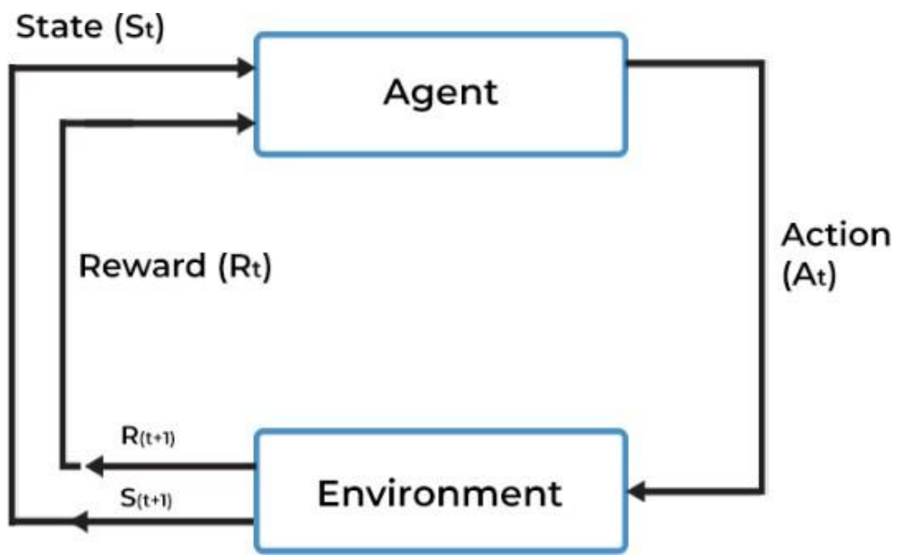
Deep Q Learning

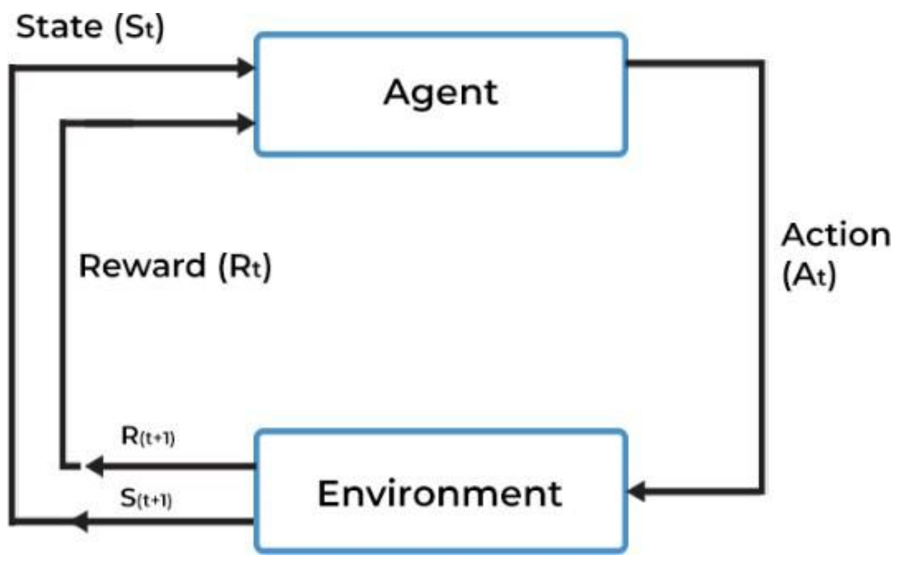
Reinforcement learning

- The challenges encompassing an agent's interaction with an environment, wherein numeric rewards are given.
- Objective of acquiring the ability to make decisions that optimize the reward outcome.



