Mahindra™ University
Global Thinkers. Engaged Leaders.

Reinforcement Learning and Autonomous Systems (CS4122)

Lecture 26 (16/10/2024)
Lecture 27 (21/10/2024)
Lecture 29 (04/11/2024)
Lecture 30 (05/11/2024)
Lecture 31 (06/11/2024)
Instructor: Gourav Saha

Lecture Content

- Drawbacks of RL.

- Fundamental idea behind Deep-RL.

- Deep Q-Learning.
    - DQN Architecture 1.
    - DQN Architecture 2.

- Basic idea of training DQN.

- Training DQN
    - Online DQN
    - DQN with Replay Buffer
    - DQN with Replay Buffer and Target Network
    - Double DQN

Drawbacks of RL

- Before starting Deep-RL let's quickly recollect a bare basic concept, i.e. the minute we have the optimal Q-function or more realistically an estimate of the optimal Q-function $q(x, a)$, we can find the optimal policy or more realistically a near-optimal policy $\pi(x)$ as follows,

$$\pi\left(x\right) = \underset{a \in \mathcal{A}(x)}{\arg\max}\, q\left(x, a\right) \tag{1}$$

- Those RL and Deep-RL algorithms where the optimal policy is **implicitly captured by the Q-function** are called **value-based policies**.
  - You have encountered a very similar concept in contextual bandits where lin-greedy and lin-ucb algorithms were value-based policies because it was learning the context-action value.
  - The other approach is **policy gradient** which you have also studied in contextual bandits.

- We will first talk about value-based policies in Deep RL and hence we will just talk about learning the optimal Q-function. It should be understood that learning the optimal Q-function is same as learning the optimal policy because of equation (1).

Drawbacks of RL

SARSA:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x,a)$ to any real value.

Update step:

$$\mathrm{q}(x,a) = q(x,a) + \alpha\big(r + \beta q(x',a') - q(x,a)\big)$$

Q-Learning:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x,a)$ to any real value.

Update step:

$$q(x,a) = q(x,a) + \alpha\left(r + \beta \max_{a' \in \mathcal{A}(x')} q(x',a') - q(x,a)\right)$$

- We will now talk about the drawbacks of RL.
- Consider the initialization steps of SARSA and Q-Learning. We are initializing Q-values **for all state action pairs** **individually**.
- There are some **obvious** drawbacks with this approach:

  1) **Drawback 1: High memory requirement** to store Q-values for each state-action pair when the **number of state-action pair is large**.
     - This is not a major drawback really because even current day deep neural networks also have a lot of training parameters.

Drawbacks of RL

SARSA:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x,a)$ to any real value.

Update step:

$$\mathrm{q}(x,a) = q(x,a) + \alpha\big(r + \beta q(x',a') - q(x,a)\big)$$

Q-Learning:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x,a)$ to any real value.

Update step:

$$q(x,a) = q(x,a) + \alpha\left(r + \beta \max_{a' \in \mathcal{A}(x')} q(x',a') - q(x,a)\right)$$

- We will now talk about the drawbacks of RL.
- Consider the initialization steps of SARSA and Q-Learning. We are initializing Q-values **for all state action pairs** **individually**.
- There are some **obvious** drawbacks with this approach:

2) **Drawback 2:** Not applicable when **state space and action space is continuous** because in such case we need to store Q-values for **infinite** state-action pairs.
   - The basic Deep-RL algorithms that you will learn in this course can handle continuous state space bot not continuous action space.
   - There are algorithms like DDPG that can handle continuous action space also.

Drawbacks of RL

SARSA:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x, a)$ to any real value.

Update step:

$$\mathrm{q}(x,a) = q(x,a) + \alpha\big(r + \beta q(x', a') - q(x,a)\big)$$

Q-Learning:

Initialization step:

For all $x \in \mathcal{S}$ and all $a \in \mathcal{A}(x)$ arbitrarily initialize $\mathrm{q}(x, a)$ to any real value.

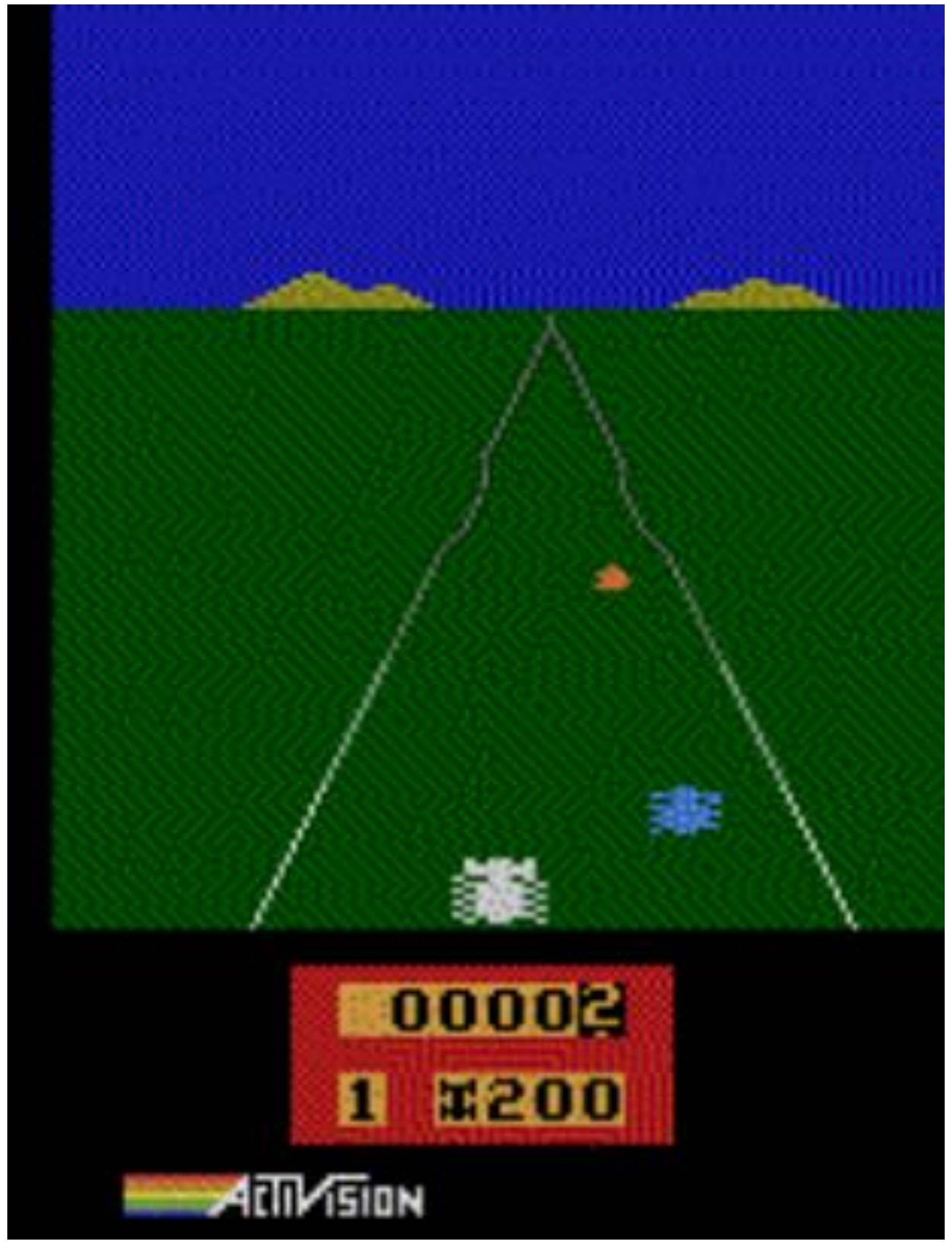
Update step:

$$q(x,a) = q(x,a) + \alpha\left(r + \beta \max_{a' \in \mathcal{A}(x')} q(x', a') - q(x,a)\right)$$

- We will now talk about the drawbacks of RL.
- Consider the initialization steps of SARSA and Q-Learning. We are initializing Q-values **for all state action pairs individually**.
- There are a **"not so obvious"** drawback:

  3) **Drawback 3:** Updating Q-values of all the state-action pairs till **convergence to their optimal value is time consuming** when the number of state-action pairs is large. This situation gets even more involved when we realize that update of **Q-values of state-actions pairs are coupled**, i.e. the "goodness" of update of $q(x, a)$ is dependent on how good $q(x', a')$ is.

Drawbacks of RL



- To understand how large the number of state-action pairs can get, let's take two examples.

Example-1 (Enduro)

- It is a Atari racing game. This environment is there in OpenAI Gym:

  https://www.gymlibrary.dev/environments/atari/enduro/

- The observation is an image of dimension 250×160.
  - For the time being **let's assume that the observation itself is the state** (for this setup, there is good reason to believe that state and observation is not same. Can you reason why? Also, do you remember the difference between states and observation?)

Drawbacks of RL

- To understand how large the number of state-action pairs can get, let's take two examples.

Example-1 (Enduro)

- Each pixel of an 8-bit RGB image can assume $(2^8)^3$ values.
- There are $250 * 160$ pixels and hence the size of the state space is,

$$2^{8*3*250*160}$$

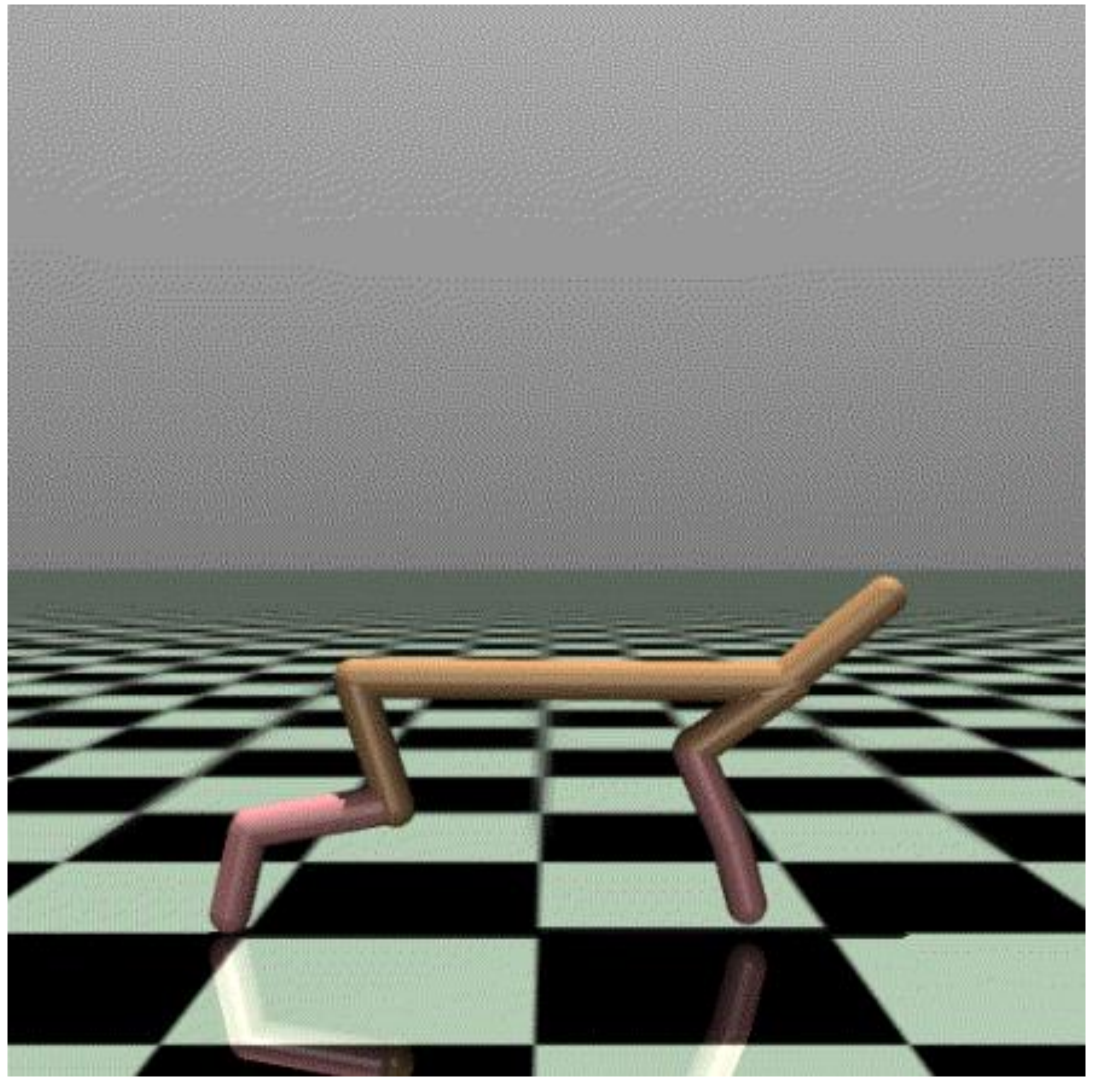
which is a huge number! The cardinality of the state space is the above number.

- There are 9 actions (check OpenAI Gym to know more about these actions) and hence the number of state-action pairs is,

$$(2^{8*3*250*160}) * 9$$

This is a huge number!

Drawbacks of RL



- To understand how large the number of state-action pairs can get, let's take two examples.

Example-2 (Half-Cheetah)

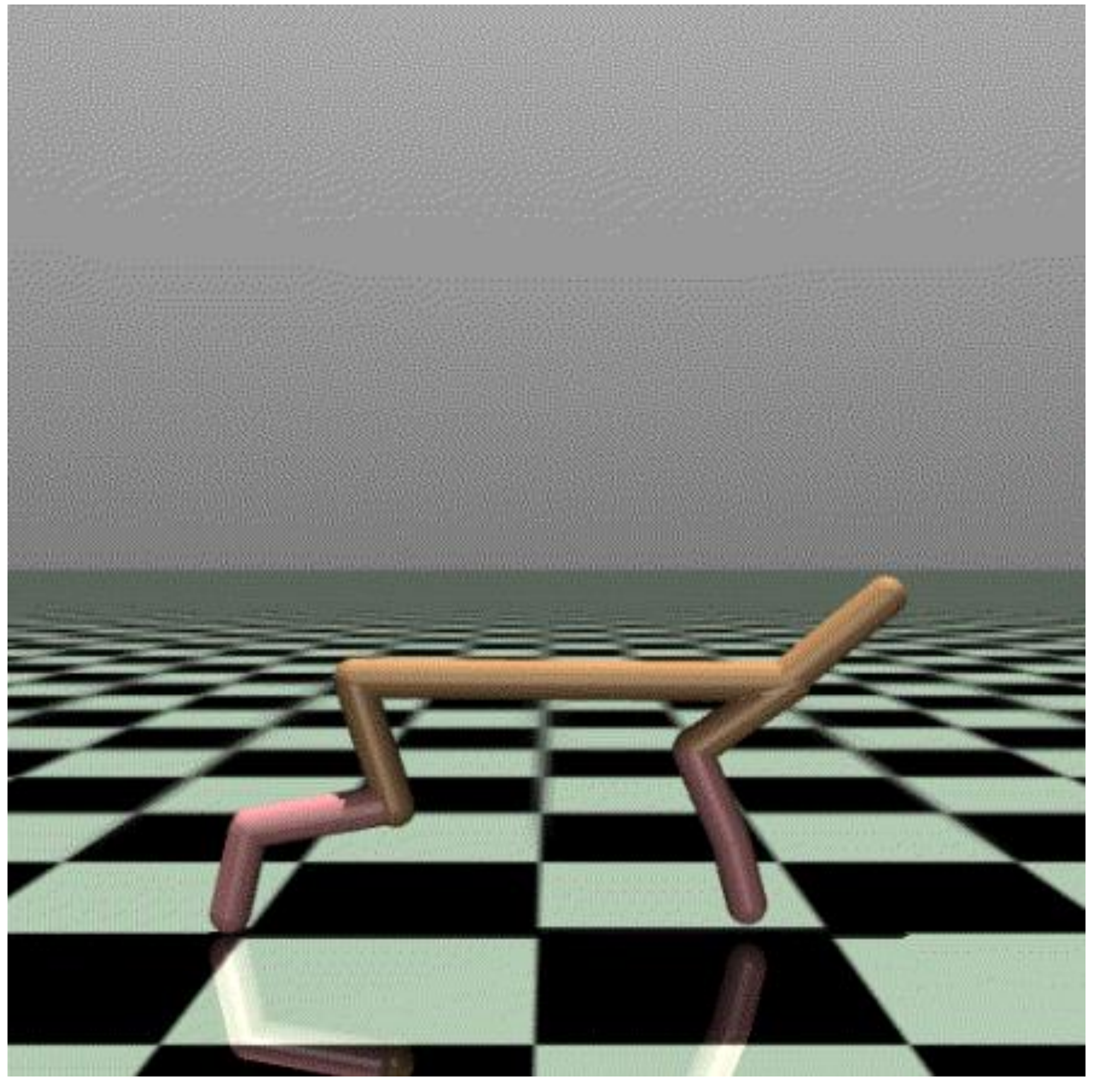
- This environment is there in OpenAI Gym:

https://www.gymlibrary.dev/environments/mujoco/half_cheetah/

The objective of this environment is to make the "cheetah like" robot run as fast as possible.

- The state space is continuous and it's dimension is **17**. The state space mainly consists of position and velocity of various joints.
- The action space is also continuous and it's dimension is 6. The action space consists of control torques to be applied at various joints.

Drawbacks of RL



- To understand how large the number of state-action pairs can get, let's take two examples.

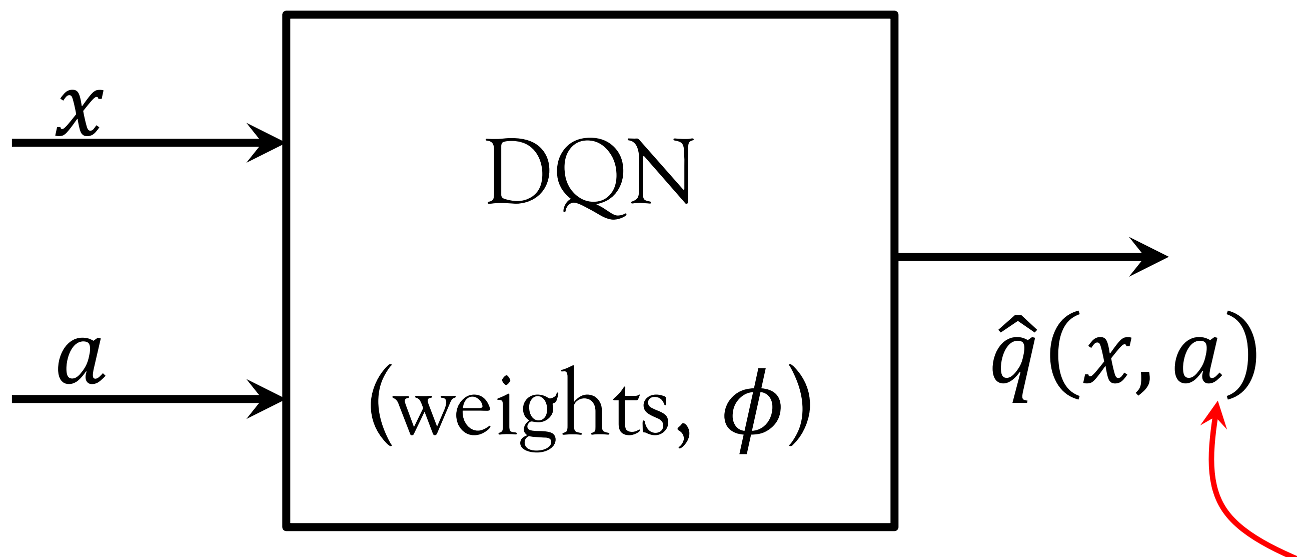
Example-2 (Half-Cheetah)

- Since we are dealing with continuous state and action space, we can't apply "conventional" RL techniques like Q-Learning and SARSA directly anyway. Why? Check this (click here) slide.

- We have to first discretize the state and action space.
  - Discretizing a continuous value means to put the value into discrete bins.

- Let's discretize each component of the state and action space in to 10 bins (my intuition suggest that 10 bins is actually less for this setup). Then the cardinality of state and action spaces are 10^{17} and 10^{6} respectively and hence the number of state-action pairs is $10^{17} \cdot 10^{6} = 10^{23}$ which is huge!

Fundamental Idea of Deep RL



Not writing ϕ to simplify notations.

- The fundamental idea behind Deep Q-Learning (among the most popular Deep-RL method) is to **not learn optimal Q-value for each state-action pair** but rather to **create a parameterized Q-function** and **learn the parameters of this Q-function**.

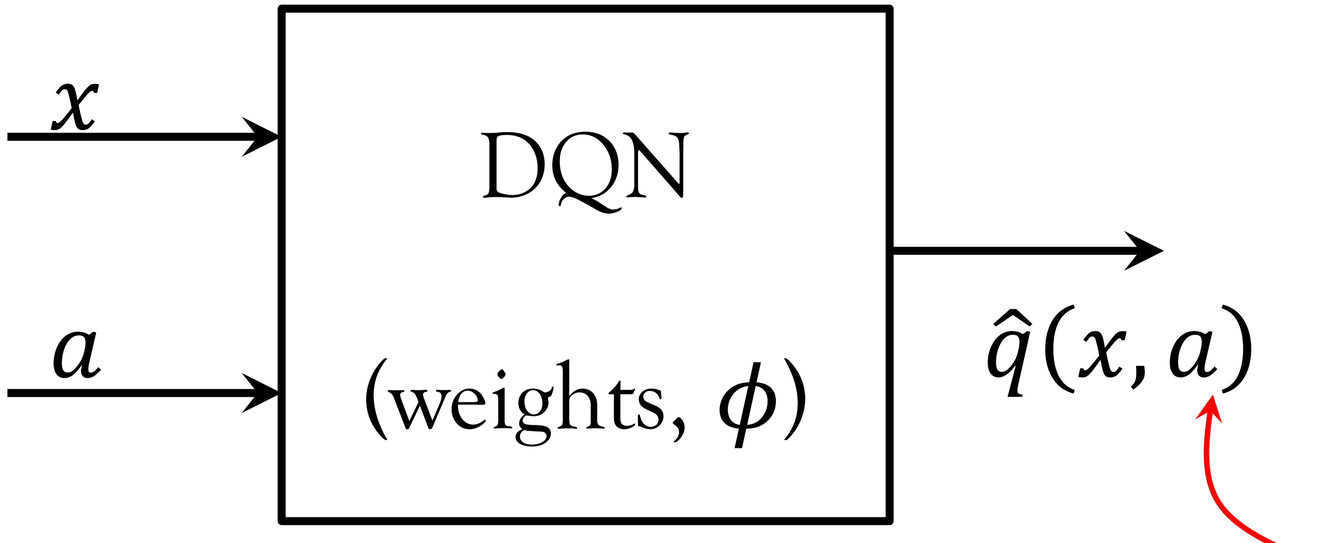
- Mathematically, we approximate $q(x, a)$ with a function $\hat{q}(x, a; \phi)$,

$$q(x,a) \approx \hat{q}(x,a;\phi)$$

  with ϕ as the parameters of this function, state x and action a as the inputs to the function, and output of this function is (optimal) Q-value for state-action pair (x, a).

- An obvious choice of such a function is a **neural network**. The neural network is called a Deep-Q network (DQN) and is shown in the left.

Fundamental Idea of Deep RL



Not writing ϕ to simplify notations.

