Q – Learning

$State, s \in S$

$Action, a \in A(s)$

$Reward, r \in R(s, a, s')$

$Transition\ Probability, P(s, a, s')$

$Optimal\ Policy, \pi^*(s)$

$Return, G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

$State\ value, V^{\pi}(s)$

$Action\ value, Q^{\pi}(s, a)$

Optimal value function:

$V^*(s) = \max_a \sum_{s'} P(s'|s, a)\big(r + \gamma V^*(s')\big)$

Value – iteration:

$V^*_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)[r + \gamma V^*_k(s')]$

Bellman optimal equation:

$Q^*(s, a) = \sum_{s'} p(s'|s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$

Q – value iteration:

$Q_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, a,)\left[r + \gamma \cdot \max_{a'} Q_i(s', a')\right]$

Policy Evaluation

$V^{\pi_k}_{i+1}(s) = \sum_{s'} P(s'|s, \pi_k(s))\left[r + \gamma V^{\pi_k}_i(s')\right]$

Policy Improvement

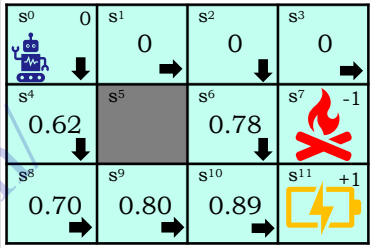
$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a)[r + \gamma V^{\pi_k}(s')]$

Q – Learning

- Goal: To estimate Q – function
- Solution approaches:
  - Tabular method
  - Function Approximator: Neural Networks

Q – Table Update:

$$Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \alpha\left(r + \gamma \cdot \max_{a'} Q_i(s', a') - Q_i(s, a)\right)$$



- Discrete States
- Discrete Actions

$Q(s,a)$	→	←	↑	↓
s_0				
s_1				
s_2				
s_3				
s_4				
s_6				
s_8				
s_9				
s_{10}				

Algorithm 1: Q – Learning for finite states

Initialize Q parameters: $\alpha, \in (\mathbf{0}, \mathbf{1}], \mathbf{s} \in \mathcal{S}^+, \mathbf{a} \in \mathcal{A}(\boldsymbol{s}), \boldsymbol{Q}(\boldsymbol{s^T}, \cdot) = \mathbf{0}$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$
  For t = 1, T **do**
    with probability ϵ select a random action $a_t \in \mathcal{A}(s_t)$
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and next state s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)\right]$
    $s \leftarrow s'$
    If $s \leftarrow s^T$
      break For
    end If
  end For
end For

$Q(s,a)$	→	←	↑	↓
s_0				
s_1				
s_2				
s_3				
s_4				
s_6				
s_8				
s_9				
s_{10}				

https://www.profrkumar.co.in/

Q – Learning: Predict Q-values with a Neural Network

$State, s \in S$

$Action, a \in A(s)$

$Reward, r \in R(s, a, s')$

$Transition\ Probability, P(s, a, s')$

$Optimal\ Policy, \pi^*(s)$

$Return, G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

$State\ value, V^{\pi}(s)$

$Action\ value, Q^{\pi}(s, a)$

Optimal value function:

$V^*(s) = \max_a \sum_{s'} P(s'|s, a)(r + \gamma V^*(s'))$

Value – iteration:

$V^*_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)[r + \gamma V^*_k(s')]$

Bellman optimal equation:

$Q^*(s, a) = \sum_{s'} p(s'|s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$

Q – value iteration:

$Q_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, a,)\left[r + \gamma \cdot \max_{a'} Q_i(s', a')\right]$