Reinforcement learning

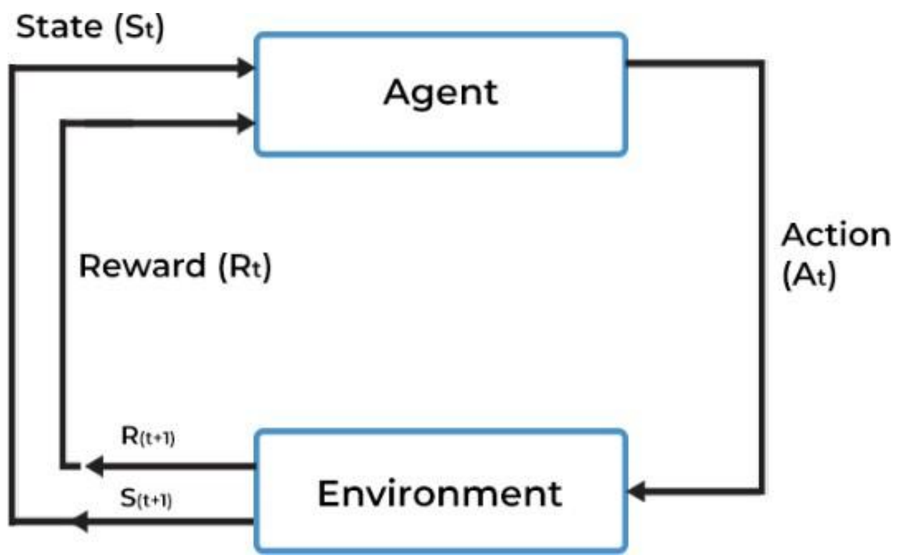
- The challenges encompassing an agent's interaction with an environment, wherein numeric rewards are given.
- Objective of acquiring the ability to make decisions that optimize the reward outcome.

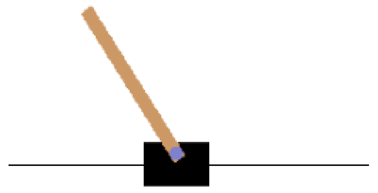


Typically in a RL system one observes the following sequence state, action, reward, new state…

Reinforcement learning

Examples

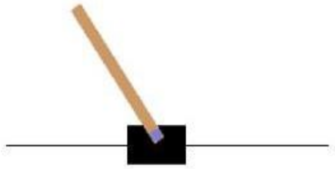


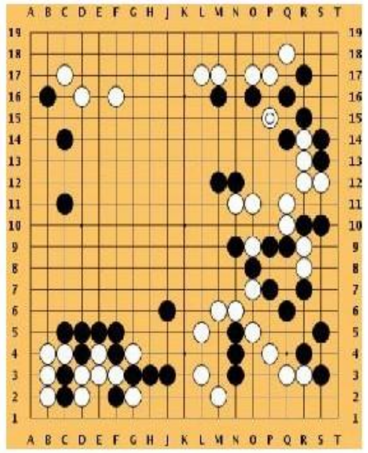


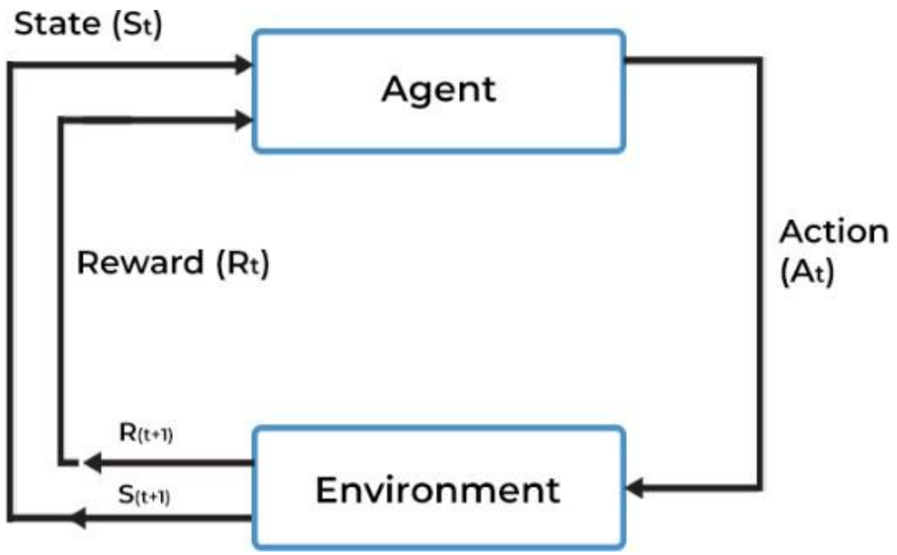
Objective: Balance a pole on top of a movable cart
State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Reinforcement learning