To summarize:

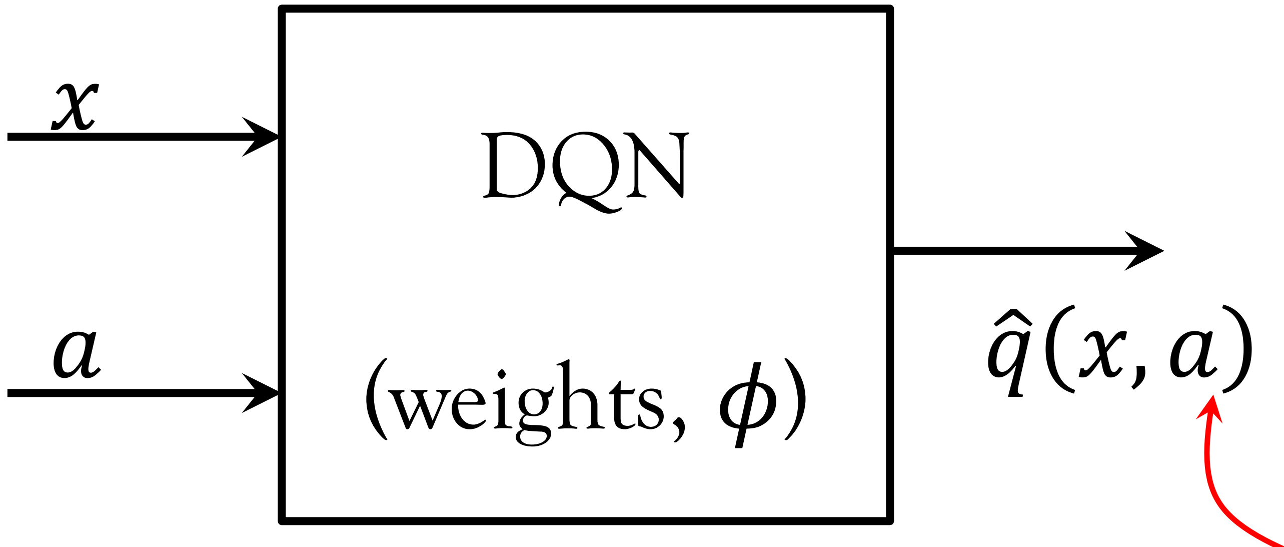
- In Q-Learning (and SARSA) we store $q(x,a)$ for each state-action pair. And, we learning $q(x,a)$ for each state-action pair.

- In Deep Q-Learning, we have a DQN parameterized by weights ϕ, $\hat{q}(x,a;\phi)$.
  - The weights ϕ is learned such that **mean squared error** between the **true optimal Q-value $q(x,a)$** and the **estimated optimal Q-value $\hat{q}(x,a;\phi)$** **across all state-action pair is minimized**. So the loss function is,

$$\mathcal{L}(\phi) = \sum_{x\in\mathcal{S}} \sum_{a\in\mathcal{A}(x)} \left(q(x,a) - \widehat{q}(x,a;\phi)\right)^2$$

    Note: This is not the exact loss function when we train the DQN. We minimize this loss function only in expectation.

Fundamental Idea of Deep RL

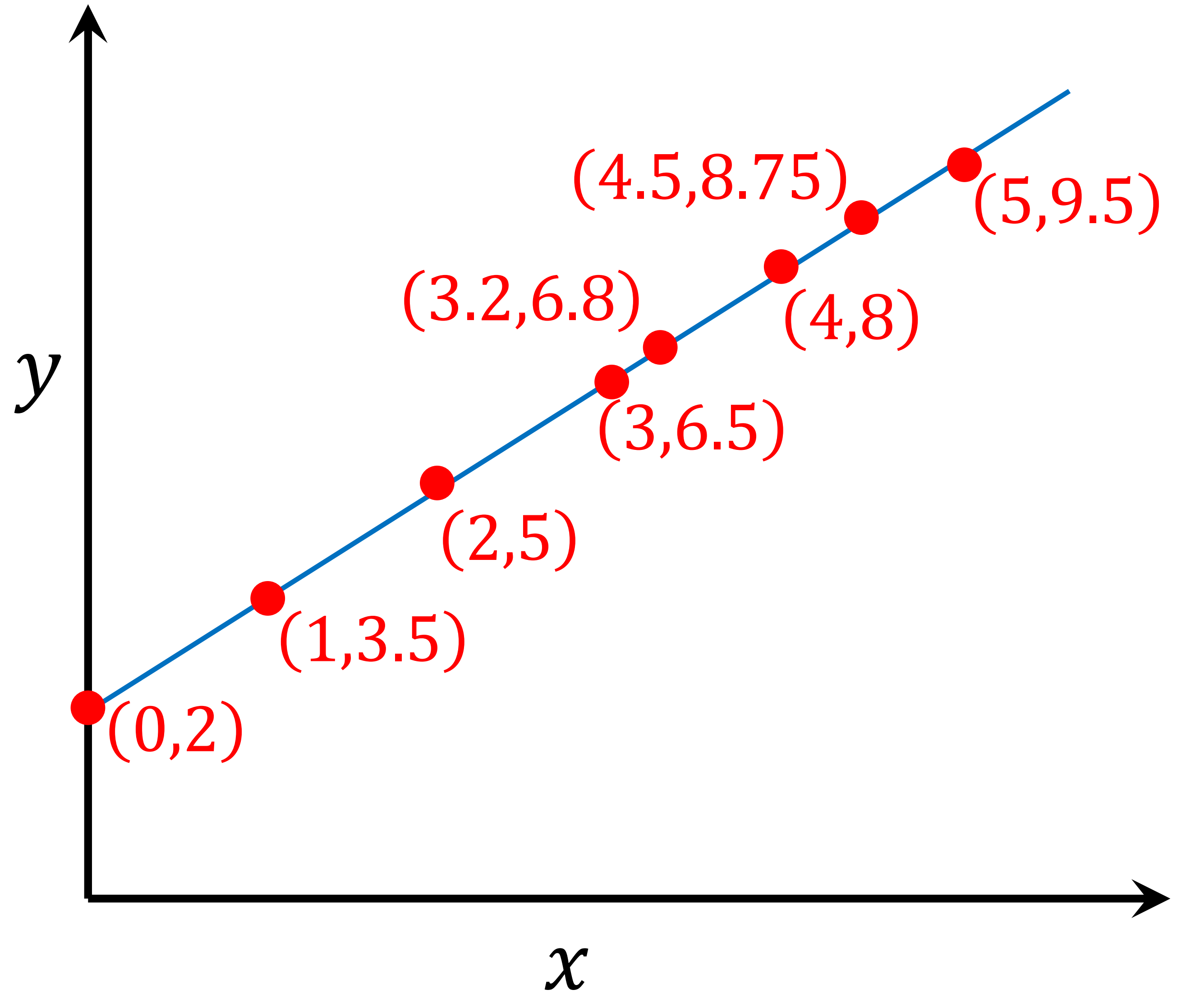


Not writing ϕ to simplify notations.

- In general, the number of parameters in ϕ is less than the number of state-action pairs. This leads to the following **advantages of Deep RL over RL**:
  - Lower memory requirement.
  - The convergence to the optimal policy MAY BE faster since the number of free parameters to learn is lesser.
- The other advantage of representing Q-values as parametrized functions is that a function captures **correlation among the dependent variable across different independent variables**.
  - In our case, Q-values are the dependent variables and state-action pairs are the independent variables.

Let's take a few examples to appreciate this point...

Fundamental Idea of Deep RL



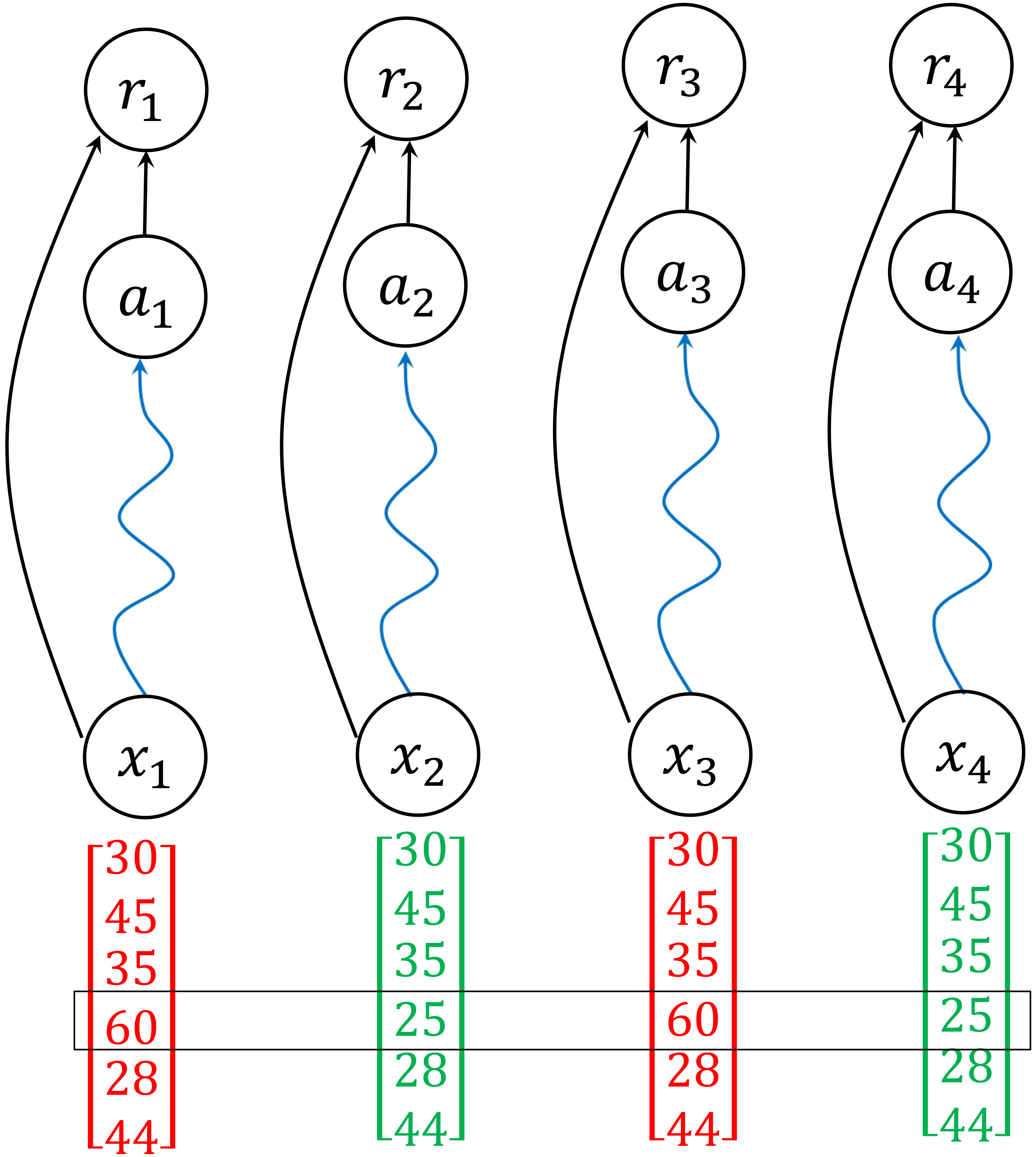
Example 1 (Line)

- A straight-line can be represented by enumerating all the (x, y) coordinates for all possible values of x. Example: The **blue** line shown in the left can be represented using the set of (x, y) coordinates shown in **red**.

- But, we never represent a line using (x, y) coordinates like this (possibly because we will need infinite number of such coordinates). We rather represent a line using the function $y = mx + c$ where m and c are the parameters. Example: The blue line shown in the left can be represented using the function $y = 1.5x + 2$.

- The very fact that we can capture a line using just two parameters m and c implies that there is correlation among the y values across different values of x.

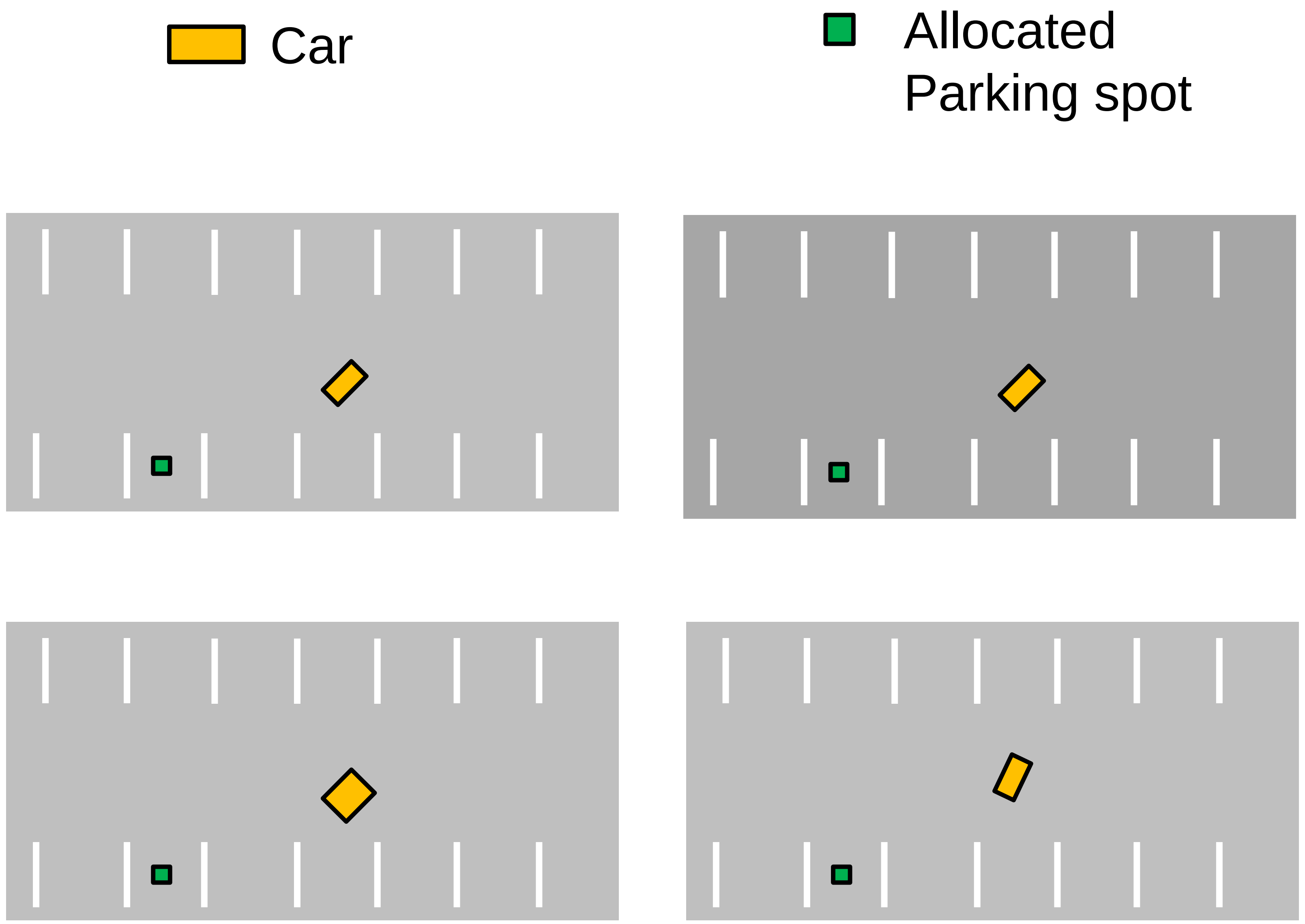
Fundamental Idea of Deep RL


r_1
r_2
r_3
r_4
a_1
a_2
a_3
a_4
x_1
x_2
x_3
x_4
$\begin{bmatrix}30\\45\\35\\60\\28\\44\end{bmatrix}$
$\begin{bmatrix}30\\45\\35\\25\\28\\44\end{bmatrix}$
$\begin{bmatrix}30\\45\\35\\60\\28\\44\end{bmatrix}$
$\begin{bmatrix}30\\45\\35\\25\\28\\44\end{bmatrix}$

Example 2 (Contextual bandits)

- A straight-s

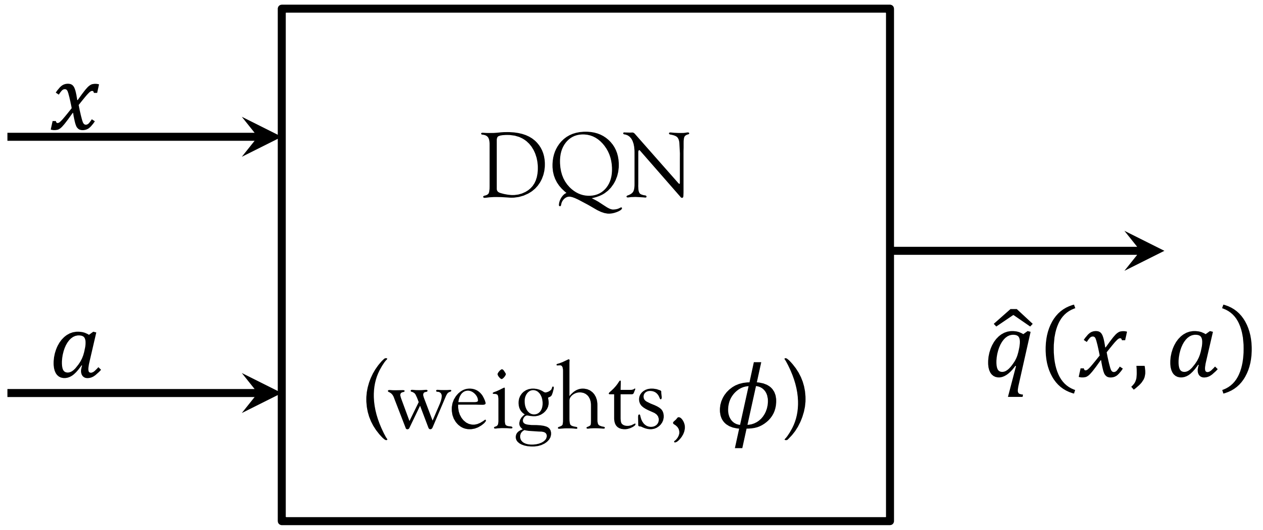
Fundamental Idea of Deep RL



Example 3 (Car parking)

- A straight-s
- s

Training DQN



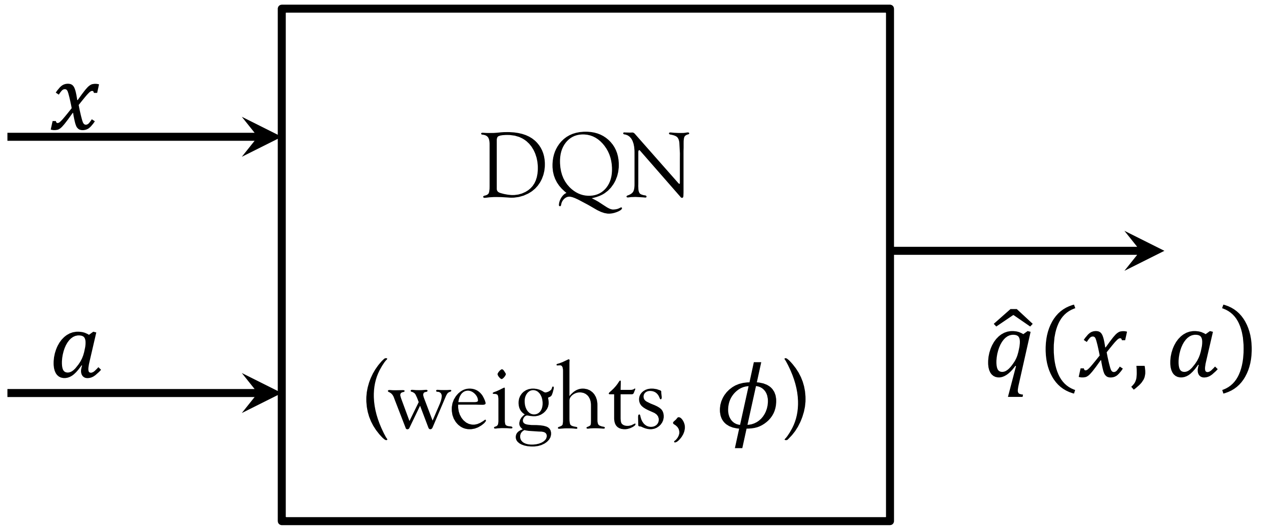
- What is the training data?

- Regression or classification?

- What is the loss function?

- What are the features (input)?

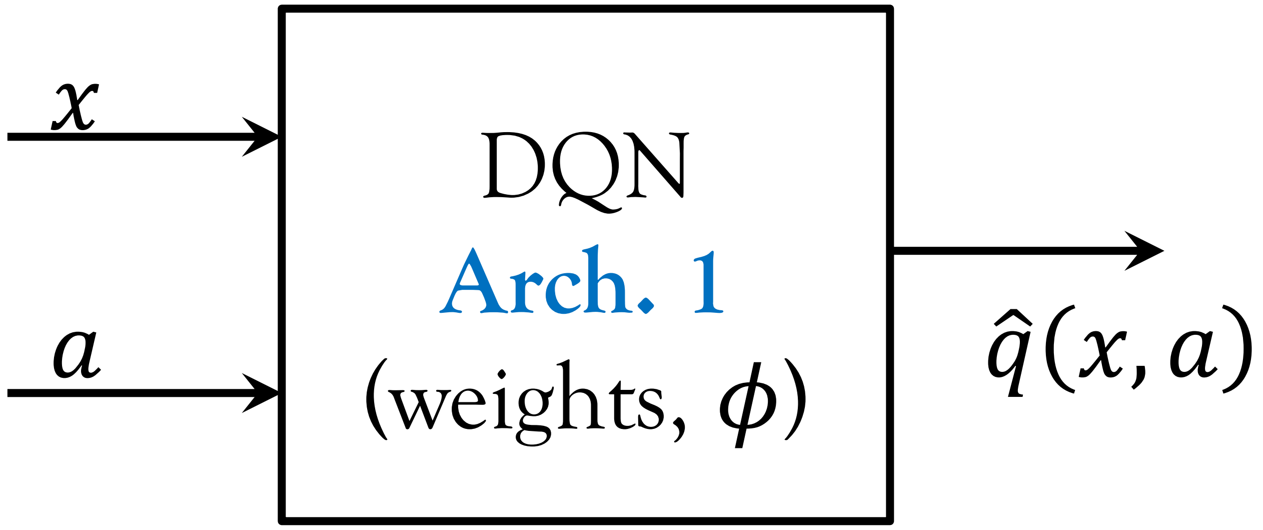
Training DQN



- What are the target/label?

- What is the loss function?

