Reinforcement Learning
DQN & SARSA

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

June 30, 2025

Table of Contents

Learning Outcomes

By the end of this session, you will be able to:

- Understand and implement **Deep Q-Networks (DQN)**.
- Grasp the **SARSA algorithm** and how it contrasts with Q-learning.
- Analyze the **differences between off-policy and on-policy learning**.
- Evaluate the strengths and weaknesses of both algorithms.
- Understand practical applications and limitations of DQN and SARSA.

Why DQN? (Motivation)

- **Tabular Q-learning does not scale** to large or continuous state spaces.
- **Function approximation** is needed to generalize across similar states; deep learning provides powerful tools for this.
- **Deep Q-Networks (DQN)** combine Q-learning with deep neural networks, enabling reinforcement learning for high-dimensional inputs such as images.
- **Breakthrough:** DQN achieved human-level performance on Atari 2600 games directly from raw pixel inputs.

أكاديمية كاوست
KAUST ACADEMY

Reinforcement Learning: **Deep Q-Network (DQN) Overview**

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Solution**: use a function approximator to estimate $Q(s, a)$. E.g. a neural network!

$$Q(s, a, ; \theta) \approx Q^{\star}(s, a)$$

Deep Q-Learning (DQN)

- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair
- What's the problem with this?
- Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Solution**: use a function approximator to estimate $Q(s, a)$. E.g. a neural network!

$$Q(s, a, ; \theta) \approx Q^{\star}(s, a)$$

- If the function approximator is a deep neural network $\Rightarrow$ Deep Q-Learning (DQN)

Deep Q-Learning (DQN)

- Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^\star(s,a) = \mathbb{E}_{p(s'|s,a)}\left[r(s,a) + \gamma \max_{a'} Q^\star(s',a')|s,a\right]$$

- Forward Pass:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a\sim p(.)}\left[(y - Q(s,a;\theta))^2\right]$$

$$\text{where} \quad y = \mathbb{E}_{p(s'|s,a)}\left[r(s,a) + \gamma \max_{a'} Q(s',a';\theta)|s,a\right]$$

- Backward Pass:
  - Gradient update (with respect to Q-function parameters θ):

-2cm-2cm

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{s,a\sim p(.),p(s'|s,a)}\left[\left(r(s,a) + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta)\right)\nabla_\theta Q(s,a;\theta)\right]$$

Reinforcement Learning: **Training the Q-network**

Training the Q-network: Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated $\Rightarrow$ inefficient learning.
  - The current Q-network parameters determine the next training samples (e.g., if the maximizing action is to move left, training samples will be dominated by transitions from the left-hand side), which can lead to bad feedback loops.

Training the Q-network: Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated $\Rightarrow$ inefficient learning.
  - The current Q-network parameters determine the next training samples (e.g., if the maximizing action is to move left, training samples will be dominated by transitions from the left-hand side), which can lead to bad feedback loops.
- These problems can be addressed using experience replay:
  - Maintain a replay memory buffer of transitions (s_t, a_t, r_t, s_{t+1}) as episodes are played.
  - Train the Q-network on random minibatches of transitions sampled from the replay memory, instead of using consecutive samples.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

DQN with Target Network

- DQN, as stated above, will have learning problems because the target and the prediction are not independant as they both rely on the same network.
- It's like a dog chasing it's own tail.

DQN with Target Network

- DQN, as stated above, will have learning problems because the target and the prediction are not independant as they both rely on the same network.
- It's like a dog chasing it's own tail.
- **Solution:** Use two separate Q-value estimators, each of which is used to update the other.
- The target values are calculated using a target Q-network. The target Q-network's parameters are updated to the current networks every C time steps.
- Target network prevents the network from spiraling around.

DQN with Target Network Algorithm

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

[0] Minh et al. https://www.nature.com/articles/nature14236

Reinforcement Learning: **Double DQN**

Double DQN

- Q-learning suffers from a maximization bias.
- This is because the update target is $r + \max_a Q^\star(s, a)$. If Q-value is slightly overestimated then this error gets compounded.