Examples







Markov Decision Process

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Process

At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
Then, for t=0 until done:

  Agent selects action a_t
  Environment samples reward $r_t \sim R(\,.\,|s_t\,,a_t\,)$
  Environment samples next state $s_{t+1} \sim P(\,.\,|s_t\,,a_t\,)$
  Agent receives reward rt and moves to next state s_{t+1}

A policy **π** is a function from S to A that specifies what action to take in each state –

Objective: find policy **π*** that maximizes cumulative discounted reward:

$$\sum_{t\geq 0} \gamma^t r_t$$

Markov Decision Process

Typically, we do not have access to most of the information given in an MDP.
Instead, what one typically have in practice is something along the line of the sequence state, action, reward, next state.

Q-function

The total reward is the discounted sum of all rewards obtained from time t:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Where:

- γ is the discount factor.
- R_t is the total reward.

The Q-function is defined as:

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

Where:

- $Q(s_t, a_t)$ captures the expected total future reward an agent in state s_t can receive by executing a certain action a_t.
- s_t is the state.
- a_t is the action.

Q-function

Given the Q function, how would the agent use it to navigate the environment:

Q-function

Given the Q function, how would the agent use it to navigate the environment:

Answer: the agent needs a policy $\pi(s)$ to infer the best action at its state, s

Q-function

Given the Q function, how would the agent use it to navigate the environment:

Answer: the agent needs a policy $\pi(s)$ to infer the best action at its state, s

Strategy : choose an action that maximizes the future reward

$$\pi^* (s) = argmax_a \, Q(s, a)$$

Q function

A practical solution to the MDP is given by solving the state-action value function :

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$ Bellman Equation

Q(s,a) can be understood as : how “good” a given state, action pair is.

Q function

A practical solution to the MDP is given by solving the state-action value function :

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$ Bellman Equation

The optimal policy can be found via :

$$\pi^*(s) = \operatorname{argmax} Q^*(s,a)$$

Q function

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$ Bellman Equation

How can we learn Q?

Q Learning

The most primitive form of the so-called Q-learning algorithm learns the correct Q-function by iteratively reducing the discrepancy between Q value estimations for two consecutive states. More precisely, the Q learning update rule is given by

$$Q(s_t, a_t) = r_t + \gamma max_a Q(s_{t+1}, a)$$

Q Learning

The most primitive form of the so-called Q-learning algorithm learns the correct Q-function by iteratively reducing the discrepancy between Q value estimations for two consecutive states. More precisely, the Q learning update rule is given by