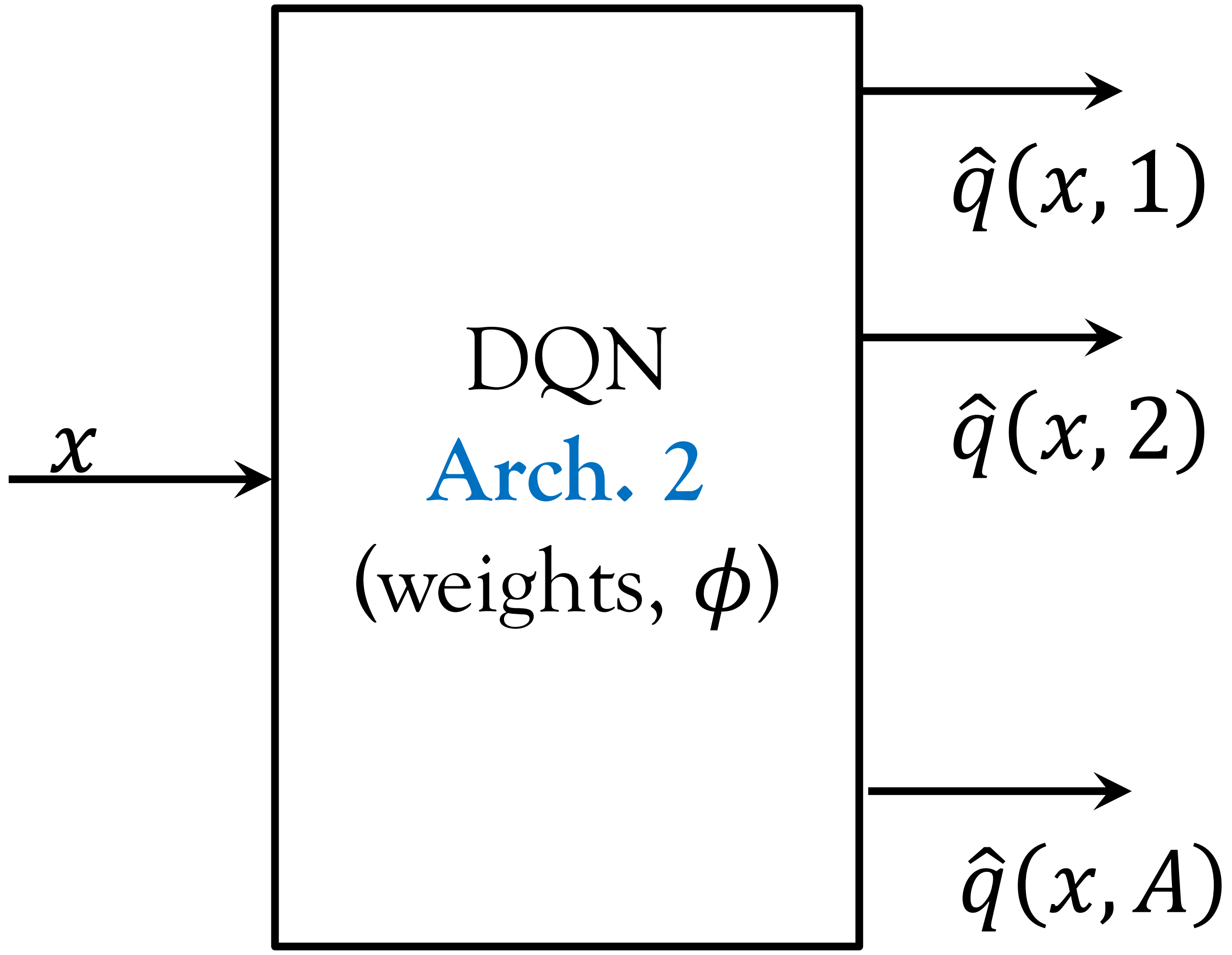
DQN Architectures



DQN Architecture 1

- Already discussed. Also, discussed also how to train it.
- Input: State action pair (x, a).
- Output: Estimated Q-value of the input state-action pair (x, a).

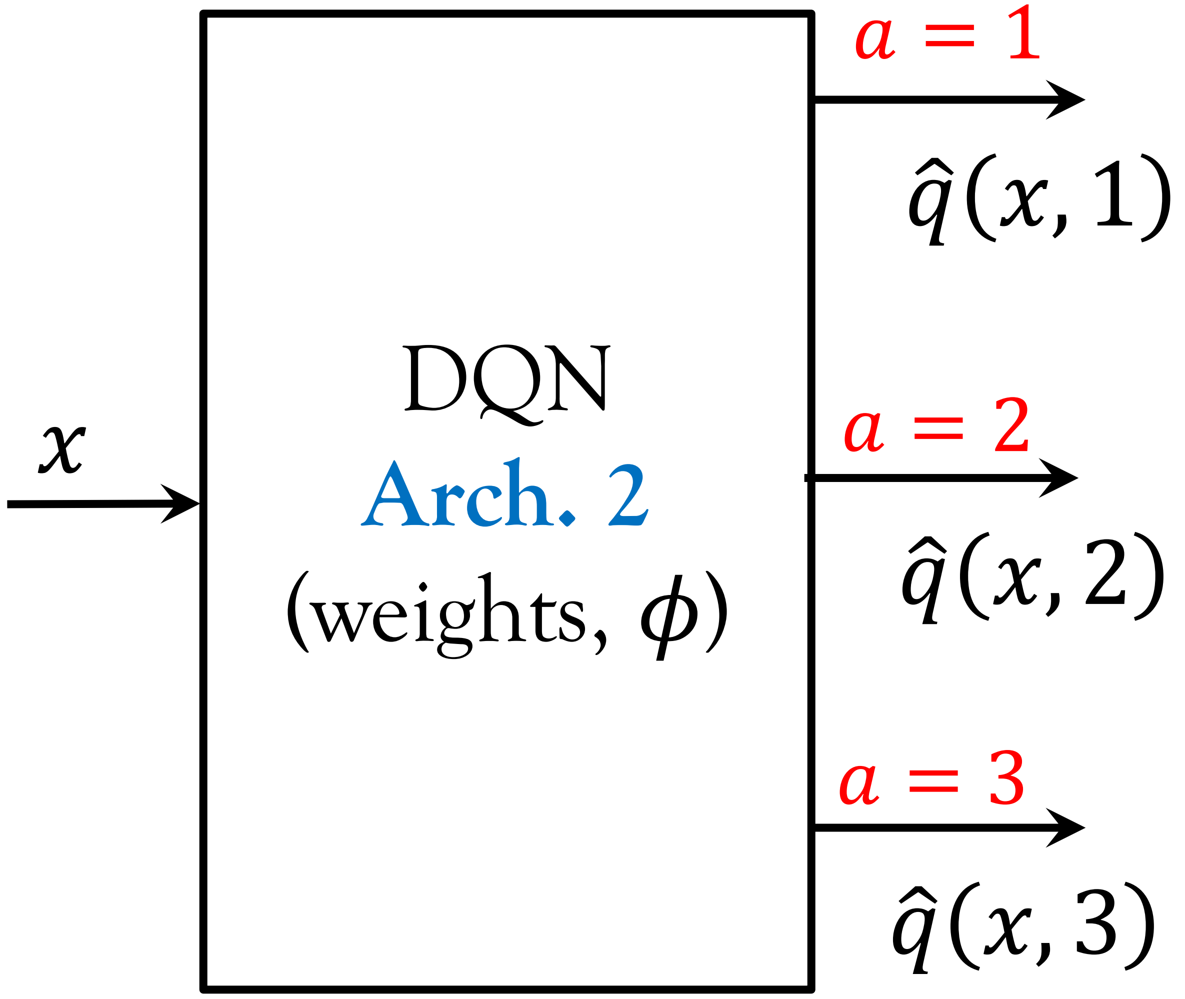
DQN Architectures



DQN Architecture 2

- Input: State x.
- Output: Estimated Q-value of **all the actions** corresponding to input state x.
- This is the more **conventional** DQN architecture.

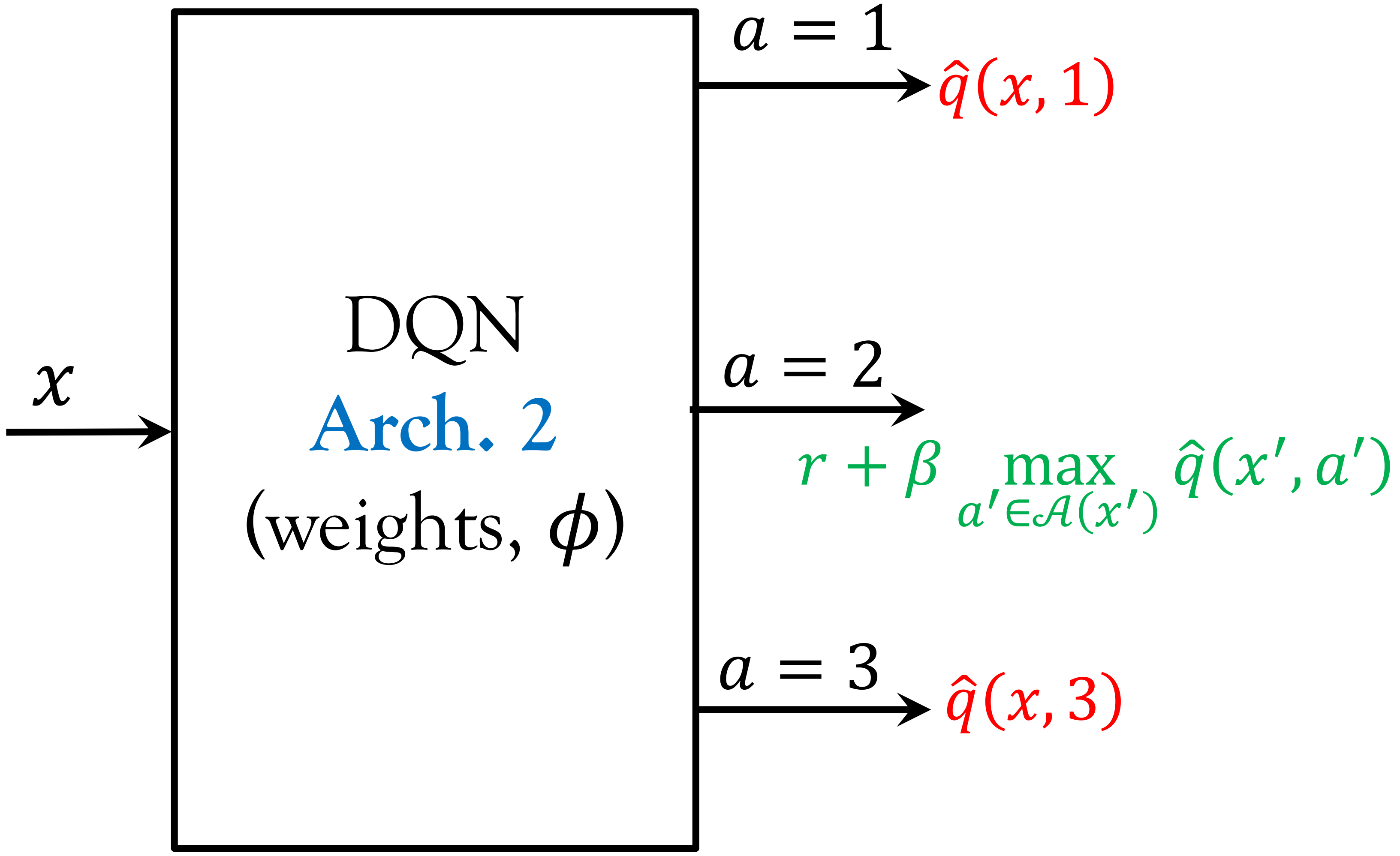
DQN Architectures



DQN Architecture 2: How to train it?

- What is the training data?
- Regression or classification?
- What is the loss function?
- What are the features (input)?

DQN Architectures



DQN Architecture 2: How to train it?

- What are the target/label?

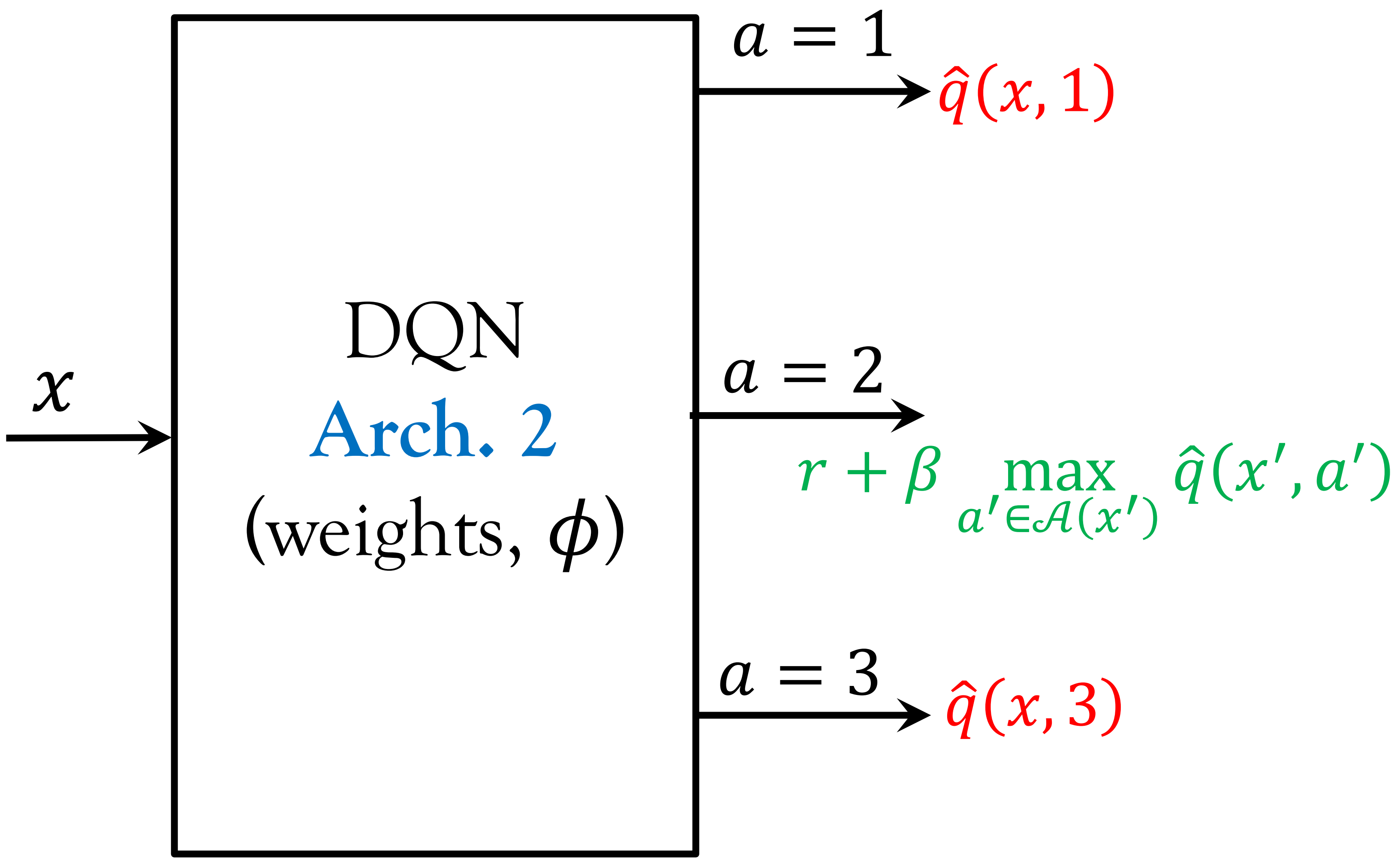
- For illustration only three actions, i.e. $A = 3$.

- Let the sample be $\langle x, 2, r, x' \rangle$.

- Labels are shown in the figure in green and red.

- Need **two forward passes of the DQN** to generate the labels:
  - One with input state x. Dummy targets. REASON?
  - The other with input state x'. Main target.

DQN Architectures



x

DQN

Arch. 2

(weights, ϕ)

$a = 1$

$\hat{q}(x, 1)$

$a = 2$

$r + \beta \max_{a' \in \mathcal{A}(x')} \hat{q}(x', a')$

$a = 3$

$\hat{q}(x, 3)$

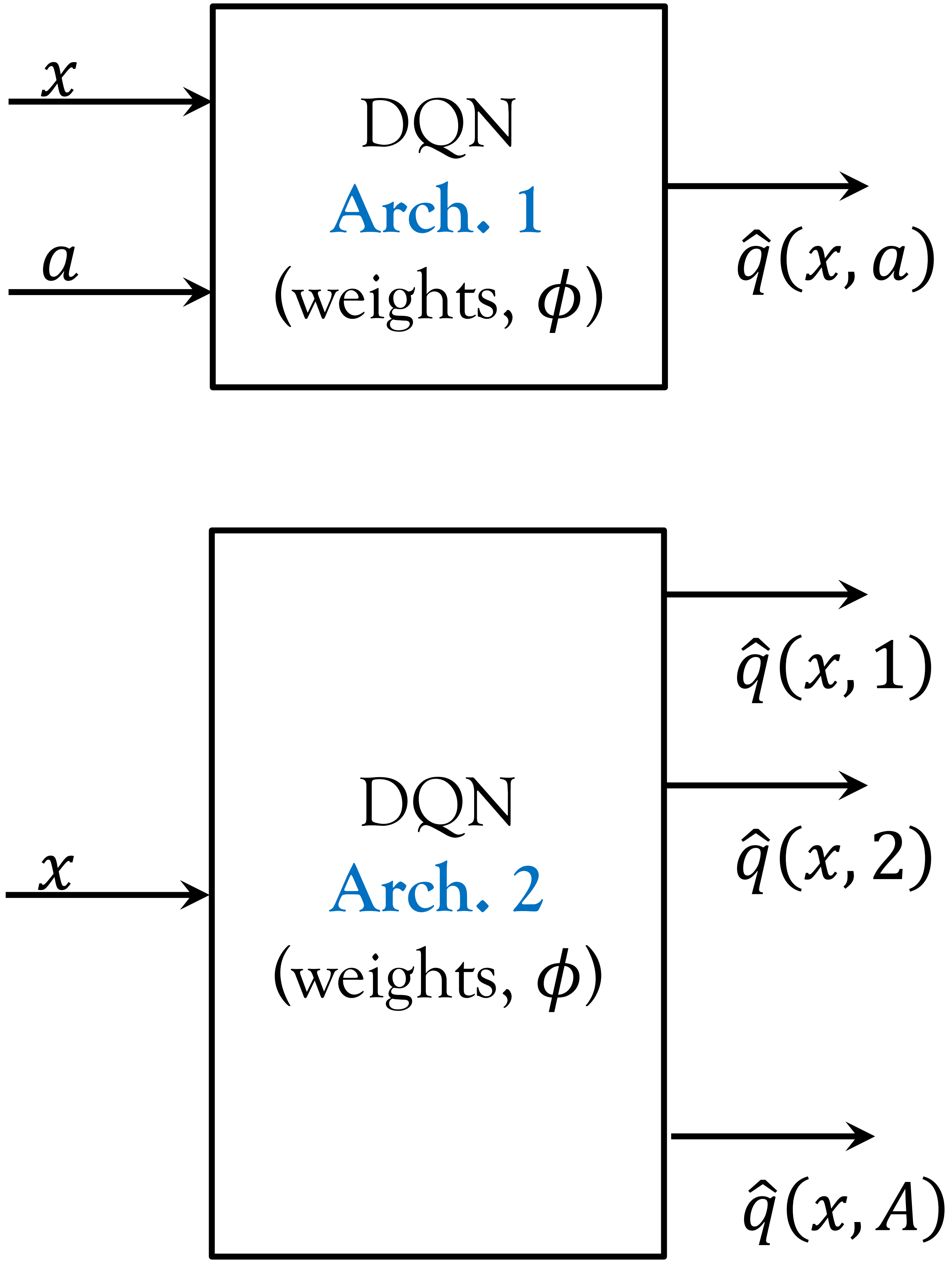
DQN Architecture 2: How to train it?

- What are the target/label?

The general formula: If the sample is $\langle x, a, r, x' \rangle$ then the corresponding target y_i for action i is,

$$y_i = \begin{cases} r + \max_{a' \in \mathcal{A}(x')} \hat{q}\left(x', a'\right) & , i = a \\ \hat{q}(x, i) & , i \neq a \end{cases}$$

DQN Architectures



DQN Architecture 2: Comparison

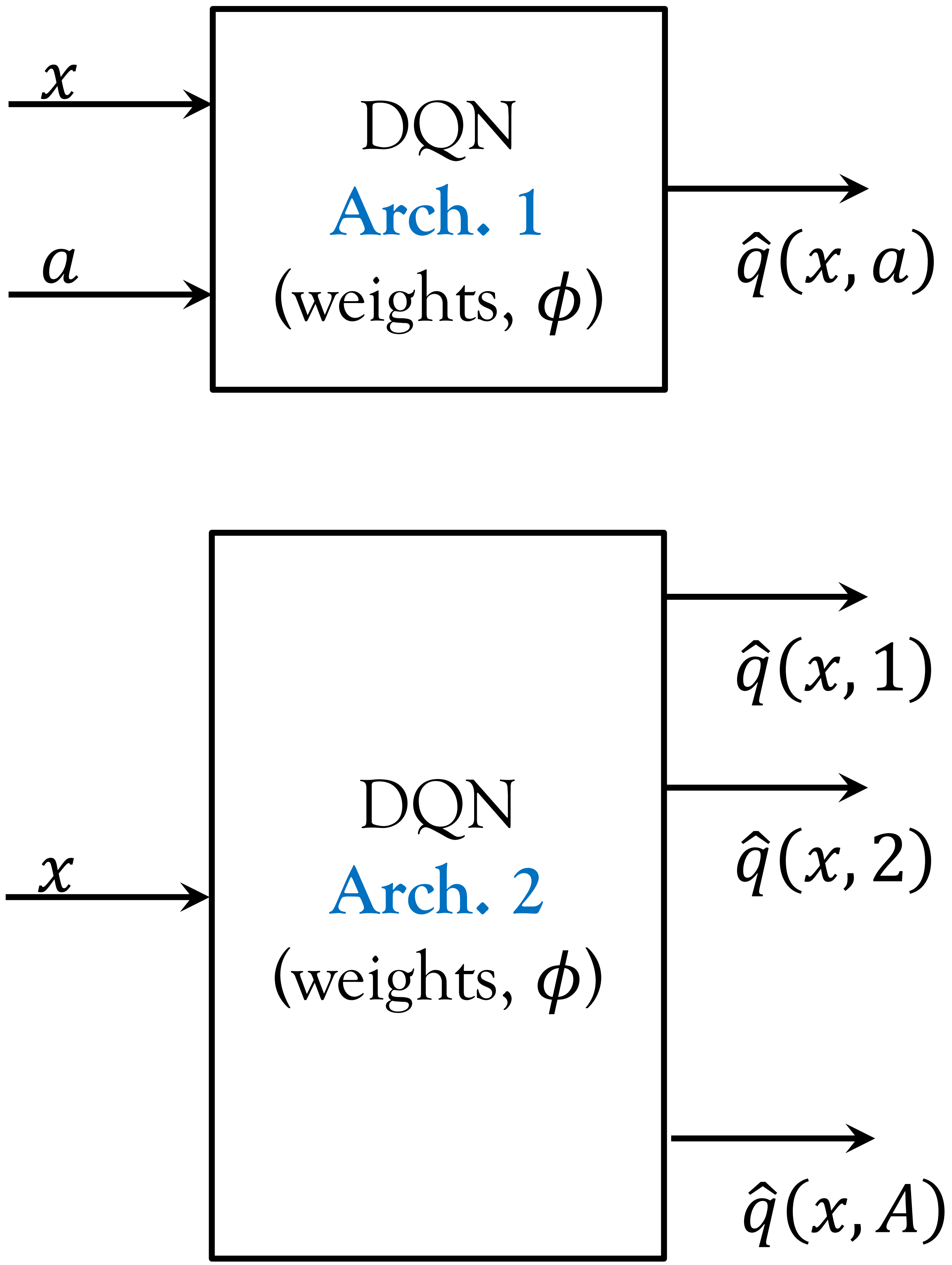
Pros:

- More **conventional**.
- Only **one forward pass required** to compute the optimal action.

$$\pi(x) = \underset{a \in \mathcal{A}(x)}{\arg\max}\, \widehat{q}(x, a)$$

For architecture 1, we need $|\mathcal{A}(x)|$ forward pass.

