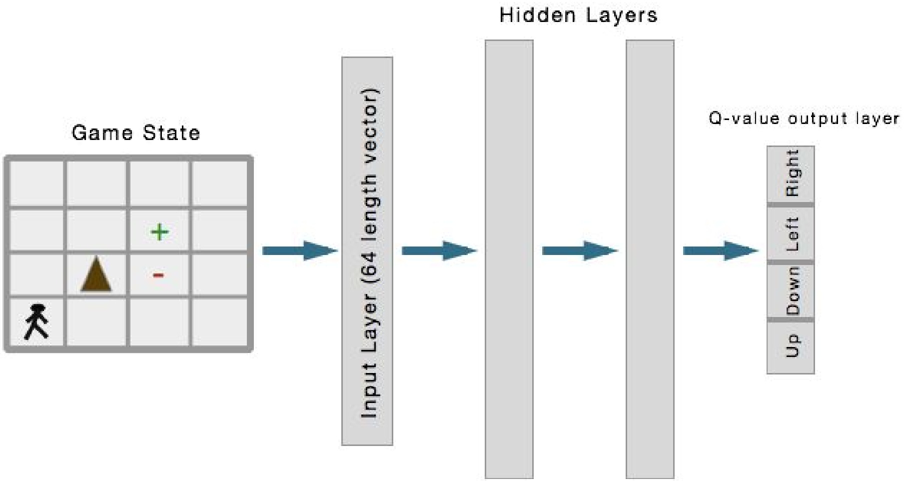
$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

- Forgets about the past (distant past values were wrong anyway)
- Easy to compute from the running average
- Here α is the learning rate

$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

A Model Free Approach : Deep Q Learning

- Approximate Q function using a neural network f(x,ϴ) where ϴ are learnable parameters of a neural network,x is input.
- The input to the neural net is the current state of the environment.
- Produce 4 Q-values for each of the action and take the maximum value among them



Loss Function and Backpropagation

- Do a feedforward pass for the current state s to get predicted Q-values for all actions.
- Do a feedforward pass for the next state s' and calculate maximum overall network outputs max a' Q(s', a')
- Set Q-value target for action to r + γmax a' Q(s', a')
- Update the weights using backpropagation

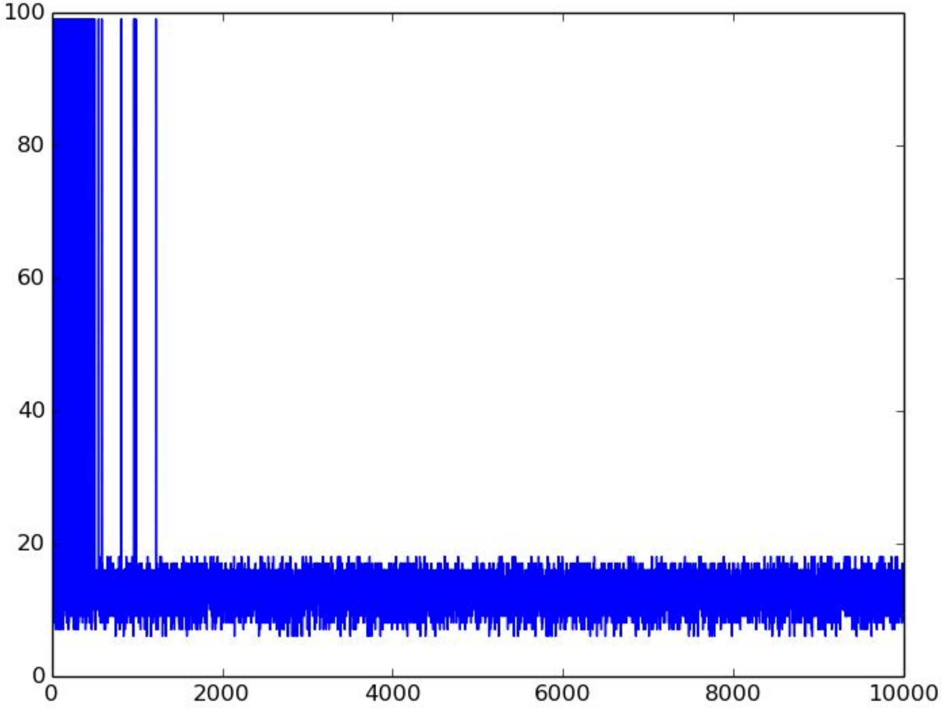
$$L = \frac{1}{2}[\underbrace{r + max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Replay Memory