$$Q(s_t, a_t) = r_t + \gamma max_a Q(s_{t+1}, a)$$

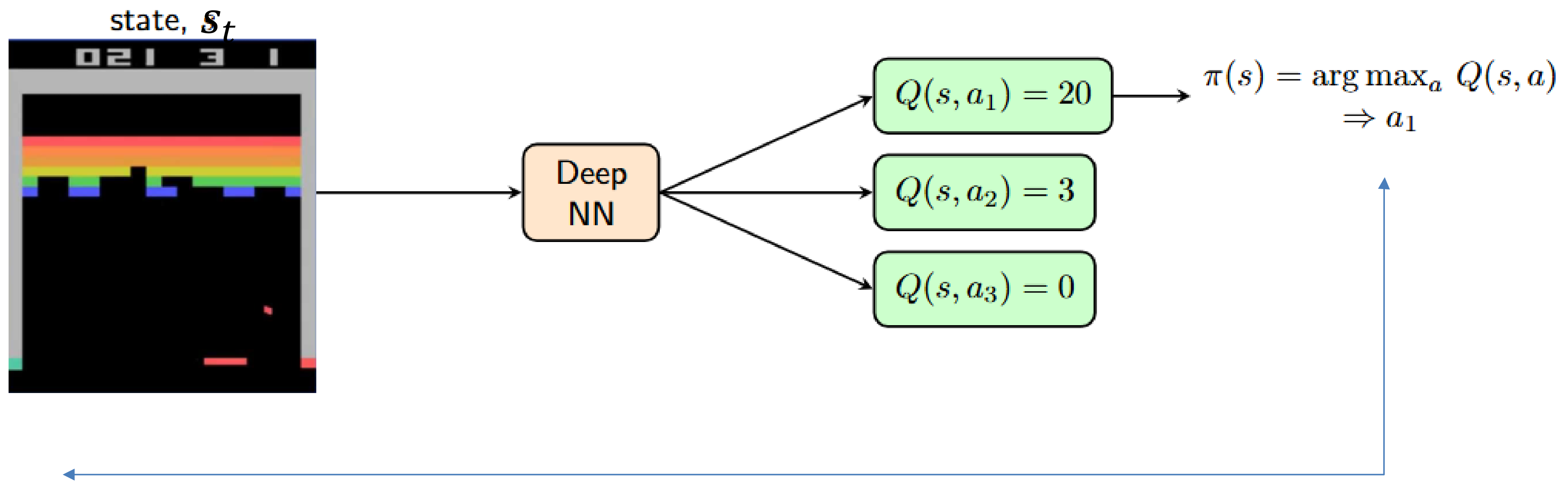
where s_t, a_t, r_t are the state, action, reward received by the agent at time t.

In a deep learning setting, we set the function Q(s, a) as a neural network Q(s, a, θ) and we try to optimize θ by specifying the loss

$$||Q(s_t, a_t) - (r_t + \gamma max_a Q(s_{t+1}, a))||^2$$

Atari game as an example

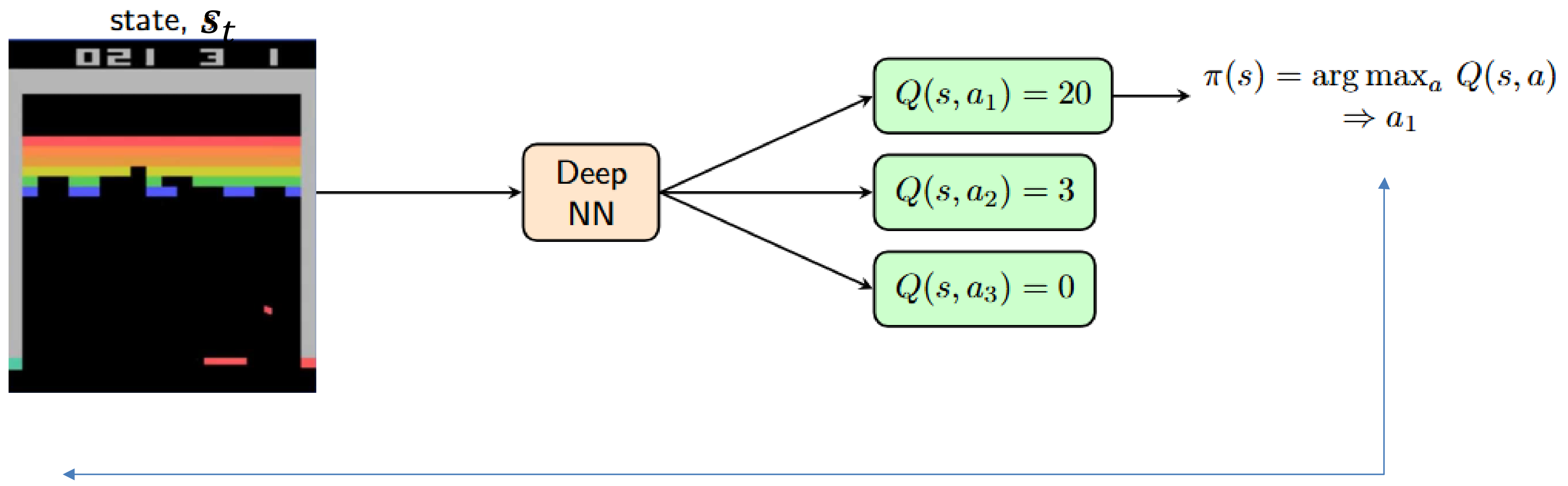
State is given as a pixel values and Q function is represented as a NN that takes as input the state and return Q(s,a) for all possible actions a. Then we take the max of these values to use it in the next state generation.