DQN Architectures

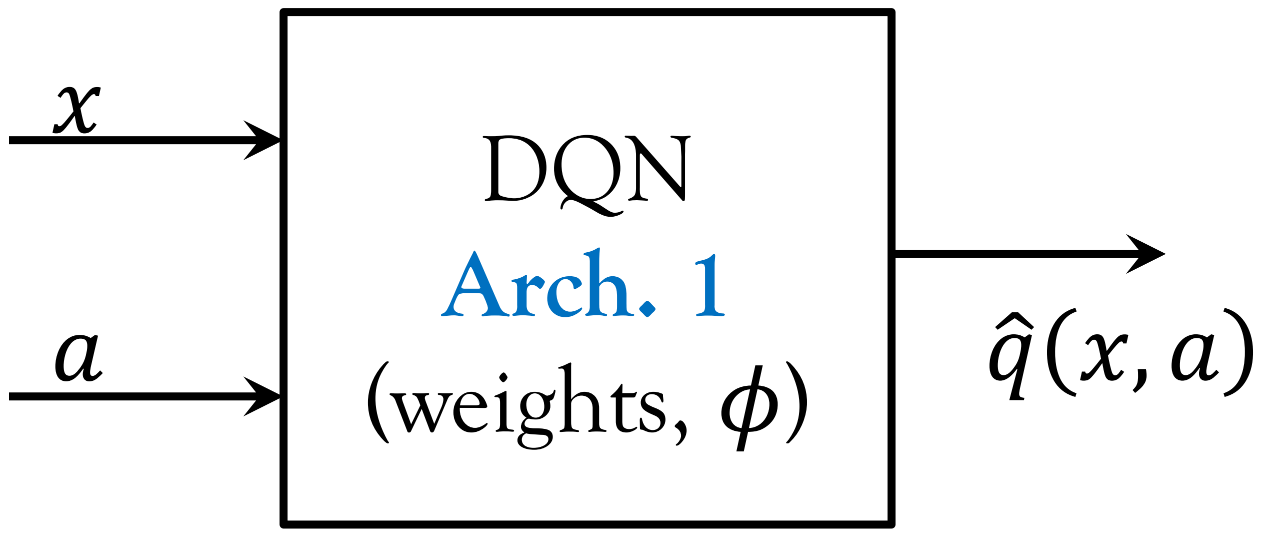


DQN Architecture 2: Comparison

Cons:

- Not advisable when **number of action is large**. More training parameters in the last layer.
- Challenging when **action space is a function of state**.
- Not possible when **action space is continuous**.
  - DQN architecture 1 is the only option.
  - Even for DQN architecture 1, we can compute $\hat{q}(x,a)$ but find the maximum over all action is still a challenge. This is where DDPG comes into picture.

DQN Architectures

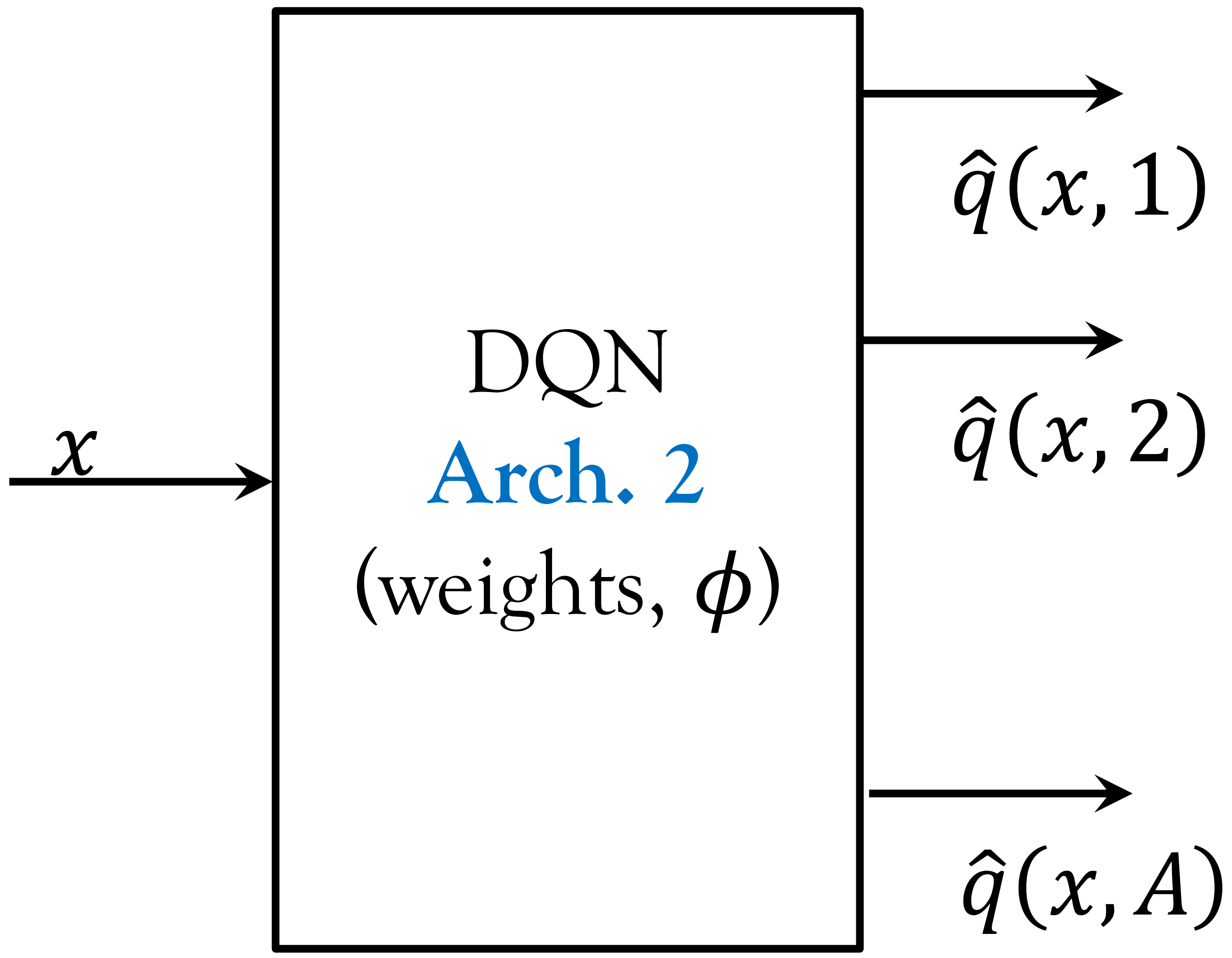


x

a

DQN
Arch. 1
(weights, ϕ)

$\hat{q}(x, a)$



x

DQN
Arch. 2
(weights, ϕ)

$\hat{q}(x, 1)$

$\hat{q}(x, 2)$

$\hat{q}(x, A)$

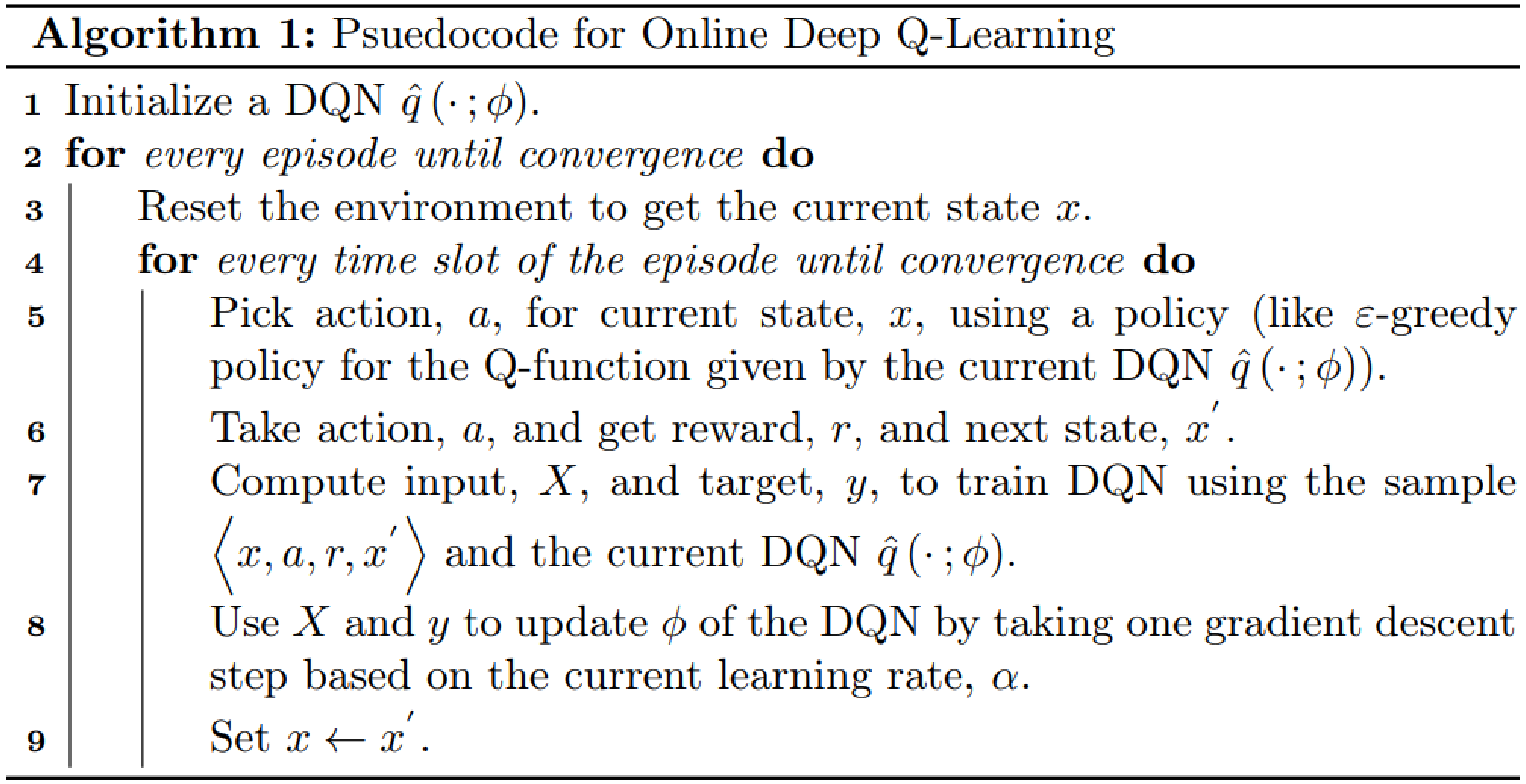
Inputs to DQN Architectures 1 and 2

Training DQN

- What we have discussed till now is the fundamental idea behind DQN and how to train it.

- Now we convert this fundamental idea into **five psuedocodes** of training DQN; each one better than the one discussed before it.

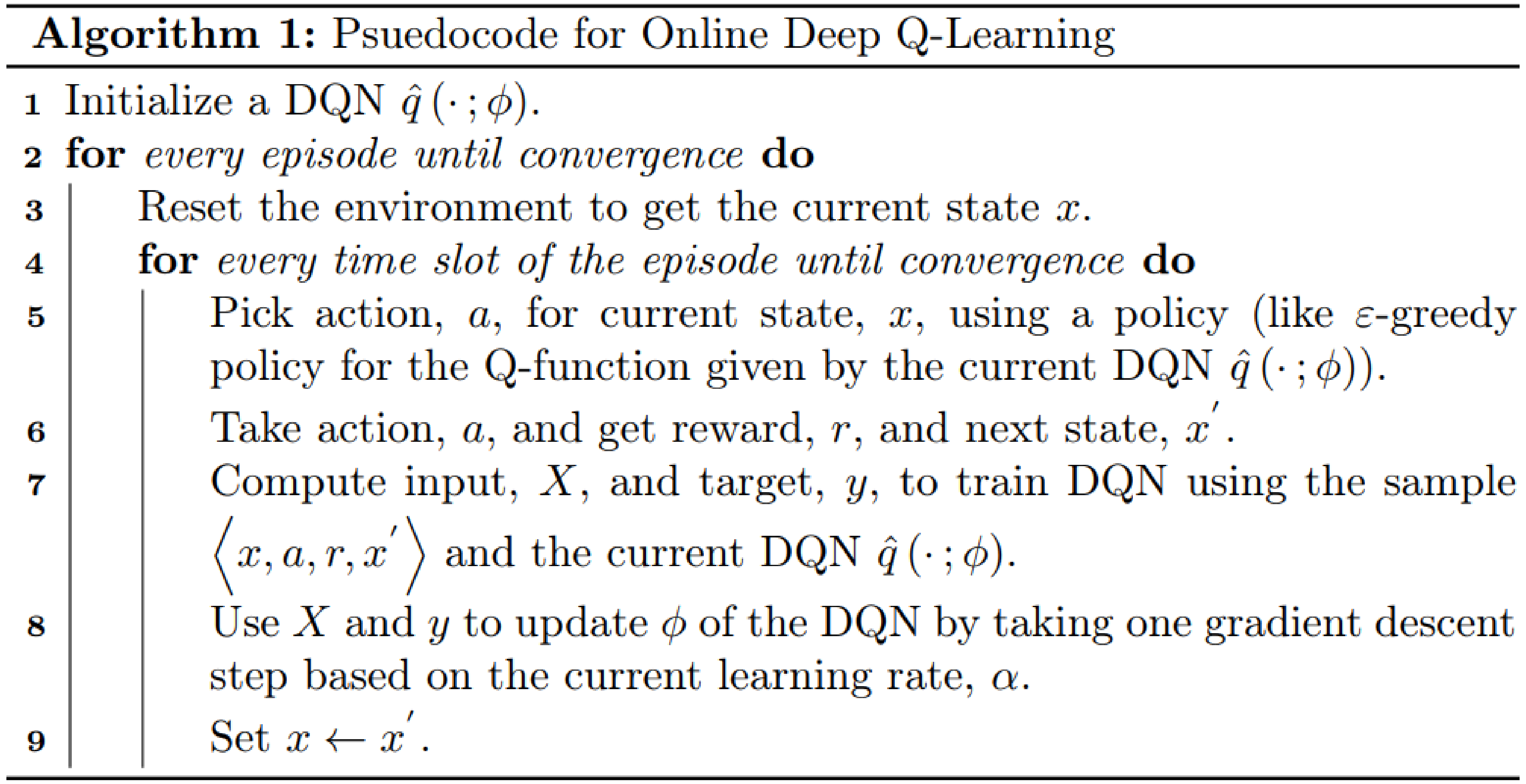
Training DQN: Online Deep Q-Learning



```
Algorithm 1: Psuedocode for Online Deep Q-Learning
1 Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
2 for every episode until convergence do
3     Reset the environment to get the current state $x$.
4     for every time slot of the episode until convergence do
5         Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
          policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
6         Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
7         Compute input, $X$, and target, $y$, to train DQN using the sample
          $\left\langle x, a, r, x^{'} \right\rangle$ and the current DQN $\hat{q}(\cdot\,;\phi)$.
8         Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent
          step based on the current learning rate, $\alpha$.
9         Set $x \leftarrow x^{'}$.
```

- Called **“online”** because training of DQN is done in every time slot based on the **most recently collected sample** $\langle x, a, r, x' \rangle$.
- **Line 1**: Either DQN architecture 1 or 2.
- **Lines 2 and 4**: You can get innovative about the convergence criteria. One possible example is **sum of reward** in an episode is greater than a threshold.
    - Sum of reward and NOT discounted reward. **Discounted reward** is used for its good mathematical properties.
- **Line 5:** Any policy that ensures that **every state-action pair** is sampled **infinitely often**.
    - A common choice is ε-greedy policy corresponding to the Q-function given by the current DQN. What does this mean?

Training DQN: Online Deep Q-Learning



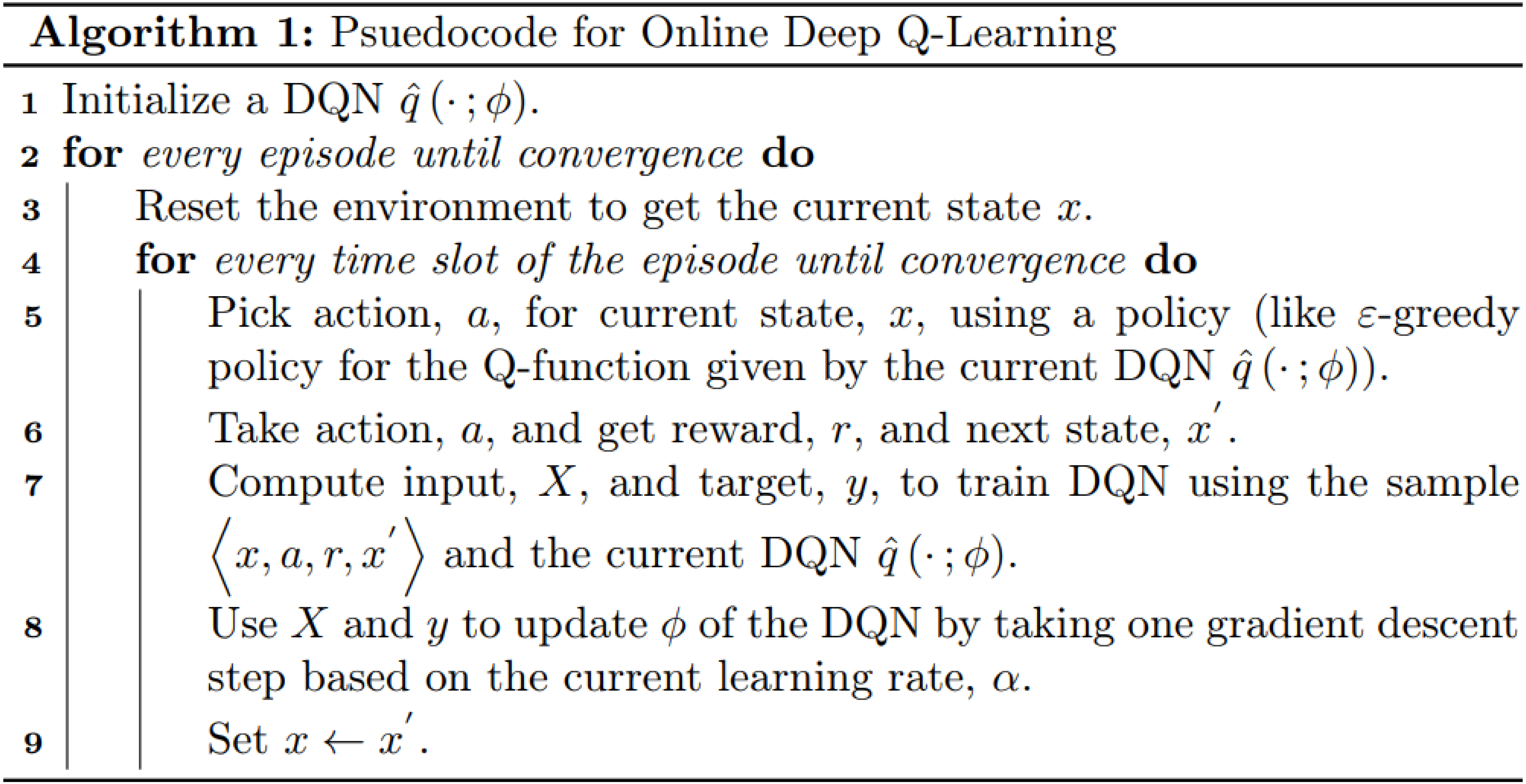
Algorithm 1: Psuedocode for Online Deep Q-Learning

```
1 Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
2 for every episode until convergence do
3     Reset the environment to get the current state $x$.
4     for every time slot of the episode until convergence do
5         Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
6         Take action, $a$, and get reward, $r$, and next state, $x'$.
7         Compute input, $X$, and target, $y$, to train DQN using the sample $\left\langle x, a, r, x' \right\rangle$ and the current DQN $\hat{q}(\cdot\,;\phi)$.
8         Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
9         Set $x \leftarrow x'$.
```

- **Line 7**: Generates training sample (feature and target) to train the DQN. **Depends on the architecture.** We have already discussed how to generate training samples for both the architecture in the previous slides.

- **Line 8:** We train the DQN on a **batch size of 1**. We take one **gradient descent** step to **minimize the mean squared error** between the predicted Q-value and the target/labels generated in the previous step.