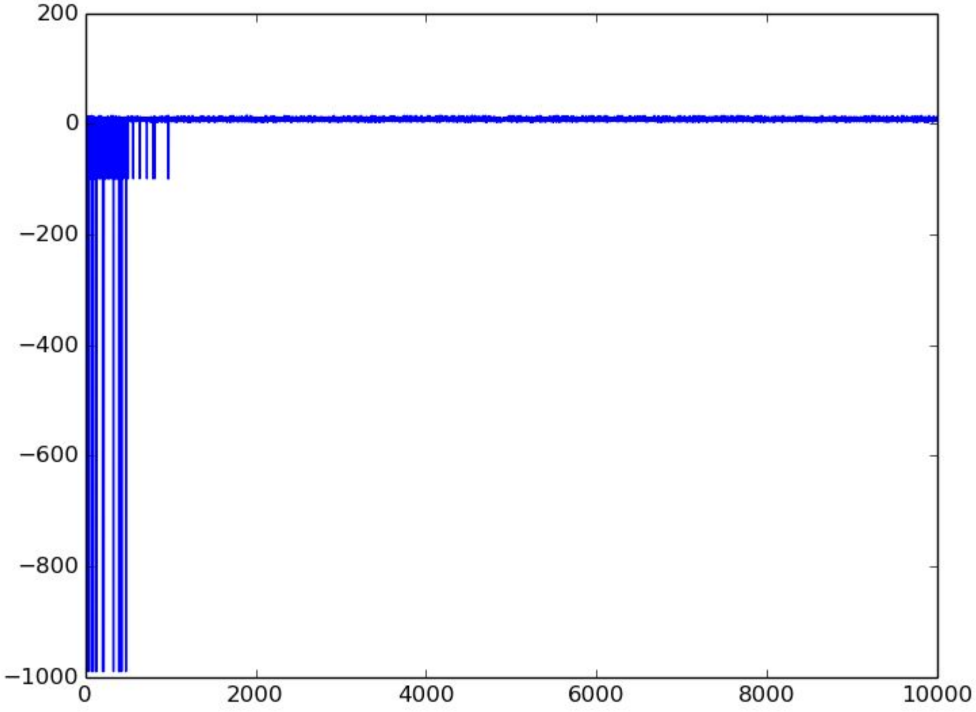
- Approximation of Q-values using non-linear functions is not very stable
- So,during gameplay all the experiences < s, a, r, s’ > are stored into a replay memory
- When training the network, random mini batches from the replay memory are used instead of the most recent transition
- Breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum
- Makes training task similar to Supervised Learning.