Use this action to move to the next state s_{t+1}

Atari game as an example

State is given as a pixel values and Q function is represented as a NN that takes as input the state and return Q(s,a) for all possible actions a. Then we take the max of these values to use it in the next state generation.



Use this action to move to the next state s_{t+1}

Finally use the Q function to optimize the function:

$$||Q(s_t,a_t)-(r_t+\gamma max_a Q(s_{t+1},a))||^2$$

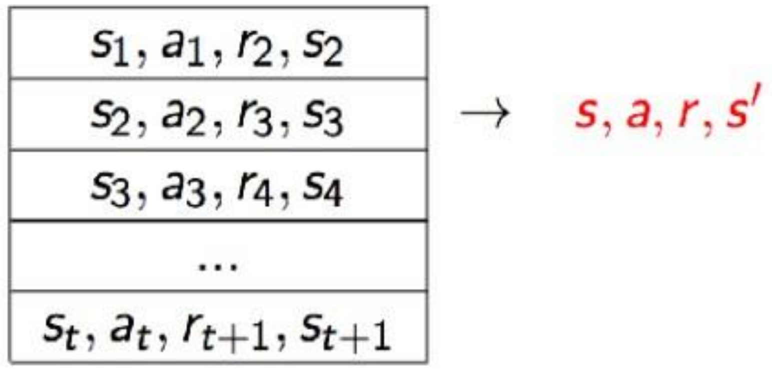
Q Learning

Problems:

1. Correlations between samples
2. 2. Non-stationary targets

Q-learning with experience replay

- To remove correlations, build data-set from agent's own experience



s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\dots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow$ s, a, r, s'

- Sample experiences from data-set and apply update

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^{-}) - Q(s, a, \mathbf{w}) \right)^2$$