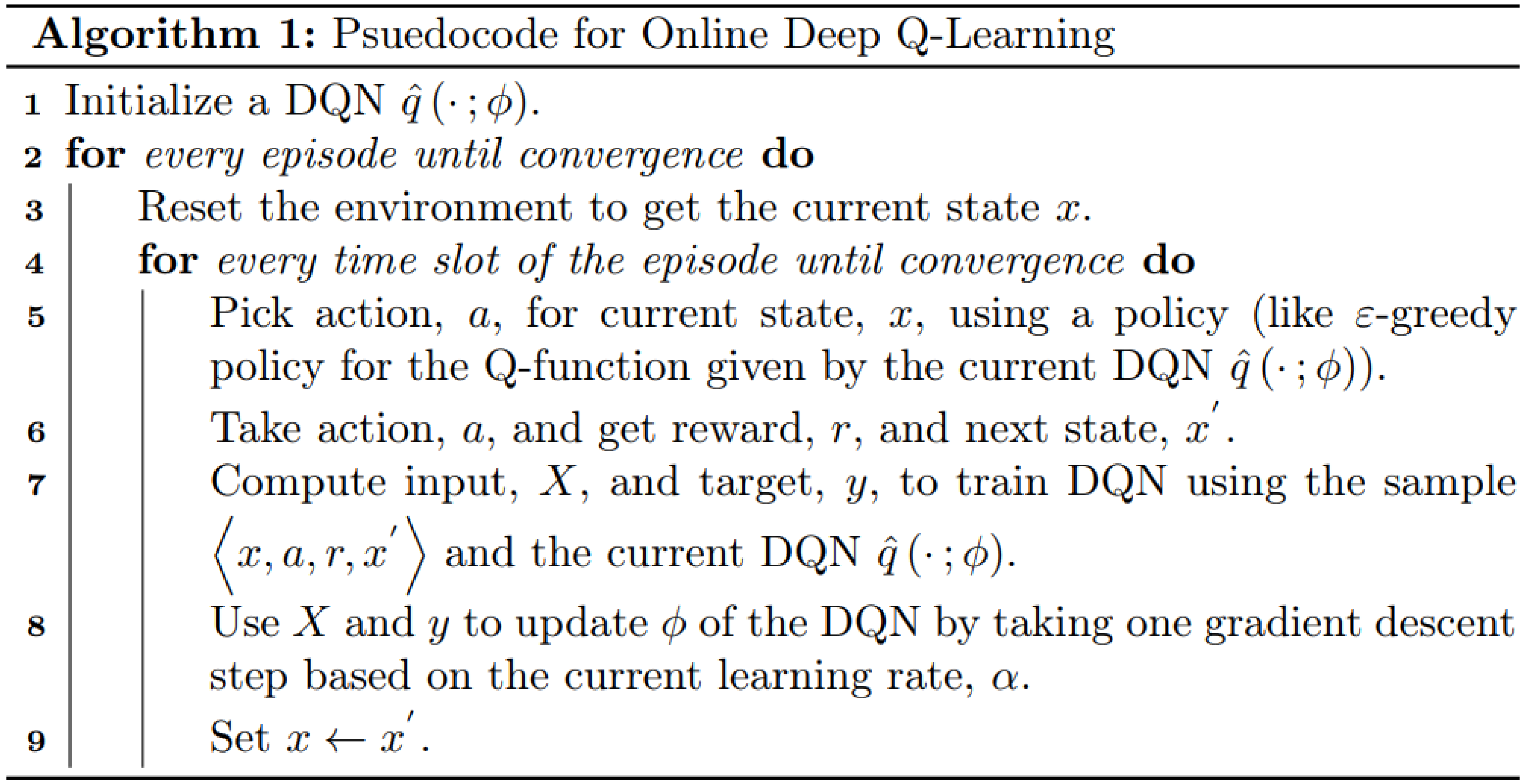
Training DQN: Online Deep Q-Learning



Algorithm 1: Psuedocode for Online Deep Q-Learning

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Compute input, $X$, and target, $y$, to train DQN using the sample $\left\langle x, a, r, x^{'} \right\rangle$ and the current DQN $\hat{q}(\cdot\,;\phi)$.
        Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$.
```

Training DQN: With Replay Buffer



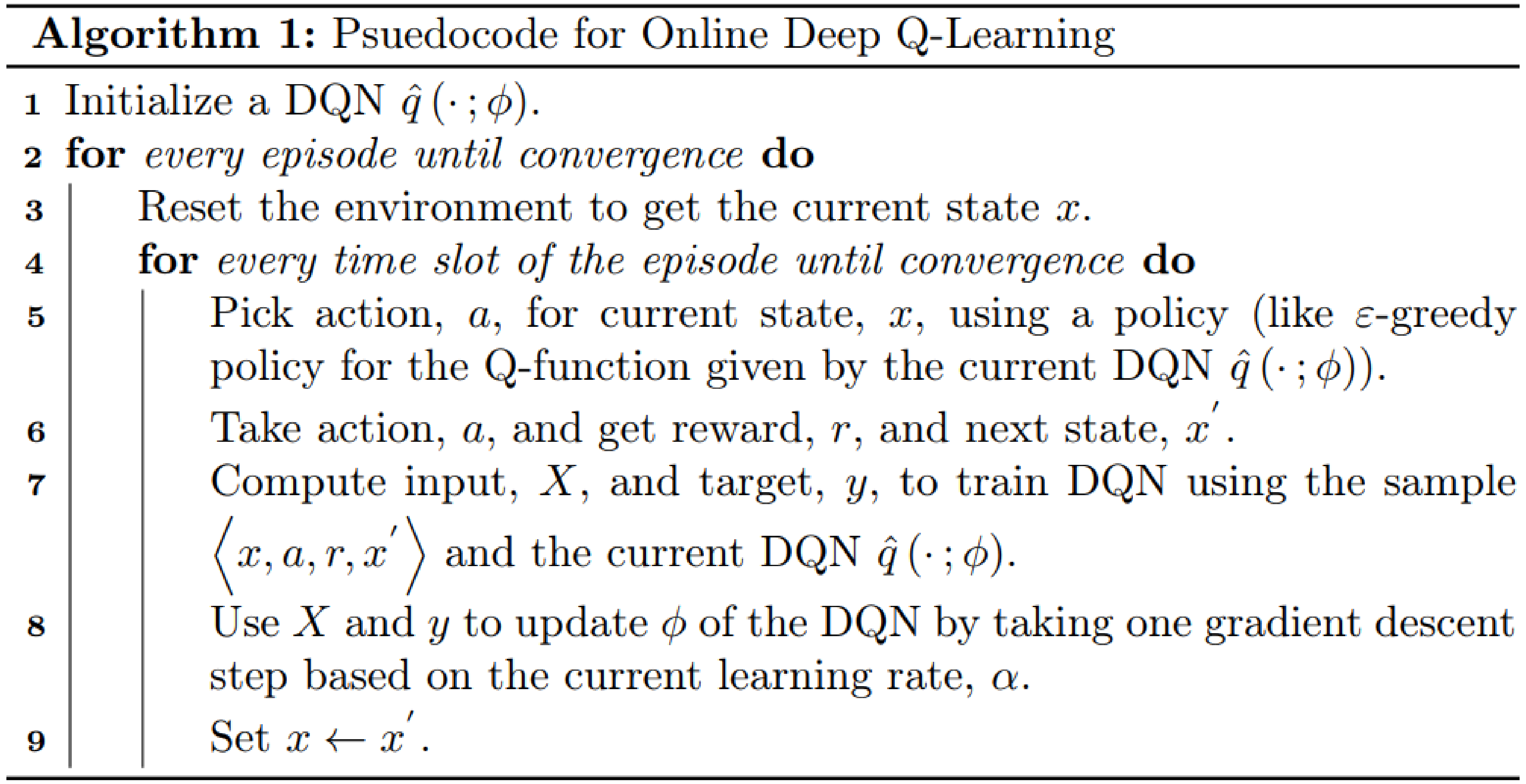
Algorithm 1: Psuedocode for Online Deep Q-Learning

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Compute input, $X$, and target, $y$, to train DQN using the sample $\left\langle x, a, r, x^{'} \right\rangle$ and the current DQN $\hat{q}(\cdot\,;\phi)$.
        Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$.
```

Drawbacks of Online Deep Q-Learning?

- **Drawback 1:** Poor sample efficiency.
  - Each sample $\langle x, a, r, x' \rangle$ is used only once. This is unlike supervised learning.
  - Generating samples is the bottleneck; needs interaction with the environment. Makes GPUs not that helpful.
  - Neural network is complex. Requires lot of small gradient step (small learning rate) to learn. In online Deep Q-Learning even smaller learning rate because batch size is 1. If each gradient step requires a new sample, time complexity to train will be enormous.
- **Drawback 2:** Correlated training samples.

Training DQN: With Replay Buffer



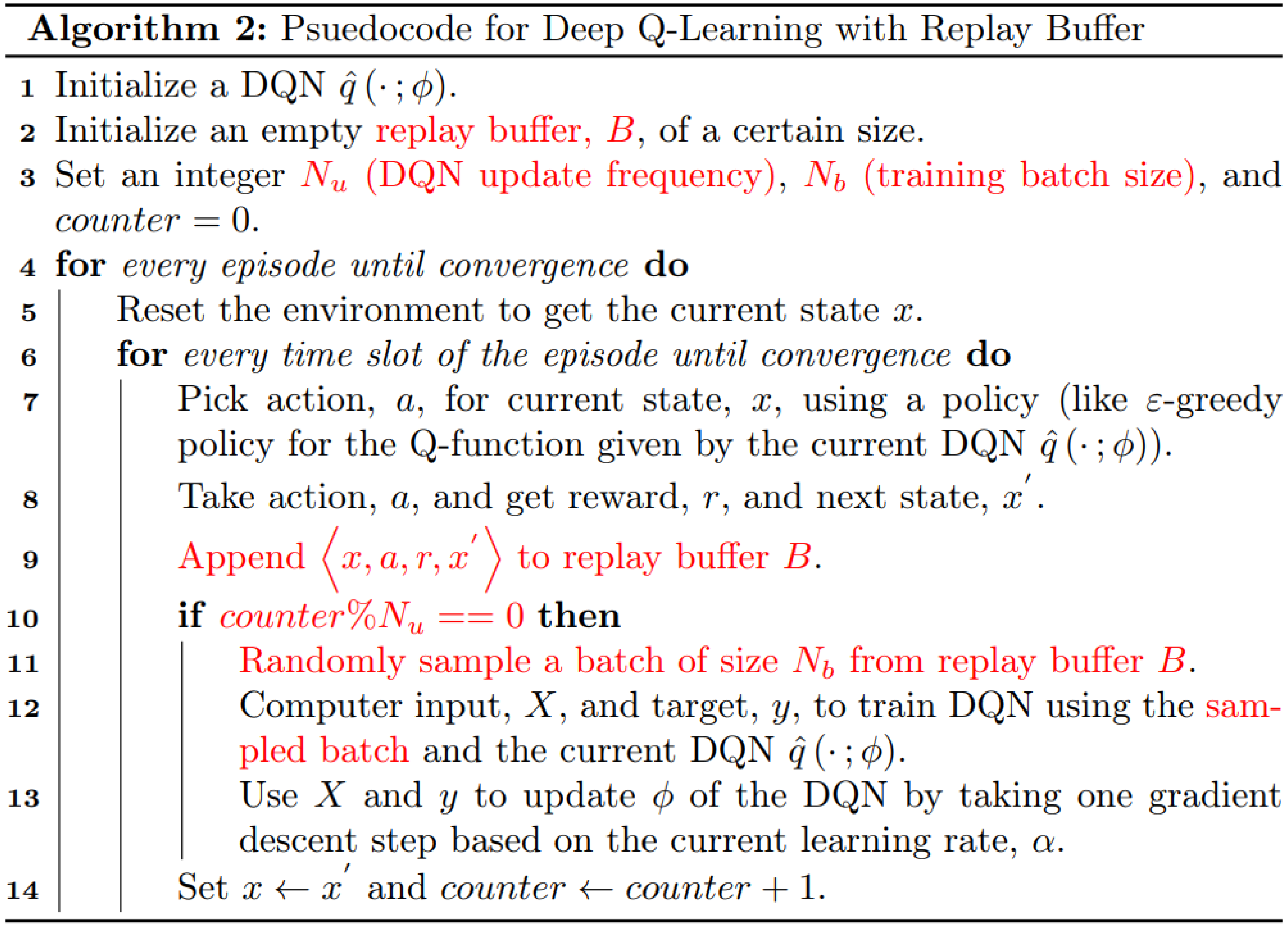
Algorithm 1: Psuedocode for Online Deep Q-Learning

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
        policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x'$.
        Compute input, $X$, and target, $y$, to train DQN using the sample
        $\left\langle x, a, r, x' \right\rangle$ and the current DQN $\hat{q}(\cdot\,;\phi)$.
        Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent
        step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x'$.
```

Drawbacks of Online Deep Q-Learning?

- **Drawback 2:** Correlated training samples.
    - Samples $\langle x, a, r, x' \rangle$ in two successive time slots is correlated, hence dependent.
    - ML/DL assumes samples are iid.

Training DQN: With Replay Buffer



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

```
1  Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and
   $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      for every time slot of the episode until convergence do
7          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
8          Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
9          Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
10         if $counter \% N_u == 0$ then
11             Randomly sample a batch of size $N_b$ from replay buffer $B$.
12             Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
13             Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient
               descent step based on the current learning rate, $\alpha$.
14         Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

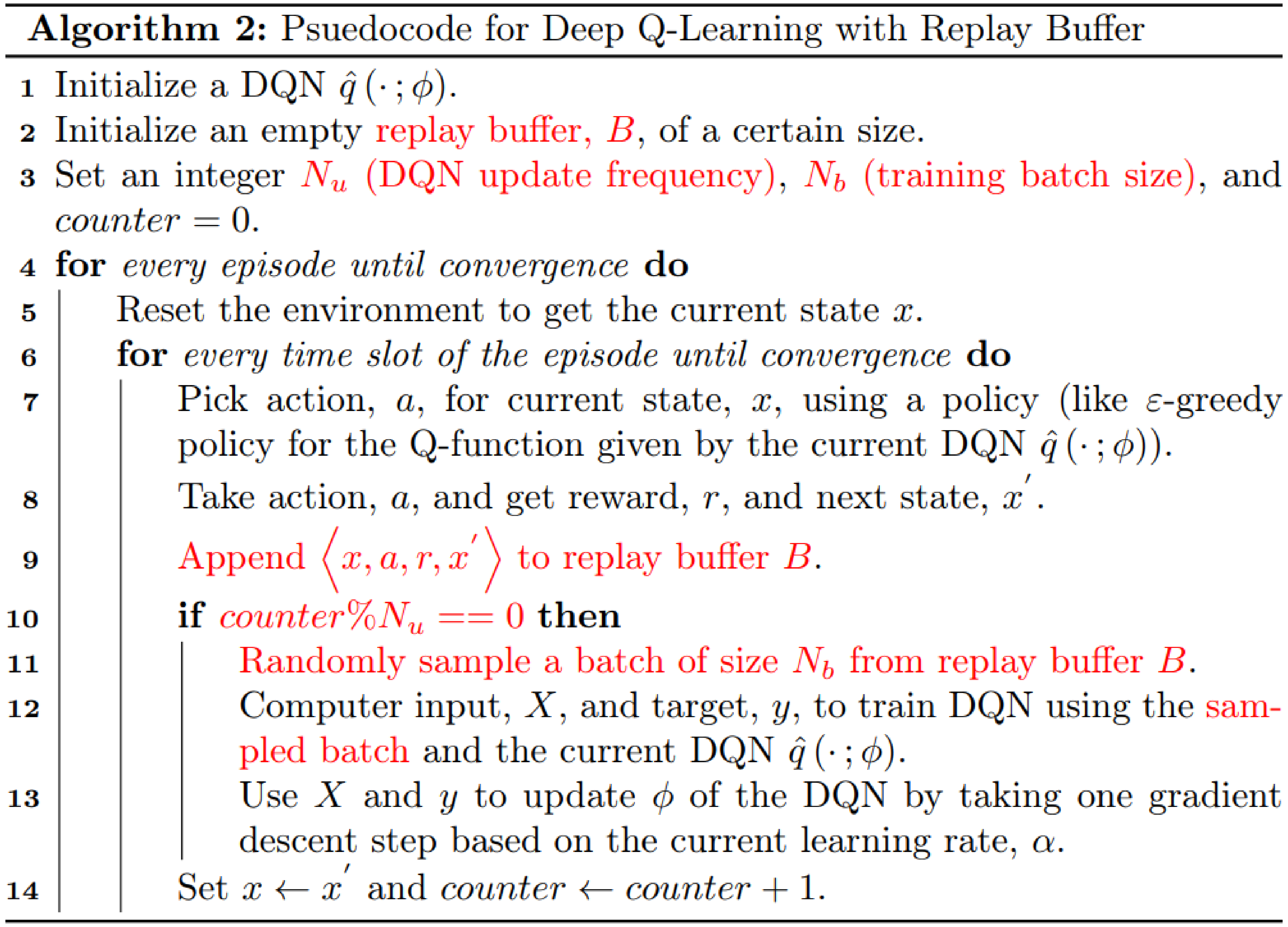
- **Line 2**: In order to address the drawbacks of online Deep Q-Learning, we add a buffer, B, called the "replay buffer", also called "experience replay".

- **Line 9:** Training samples $\langle x, a, r, x' \rangle$ that gets generated is not directly used for training. It is first added to replay buffer.
  - Replay buffer has a certain size. After it gets filled the oldest data is removed and new data gets added (like FIFO queue).

- **Line 11:** Random batch size of size N_b is generated from the replay buffer. Then in line 12, the samples of this batch is used for generating features and target just like online Deep Q-Learning.
  - Advanced version: Batch selection using **prioritized experience replay**.

- **Line 10:** We don't need to train in every time slot.

Training DQN: With Replay Buffer



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

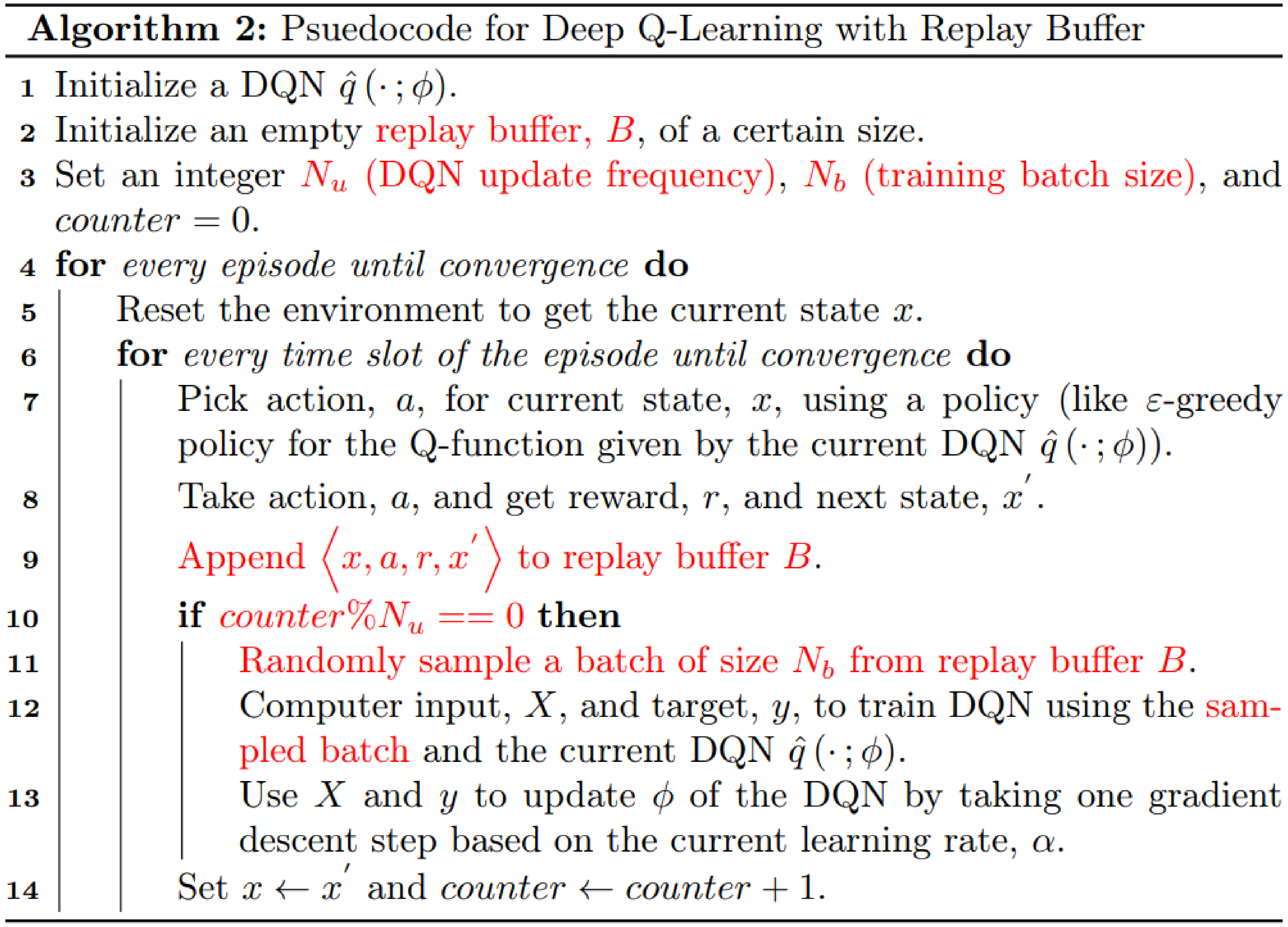
```
Initialize a DQN $\hat{q}(\cdot;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

How does it addresses the two drawbacks?

- **How it addresses drawback 1?:** Each sample $\langle x, a, r, x' \rangle$ can be sampled multiple times. So, one sample is used to train the DQN multiple times (just like in supervised learning). This increases sample efficiency.

- **How it addresses drawback 2?:** Randomly sampling from the replay buffer. Therefore, the chances of any two samples being "almost time adjacent" is very less. Hence, correlation between training samples is low.

Training DQN: With Replay Buffer



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

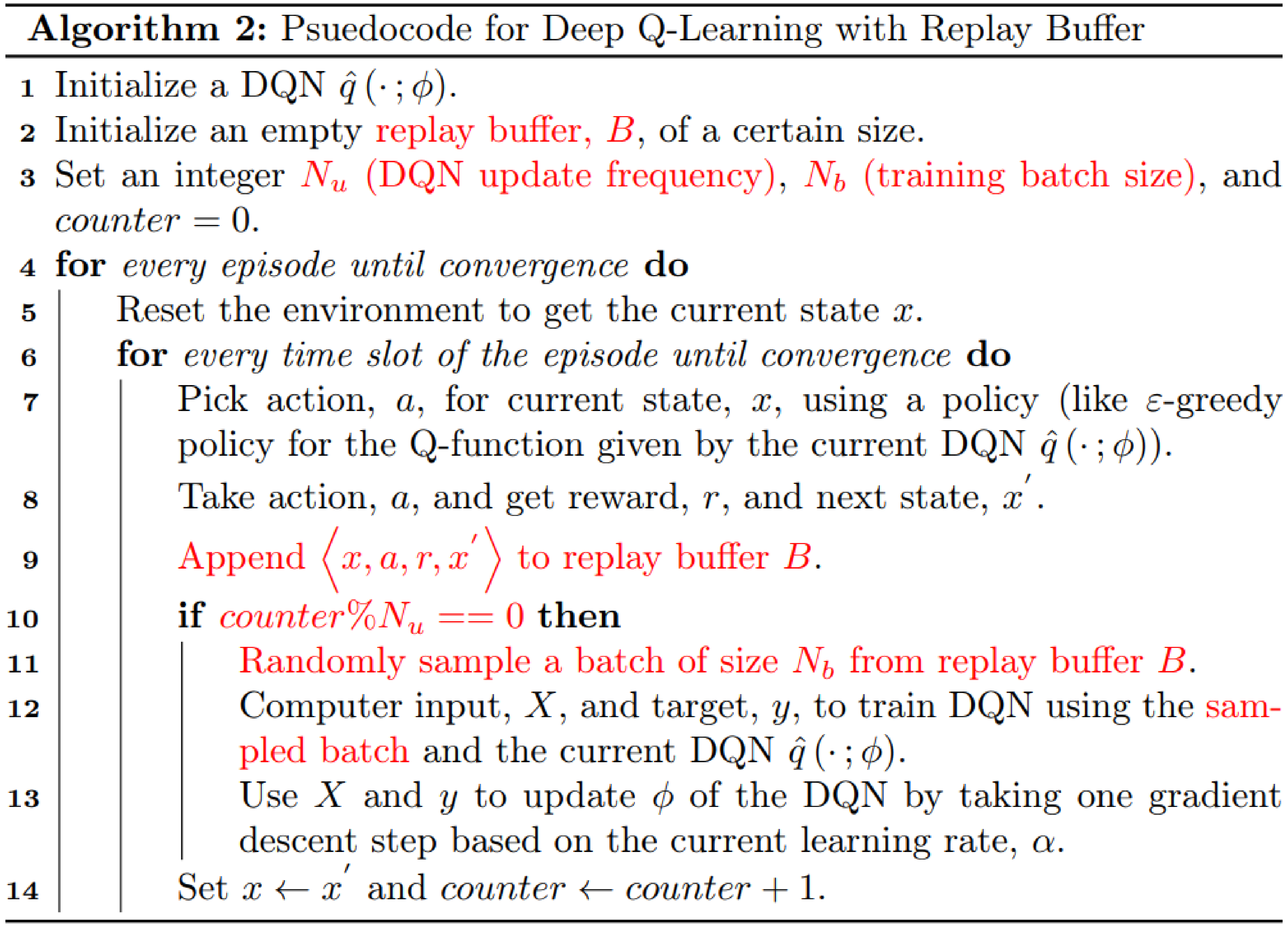
```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

Two other advantages?

- Replay buffer can be loaded with offline data. This can speed up training process. **WHY?**

- Batch size can be greater than 1. This **reduces variance in the calculation of the gradient**. Hence, training is more stable compared to online Deep Q-Learning.

