Reinforcement Learning: From Q Learning to Deep Q-Learning

Prasun De (Roll: MB2117)

ISI Kolkata

July 25, 2023

Reinforcement Learning

- Learning to map situations to actions in order to maximize a numerical reward
- Learner must **discover** which actions to take by trying them out (trial-and-error)
- Learner can face **delayed reward**: consequences of a delayed action are not immediately observable or received
- Learner faces **sequential data and interactions**, where the learner's actions influence the subsequent states and rewards it receives

Markov Decision Process (MDP)

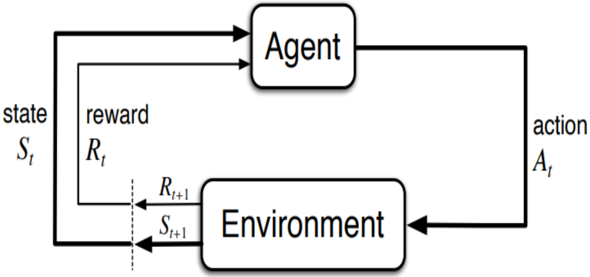


Figure: Agent–Environment Interaction

Assumptions:

- Agent gets to observe the state
- **Markov Property**:

$$P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \cdots, r_1, s_0, a_0)$$
$$= P(s_{t+1} = s', r_{t+1} = r | s_t, a_t)$$

Real Life Applications

- **Self-Driving Cars**: Q-Learning for Lane changing [2]
- **Energy Conservation**: Deepmind uses RL agents to achieve 40% decrease in energy consumption of data centers
- **Financial Applications**: Hedging portfolio in presence of transaction costs etc [1]
- **Healthcare**: Discovery and generation of optimal DTRs for chronic diseases [8]
- **Recommendation Engines**: Personalized User Recommendations [3]

MDP Setup

MDP is defined by:

- Set of States S
- Set of Actions A
- Transition function $P(s'|s,a)$
- Reward function $R(s,a,s')$
- Start state s_0
- Discount factor γ
- Horizon H

Goal: Given an MDP(S, A, P, R, γ, H) obtain **optimal policy** $\pi^\star$ for

$$\max_\pi E\left[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi\right]$$

Exact Method: Value Iteration

- **Optimal Value Function** $V_H^\star$ for horizon H

$$V_H^\star(s) := \max_\pi E\left[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi, s_0 = s\right]$$

$=$ sum of discounted rewards starting from state s and acting optimally

- We can say that $V_0^\star(s) = 0 \forall s$.

$$\begin{aligned} V_1^\star(s) &= \text{ optimal value for state } s \text{ when } H = 0 \\ &= \max_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + \gamma V_0^\star(s')) \end{aligned}$$

- In general, we can say:

$$\begin{aligned} V_k^\star(s) &= \text{ optimal value for state } s \text{ when } H = k \\ &= \max_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + \gamma V_{k-1}^\star(s')) \end{aligned}$$

Value Iteration Convergence-I

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, ... , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

This is called a value update or Bellman update/back-up

Figure: Value Iteration Algorithm

Result: Value iteration converges. At convergence, we obtain the optimal value $V^\star$ for infinite horizon problem and it satisfies:

$$V^\star(s) = \max_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + \gamma V^\star(s'))$$

Value Iteration convergence-II

- Infinite horizon policy is stationary: optimal action at a state s is the same action at all times
- Recall definitions of $V^\star(s)$ and $V_H^\star(s)$ for a state s.
- Additional reward collected over $t = H+1, H+2, \cdots$

$$\begin{aligned}\gamma^{H+1}R(s_{H+1}) + \gamma^{H+2}R(s_{H+2}) + \cdots &\le \gamma^{H+1}R_{\max} + \gamma^{H+2}R_{\max} + \cdots \\ &= \frac{\gamma^{H+1}}{1-\gamma}R_{\max} \to 0\end{aligned}$$

Intuition for $V_H^\star \to V^\star$ as $H \to \infty$

- Proof involves contractions of the max-norm

Q-values

- $Q^\star(s, a)$ is the expected utility started in s, taking action a and (thereafter) acting optimally
- **Q-value iteration** has a somewhat similar form to the Value function iteration

$$Q^\star_{k+1}(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^\star_k(s', a'))$$

- **Bellman Equation**:

$$Q^\star(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^\star(s', a'))$$

- No need to keep track of policy and value now !

