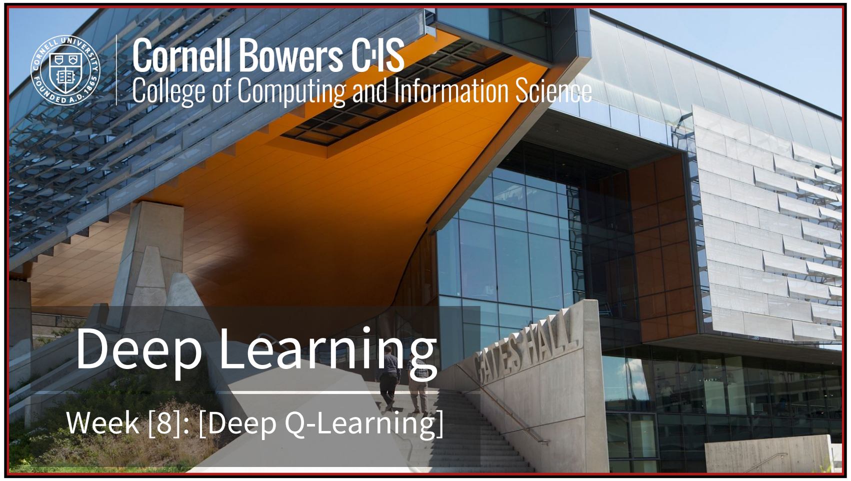


Cornell Bowers C·IS
College of Computing and Information Science

Deep Learning

Week [8]: [Deep Q-Learning]

Cornell Bowers C·IS

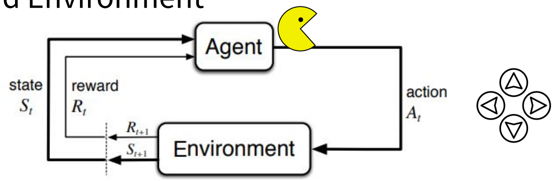
Logistics

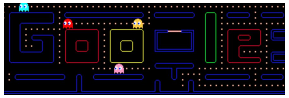
- Will release midterm grades by the end of the week
- Releasing a sign-up sheet for final presentation slots later today
  - First day of presentations is April 30th
- Announcement will include instructions for the final presentation

---

Cornell Bowers C·IS

Agent and Environment





Agent:	Environment:
- Perceives environment. - Makes decisions. - Aims to maximize reward.	- The external context in which an agent operates and interacts with - Provides feedback to agent

---

Cornell Bowers C·IS

Markov Decision Process (MDP)

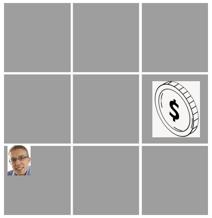
- MDPs provide a framework for modeling sequential decision-making problems.
- An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:
  - $\mathcal{S}$: Set of states representing the environment.
  - $\mathcal{A}$: Set of actions the agent can take.
  - $\mathcal{P}$: Transition probability function, $\mathcal{P}(s'|s, a)$.
  - $\mathcal{R}$: Reward function, $\mathcal{R}(s, a)$.
  - γ: Discount factor, $\gamma \in [0, 1]$.

Q-Table

- In Q-Learning, the Q-function is typically represented using a Q-table.
  - A table that stores the estimated Q-values for state-action pairs.
- Q-table has dimensions $|\mathcal{S}| \times |\mathcal{A}|$, where:
  - $|\mathcal{S}|$ is the number of states in the state space.
  - $|\mathcal{A}|$ is the number of actions in the action space.
- The Q-table is initialized with arbitrary values and iteratively updated based on the agent's experiences during the learning process.

Q-Table Example

- 9 states and 4 actions
- Initialize valid (s,a) tuples to 0s



	Up	Down	Right	Left
Bottom Left	0	-	0	-
Bottom Middle	0	-	0	0
Bottom Right	0	-	-	0
Mid Left	0	0	0	-
Mid Middle	0	0	0	0
Mid Right	0	0	-	0
Top Left	-	0	0	-
Top Middle	-	0	0	0
Top Right	-	0	-	0

Q-Learning Update Rule

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a')$$

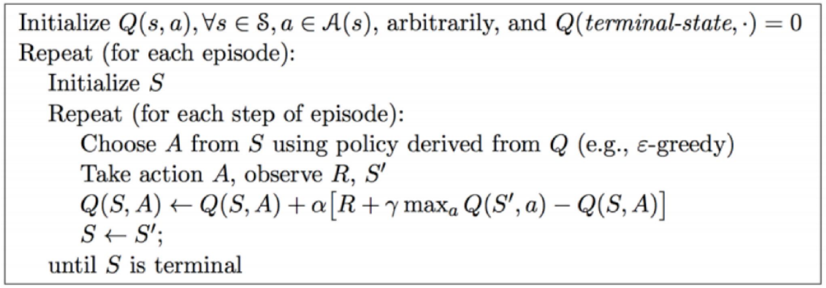
- The Q-Learning update rule can be expanded as:

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

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[\mathcal{R}(s,a) + \gamma \max_{a'} Q(s',a')\right]$$

- The update rule adjusts the current Q-value estimate $Q(s,a)$ in the direction of the target Q-value based on the Bellman error.
  - Q-values are gradually improved and converge towards the optimal Q-function.