Policy Evaluation

$V^{\pi_k}_{i+1}(s) = \sum_{s'} P(s'|s, \pi_k(s))[r + \gamma V^{\pi_k}_i(s')]$

Policy Improvement

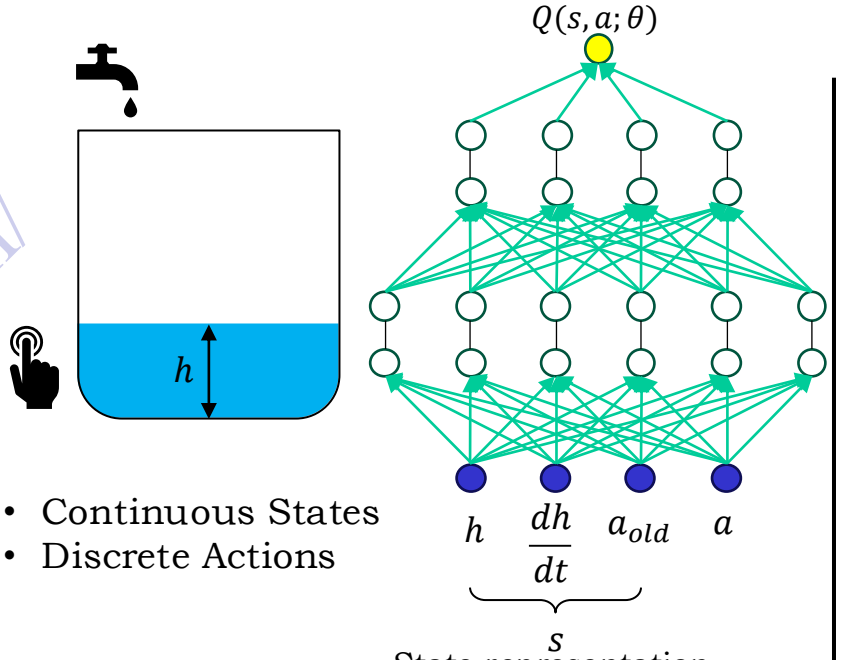
$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a)[r + \gamma V^{\pi_k}(s')]$

Neural Network (for regression)

- Input: State
- Output: Q-values of various actions.
- Learning: Gradient descent of squared temporal difference error/Bellman error loss.
  - $\left(\left(R(s, a, s') + \gamma \max_{a'} Q(s', a')\right) - Q(s, a)\right)^2$

Algorithm 1: Deep Q – Learning v1 **(DQNv1)**

Initialize Q with random weights
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$
  For t = 1, T **do**
    with probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and next state s_{t+1}
    Perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$
    $\theta \leftarrow \theta - \alpha \frac{dQ_\theta(s_i, a_i)}{d\theta}\left(Q_\theta(s_i, a_i) - \left[r_i + \gamma \max_{a'} Q_\theta(s'_i, a'_i)\right]\right)$
  end For
end For



- Continuous States
- Discrete Actions

- Issue:
  - Sequential states are strongly correlated (not i.i.d.)
  - Target value is always changing.
  - Gradient descent without fixed objective causes instability or Q-value divergence.

https://www.profrkumar.co.in/

Q – Learning: Replay Buffer & Target Estimator Update

$State, s \in S$

$Action, a \in A(s)$

$Reward, r \in R(s, a, s')$

$Transition\ Probability, P(s, a, s')$

$Optimal\ Policy, \pi^*(s)$

$Return, G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

$State\ value, V^{\pi}(s)$

$Action\ value, Q^{\pi}(s, a)$

Optimal value function:

$V^*(s) = \max_a \sum_{s'} P(s'|s, a)(r + \gamma V^*(s'))$

Value – iteration:

$V^*_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)[r + \gamma V^*_k(s')]$

Bellman optimal equation:

$Q^*(s, a) = \sum_{s'} p(s'|s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$

Q – value iteration:

$Q_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, a,)\left[r + \gamma \cdot \max_{a'} Q_i(s', a')\right]$

Policy Evaluation

$V^{\pi_k}_{i+1}(s) = \sum_{s'} P(s'|s, \pi_k(s))\left[r + \gamma V^{\pi_k}_i(s')\right]$

Policy Improvement

$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a)[r + \gamma V^{\pi_k}(s')]$

Issue with DQNv1:

- Sequential states are strongly correlated (not i.i.d.)

