Markov Decision Process

Markov Property

$$\mathbb{P}[S_{t+1}|S_t, A_t] = \mathbb{P}[S_{t+1}|S_1, A_1, S_2, A_2 \ldots, S_t, A_t]$$

$$\mathbb{P}[R_t|S_t, A_t] = \mathbb{P}[R_t|S_1, A_1, S_2, A_2 \ldots, S_t, A_t] = 1$$

Markov Decision Process $\langle \mathcal{S}, \mathcal{S}_F, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is an infinite state space
- $\mathcal{S}_F$ is a set of final states
- $\mathcal{A}$ is a finite $(|\mathcal{A}| = m)$ action space
- $\mathcal{P}$ is an uknown transition probability function

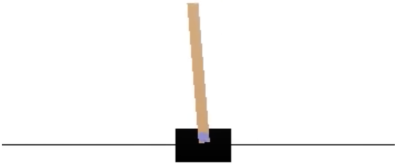
$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- $\mathcal{P}_0$ is an uknown initial state probability function
- $\mathcal{R}$ is an uknown reward function

$$\mathcal{R}(s, a) = R_t \quad \Leftrightarrow \quad \mathbb{P}[R_t|S_t = s, A_t = a] = 1$$

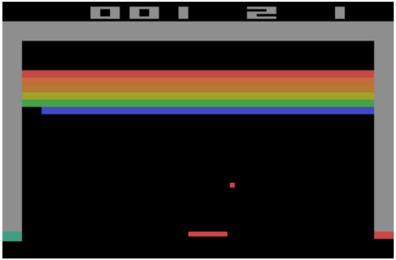
- $\gamma \in [0, 1]$ is a discount coefficient

Example: Cartpole



- States: $\mathbb{R}^4$
- Actions: $\rightarrow$, $\leftarrow$, «0»
- Rewards: $+1$ for each step
- Final states: when the pole falls down

Example: Breakout Atari Game



- States: pixels
- Actions: →, ←, «0»
- Rewards: points
- Final states: when the ball falls down

Monte-Carlo Algorithm

Let $Q(s,a) = 0$, $N(s,a) = 0$ and $\varepsilon = 1$.
For each $k \in \overline{1, K}$, do

- According to $\pi = \varepsilon$-greedy(Q), get trajectory $\tau = (S_0, A_0, \ldots, S_T)$ and rewards $(R_0, \ldots, R_{T-1})$. Define $(G_0, \ldots, G_{T-1})$.
- For each $t \in \overline{0, T-1}$, update Q and N:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t) + 1}\big(G_t - Q(S_t, A_t)\big),$$

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

Define $\varepsilon = 1/k$

Monte-Carlo Algorithm

Let $Q(s,a)=0$, $N(s,a)=0$ and $\varepsilon=1$.
For each $k \in \overline{1,K}$, do

- According to $\pi = \varepsilon$-greedy(Q), get trajectory $\tau = (S_0, A_0, \dots, S_T)$ and rewards $(R_0, \dots, R_{T-1})$. Define $(G_0, \dots, G_{T-1})$.
- For each $t \in \overline{0, T-1}$, update Q and N:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \frac{1}{N(S_t,A_t)+1}\big(G_t - Q(S_t,A_t)\big),$$

$$N(S_t,A_t) \leftarrow N(S_t,A_t) + 1$$

Define $\varepsilon = 1/k$

SARSA Algorithm

```
Let $Q(s, a) = 0$, $K > 0$, and $\varepsilon = 1$.
For each $k \in \overline{1, K}$, do
    During trajectory
        • From the state $S_t$, acting $A_t \sim \pi(\cdot|S_t)$, where $\pi = \varepsilon$-greedy$(Q)$,
          get $R_t$, go to the next state $S_{t+1}$, and act $A_{t+1} \sim \pi(\cdot|S_{t+1})$
        • According to $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$, update $Q$:

              $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

    Put $\varepsilon = 1/k$
```

Q-Learning Algorithm

Let $Q(s,a)=0$, $K>0$, and $\varepsilon=1$.
For each $k \in \overline{1,K}$, do

During trajectory

- From the state S_t, acting $A_t \sim \pi(\cdot|S_t)$, where $\pi = \varepsilon$-greedy(Q), get R_t, and go to the next state S_{t+1}
- According to (S_t, A_t, R_t, S_{t+1}), update Q:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$$

