Documentation for Deep Q Network

CartPole Balancing

- Combines Q learning with deep neural networks
- Based on Markov Decision Process (MDP)
  - State space (S)
  - Action space (A)
  - Transition function $P(s'|s,a)$
  - Reward function $R(s, a, s')$
  - Discount factor γ

Q value quantifies the expected long term return of taking specific action in specific state. They predict a total reward for taking action 'a' in state 's'. Higher is the value, better is the action. The policy is derived by choosing the action with highest Q value.

In brief difference between Tabular Q learning and Deep Q network -

1) Tabular Q learning -

- Uses a matrix to store Q values
- Each state action pair (s,a) has its own entry
- Q values are stored and updated individually

eg -> A room with action of (0) -> heating off and (1) -> heating on
There are three states - cold, comfortable and hot.

Q -Table ->

$$
\begin{array}{r c}
 & \begin{array}{cc} (0) & (1) \end{array} \leftarrow \text{actions} \\
\begin{array}{r} \text{cold} \\ \text{comfortable} \\ \text{hot} \end{array} & \begin{bmatrix} 0.1 & 0.9 \\ 0.7 & 0.8 \\ 0.5 & 0.2 \end{bmatrix}
\end{array}
$$

Tabular Q learning expects discrete state spaces and suffers with computational inefficiency as state space grows exponentially with dimensions.

eg -> Atari games have 10^{60} possible states. Makes problem impossible to solve.

The update happens for single table entries -

eg => For above example -

$R(\text{comfortable} \mid \text{cold}, a=1) = 1$ ... known / given. When heating is 'on' and we move from cold to comfortable state, we get Reward of 1.

$$\hat{Q}(\text{comfortable}) = R(\text{comfort} \mid \text{cold}, a=1) + \gamma \max Q_k(\text{comfortable} \mid a=1)$$

$$= 1 + 0.9 \max(0.7, 0.8)$$ → discount factor

$$= 1 + 0.9(0.8)$$

$$= 1 + 0.72 = 1.72$$

/ /

The impact of new learning is incorporated by running average,

$$Q_{K+1}(s,a) = (1-\alpha)Q_K(s,a) + \alpha\hat{Q}(s')$$

$$= (1-\alpha)Q_K(\text{cold}, a=1) + \alpha(1.72)$$

$$= 0.9(0.9) + 0.1(1.72)$$

$$= 0.81 + 0.172 = 0.982$$

α = learning rate $= 0.1$

- Tabular Q learning gauantees to converge to optimal Q^*. Requires all state-action pairs to be visited. Hence, convergence can be slow.

2) Deep Q Networks (DQN) -

- Uses neural network to approximate Q values
- The network takes state as input and outputs the Q value for all action
- Basically, learns a function $Q(s,a,\theta)$ where θ are neural network parameters.

The greatest advantage is -
- can handle continuous and discrete state spaces
- scaling is possible
- memory usage depends on network complexity

The network parameters are handled via backpropogation. Can divege or become unstable without proper techniques like target networks, expirience replay.

The code and implementation is explained by comments directly in code.

Expirence Replay and Target Network understanding -

1) Expirence Replay - This is used to avoid the phenamenon called as catastrophic forgetting.

The core problem of neural networks and hence it percolates to decision making is that the previously learned (state, action) pair is updated. Hence, the successes and mistakes are lost and this can lead to very slow convergence or no learning.
The neural networks use stochastic gradient to update the weights and hence newer expinences may tend to dominate.

2) Target Network - Two seperate networks are used. One is the policy network which updates after every step and on other hand there is target network which updates slowly: after some steps. This target network is used to estimate future rewards. The target network provides stable targets for learning.