Training DQN: With Replay Buffer

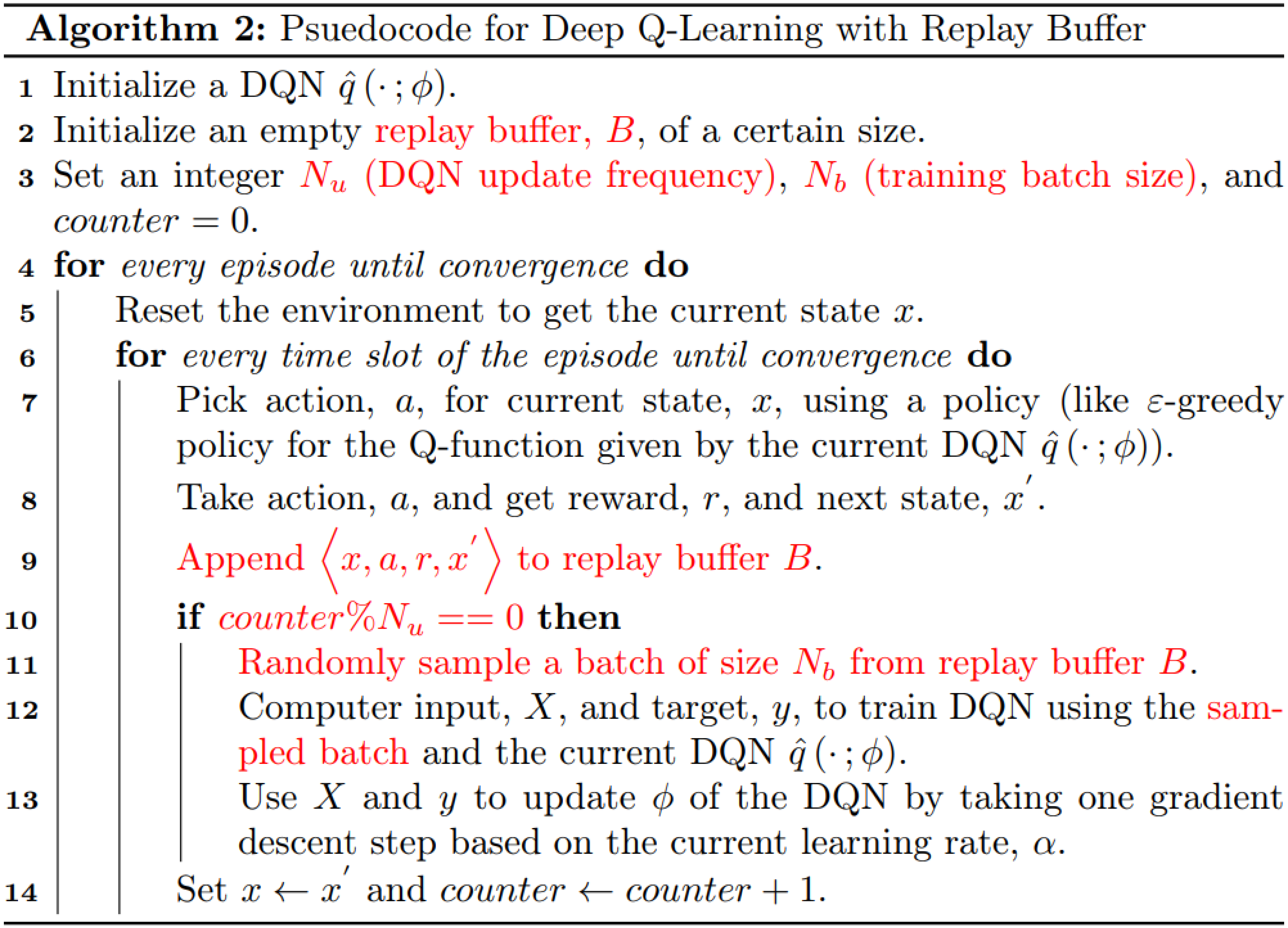


Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

- VERY IMPORTANT: Replay buffer is possible because Deep Q-Learning is off-policy algorithm. Hence, previous samples that were collected using another policy (for Deep Q-Learning, policy is determined by the weights of the DQN) can be used to train the current DQN.

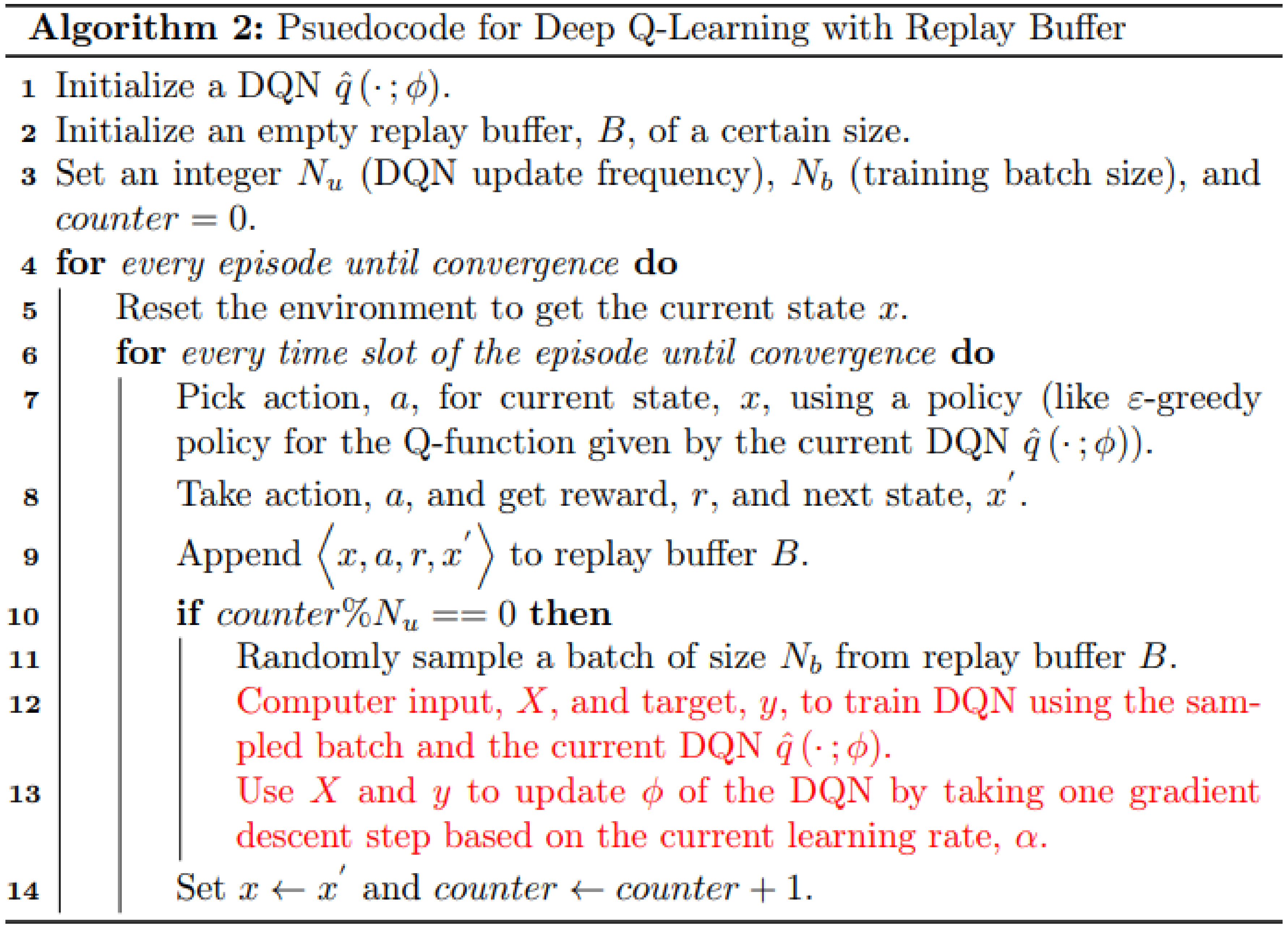
Training DQN: With Replay Buffer



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

Training DQN: Target Network



```
Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer
1  Initialize a DQN $\hat{q}(\cdot;\phi)$.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and
   $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      for every time slot of the episode until convergence do
7          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the Q-function given by the current DQN $\hat{q}(\cdot;\phi)$).
8          Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
9          Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
10         if $counter \% N_u == 0$ then
11             Randomly sample a batch of size $N_b$ from replay buffer $B$.
12             Computer input, $X$, and target, $y$, to train DQN using the sam-
               pled batch and the current DQN $\hat{q}(\cdot;\phi)$.
13             Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient
               descent step based on the current learning rate, $\alpha$.
14         Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

The issue of non-stationary target:

- Both the psuedocodes (online and with replay buffer) that we discussed till now have one drawback related to non-stationary target that we are going to discuss now. To explain this drawback we consider DQN architecture 1*, i.e. state-action pair as input and the Q-value of the state-action pair as output.

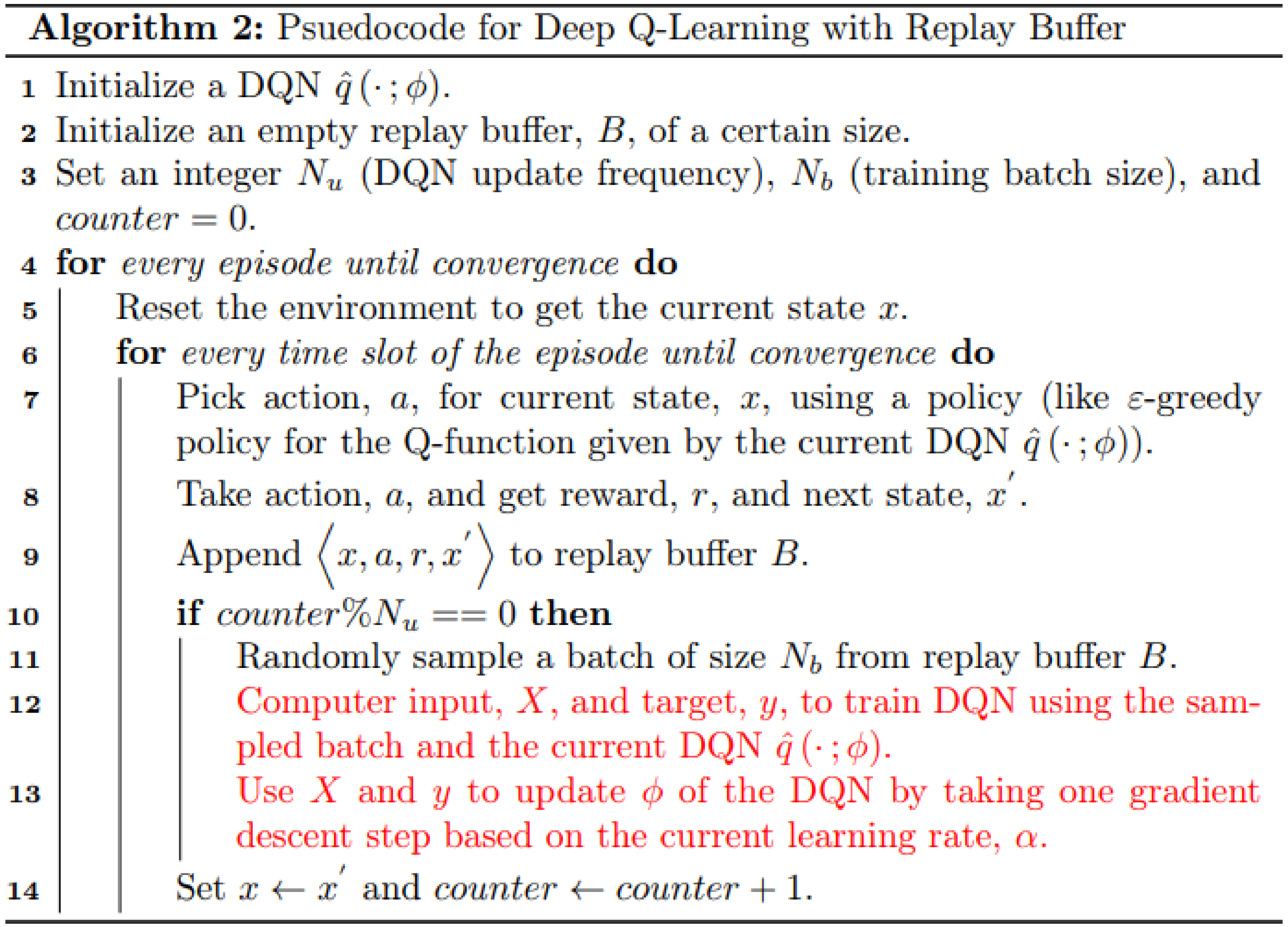
- Consider steps 12. Say that one of the sample in the batch is $\langle x, a, r, x' \rangle$. Then the input $X = (x, a)$ and the target,

  $$y = r + \max_{a^{'} \in \mathcal{A}(x^{'})} \widehat{q}\left(x^{'}, a^{'}; \phi\right)$$

  where ϕ is the current parameter of the DQN.

- Now in step 13, X and y generated in step 12 is used to used to update DQN parameter ϕ using gradient descent. Let the updated parameters be $\tilde{\phi}$.

Training DQN: Target Network



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

```
1  Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and
   $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      for every time slot of the episode until convergence do
7          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
8          Take action, $a$, and get reward, $r$, and next state, $x'$.
9          Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
10         if $counter \% N_u == 0$ then
11             Randomly sample a batch of size $N_b$ from replay buffer $B$.
12             Computer input, $X$, and target, $y$, to train DQN using the sam-
               pled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
13             Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient
               descent step based on the current learning rate, $\alpha$.
14         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

The issue of non-stationary target:

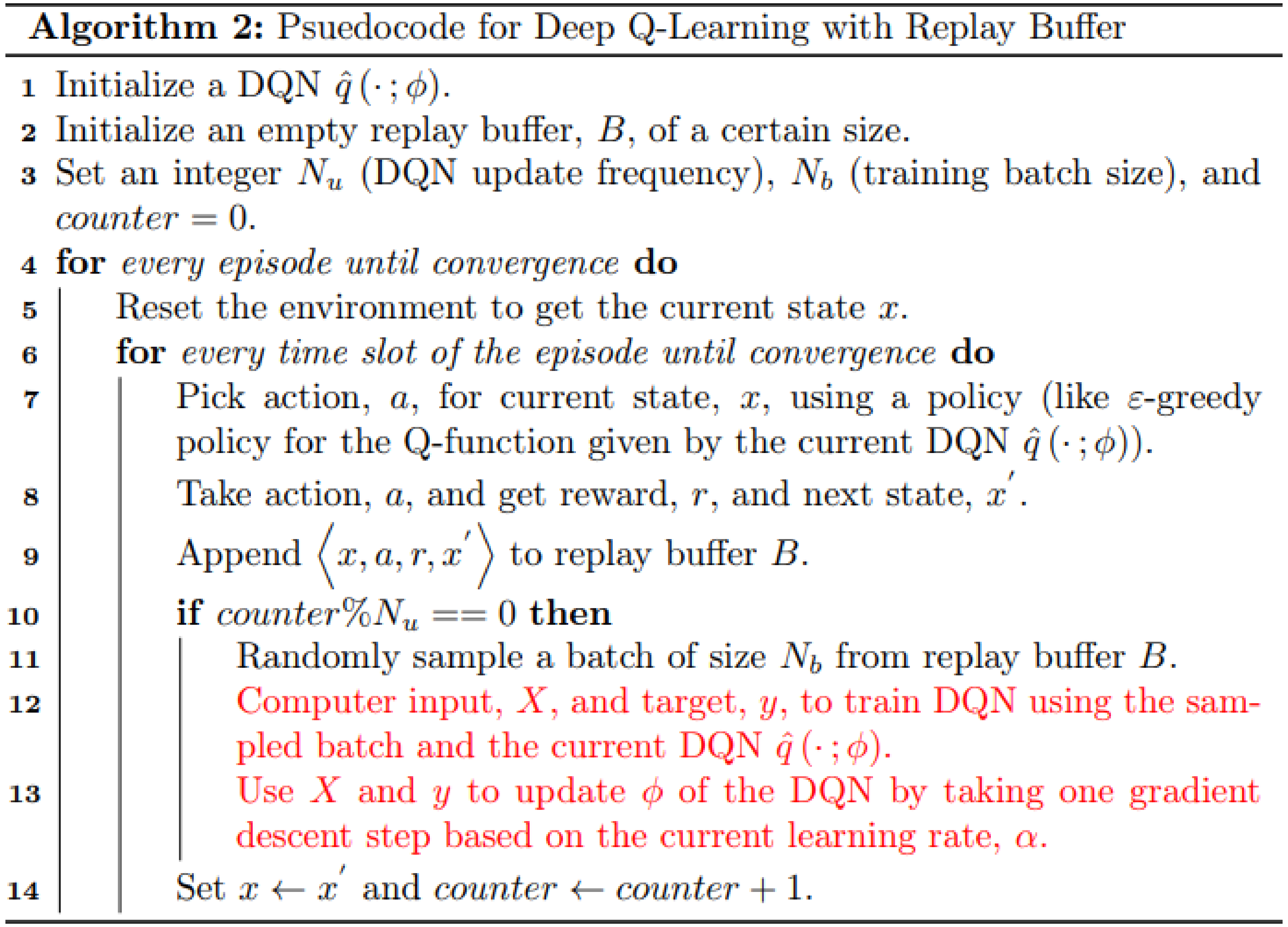
- Now consider the next instant we want to update DQN parameters. So we come back to step 12 and let's say that the same sample $\langle x, a, r, x' \rangle$ that was there in the previous batch is there in the current batch also. Then the input remains same as before, i.e. $X = (x, a)$. But the **target $\boldsymbol{y}$ has changed** to,

$$y = r + \max_{a' \in \mathcal{A}(x')} \hat{q}\left(x', a'; \tilde{\phi}\right)$$

where $\tilde{\phi}$ is the **updated parameter** of the DQN.

- So essentially, between two successive updates, the input remained same but the target changed. Target changed because the parameter of the DQN that is used to generate the target has changed between two successive updates.

Training DQN: Target Network



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

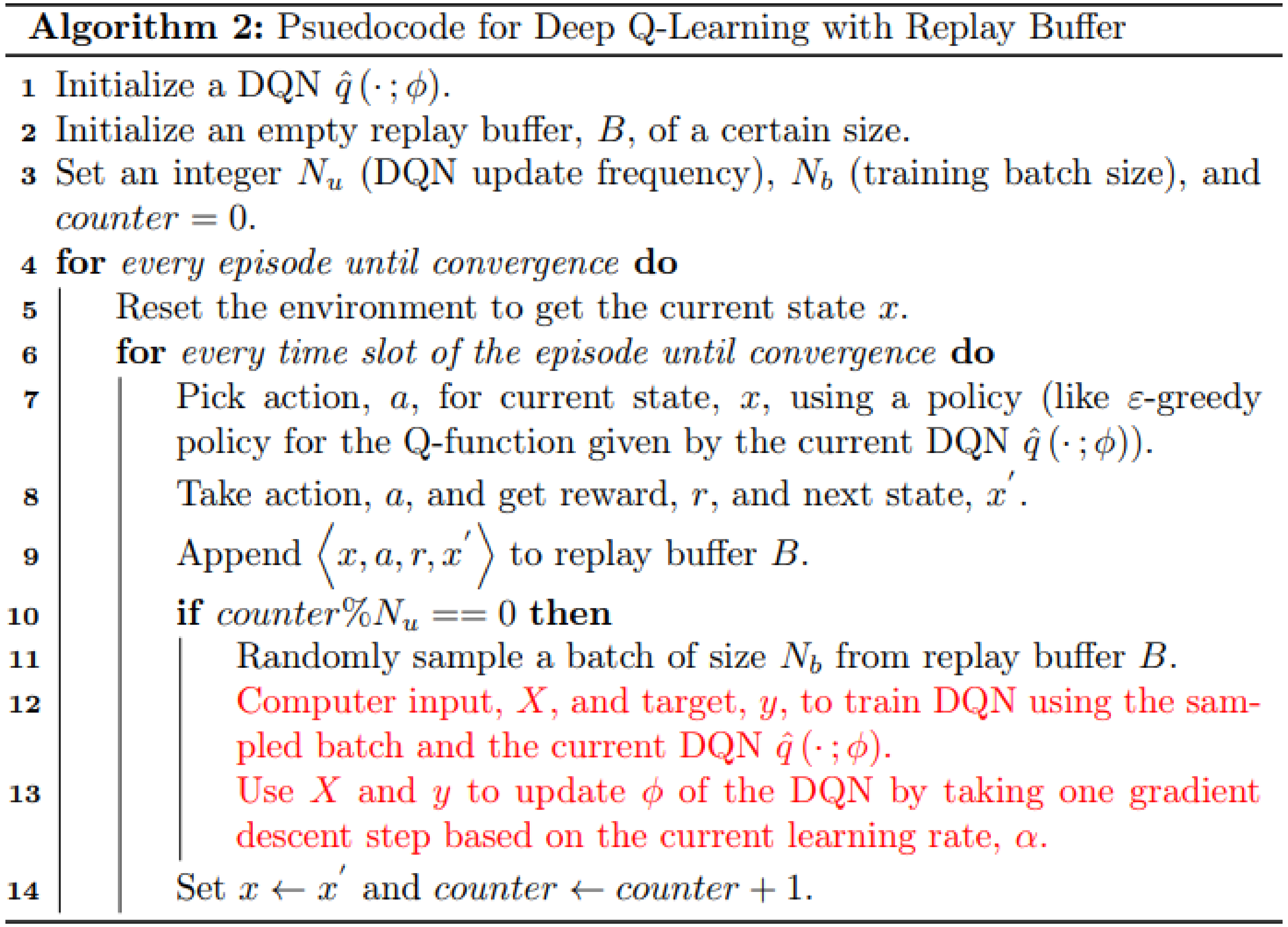
```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sampled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

The issue of non-stationary target:

- This rapidly* changing target is called a **non-stationary target** and it is a concern because by the time DQN parameters updates itself to reduce the mean-squared error between its prediction and the current target, the target changes!

* Rapidly because the target is changing between successive DQN parameter update.

Training DQN: Target Network



Algorithm 2: Psuedocode for Deep Q-Learning with Replay Buffer

```
Initialize a DQN $\hat{q}(\cdot\,;\phi)$.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (DQN update frequency), $N_b$ (training batch size), and
  $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
          policy for the Q-function given by the current DQN $\hat{q}(\cdot\,;\phi)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train DQN using the sam-
              pled batch and the current DQN $\hat{q}(\cdot\,;\phi)$.
            Use $X$ and $y$ to update $\phi$ of the DQN by taking one gradient
              descent step based on the current learning rate, $\alpha$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

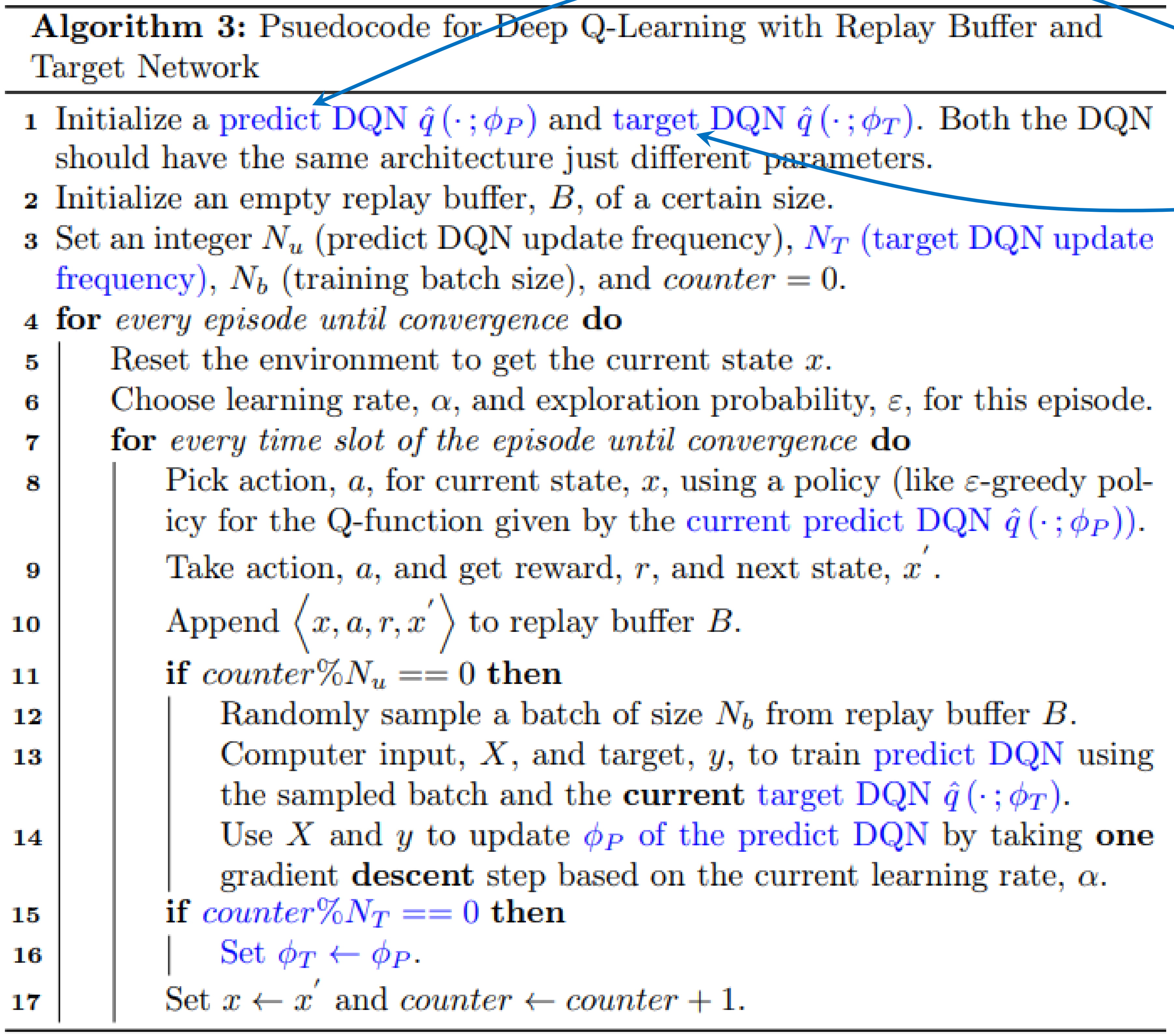
The issue of non-stationary target:

- This rapidly* changing target is called a **non-stationary target** and it is a concern because by the time DQN parameters updates itself to reduce the mean-squared error between its prediction and the current target, the target changes!