Put $\varepsilon = 1/k$

Approximation

Idea

- Define $Q^{\theta}(s,a)$ parameterized by $\theta \in \mathbb{R}^N$
- Find θ such that

$$Q^{\theta}(s,a) \approx q_{\pi}(s,a) \quad \text{or} \quad Q^{\theta}(s,a) \approx q_{*}(s,a)$$

Differentiable Approximators

- Linear combinations
- Neural networks

Linear Combinations

$$Q^\theta(s,a) = \sum_{i=1}^{n} \theta_i \varphi_i(s,a),$$

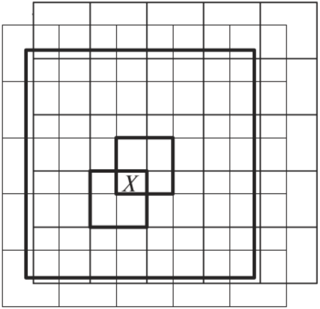
where $\varphi_i(s,a)$ are fixed functions

Gradient

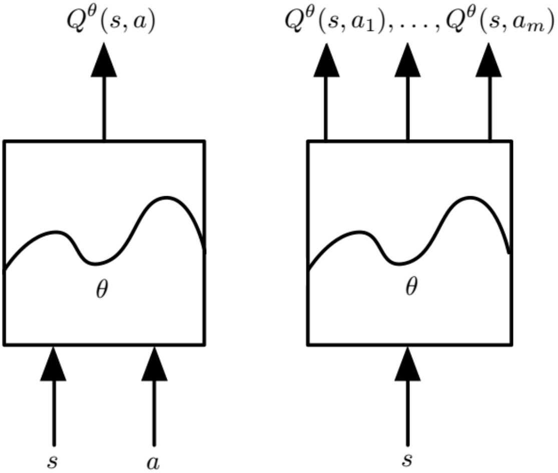
$$\nabla_\theta Q^\theta(s,a) = \begin{pmatrix} \varphi_1(s,a) \\ \vdots \\ \varphi_n(s,a) \end{pmatrix}$$

Example $\varphi_{i,j}(s,a)$

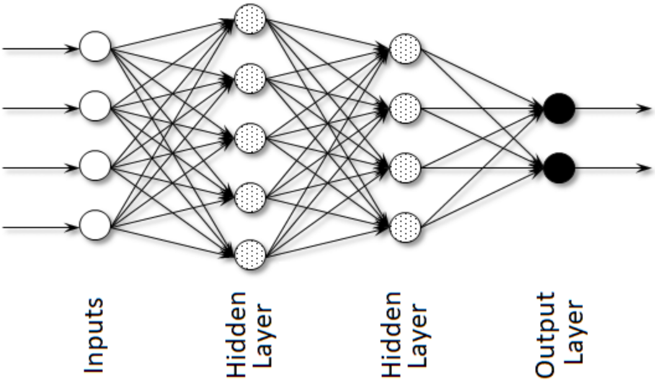
$$\begin{array}{c|cccc} & & j & & \\ \hline & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 \\ i & 0 & 1 & 0 & 0 \\ & 0 & 0 & 0 & 0 \end{array}$$



Neural Networks



Neural Networks



$$F_j^\theta(X) = f_{out}\left(b_j + \sum_{k=1}^{4} w_{j,k} f\left(\hat{b}_k + \sum_{l=1}^{5} \hat{w}_{k,l} f\left(\tilde{b}_l + \sum_{i=1}^{4} \tilde{w}_{l,i} x_i\right)\right)\right), \quad j \in \overline{1,2}.$$

$$F^\theta(X) \in \mathbb{R}^2, \quad X \in \mathbb{R}^4, \quad \theta \in \mathbb{R}^{59}$$

Monte-Carlo Update

Q-Function

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a]$$

$$\Downarrow$$

Monte-Carlo Update for Q

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t) + 1}\big(G_t - Q(S_t, A_t)\big),$$