- To deal with non-stationarity, target parameters w− are held fixed

Q-learning with experience replay

```
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** $t = 1,\mathrm{T}$ **do**
        With probability $\varepsilon$ 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 $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```

Initialize the replay memory and two identical Q approximators (DNN). $\hat{Q}$ is our target approximator.

Q-learning with experience replay

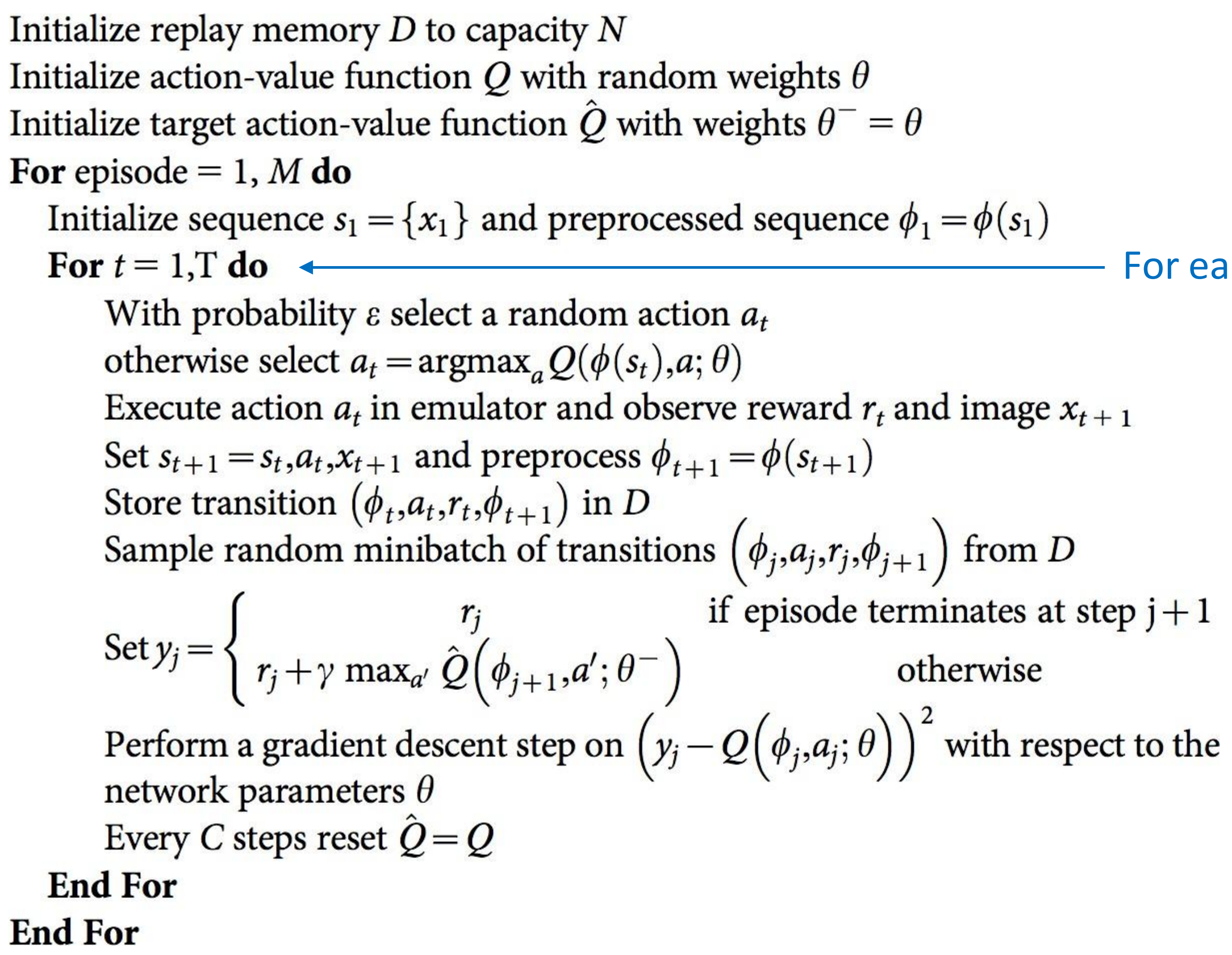
```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode = 1, $M$ **do**                                        Play *m* episodes (full games)
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    **For** $t$ = 1,T **do**
        With probability $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```

Q-learning with experience replay

```
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 $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

Start episode from x_1 (pixels at the starting screen).
Preprocess the state (include 4 last frames, RGB to grayscale conversion, downsampling, cropping)

Q-learning with experience replay



```
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 $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```