(Tabular) Q-Learning

- **Q-value iteration**

$$Q^\star_{k+1}(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^\star_k(s',a'))$$

- Rewriting it as an expectation:

$$Q^\star_{k+1}(s,a) = E_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q^\star_k(s',a') \right]$$

- Tabular Q-Learning: Replace expectation by samples.
  - For state-action pair (s,a), receive $s' \sim P(s'|s,a)$
  - Consider **old estimate** $Q_k(s,a)$
  - Obtain **new sample estimate**:

$$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

  - Incorporate new estimate into running average

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha \text{target}(s')$$

Tabular Q-Learning Algorithm

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s’

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

$$s \leftarrow s'$$

Figure: Tabular Q-Learning Algorithm

We can sample actions as follows:

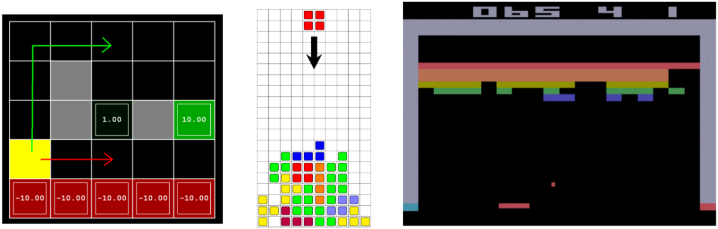
- Choose *a* randomly
- Choose *a* that maximizes $Q_k(s, a)$ greedily
- $\epsilon-$Greedy: Choosing random action w.p. ϵ and greedily w.p. $1 - \epsilon$

Q-Learning Convergence

- **Q-Learning Converges** to the optimal policy (under some weak conditions)
- **Off-policy learning**
- Explore enough (s, a) pairs and also exploit the learnt $Q - values$
- Eventually decrease learning rate gradually
- **Convergence guarantee** for bounded reward and finite state-action space [7]:
  - $n^i(s, a)$ is the index of i-th time that action a is tried in state s
  - $|r_n| \leq R_{\max}$
  - $\alpha_n \in [0, 1]$
  - $\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty$
  - $\sum_{i=1}^{\infty} \alpha^2_{n^i(s,a)} < \infty$

  Then, $Q_n(s, a) \to Q^\star(s, a) \forall s, a$

Scaling Tabular Methods



Gridworld
10^1

Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

Figure: Size of State Space

Challenges with scaling

- Basic Q-Learning keeps a table of Q-values.
- In realistic situation, cannot possibly learn about every single state.
  - Too many states to visit all during training
  - Too many state-action pairs to hold Q-table in memory
- We would instead want to generalize:
  - Learn about small number of training states from experience
  - Generalize the experience to new similar situations

Approximate Q-Learning

- We have a parameterized Q-function $Q_\theta(s,a)$ instead of a table.
  - $Q_\theta(s,a) = \sum_{k=0}^{n} \theta_k f_k(s,a)$ or
  - Can be a neural network
- **Learning Rule**: We can try to change the
  - $\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_{\theta_k}(s',a')$
  - Gradient update on θ: $\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2\right] |_{\theta=\theta_k}$

Connection to Tabular Q-Learning

- Suppose $\theta \in \mathbb{R}^{|S| \times |A|}$ and $Q_\theta(s, a) = \theta_{sa}$

$$
\begin{aligned}
&\nabla_{\theta_{sa}} \left[\frac{1}{2} (Q_{\theta_{sa}}(s, a) - \text{target}(s'))^2 \right] \\
&= \nabla_{\theta_{sa}} \left[\frac{1}{2} (\theta_{sa} - \text{target}(s'))^2 \right] \\
&= \theta_{sa} - \text{target}(s')
\end{aligned}
$$

- Now, we may plug it into our update:

$$
\begin{aligned}
\theta_{sa} &\leftarrow \theta_{sa} - \alpha(\theta_{sa} - \text{target}(s')) \\
&= (1 - \alpha)\theta_{sa} + \alpha[\text{target}(s')]
\end{aligned}
$$

- Compare with tabular Q-Learning Update:

$$
Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha[\text{target}(s')]
$$

- Deep RL essentially uses powerful function approximators (eg; CNN) to represent Q-values (**with some care !**)

Approximate Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s’

else:

Chasing a nonstationary target!