Implementation On Taxi-V2 and FrozenLake-V0

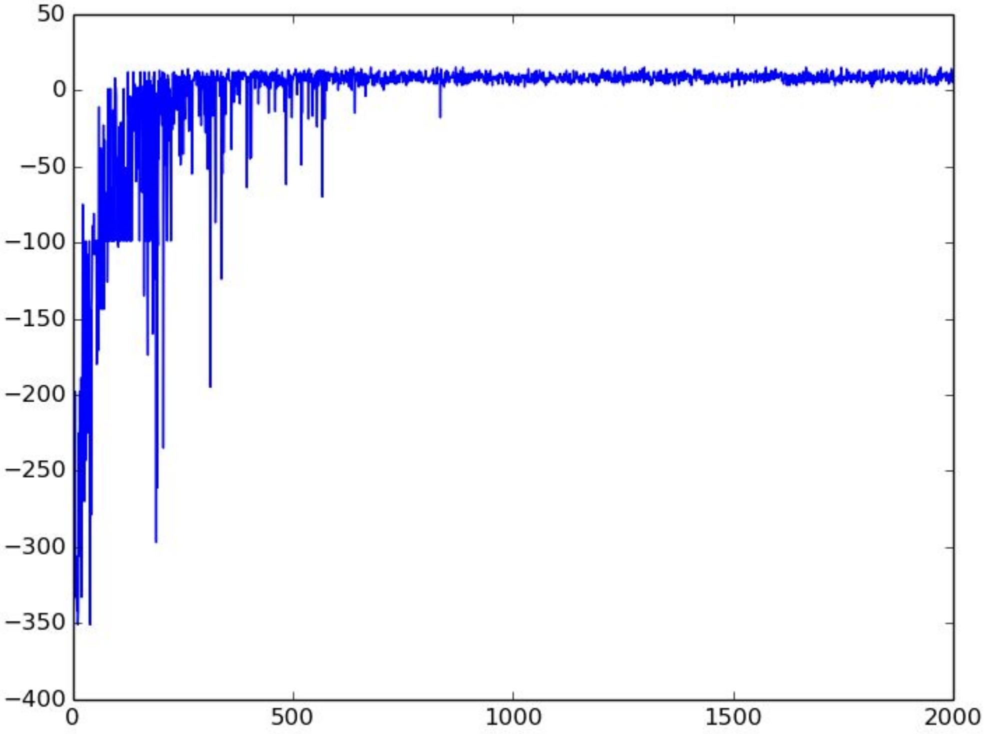
Steps Per Episode on Taxi-V2



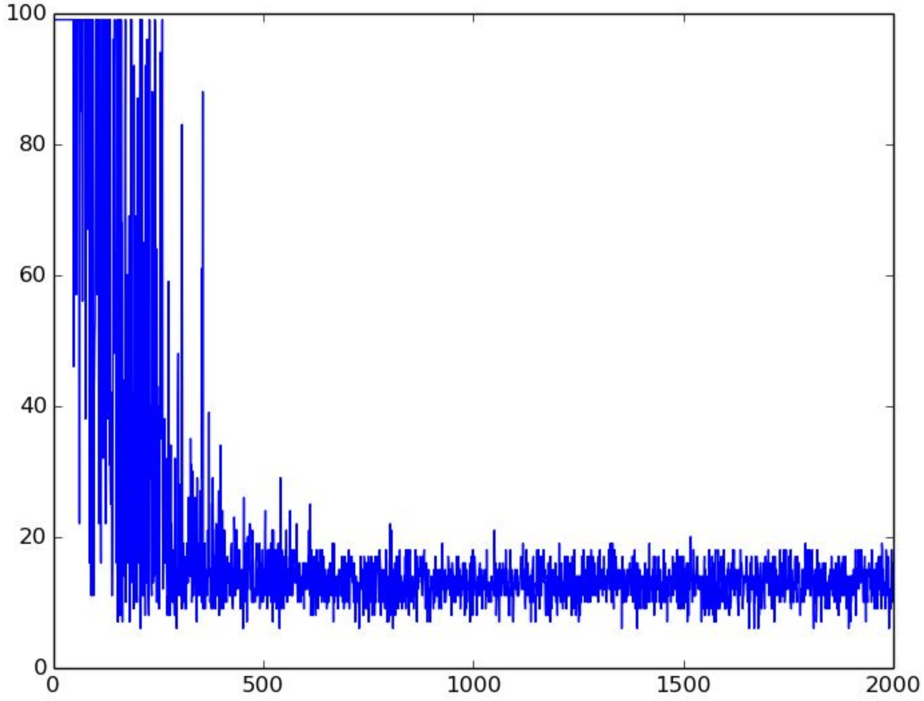
Rewards Per Episode on Taxi-V2



Reward Per Episode on FrozenLake-V0



Steps Per Episode on FrozenLake-V0



Future Plans

Self driving car simulator:

- Planning to train an agent to play in a conti-nuous environment by looking at subseque-nt image frames and taking actions
- Implement this with the help of CNN's