- Wait a minute!! This same issue of non-stationary target should be there in Q-learning as well. Why is it not an issue for Q-learning?
    - Because for Q-learning we can theoretically prove convergence to the optimal policy.
    - There does not exist such theoretical guarantees for DQN (that I know of).

Essentially, TD learning which is the basis of DQN and Q-learning is an intuition which works for Q-learning by needs modification for DQN.

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

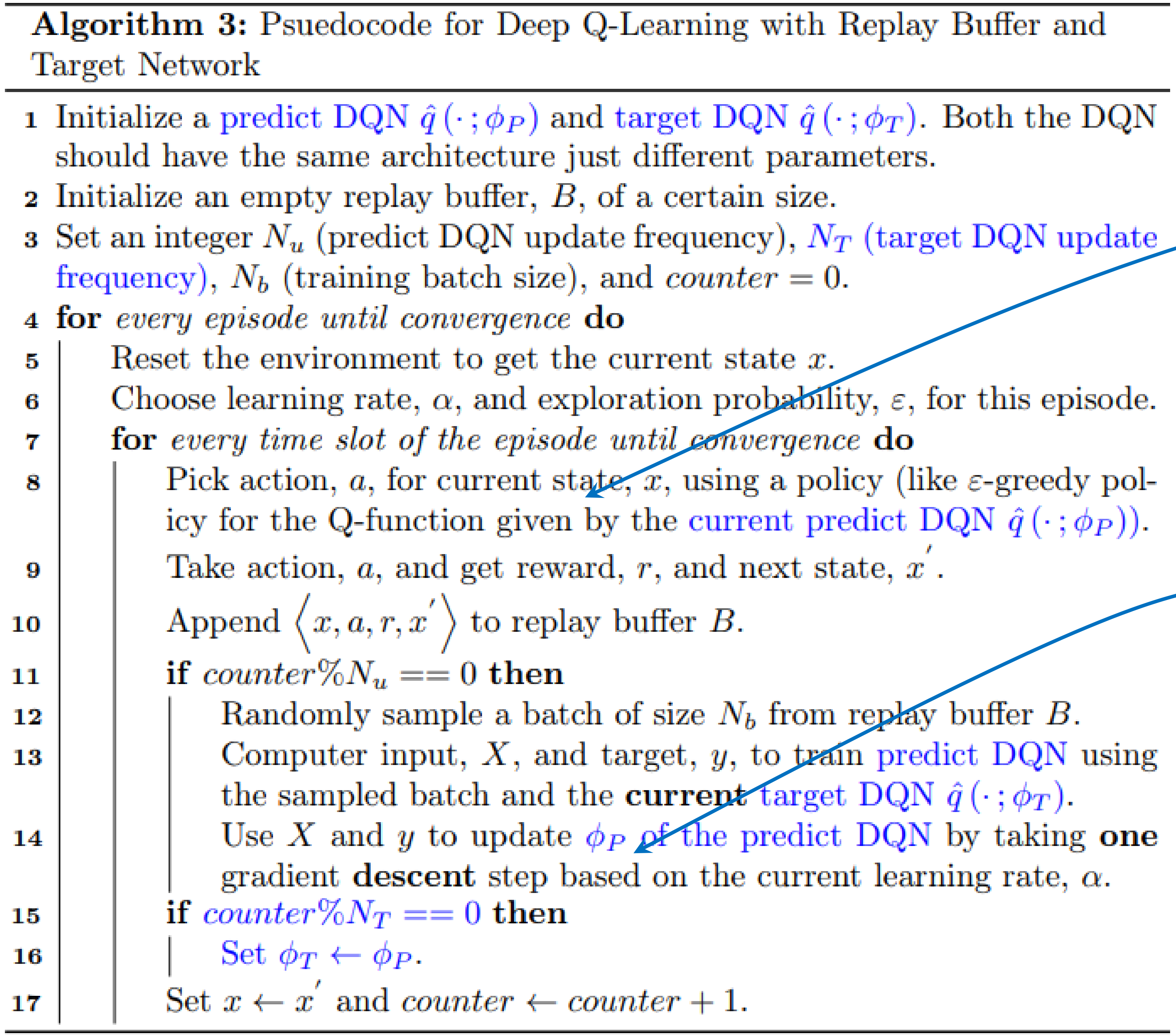
```
Initialize a predict DQN $\hat{q}(\cdot\,;\phi_P)$ and target DQN $\hat{q}(\cdot\,;\phi_T)$. Both the DQN should have the same architecture just different parameters.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot\,;\phi_P)$).
        Take action, $a$, and get reward, $r$, and next state, $x'$.
        Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch and the current target DQN $\hat{q}(\cdot\,;\phi_T)$.
            Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        if $counter \% N_T == 0$ then
            Set $\phi_T \leftarrow \phi_P$.
        Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN’s:

1. Predict DQN (also called the predict network).
2. Target DQN (also called target network).

- Both these neural networks must have the same architecture but different parameters.

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

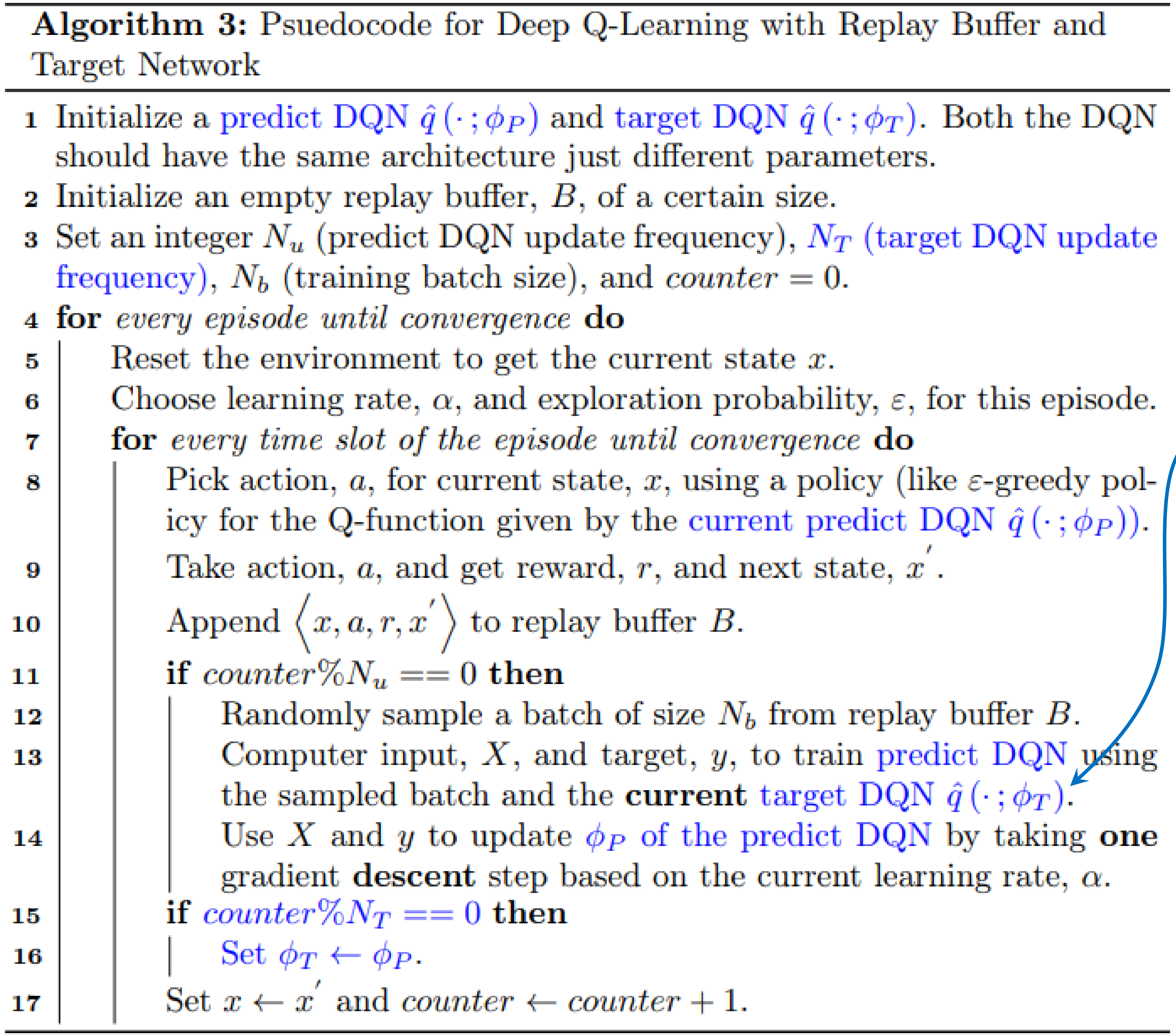
```
1  Initialize a predict DQN $\hat{q}(\cdot\,;\phi_P)$ and target DQN $\hat{q}(\cdot\,;\phi_T)$. Both the DQN
   should have the same architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
   frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy pol-
           icy for the Q-function given by the current predict DQN $\hat{q}(\cdot\,;\phi_P)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Computer input, $X$, and target, $y$, to train predict DQN using
               the sampled batch and the current target DQN $\hat{q}(\cdot\,;\phi_T)$.
14             Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one
               gradient descent step based on the current learning rate, $\alpha$.
15         if $counter \% N_T == 0$ then
16             Set $\phi_T \leftarrow \phi_P$.
17         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN's:

Predict DQN:

- This is the "usual DQN" that we used till now.
- Predict DQN is used for action selection. This happens in **line 8**.
- The parameters ϕ_P of the predict DQN is trained using gradient descent as before. This happens in **line 14**.
- After training is complete, **predict DQN is the final deliverable**. So essentially, while testing the trained DQN, we are going to use the predict DQN (target DQN doesn't come into question during testing).

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

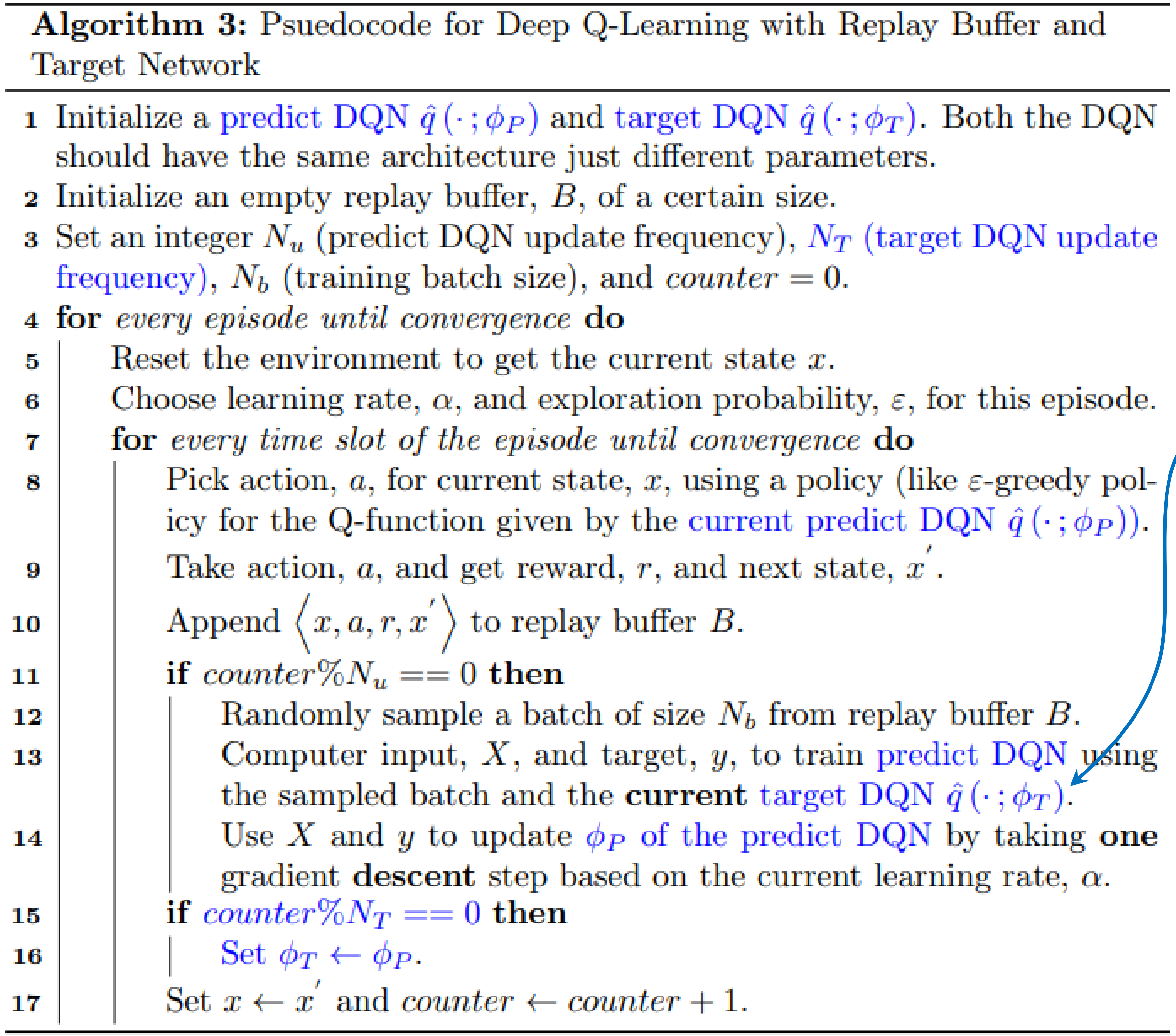
```
1  Initialize a predict DQN $\hat{q}(\cdot\,;\phi_P)$ and target DQN $\hat{q}(\cdot\,;\phi_T)$. Both the DQN should have the same architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot\,;\phi_P)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch and the current target DQN $\hat{q}(\cdot\,;\phi_T)$.
14             Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
15         if $counter \% N_T == 0$ then
16             Set $\phi_T \leftarrow \phi_P$.
17         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN's:

Target DQN:

- Target DQN is the newly added DQN.
- Target DQN is used ONLY to generate target values to train the predict DQN in **line 13**.
- **Overall intuition:** The entire idea is to **update the parameters of the target DQN MUCH SLOWER compared to predict DQN**. Therefore the target values that is used to train the predict DQN don't change immediately when the parameters ϕ_P of the predict DQN changes. This is how the issue of **non-stationarity of target is avoided** by using target network.

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

```
1  Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN
   should have the same architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
   frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy pol-
           icy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Computer input, $X$, and target, $y$, to train predict DQN using
               the sampled batch and the current target DQN $\hat{q}(\cdot;\phi_T)$.
14             Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one
               gradient descent step based on the current learning rate, $\alpha$.
15         if $counter \% N_T == 0$ then
16             Set $\phi_T \leftarrow \phi_P$.
17         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN's:

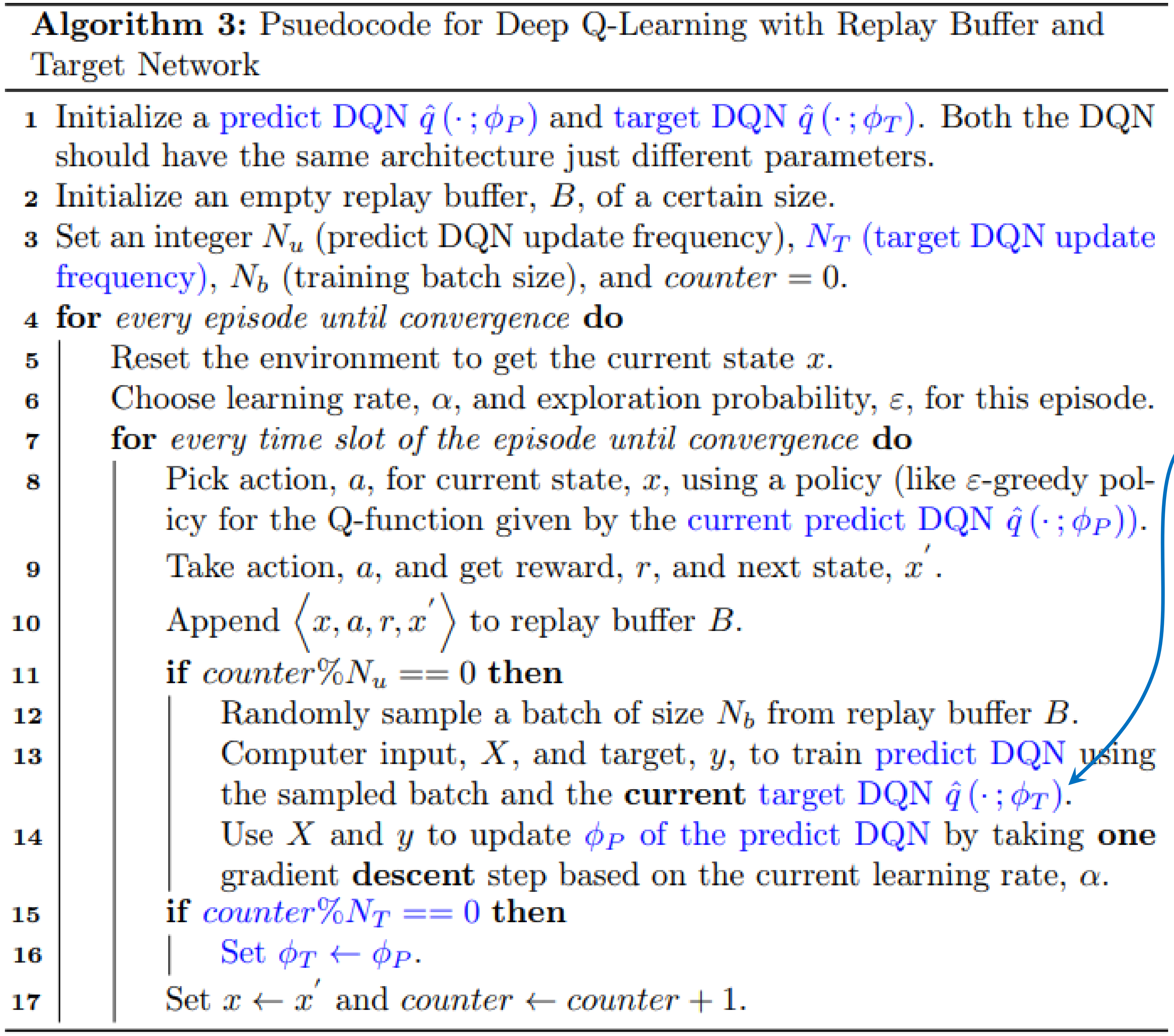
Target DQN:

- Target DQN is the newly added DQN.
- Target DQN is used ONLY to generate target values to train the predict DQN in **line 13**. Example: Say that one of the sample in the batch is $\langle x, a, r, x' \rangle$.
  - **Architecture 1:** Input $X = (x, a)$ and the target,

$$y = r + \beta \max_{a' \in \mathcal{A}(x')} \hat{q}\left(x', a'; \phi_T\right)$$

Target network

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

```
1  Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN should have the same architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch and the current target DQN $\hat{q}(\cdot;\phi_T)$.
14             Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
15         if $counter \% N_T == 0$ then
16             Set $\phi_T \leftarrow \phi_P$.
17         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN's:

Target DQN:

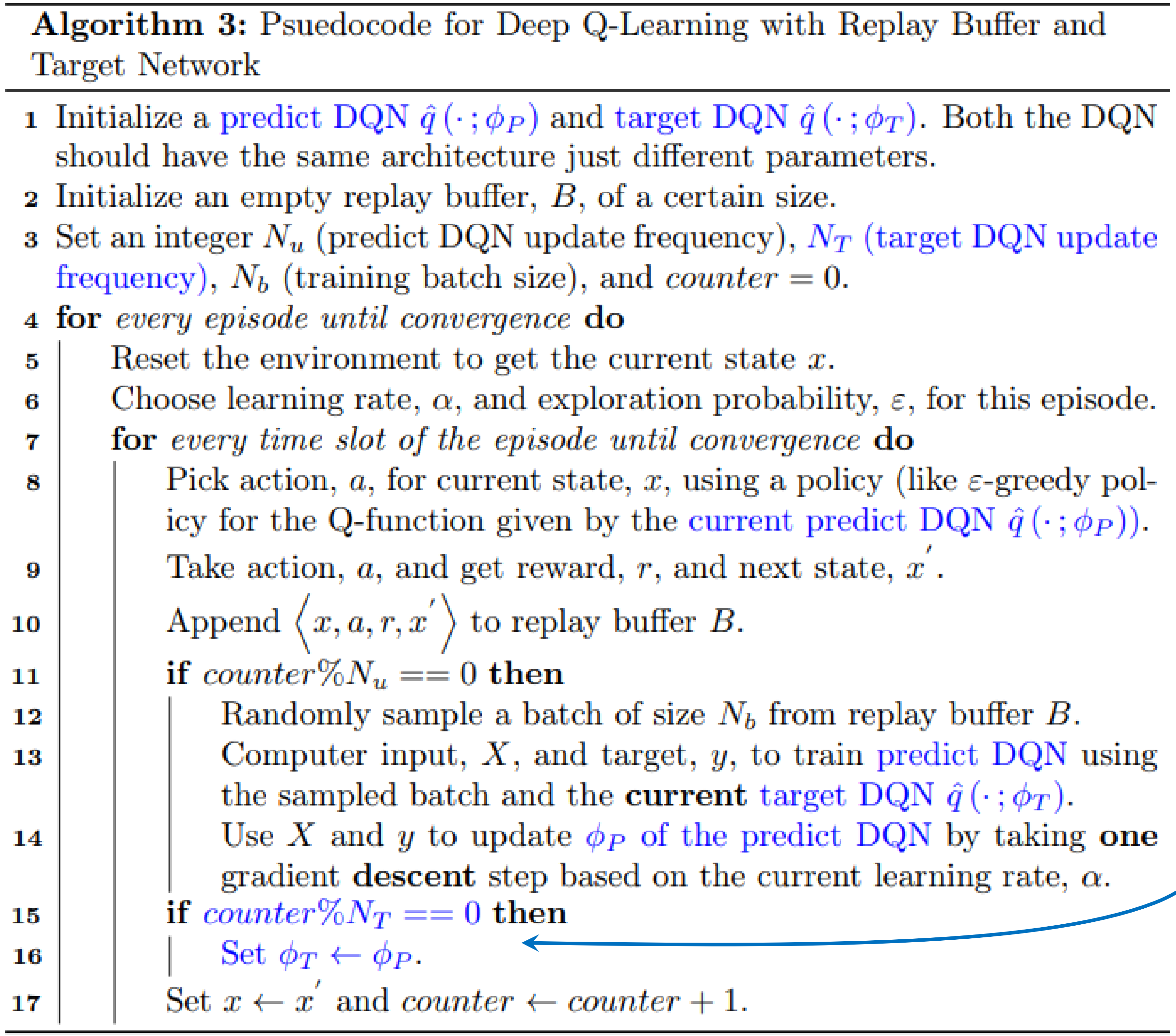
- Target DQN is the newly added DQN.
- Target DQN is used ONLY to generate target values to train the predict DQN in **line 13**. Example: Say that one of the sample in the batch is $\langle x, a, r, x' \rangle$.
  - **Architecture 2:** Input $X = x$ and the target y_i for action i is,

$$y_i = \begin{cases} r + \beta \max_{a' \in \mathcal{A}(x')} \hat{q}\left(x', a'; \phi_T\right) & , i = a \\ \hat{q}\left(x, i; \phi_P\right) & , i \neq a \end{cases}$$

Predict network is used to generate the "dummy targets". This is so that these targets contributes to zero loss during training.

Target network is used to generate the "main target".

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