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$\Downarrow$$

Monte-Carlo Update for Q^θ

$$Loss(\theta) = \big(G_t - Q^\theta(S_t, A_t)\big)^2$$

$$\nabla_\theta Loss(\theta) = -2\big(G_t - Q^\theta(S_t, A_t)\big)\nabla_\theta Q^\theta(S_t, A_t)$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

SARSA Update

Bellman Expectation Equation

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s,\ A_t = a]$$

$$\Downarrow$$

SARSA Update for Q

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big)$$

$$\Downarrow$$

SARSA Update for Q^θ

$$Loss(\theta) = \big(R_t + \gamma Q^\theta(S_{t+1}, A_{t+1}) - Q^\theta(S_t, A_t)\big)^2$$

$$\nabla_\theta Loss(\theta) \approx -2\big(R_t + \gamma Q^\theta(S_{t+1}, A_{t+1}) - Q^\theta(S_t, A_t)\big)\nabla_\theta Q^\theta(S_t, A_t)$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

Q-Learning Update

Bellman Optimality Equation

$$q_*(s,a) = \mathbb{E}[R_t + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s,\ A_t = a]$$

$$\Downarrow$$

Q-Learning Update for Q

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\big)$$

$$\Downarrow$$

Q-Learning Update for Q^θ

$$Loss(\theta) = \big(R_t + \gamma \max_{a'} Q^\theta(S_{t+1}, a') - Q^\theta(S_t, A_t)\big)^2$$

$$\nabla_\theta Loss(\theta) \approx -2\big(R_t + \max_{a'} \gamma Q^\theta(S_{t+1}, a') - Q^\theta(S_t, A_t)\big)\nabla_\theta Q^\theta(S_t, A_t)$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

Finite Case (for SARSA)

$$\mathcal{S} = \{0, 1, \ldots, n-1\}, \quad \mathcal{A} = \{0, 1, \ldots, m-1\}$$

Set

$$\varphi_{i,j}(s,a) = \begin{cases} 1, & \text{if } s = i,\ a = j, \\ 0, & \text{otherwise}, \end{cases} \qquad i \in \mathcal{S}, \quad j \in \mathcal{A}$$

$$Q^\theta(s,a) = \theta_{s,a}$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

$$\Downarrow$$

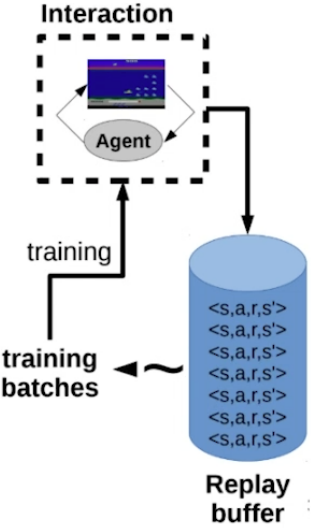
$$\begin{pmatrix} \theta_{0,0} \\ \vdots \\ \theta_{s,a} \\ \vdots \end{pmatrix} = \begin{pmatrix} \theta_{0,0} \\ \vdots \\ \theta_{s,a} \\ \vdots \end{pmatrix} - \alpha \begin{pmatrix} 0 \\ \vdots \\ -(R_t + \gamma Q^\theta(S_{t+1}, A_{t+1}) - Q^\theta(S_t, A_t)) \\ \vdots \end{pmatrix}$$

Convergence

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗

(✓) = chatters around near-optimal value function

Experience Replay



- Store $(S_t, A_t, R_t, S_{t+1}) \rightarrow Memory$
- Learning by minibatch $\{(s_i, a_i, r_i, s_i')\}_{i=1}^{n} \leftarrow Memory$

DQN Algorithm

Initialize a neural network Q^θ. Let $\varepsilon = 1$.
For each episode, do:
While the episode is finished, do:

- Being in a state S_t, act $A_t \sim \pi(\cdot|S_t)$, where $\pi = \varepsilon$-greedy(Q^θ), get a reward R_t, and transfer to a next state S_{t+1}.
  Store $(S_t, A_t, R_t, S_{t+1}) \to Memory$
- Get a minibatch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^n \leftarrow Memory$, determine target values