For each time step during the episode

Q-learning with experience replay

```
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 $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

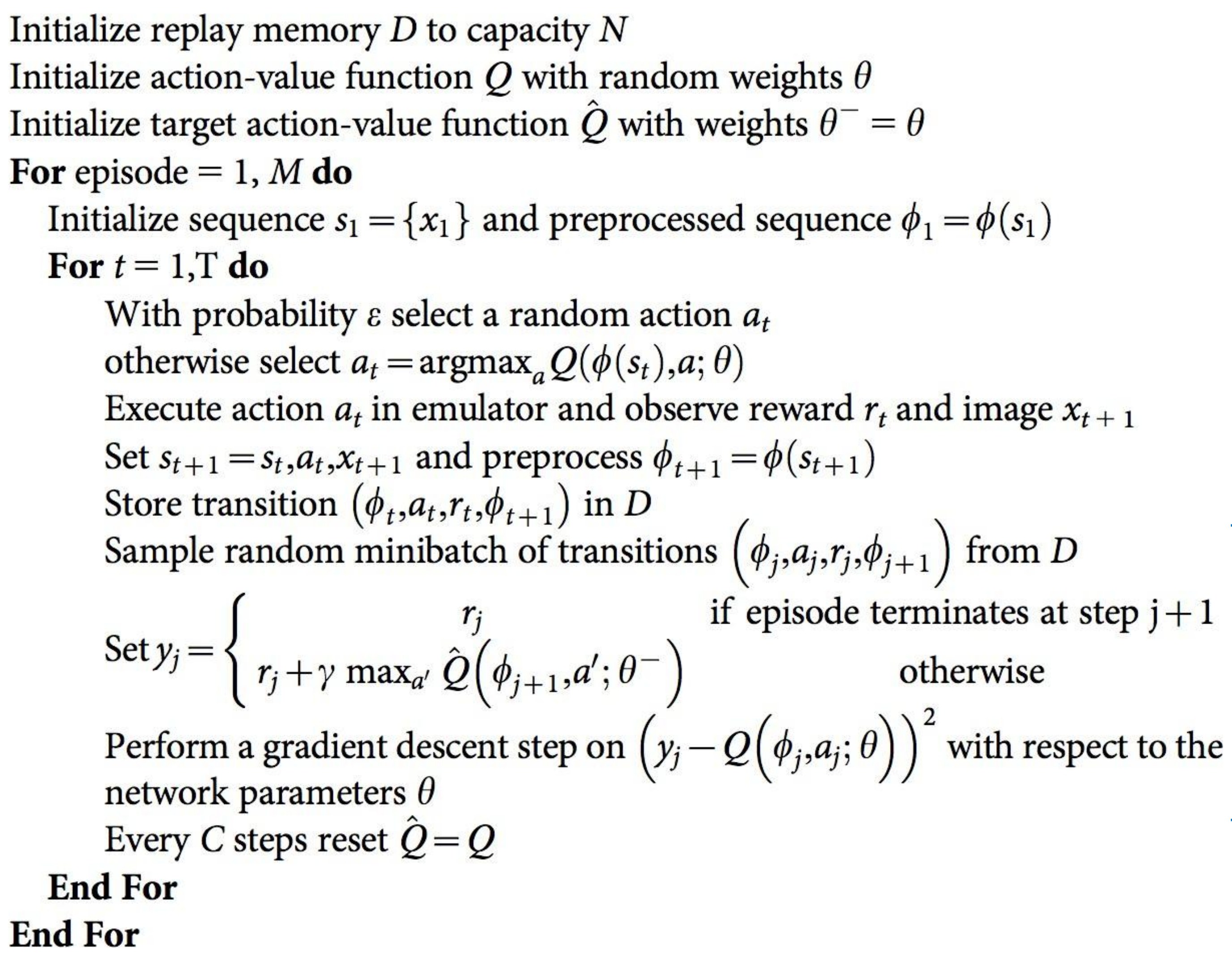
With small probability select a random action (explore), otherwise select the, currently known, best action (exploit).

Q-learning with experience replay

```
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 $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $(\phi_j, a_j, r_j, \phi_{j+1})$ 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 $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

Execute the chosen action and store the (processed) observed transition in the replay memory