Double DQN

- Q-learning suffers from a maximization bias.
- This is because the update target is $r + \max_a Q^{\star}(s, a)$. If Q-value is slightly overestimated then this error gets compounded.
- **Solution:** Decompose the max operation in the target into action selection and action evaluation.
- Use the current network to select the max action for the next state and then use the target network to get the target Q-value for that action.
- Using these independent estimators, we can have unbiased Q-value estimates of the actions selected using the opposite estimator.
- We can thus avoid maximization bias by disentangling our updates from biased estimates.

DDQN Algorithm

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$

$$y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$$

    Do a gradient descent step with loss $\| y_j - Q(s, a; \theta) \|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

[0] https://leejungi.github.io/posts/Dueling-DQN/

On-Policy vs. Off-Policy Algorithms

- **On-Policy**:
  - Agent learns by doing.
  - Follows one policy for both acting and learning.
  - Example: Tries out actions, learns from its own experience.
- **Off-Policy**:
  - Agent learns from others or past experiences.
  - Can follow one policy but learn about another.
  - Example: Watches someone else, learns what would have happened if it acted differently.

Reinforcement Learning: **SARSA**

SARSA: Overview

- **SARSA** stands for:

$$\begin{aligned}&\text{State}\\ \rightarrow\ &\text{Action}\\ \rightarrow\ &\text{Reward}\\ \rightarrow\ &\text{State}'\\ \rightarrow\ &\text{Action}'\end{aligned}$$

- **On-policy learning:** Learns Q-values by following the current behavior policy.
- **Update Rule:**

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

SARSA: Key Points

- Uses the action the agent actually takes, not just the best possible one.
- Learns from the agent's real experience, not hypothetical alternatives.
- Tends to be safer—useful in situations where mistakes are costly.
- Helps the agent improve its actual behavior step by step.

SARSA vs. Q-Learning

- The SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning is an **off-policy** method and uses a greedy approach to learn the Q-values.
- SARSA, on the other hand, is an **on-policy** method and uses the action performed by the current policy to update the Q-values.

SARSA vs. Q-Learning

- The SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning is an **off-policy** method and uses a greedy approach to learn the Q-values.
- SARSA, on the other hand, is an **on-policy** method and uses the action performed by the current policy to update the Q-values.

Q-Learning: $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

- The SARSA algorithm is a slight variation of the Q-Learning algorithm.
- Q-Learning is an **off-policy** method and uses a greedy approach to learn the Q-values.
- SARSA, on the other hand, is an **on-policy** method and uses the action performed by the current policy to update the Q-values.

Q-Learning: $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

SARSA: $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- The update equation for SARSA depends on the current state, current action, reward obtained, next state, and next action.
- This observation led to the naming of the learning technique as **SARSA**, which stands for **State-Action-Reward-State-Action**, symbolizing the tuple (s, a, r, s', a').

SARSA

- The update equation for SARSA depends on the current state, current action, reward obtained, next state, and next action.
- This observation led to the naming of the learning technique as **SARSA**, which stands for **State-Action-Reward-State-Action**, symbolizing the tuple (s, a, r, s', a').
- Similar to DQN, there is also a Deep SARSA variant.