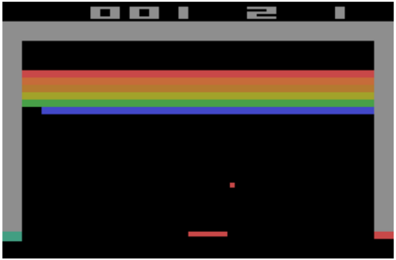
$$y_j = \begin{cases} r_j, & \text{if } s'_i \text{ is final,} \\ r_j + \gamma \max\limits_{a'} Q^\theta(s'_j, a'), & \text{otherwise} \end{cases}$$

  a loss function $Loss(\theta) = \frac{1}{n} \sum\limits_{i=1}^{n} \big(y_i - Q^\theta(s_i, a_i)\big)^2$ and update θ:

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

- Decrease ε

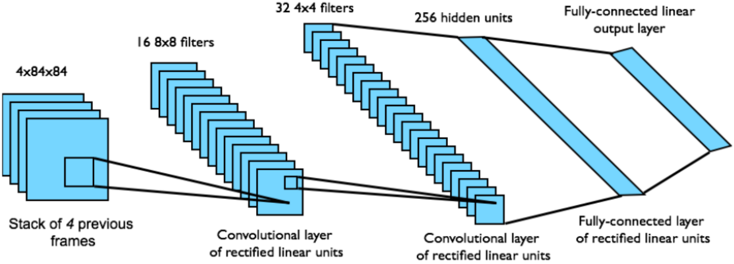
Example: Breakout Atari Game



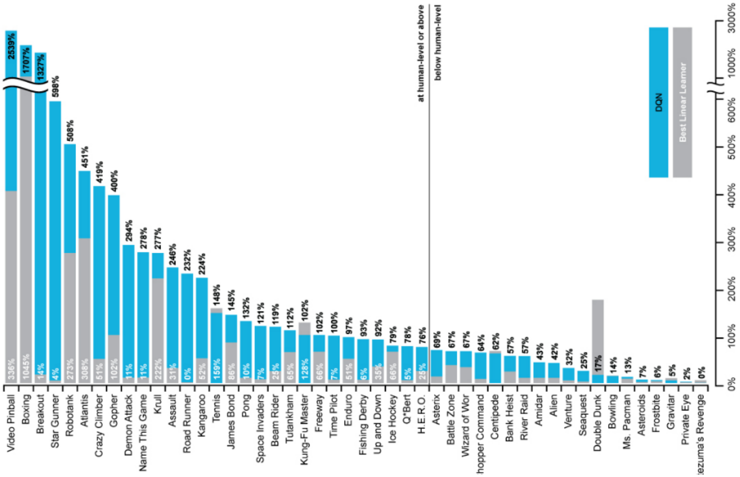
- States: pixels
- Actions: →, ←, «0»
- Rewards: points
- Final states: when the ball falls down

Neural Network for Atari Games

- The last 4 pre-processed screen images is input
- Output $Q^\theta(s, a_1), \ldots, Q^\theta(s, a_m)$
- Neural network structure and learning hyperparameters are the same for all games


4x84x84
16 8x8 filters
32 4x4 filters
256 hidden units
Fully-connected linear output layer
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

Performance for Atari Games



Mnih V., at el. Playing Atari with Deep Reinforcement Learning. 2013.

Can we use Experience Replay for MC and SARSA?

No

Because (S_t, A_t, G_t) and $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$ depend on Policy

Autocorrelation

Q-Learning Update for Q^θ

- $y = r + \gamma \max_{a'} Q^\theta(s', a')$
- $Loss(\theta) = \big(y - Q^\theta(s, a)\big)^2$
- $\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$

Challenge

If Reward in two close states is very different, then with Q-Learning Update it is possible to get $Q^\theta(s, a) \to \infty$

Hard Target Networks $Q^{\theta'}(s, a)$

- Set $\theta = \theta'$
- Do a lot of iterations:
    - $y = r + \gamma \max_{a'} Q^{\theta'}(s', a')$
    - $Loss(\theta) = \big(y - Q^{\theta}(s, a)\big)^2$
    - $\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$
- Update $\theta' = \theta$

Soft Target Networks $Q^{\theta'}(s, a)$

- $y = r + \gamma \max_{a'} Q^{\theta'}(s', a')$
- $Loss(\theta) = \big(y - Q^{\theta}(s, a)\big)^2$
- $\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$
- $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$