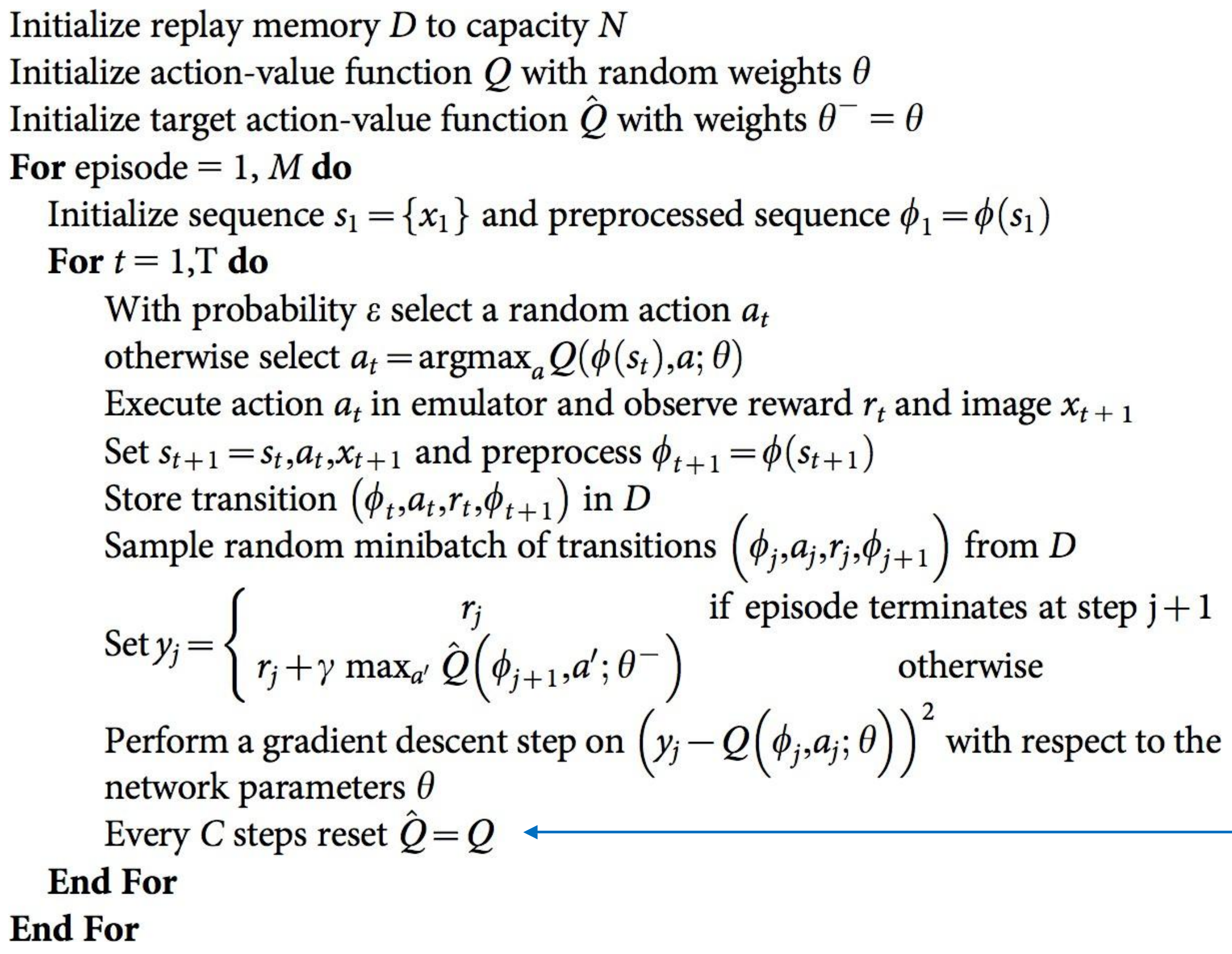
Q-learning with experience replay



```
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 $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
     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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
     Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
     Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Experience replay:
Sample a random minibatch of transitions from replay memory and perform gradient decent step on Q (not on $\hat{Q}$)

Q-learning with experience replay



```
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 $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Once every several steps set the target function, $\hat{Q}$, to equal Q

Q-learning with experience replay

```
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 $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Such delayed online learning helps in practice:
"This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases the target y_j, possibly leading to oscillations or divergence of the policy" [Human-level control through deep reinforcement learning. Nature 518.7540 (2015): 529.]

Refs

lecture_DQL.key (cmu.edu)

12DQN.pptx (live.com)

Lecture 6: CNNs and Deep Q Learning =1[1]With many slides for DQN from David Silver and Ruslan Salakhutdinov and some vision slides from Gianni Di Caro and images from Stanford CS231n, http://cs231n.github.io/convolutional-networks/