

1 Deep reinforcement Q-learning

Paper 1: Playing Atari with Deep Reinforcement Learning:

<https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>

Paper 2: Human-level control through deep reinforcement learning:

<https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf>

1.1 Optimal action-value function

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} \mathbf{E}[R_t | s_t = s, a_t = a, \pi] \\ &= \max_{\pi} \mathbf{E}_{s' \sim \varepsilon} [r + \gamma \max_{a'} Q^*(s', a') | s, a] \end{aligned}$$

with:

- Environment: ε
- State (observation): s
- Action: $a \in$ set of discrete actions $A = \{1, 2, \dots, k\}$
- Policy mapping observation (s) to action (a): $\pi = P(a|s)$
- Reward: r , Discount factor: $\gamma \in]0, 1[$, Horizon: T
- Future discounted return at time t :

$$R_t = \sum_{t'=t}^T \gamma^{(t'-t)} \cdot r_{t'} = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots$$

1.2 Tabular Q-learning

Update the Q-table with the Bellman equation at each iteration i :

$$Q_{i+1}(s, a) = \mathbf{E}[r + \gamma \max_{a'} Q_i(s', a') | s, a]$$

$$Q_i \longrightarrow Q^* \text{ when } i \longrightarrow \infty$$

Problems:

- No generalisation: Q is estimated separately at each iteration.
- No scaling: The MDP and Q-table grow very fast very large.
- No learning in large state space: every state is a new state.

1.3 DQN

1.3.1 Problem with function approximators

Non-linear Q-function approximators (such as NN) make RL unstable:

- Consecutive actions in the observation sequence are highly correlated.
- Small updates to Q may significantly change the policy.

Solution, 2 key ideas:

- Replay memory.
- Target network.

1.3.2 Replay memory

Role: Removing correlations in the observation sequences and smoothing over changes in the data distribution.

Dataset of agent experiences over N last timesteps, with:

- Experience at timestep t : $e_t = (s_t, a_t, r_t, s_{t+1})$
- Replay memory dataset of past experiences: $D = \{e_{t-1}, e_{t-2}, \dots, e_{t-N}\}$

The Q-network is updated over a sample of mini-batch experiences linearly drawn at random from the pool of stored experiences in the replay memory.

1.3.3 Target network

Role: Adjusting the action-values toward target values that are only periodically updated, thereby reducing correlations with the targets. Generating the targets using an older set of weights adds a delay between an update to Q and the time this update affects the targets y_i , reducing divergence.

Target network \widehat{Q} , with:

- Weights: θ_i^-
- Target network update frequency: C

The Q-network, with weights θ_i is cloned every C steps to obtain the target network \widehat{Q} , with weights θ_i^- . This separate network is held fixed for the next C updates and is used for generating the targets y_i in the Q-learning updates.

1.3.4 Learning strategy

Off-policy, ϵ -greedy strategy:

$$a = \begin{cases} \operatorname{argmax}_a Q(s, a, \theta) & \text{with probability } 1 - \epsilon \\ \text{rand } a \in A & \text{with probability } \epsilon \end{cases}$$

Model-free: does not construct an estimate of the environment ε .

1.3.5 State space

A state is a sequence of observations and actions, with:

- Observation: x
- State: $s_t = (x_1, a_1, x_2, a_2, \dots, x_{t-1}, a_{t-1}, x_t) = (s_{t-1}, a_{t-1}, x_t)$
- State observation preprocessing function: $\phi_s = \phi(s) \sim s$

Assuming a finite horizon, the MDP is large but finite.

1.3.6 Q-Network

Neural network function approximator with weights θ :

$$Q(s, a, \theta) \approx Q^*(s, a)$$

Loss function at iteration i :

$$\begin{aligned} L_i(\theta_i) &= \mathbf{E}_{(s,a,r,s') \sim U(D)} [(y_i - Q(s, a, \theta_i))^2] \\ &= \mathbf{E}_{(s,a,r,s') \sim U(D)} [(r + \gamma \cdot \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i))^2] \end{aligned}$$

with:

- Q updates on samples (mini-batch) of experiences (s,a,r,s') : $U(D)$
- Target: $y_i = r + \gamma \cdot \max_{a'} Q(s', a', \theta_i^-)$
 - Weights of the target network: θ_i^-
 - Error clipping between -1 and 1
 - The target y_i changes over time, contrary to supervised learning
- Minimize with mini-batch gradient descent.

Gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbf{E}_{(s,a,r,s') \sim U(D)} [(r + \gamma \cdot \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)) \cdot \nabla_{\theta_i} Q(s, a, \theta_i)]$$

1.4 DQN algorithm

DQN algorithm

```
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 sequence  $s_1 = x_1$  and preprocessed sequence  $\phi_1 = \phi(s_1)$ 
    for  $t = 1, T$  do
        With probability  $\epsilon$  select a random action  $a_t$ 
        otherwise select action  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a, \theta)$ 
        Execute  $a_t$  in emulator  $\varepsilon$  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  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
        Set  $y_i = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \cdot \max_{a'} \hat{Q}(\phi_{j+1}, a', \theta^-) & \text{otherwise} \end{cases}$ 
        Perform a gradient descent step on  $(y_i - Q(\phi_j, a_j, \theta))^2$  w.r.t.  $\theta$ 
        Every  $C$  steps reset  $\hat{Q} = Q$ 
    end for
end for
```

1.5 Atari Q-network

1.5.1 Input

Fixed length state representation preprocessed by function ϕ , with:

- Size: $32 \times (84 \times 84 \times 4)$:
 - Mini-batch size: 32
 - Image cropped and reduced x : 84×84 pixels
 - Agent history length: 4 frames

1.5.2 Neural network

CNN:

- CONV1: $\text{hwc} = 8 \times 8 \times 32, \text{s}=4$

- Relu
- CONV2: $\text{hwc} = 4 \times 4 \times 64$, $\text{s}=2$
- Relu
- CONV3: $\text{hwc} = 3 \times 3 \times 64$, $\text{s}=1$
- Relu
- FC1: 256 logits
- Relu
- FC2: [4,18] logits

with:

- Optimizer: RMSprop or Adam
- Loss function: huber (not quadratic)

1.5.3 Output

Actions:

- Predicted Q-values of the individual action for the input state.
- Compute Q-values for all actions in a state with a single forward pass.

1.5.4 Hyperparameters

- Training: 10000000 timesteps
- Replay memory dataset size: $N = 1000000$ timesteps
- Target network update frequency: $C = 10000$ timesteps
- Reward: $r \in \{-1, 0, 1\}$
- Frame Skipping: $k = 4$ frames:
 - The agent sees 1/4 frames and the action is repeated over 4 frames.
- α , γ , ϵ , RMSprop, other: see paper 2.

1.6 Double DQN

Paper 3: Deep Reinforcement Learning with Double Q-learning:
<https://arxiv.org/pdf/1509.06461.pdf>

1.6.1 Overestimation in DQN

DQN sometimes learns unrealistically high value functions because it includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values. The max operator uses the same values to select and to evaluate an action for the targets, i.e. $\max_{a'} Q(s', a', \theta_i^-)$, resulting in overoptimistic value estimates and suboptimal policies.

1.6.2 Decoupling action selection and action evaluation

Double DQN solution to overestimation:

- Decoupling action selection and action evaluation for the target values.

Double DQN reduces overestimation by decomposing the max operation when approximating the target values into action selection and action evaluation:

- The action is selected by the online network.
- The action is evaluated by the target network.

1.6.3 Double DQN target

DQN target:

$$y_i = r + \gamma \cdot \max_{a'} Q(s', a', \theta_i^-)$$

Double DQN target:

$$y_i = r + \gamma \cdot Q(s', \operatorname{argmax}_{a'} Q(s', a', \theta_i), \theta_i^-)$$

1.6.4 Hyperparameters

- Target network update frequency: $C = 30000$ timesteps
- Mostly unchanged from DQN: see paper 3.