Q-Learning Algorithm

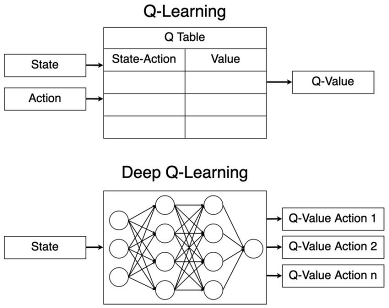


```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

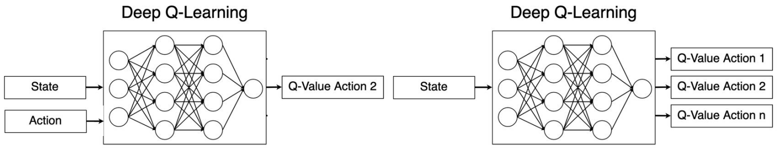
ε-Greedy Policy

- Simple solution to balance exploration and exploitation: ϵ-greedy policy
- ϵ-greedy policy:
  - With probability $1 - \epsilon$, choose the optimal action according to the learned Q-values.
  - With probability ϵ, choose a random action.
- The ϵ-greedy policy ensures that the agent explores the environment while still exploiting the learned knowledge.
- Despite its simplicity, ϵ-greedy is still widely used in practice and often yields good results.

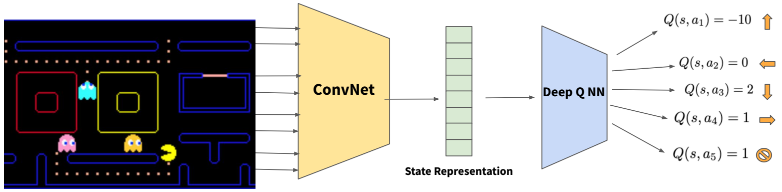
How can we use deep learning to improve classical RL techniques?



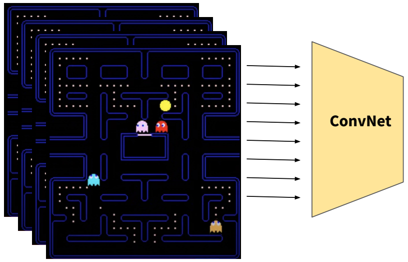
Discussion: Which do we prefer?



Play Pacman with DQN



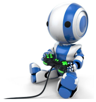
We feed in multiple



Training DQN

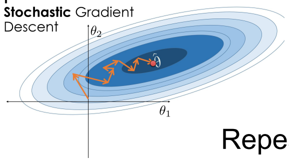
Naive Approach

Agent performs action



(s_t, a_t, r_t, s'_t)

Gradient step



Repeat

Experience Replay

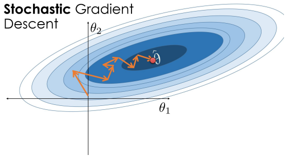
Store Transitions

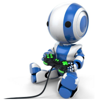
(s_1, a_1, r_1, s'_1)
…
…
…
(s_t, a_t, r_t, s'_t)

Sample Minibatch

(s_1, a_1, r_1, s'_1)
…
(s_j, a_j, r_j, s'_j)

Training





- Efficient use of training data
- Avoid forgetting previous experiences
- Remove correlations

How do we compute loss?

$$||y - Q(s, a)||_2^2$$

"True" Q value

Temporal Difference Target

- Consider the Bellman update

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[\mathcal{R}(s,a) + \gamma \max_{a'} Q(s',a')\right]$$

- When does this converge?

$$Q(s,a) = \mathcal{R}(s,a) + \gamma \max_{a'} Q(s',a')$$

Discussion: Why might issues arise with this loss function?

$$\|\underbrace{\boxed{r + \gamma \max_{a} Q(s',a)}}_{\text{Target}} - \underbrace{\boxed{Q(s,a)}}_{\text{Prediction}}\|_2^2$$

Chasing a moving target!

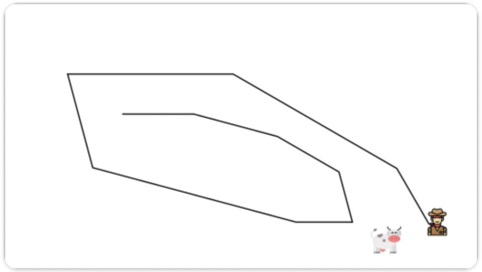
When $Q(s,a)$ changes, so does y

https://huggingface.co/learn/deep-rl-course/en/unit3/deep-q-algorithm

---

Chasing a moving target!

When $Q(s,a)$ changes, so does y



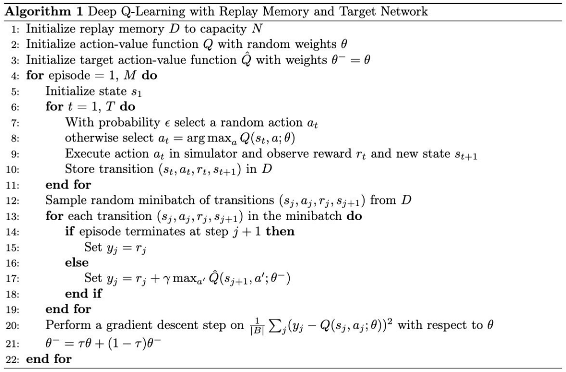
Solution: Fix the target to stabilize training

$$||r + \gamma \max_a \boxed{Q}(s', a) - Q(s, a)||_2^2$$

- Use a frozen network to compute the target
- Update the frozen network periodically

Deep Q-learning

- Sampling
  - Perform actions and store the observed experience tuples in a replay memory
  - Tuples are of the form (s_t, a_t, r_t, s_t')
- Training
  - Select a batch from the replay buffer
  - Update neural network based on the batch



Algorithm 1 Deep Q-Learning with Replay Memory and Target Network

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ do
    Initialize state $s_1$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
        Execute action $a_t$ in simulator and observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    end for
    Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
    for each transition $(s_j, a_j, r_j, s_{j+1})$ in the minibatch do
        if episode terminates at step $j+1$ then
            Set $y_j = r_j$
        else
            Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
        end if
    end for
    Perform a gradient descent step on $\frac{1}{|B|} \sum_j (y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
    $\theta^- = \tau\theta + (1 - \tau)\theta^-$
end for
```