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[(Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

$$s \leftarrow s'$$

Updates are correlated within a trajectory!

Figure: Approx. Q-Learning Algorithm

Deep Q-Networks

- The high-level idea is to make Q-Learning look like supervised learning
- Two essential ideas to stabilize training:
  - **Experience Replay Buffer** [4]
  - Previously used for better data efficiency
  - Makes data distribution more stationary
  - Use an **older set of weights to compute targets**
  - Keeps target function from changing too quickly

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

Figure: DQN update

Replay Buffer

- Most recent k transitions $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a **replay buffer** $D_T = \{e_1, e_2, \cdots, e_T\}$
- Sample uniformly a batch of N transitions to update the Q-network
- Helps in improving data efficiency, reducing sample correlations

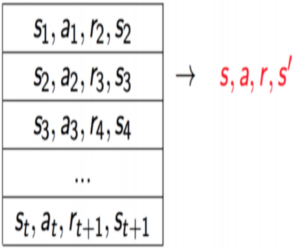


Figure: Experience Replay

Target Network Intuition

- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.

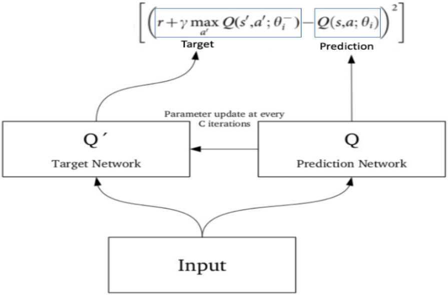


Figure: Target network

DQN 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 $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
    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

Figure: Target network

DQN on Atari

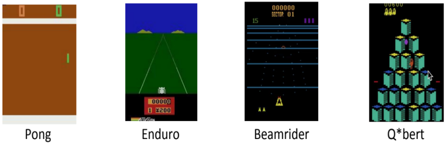


Figure: Some Atari Games

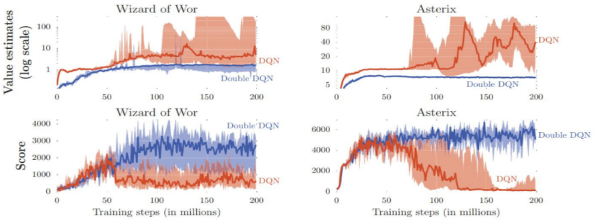
- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

Double DQN

- There is an **upward bias** in $\max_a Q(s, a; \theta)$ [6]
- DQN maintains two sets of weight θ and θ^-, so reduce bias by using:
  - θ to **select** best action
  - θ^- to **evaluate** best action
- **Double DQN** [6] loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\ D}\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta_i^-) - Q(s, a; \theta_i)\right)^2$$

Figure: Double DQN Loss



Prioritized Experience Replay

- Replaying all transitions with equal probability is highly suboptimal.
- Replay transitions in proportion to absolute Bellman error [5]:

$$|r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a, \theta)|$$

- Generally leads to faster learning

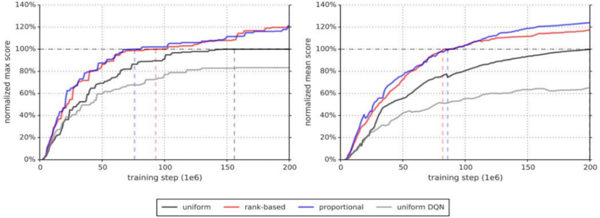


Figure: PER

Thank You !

[1] Hans Bühler et al. *Deep Hedging*. 2018. arXiv: 1802.03042 [q-fin.CP].

[2] B Ravi Kiran et al. "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2021), pp. 4909–4926.

[3] Lihong Li et al. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web*. ACM, Apr. 2010. DOI: 10.1145/1772690.1772758. URL: https://doi.org/10.1145%2F1772690.1772758.

[4] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[5] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG].

[6] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In:

Proceedings of the AAAI conference on artificial intelligence. Vol. 30. 1. 2016.

[7] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8 (1992), pp. 279–292.

[8] Chao Yu, Jiming Liu, and Shamim Nemati. *Reinforcement Learning in Healthcare: A Survey*. 2020. arXiv: 1908.08796 [cs.LG].