Input : States, $s \in S$, Actions $a \in A(s)$, Initialize Q(s, a), α, γ, π to an arbitrary policy (non-greedy)

Output: Optimal action value Q(s, a) for each stste-action pair

while *True* do

  for ($i = 0; i \leq \#\, of\, episodes; i++$) do

    Initialize s

    Choose a from s, using policy derived from Q

    Repeat(for each step of episodes):

    Take action a; observe reward, r, and next state, s'

    Choose action a' from state s' using policy derived from Q

    $Q(s, a) \longleftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

    $s \longleftarrow s'$; $a \longleftarrow a'$;

    until s is terminal

  end

end

[0] Lockery & Peters, Adaptive learning by a target-tracking system

Q-Learning vs SARSA: Comparison Table

Feature	**Q-Learning**	**SARSA**
Policy Type	Off-policy	On-policy
Target	$\max Q(s', a')$	$Q(s', a')$ actually taken
Exploration Aware	No	Yes
Risk Sensitivity	More aggressive	More conservative
Use Case	Goal-seeking behavior	Risk-aware behavior

Case Study: **Playing Atari Games**

Case Study: Playing Atari Games

- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

[0] [Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games

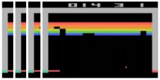
$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[0] [Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games

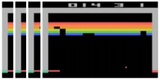
$Q(s, a; \theta)$:
neural network
with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

Number of actions between 4-18 depending on Atari game

https://www.youtube.com/watch?v=V1eYniJ0Rnk

[0] [Mnih et al. NIPS Workshop 2013; Nature 2015]

DDQN & SARSA: **Summary**

When to Use Which Algorithm?

Scenario	Recommended Algorithm
Environment is stochastic	SARSA (conservative)
Maximizing reward is the priority	Q-Learning or DQN
Large state space (e.g., images)	DQN
Limited computational resources	SARSA (simpler model)

Future Directions

- **Dueling DQN**: Separates value and advantage estimation.
- **Double DQN**: Reduces overestimation bias.
- **Prioritized Replay**: Samples important experiences more frequently.
- **Distributional RL**: Models the full return distribution, not just the expectation.
- **Safe RL**: Ensures safety in high-stakes environments (e.g., medical, robotics).

Summary

- A Markov Decision Process (MDP) is the mathematical formulation of the reinforcement learning problem, defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$.
- Each state satisfies the Markov property, i.e., the future is independent of the past given the present.
- The agent and the environment interact in a sequential loop. The policy π determines how the agent chooses actions.
- The value function estimates how good a state is, while the Q-value function estimates the quality of a state-action pair.
- The Bellman equation is a recursive formula for the Q-value function.

Summary (cont.)

- Q-learning is an algorithm that repeatedly adjusts Q-values to minimize the Bellman error.
- When the Q-value function approximator is a deep neural network, we obtain Deep Q-Learning.
- DQN is powerful for complex, high-dimensional inputs but sensitive and data-hungry.
- SARSA is an on-policy variation of Q-learning.
- SARSA offers a safer, on-policy alternative, better suited for uncertain environments.
- Choice depends on task risk, dimensionality, and training stability.

DDQN & SARSA: **References**

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015).
Human-level control through deep reinforcement learning.
Nature.

[2] Van Hasselt, H., Guez, A., Silver, D. (2016).
Deep Reinforcement Learning with Double Q-learning.
AAAI.

[3] Wang, Z., Schaul, T., Hessel, M., et al. (2016).
Dueling Network Architectures for Deep RL.
ICML.

[4] Rummery, G. A., & Niranjan, M. (1994).
On-line Q-learning using connectionist systems.
Technical Report.

[5] Sutton, R. S., & Barto, A. G. (2018).
Reinforcement Learning: An Introduction.

[6] Berkeley CS285 Deep RL Lectures.
https://rail.eecs.berkeley.edu/deeprlcourse/

[7] David Silver's RL Series.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

[8] Emma Brunskill, Stanford CS234: Reinforcement Learning.
https://web.stanford.edu/class/cs234/

References (cont.)

[9] Fei-Fei Li, Yunzhu Li & Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision.
http://cs231n.stanford.edu/

[10] Ashwin Rao, Stanford CME241: Foundations of Reinforcement Learning with Applications in Finance.
https://web.stanford.edu/class/cme241/

[11] Jimmy Ba & Bo Wang, UofT CSC413/2516: Neural Networks and Deep Learning.
https://uoft-csc413.github.io/2022/

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.