Overestimation in DQN

$$y = r + \max_a Q(s', a; \theta^-)$$

Taking the max of noisy random variables

Overestimation Intuition

- There are 300 people with the same weight: 150 pounds
- There is a weight scale that measures with an error of +/- 5
- Suppose someone (who doesn't know the original weights) and wants to estimate the max weight
  - Method 1:
    - sample n people and weigh them to obtain a list of weights $x_1,\ldots,x_n$
    - Output $\max(x_1,\ldots,x_n)$
  - Method 2:
    - Weigh each if the n people twice to obtain x1,...,xn and x'1,...,x'n
    - Find i = argmax(x1,...,xn) and then output x'_i

Overestimation in DQN

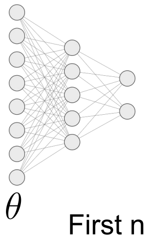
$$y = r + \max_a Q(s', a; \theta^-)$$

Taking the max of noisy random variables

Idea: Use different
networks to select action
& evaluate action

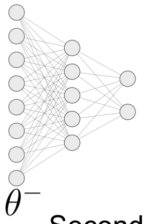
Double DQN

$$a^* = \arg\max_a Q(s', a; \theta)$$



First network
selects best action

$$y = r + \gamma Q(s', a^*; \theta^-)$$



Second network
evaluates action

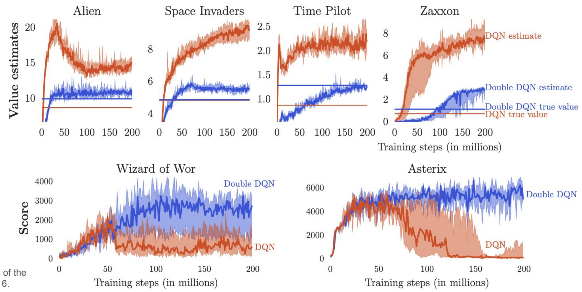
Discuss: What's the difference between Fixed Q-Targets & Double DQN?

Algorithm 2 Double Deep Q-Learning

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ do
    Initialize state $s_1$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
        Execute action $a_t$ in simulator and observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    end for
    Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
    for each transition $(s_j, a_j, r_j, s_{j+1})$ in the minibatch do
        if episode terminates at step $j+1$ then
            Set $y_j = r_j$
        else
            $a^* = \max_{a'} Q(s_{j+1}, a'; \theta)$
            Set $y_j = r_j + \gamma \hat{Q}(s_{j+1}, a^*; \theta^-)$
        end if
    end for
    Perform a gradient descent step on $\frac{1}{|B|} \sum_j (y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
    $\theta^- = \tau\theta + (1-\tau)\theta^-$
end for
```

Impact of Double DQN

- Double DQN reduces over-estimation
- Stabilizes training



Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

Summary of Models

Q Learning	Vanilla DQN	Double DQN
Basic reinforcement learning algorithm Tabular storage of q-values	Use ConvNet to represent environment Neural net for Q-values	Fixes the overestimation problem by using different networks to find and evaluate the best action

---

Review

- Tabular q-learning does not scale well when there a lot of states
- Deep q-learning uses a neural networks and is able to handle problems with large state-action spaces
- Issues with vanilla deep q-learning:
    - Correlation between subsequent time steps
    - Moving target
- Vanilla deep q-learning can be further improved by fixing the overestimation problem and using double deep q-learning