Experience Replay

- Q-learning is off-policy.
  - → Maintain buffer of past experiences.
  - → Update Q-values by utilizing a sample obtained from the replay buffer.
  - → Advantages:
    - Breaks correlations between consecutive samples.
    - Multiple gradient updates are affected with each step's execution.

Replay Buffer
$\{s_1, a_1, r_1, s_2\}$
$\{s_2, a_2, r_2, s_3\}$
…
$\{s_j, a_j, r_j, s_{j+1}\}$

Algorithm 2: Q – Learning with Neural Network (Online) & Replay Buffer $\mathcal{D}$ | Deep Q-learning v2

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize Q with random weights
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$
  For t = 1, T **do**
    with probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Perform a gradient descent step:
    $\theta \leftarrow \theta - \alpha \sum_j \frac{dQ_\theta(s_i, a_i)}{d\theta}\left(Q_\theta(s_i, a_i) - \left[r_i + \gamma \max_{a'} Q_\theta(s'_i, a'_i)\right]\right)$
  end For
end For

https://www.profrkumar.co.in/

Q – Learning: Replay Buffer & Target Estimator Update

$State, s \in S$

$Action, a \in A(s)$

$Reward, r \in R(s, a, s')$

$Transition\ Probability, P(s, a, s')$

$Optimal\ Policy, \pi^*(s)$

$Return, G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

$State\ value, V^{\pi}(s)$

$Action\ value, Q^{\pi}(s, a)$

Optimal value function:

$V^*(s) = \max_a \sum_{s'} P(s'|s, a)(r + \gamma V^*(s'))$

Value – iteration:

$V^*_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)[r + \gamma V^*_k(s')]$

Bellman optimal equation:

$Q^*(s, a) = \sum_{s'} p(s'|s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$

Q – value iteration:

$Q_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, a,)\left[r + \gamma \cdot \max_{a'} Q_i(s', a')\right]$

Policy Evaluation

$V^{\pi_k}_{i+1}(s) = \sum_{s'} P(s'|s, \pi_k(s))\left[r + \gamma V^{\pi_k}_i(s')\right]$

Policy Improvement

$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a)[r + \gamma V^{\pi_k}(s')]$

Issue with DQNv2:

- Target value is always changing.
  - Gradient descent without fixed objective causes instability or Q-value divergence.

Update Target Network

- Targets don't change inside the inner-loop (episodic loop)
- Stable targets lead to SUPERVISED REGRESSION.

Algorithm 3: Q – Learning with Neural Network (Online) & Replay Buffer $\mathcal{D}$ | Deep Q-learning v3

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize Q with random weights
Update $\theta' \leftarrow \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$
  For t = 1, T **do**
    with probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) \end{cases}$
    Perform gradient descent: $\left(y_j - Q(s_j, a_j; \theta)\right)^2$
    $\theta \leftarrow \theta - \alpha \sum_j \frac{dQ_\theta(s_i, a_i)}{d\theta}\left(Q_\theta(s_i, a_i) - \left[r_i + \gamma \max_{a'} Q_{\theta'}(s'_i, a'_i)\right]\right)$
  end For
end For

- Solution
  - Use two neural networks to provide STABILITY.
  - Net-1: Update regularly
  - Net-2: Update occasionally

$$\left(Q_\theta(s, a) - \left(r_i + \gamma \max_{a'} \boldsymbol{Q_{\theta'}(s', a')}\right)\right)^2$$

https://www.profrkumar.co.in/

Deep Q – Network

$State, s \in S$

$Action, a \in A(s)$

$Reward, r \in R(s, a, s')$

$Transition\ Probability, P(s, a, s')$

$Optimal\ Policy, \pi^*(s)$

$Return, G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

$State\ value, V^{\pi}(s)$

$Action\ value, Q^{\pi}(s, a)$

Optimal value function:

$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left(r + \gamma V^*(s')\right)$

Value – iteration:

$V^*_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[r + \gamma V^*_k(s')]$

Bellman optimal equation:

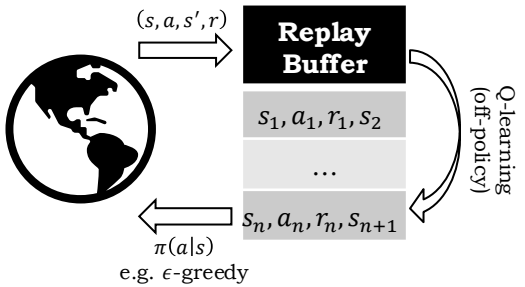
$Q^*(s,a) = \sum_{s'} p(s'|s,a)\left[r + \gamma \max_{a'} Q^*(s',a')\right]$

Q – value iteration:

$Q_{i+1}(s) \leftarrow \sum_{s'} P(s'|s,a,)\left[r + \gamma \cdot \max_{a'} Q_i(s',a')\right]$

Q – Learning:

$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha\left(r + \gamma \cdot \max_{a'} Q_i(s',a') - Q_i(s,a)\right)$

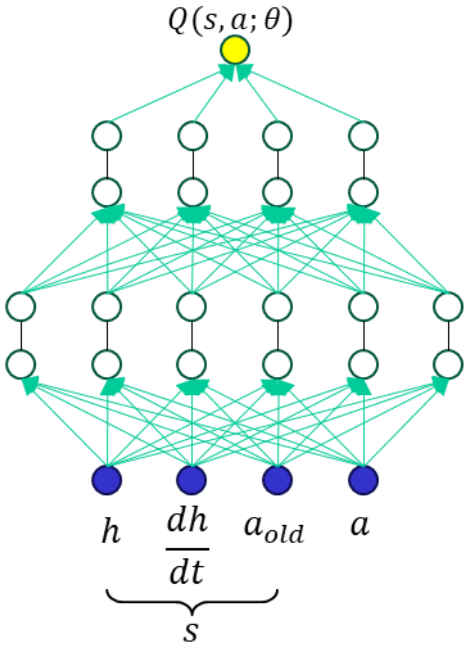


- ## Q – Network Update:
  - $\theta^{new} \leftarrow \theta - \alpha \cdot \nabla_\theta U(\theta; s, a)$
  - $U_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}[y_i - Q(s,a;\theta_i)]^2$
  - $y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r_t + \gamma \cdot \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a\right]$
  - $\rho(s,a) \rightarrow$ behavior distributions
  - Model free: no explicit construction of an estimator for $\mathcal{E}$
  - Greedy: $a^* = \max_a Q(s,a;\theta)$

- ## Experience Replay
  - $e_t = (s_t, a_t, r_t, s_{t+1}) \rightarrow$ agent's experience
  - $\mathcal{D} = e_1, e_2. e_3, \ldots, e_N \rightarrow$ replay memory
  - Q – Network:
    - Input: State / Observation
    - Output: Predicted Q values of individual actions.

Algorithm 4: Classic Deep Q – Learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize Q with random weights
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ & pre-processed sequence $\phi_1 = \phi(s_1)$
  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 next state s_{t+1}
    Preprocess $\phi_{t+1} = \phi(s_t, a_t, s_{t+1}) \rightarrow$ last four frames of history (stacked)
    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 $y_j = \begin{cases} r_j & ; for\ terminal\ \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & ; for\ non-terminal\ \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$
  end For
end For
Update $\theta' \leftarrow \theta$ after N steps



https://www.profkumar.co.in/

Mnih, Volodymyr, et al. "Playing Atari with deep reinforcement learning,"arXiv preprint arXiv: 1312.5602 (2013)