

Deep Q Learning

start
time:

Before you start, share this document with your team member(s) and then complete the form below to assign the role of speaker.

Team Role	Team Member
Speaker: shares your team's ideas with the class.	Addy, Arogya, Kelii

A. Review of Q Learning

start
time:

In classical Q-learning, we update the Q-value for state s and action a using:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

1. What is Q table?

Dictionary of state, action pair to Q Value. Q Value is estimated utility collected after taking action a on a state s until it is terminated.

$q_value = \{s:a\}$

2. How do we use the Q table to select action for a given state s ?

- We select the highest Q value for each state, action pair

3. What does each variable in the update rule/formula above represent?

alpha -> Learning rate

R -> reward

Gamma -> discounting rate

s -> state | a -> action

s' -> next state | a' -> next action

4. Why do we take the maximum over the next state's actions in the formula?

- Next best move AT the next state (s')

5. What is alpha in the formula? How does the alpha work to control the update of $Q(s,a)$?

- learning rate. How much it learns from new state or keeps previous state.
 - where 0: learn nothing
 - where 1: learn everything

- Do we update the Q table right after each move of the agent?
 - Yes; for every move we update the table

- What are the limitations of Q-learning when the state space is large or continuous?

- Computationally expensive, takes some time
- Not scalable
 - One solution: apprx_q learning
 - HOWEVER; drawbacks of apprx_q occur when features cannot be grouped or too many unique features exist

B. From Q-table to Neural Network	start time:
--	----------------

Instead of storing $Q(s,a)$ in a table, we use a neural network $Q(s,a;\theta)$ to predict the Q-value, where θ is the weight and bias of the network.

- Why can't we use a Q-table for high-dimensional states (like images)?

- Because there are too many pixels in image. Unless we zoom in very much to limit the number of pixels, it will be very difficult to store even 1 image data in q table.

- What is the input and output of the Q-network?

- Input: 1 or 2 (position: Index or (x,y)) | number of variables to describe the current state
- Output: possible actions**

- How does this structure enable the agent to **generalize** across states, without using a table?

- Dimensionality reduction
- Handles new data better
 - highly dependent on the training data used(model fit)

- How to choose the best action given s using the network?

The output layer(4) neuron that has the highest value

- How to retrieve the q value $Q(s,a)$ using the network, given s and a?

- Each output neuron represents a $Q(s, a)$ with a **q-value**

13. Do we always choose the best action for the states using the network during the training?

- a. No, sometimes we want to balance it with exploration
- b. Epsilon greedy, balancing between explore and exploit(epsilon decay)

C. Training Neural Network	start time:
-----------------------------------	-------------

DQN includes two major ideas:

- **Experience Replay Memory:** Store past experiences (s,a,r,s') and train the network on random batches.
- **Two Networks:** Use a second network - **target network** $Q'(s,a;\theta')$ - to compute the target values.

14. Why do we use a replay buffer instead of training on the most recent transition?

If the next move is vastly different then the training will not be smooth. Using the number of transitions in replay buffer means that the next transition will be smoother.

- Prevent overfitting
- Leveling exploit vs explore
 - stable model

15. What should be put in the buffer?

- Past experiences and moves, (example: previous 1000 moves)
- s, a, r, s'

16. Is the most recent move used for the next round of training?

- We should use random moves from the buffer for the next round of training, not the most recent move.

17. Describe how a new estimate at s and a: $y=r+\gamma \max Q'(s',a')$ is computed using the network.

- That is the new estimation of $Q(s,a)$ using the network
- We call the network twice because we need fro $Q(s,a)$ and also for $Q(s',a')$
 - is q_value at current state == $target_q_value$?
 - no: $new_estimate = q_value(current) - q_value(new)$
 - yes: Test

18. How to calculate the loss between current $Q(s,a)$ and the new estimate y ?

- $y = reward + gamma * (\max(Q(s',a')) - Q(s,a))$
- difference between $Q(s,a)$ & $y \rightarrow target_q_value$?

19. Should we use the same network to get the current $Q(s,a)$ and the new estimate y ?

- No
 - If we do there is a lack of generalization: overfitting to one particular scenario

20. Why does using a separate target network help training?

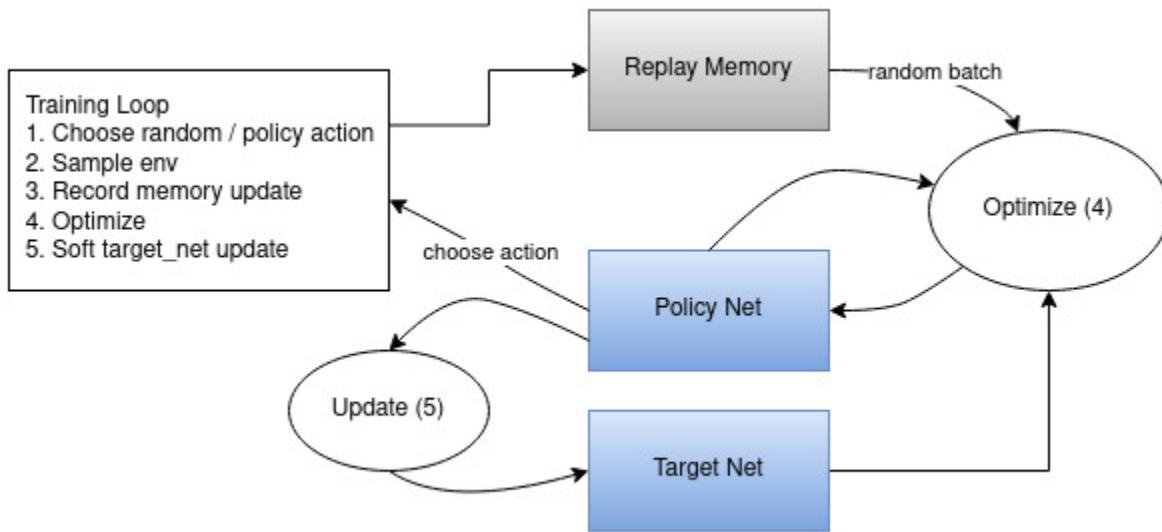
- a. One network to train -> current S
- b. One network as a target network -> Next S
- c. If we use the same network that is training and updating, then it will be unstable

21. When and how is the target network updated?

- a. In every certain interval, we copy the policy network to the target network. We don't update it regularly, but in certain intervals.
- b. Another method is using tau to update the new network. If tau is large, we use more from policy, if tau is small, we use less from policy.

D. DQN Framework

start time:



22. Use the flowchart above to describe each step of DQN.

1) Initialization

- Randomize network weights
 - Policy
 - Target
- Set size for Buffer

2) Interact with Environment

- Play the game
 - Call Epsilon Greedy to make first move
 - Save (r, s') into the buffer

3) Sample Mini-batch

- Grab the random 64 past experiences (s, a, r, s') from buffer

4) Compute Target

- $Q(s,a) = R + \text{gamma} * (\max Q(s',a'))$
- Use networks to calculate $Q(s, a)$ and $Q(s', a')$

5) Update Q-network (Policy Net)

- Call Policy_net to calculate loss
-

6) Update Target Network

- After 5 updates to the Policy Network, copy the Policy to the Target Network