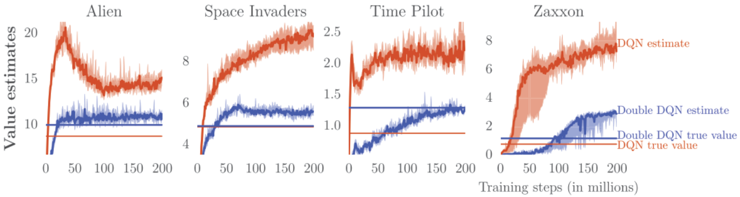
Performance of Experience Replay and Target Network

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Double DQN

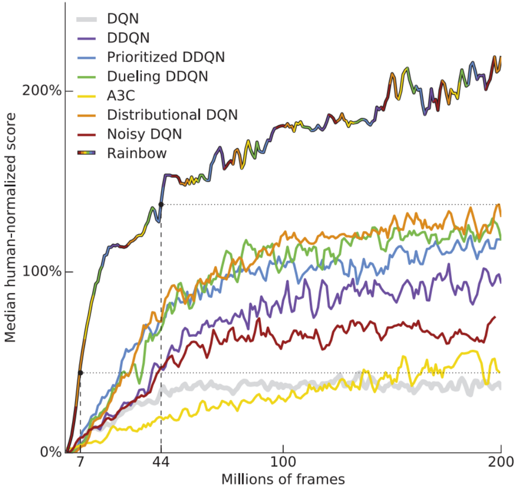
- $y = r + \gamma Q^{\theta}\big(s', \mathrm{argmax}_{a'} Q^{\theta'}(s', a')\big)$
- $Loss(\theta) = \big(y - Q^{\theta}(s, a)\big)^2$
- $\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$
- $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$

Performance of Double DQN

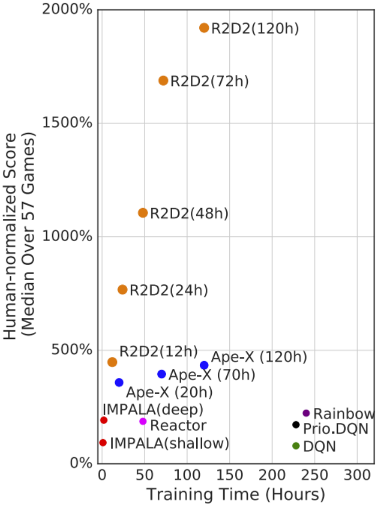


Van Hasselt H., Guez A., Silver D. Deep Reinforcement Learning with Double Q-Learning. 2016.

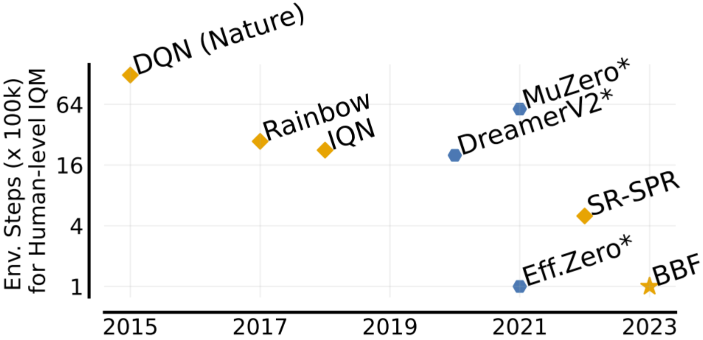
Rainbow (2018)



R2D2 (2019)


R2D2(120h)
R2D2(72h)
R2D2(48h)
R2D2(24h)
R2D2(12h)
Ape-X (120h)
Ape-X (70h)
Ape-X (20h)
IMPALA(deep)
Reactor
IMPALA(shallow)
Rainbow
Prio.DQN
DQN
2000%
1500%
1000%
500%
0%
0
50
100
150
200
250
300
Training Time (Hours)
Human-normalized Score
(Median Over 57 Games)

Bigger, Better, Faster (2023)



Markov Decision Process

Markov Property

$$\mathbb{P}[S_{t+1}|S_t, A_t] = \mathbb{P}[S_{t+1}|S_1, A_1, S_2, A_2 \ldots, S_t, A_t]$$

$$\mathbb{P}[R_t|S_t, A_t] = \mathbb{P}[R_t|S_1, A_1, S_2, A_2 \ldots, S_t, A_t] = 1$$

Markov Decision Process $\langle \mathcal{S}, \mathcal{S}_F, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is an infinite state space
- $\mathcal{S}_F$ is a set of final states
- $\mathcal{A}$ is a infinite action space
- $\mathcal{P}$ is an uknown transition probability function

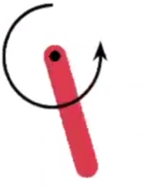
$$\mathcal{P}(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- $\mathcal{P}_0$ is an uknown initial state probability function
- $\mathcal{R}$ is an uknown reward function

$$\mathcal{R}(s,a) = R_t \quad \Leftrightarrow \quad \mathbb{P}[R_t|S_t = s, A_t = a] = 1$$

- $\gamma \in [0,1]$ is a discount coefficient

Example: Pendulum



- State space: $\mathbb{R}^2$ or screen pixels
- Action space: $[-2, 2]$
- Rewards: $-\psi^2 - 0.1\dot{\psi}^2 - 0.001a^2$

DQN Algorithm

Initialize a neural network Q^θ. Let $\varepsilon = 1$.
For each episode, do:
While the episode is finished, do:

- Being in a state S_t, act $A_t \sim \pi(\cdot|S_t)$, where $\pi = \varepsilon$-greedy(Q^θ), get a reward R_t, and transfer to a next state S_{t+1}.
  Store $(S_t, A_t, R_t, S_{t+1}) \to Memory$
- Get a minibatch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^n \leftarrow Memory$, determine target values

$$y_j = \begin{cases} r_j, & \text{if } s'_i \text{ is final,} \\ r_j + \gamma \max\limits_{a'} Q^\theta(s'_j, a'), & \text{otherwise} \end{cases}$$

  a loss function $Loss(\theta) = \frac{1}{n} \sum\limits_{i=1}^{n} \big(y_i - Q^\theta(s_i, a_i)\big)^2$ and update θ:

$$\theta \leftarrow \theta - \alpha \nabla_\theta Loss(\theta)$$

- Decrease ε

DQN Algorithm

Initialize a neural network Q^{θ}. Let $\varepsilon = 1$.
For each episode, do:
  While the episode is finished, do:

- Being in a state S_t, act $A_t \sim \pi(\cdot|S_t)$, where $\pi = \varepsilon\text{-greedy}(Q^{\theta})$, get a reward R_t, and transfer to a next state S_{t+1}.
  Store $(S_t, A_t, R_t, S_{t+1}) \to Memory$
- Get a minibatch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{n} \leftarrow Memory$, determine target values

  $$y_j = \begin{cases} r_j, & \text{if } s'_i \text{ is final,} \\ r_j + \gamma \max_{a'} Q^{\theta}(s'_j, a'), & \text{otherwise} \end{cases}$$

  a loss function $Loss(\theta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - Q^{\theta}(s_i, a_i)\big)^2$ and update θ:

  $$\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$$
- Decrease ε

Continuous DQN (Normalized Advantage Functions)

Gu S., Lillicrap T., Sutskever I., Levine S. Continuous Deep Q-Learning with Model-based Acceleration. 2016.

$$Q^{\theta}(s,a) = V^{\theta_V}(s) + A^{\theta_A}(s,a), \quad \theta = (\theta_V, \theta_A)$$

where

$$A^{\theta_A}(s,a) = -(a - \mu^{\theta_\mu}(s))^T P^{\theta_P}(s)(a - \mu^{\theta_\mu}(s)), \quad \theta_A = (\theta_\mu, \theta_P)$$

where

$$P^{\theta_P}(s) = L^{\theta_P}(s) L^{\theta_P}(s)^T$$

- $\max\limits_a Q^{\theta}(s,a) = V^{\theta_V}(s)$
- $\mathrm{argmax}_a Q^{\theta}(s,a) = \mu^{\theta_\mu}(s)$

Plaksin A., Martyanov S. Continuous Deep Q-Learning in Optimal Control Problems: Normalized Advantage Functions Analysis. NeurIPS 2022.