```
Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN should have the same architecture just different parameters.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter\%N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch and the current target DQN $\hat{q}(\cdot;\phi_T)$.
            Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        if $counter\%N_T == 0$ then
            Set $\phi_T \leftarrow \phi_P$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

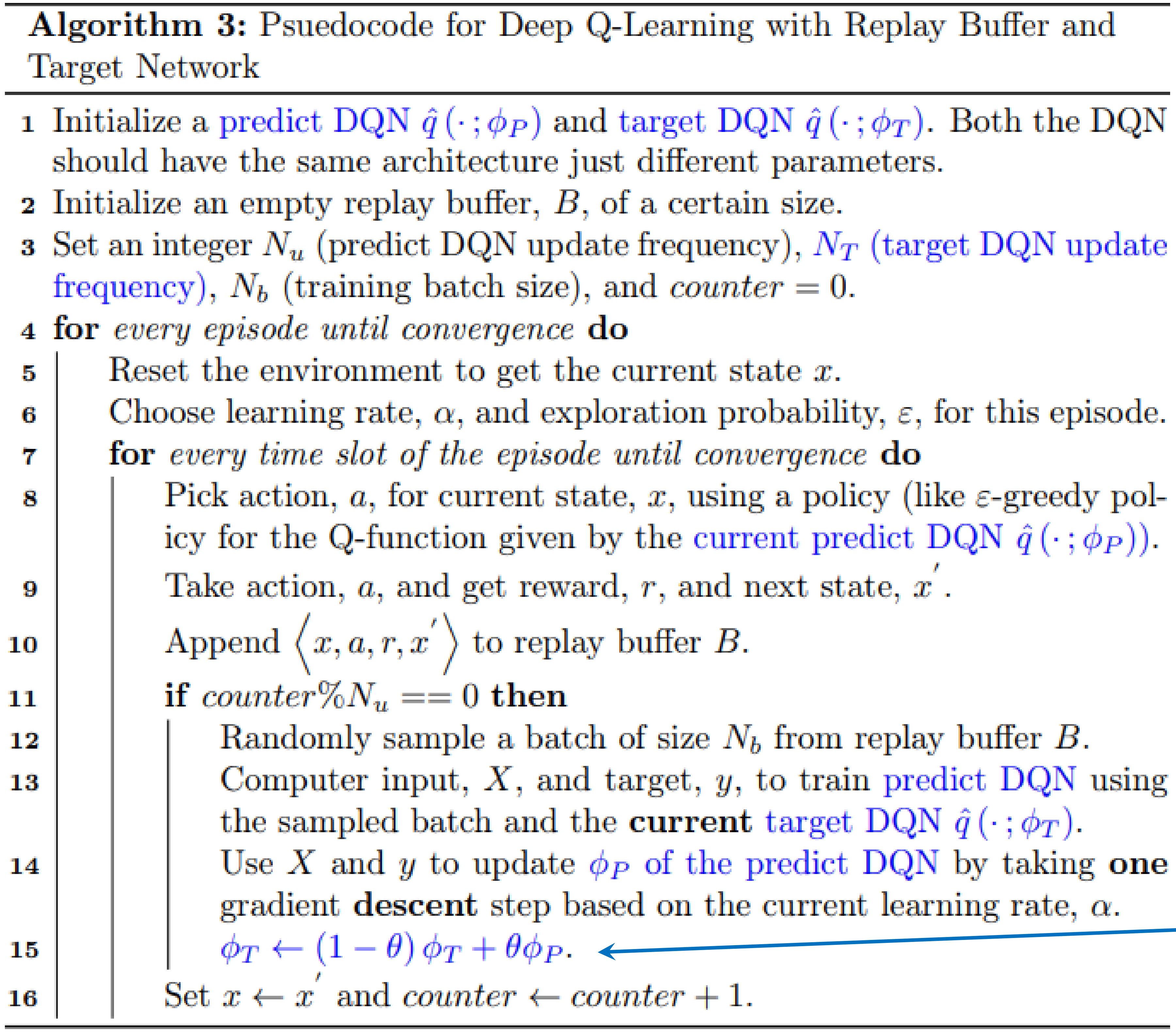
Solution of non-stationary target: Use two DQN's:

Target DQN:

- As mentioned in this slide (click here), parameter of the target DQN is updated slower compared to predict DQN. The are two approaches to achieve this:

    i. A common approach is to set $\phi_T = \phi_P$ every N_T time slots. We must have $\boldsymbol{N_T > N_u}$ to ensure that the parameters of the target DQN is updated slower compared to predict DQN (N_u is the training frequency of predict DQN).

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

```
Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN should have the same architecture just different parameters.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
        Take action, $a$, and get reward, $r$, and next state, $x'$.
        Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch and the current target DQN $\hat{q}(\cdot;\phi_T)$.
            Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
            $\phi_T \leftarrow (1-\theta)\,\phi_T + \theta\phi_P$.
        Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

Solution of non-stationary target: Use two DQN's:

Target DQN:

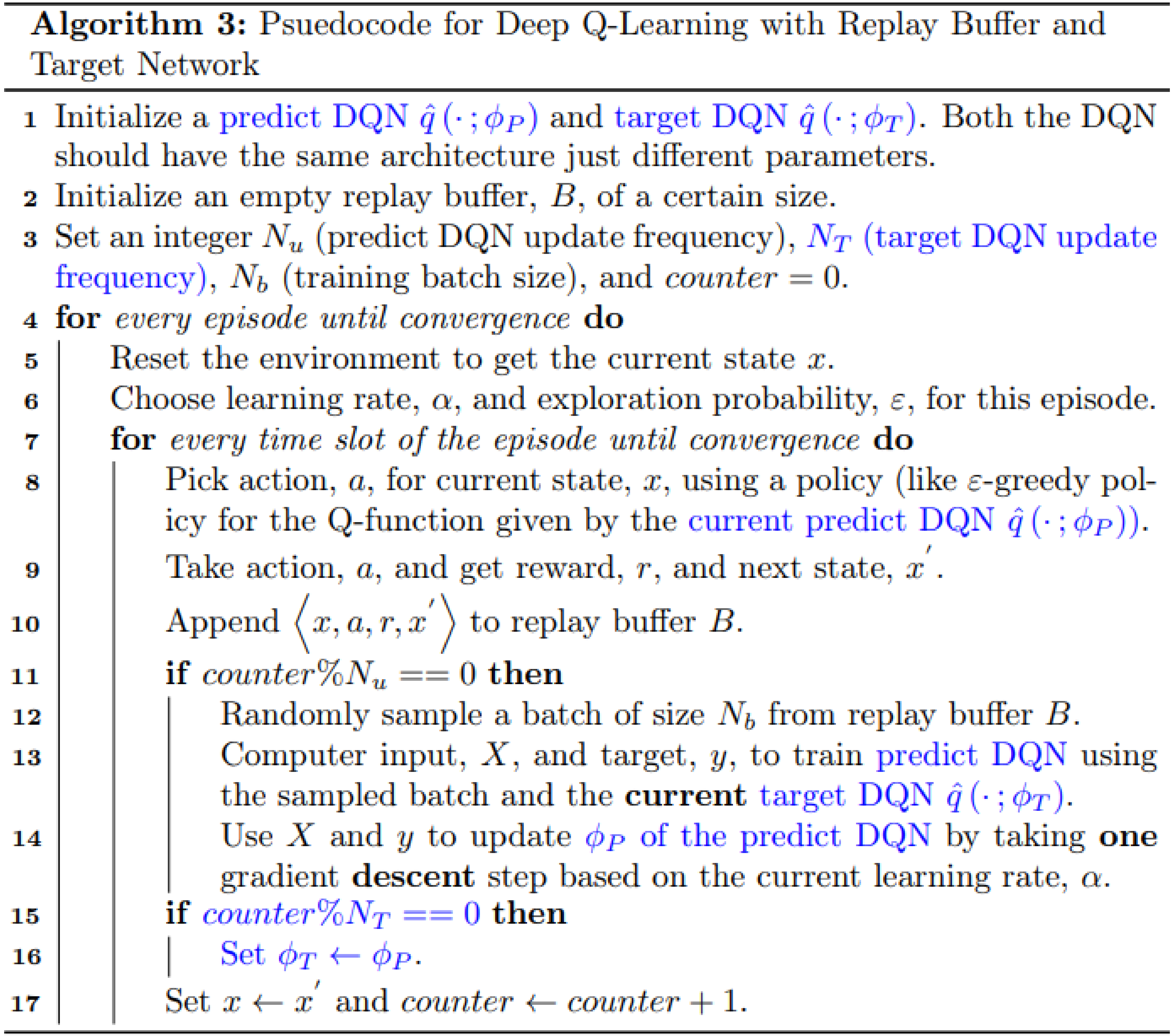
- ➢ As mentioned in this slide (click here), parameter of the target DQN is updated slower compared to predict DQN. The are two approaches to achieve this:

ii. The issue with the first approach is that the parameter ϕ_T of the target DQN is **suddenly updated** every N_T time slots. A sudden change in ϕ_T would lead to a sudden change in the target values which implies **non-stationarity** and hence not desired. We can do a smooth update of ϕ_T as follows,

$$\phi_T = (1-\theta)\phi_T + \theta\phi_P$$

where **θ is kept small in order to ensure that ϕ_T updates slowly.** $\boldsymbol{\theta = 0.001}$ works well in practice.

Training DQN: Target Network



Algorithm 3: Psuedocode for Deep Q-Learning with Replay Buffer and Target Network

```
Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN
should have the same architecture just different parameters.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy pol-
        icy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
        Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
        Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Computer input, $X$, and target, $y$, to train predict DQN using
            the sampled batch and the current target DQN $\hat{q}(\cdot;\phi_T)$.
            Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one
            gradient descent step based on the current learning rate, $\alpha$.
        if $counter \% N_T == 0$ then
            Set $\phi_T \leftarrow \phi_P$.
        Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

Training DQN

- The concept of experience replay and target network originated in the following paper:

  V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. **Human level control through deep reinforcement learning**. Nature, 518 (7540):529–533, 2015.

- **Important: Experience replay** and **target network** should be incorporated in any DQN training algorithm (like the one you will do in programming assignment 4).

Training DQN: Double DQN

Choose a random variable w uniformly at random in the interval $[0,1]$. If $w \leq 0.5$, then update $q_1\ (x, a)$:

$$q_1(x,a) = q_1(x,a) + \alpha\left(r + \beta\, q_2\left(x', \underset{a' \in \mathcal{A}(x')}{\operatorname{argmax}} q_1(x', a')\right) - q_1(x,a)\right)$$

Else update $q_2\ (x, a)$:

$$q_2(x,a) = q_2(x,a) + \alpha\left(r + \beta\, q_1\left(x', \underset{a' \in \mathcal{A}(x')}{\operatorname{argmax}} q_2(x', a')\right) - q_2(x,a)\right)$$

- Recall double Q-learning from module 3. It is used because **Q-learning overestimates the Q-value**. Similarly DQN training algorithms that we have seen till now also overestimates the Q-values. Hence the need for Double DQN.

- Let's recollect double Q-learning. The most important line of the pseudocode of double Q-learning is shown in the left.

- Main idea behind double Q-learning:
  - Keep two copies of Q-functions, q_1 and q_2.
  - Randomly select which Q-function to update.
  - While updating a Q-function, use one of them for **action selection** and the other for **evaluation**.

Training DQN: Double DQN

Choose a random variable w uniformly at random in the interval $[0,1]$. If $w \leq 0.5$, then update $q_1\,(x, a)$:

$$q_1(x,a) = q_1(x,a) + \alpha\left(r + \beta\, q_2\left(x', \underset{a' \in \mathcal{A}(x')}{\operatorname{argmax}}\, q_1(x', a')\right) - q_1(x,a)\right)$$

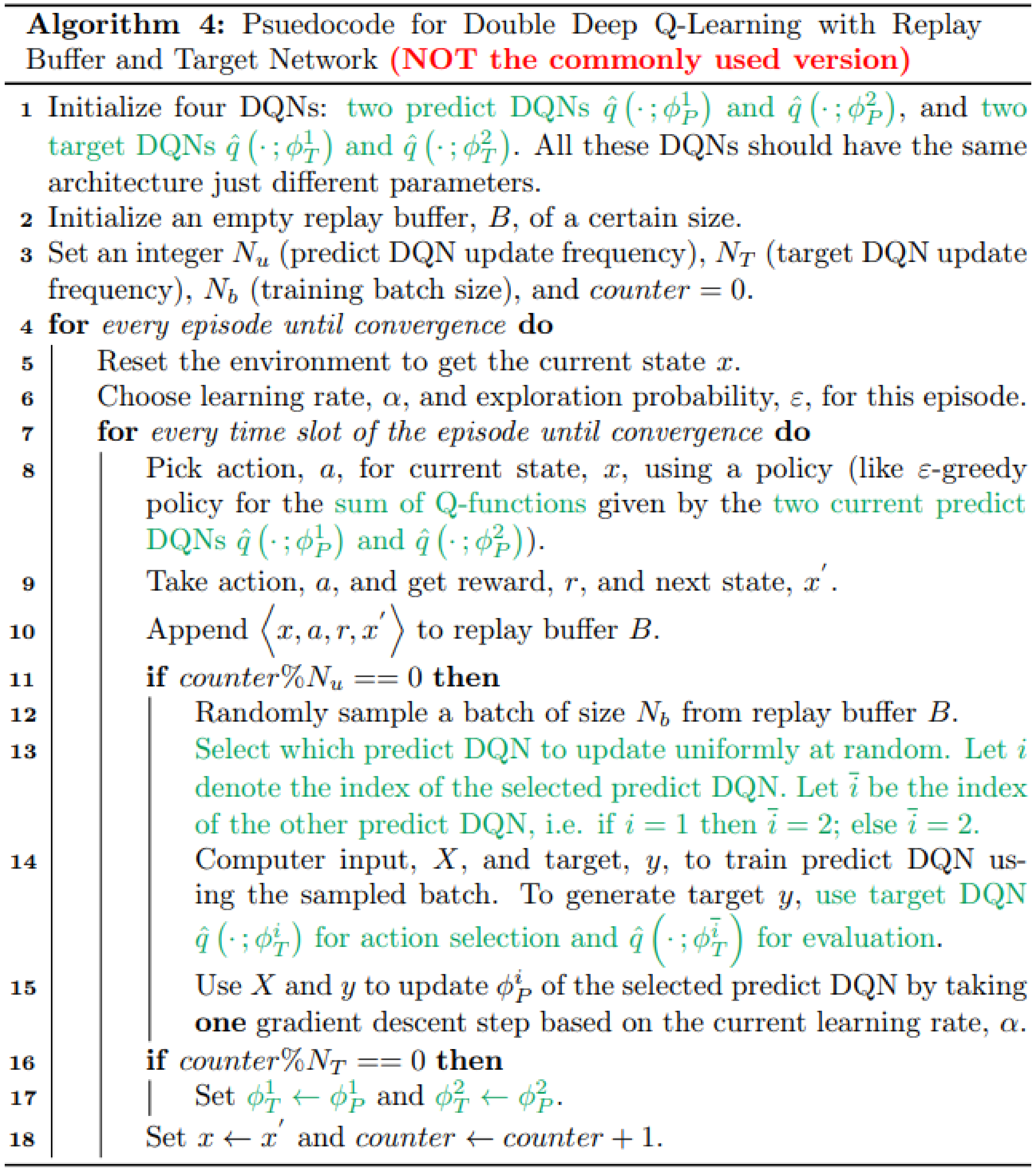
Else update $q_2\,(x, a)$:

$$q_2(x,a) = q_2(x,a) + \alpha\left(r + \beta\, q_1\left(x', \underset{a' \in \mathcal{A}(x')}{\operatorname{argmax}}\, q_2(x', a')\right) - q_2(x,a)\right)$$

- In double Q-learning, **these** are an estimate of a sample **return**. For double DQN, these are the target values.

- So, the main steps of the double DQN are:
  - Keep two copies of each of predict and target DQN.
  - Randomly select which predict network to update (using gradient descent).
  - **To generate the training data, use one target network for action selection and the other for evaluation**.
  - The target networks parameters are updated using corresponding predict network parameters (as before).

Training DQN: Double DQN



Algorithm 4: Psuedocode for Double Deep Q-Learning with Replay Buffer and Target Network **(NOT the commonly used version)**

```
Initialize four DQNs: two predict DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$, and two target DQNs $\hat{q}(\cdot;\phi_T^1)$ and $\hat{q}(\cdot;\phi_T^2)$. All these DQNs should have the same architecture just different parameters.
Initialize an empty replay buffer, $B$, of a certain size.
Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
for every episode until convergence do
    Reset the environment to get the current state $x$.
    Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
    for every time slot of the episode until convergence do
        Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the sum of Q-functions given by the two current predict DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$).
        Take action, $a$, and get reward, $r$, and next state, $x'$.
        Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
        if $counter \% N_u == 0$ then
            Randomly sample a batch of size $N_b$ from replay buffer $B$.
            Select which predict DQN to update uniformly at random. Let $i$ denote the index of the selected predict DQN. Let $\bar{i}$ be the index of the other predict DQN, i.e. if $i = 1$ then $\bar{i} = 2$; else $\bar{i} = 2$.
            Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch. To generate target $y$, use target DQN $\hat{q}(\cdot;\phi_T^i)$ for action selection and $\hat{q}(\cdot;\phi_T^{\bar{i}})$ for evaluation.
            Use $X$ and $y$ to update $\phi_P^i$ of the selected predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
        if $counter \% N_T == 0$ then
            Set $\phi_T^1 \leftarrow \phi_P^1$ and $\phi_T^2 \leftarrow \phi_P^2$.
        Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

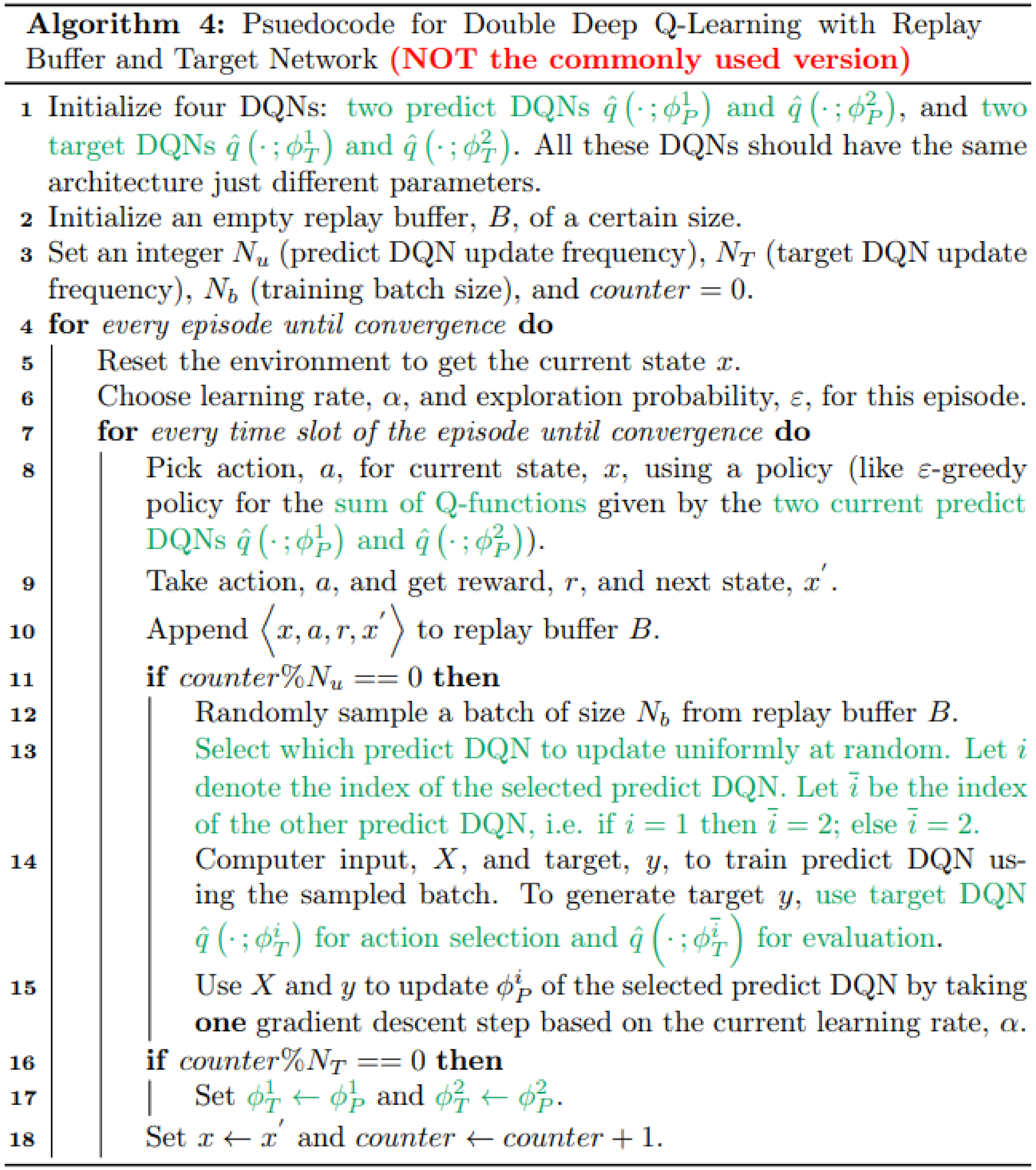
- The pseudocode of the "main idea" discussed in the previous slide is shown in the left.

Disclaimer: The original paper that discussed double DQN is:

H.V. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning", Proceedings of the AAAI conference on artificial intelligence, 2016

The pseudocode shown in the left is NOT the one used in practice nor the one discussed in the original paper. We are discussing it just to convey the main idea behind double DQN. The pseudocode that is used in practice and which is also discussed in the original paper will be discussed next.

Training DQN: Double DQN



Algorithm 4: Psuedocode for Double Deep Q-Learning with Replay Buffer and Target Network **(NOT the commonly used version)**

```
1  Initialize four DQNs: two predict DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$, and two
   target DQNs $\hat{q}(\cdot;\phi_T^1)$ and $\hat{q}(\cdot;\phi_T^2)$. All these DQNs should have the same
   architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
   frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the sum of Q-functions given by the two current predict
           DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Select which predict DQN to update uniformly at random. Let $i$
               denote the index of the selected predict DQN. Let $\bar{i}$ be the index
               of the other predict DQN, i.e. if $i = 1$ then $\bar{i} = 2$; else $\bar{i} = 2$.
14             Computer input, $X$, and target, $y$, to train predict DQN us-
               ing the sampled batch. To generate target $y$, use target DQN
               $\hat{q}(\cdot;\phi_T^i)$ for action selection and $\hat{q}(\cdot;\phi_T^{\bar{i}})$ for evaluation.
15             Use $X$ and $y$ to update $\phi_P^i$ of the selected predict DQN by taking
               one gradient descent step based on the current learning rate, $\alpha$.
16         if $counter \% N_T == 0$ then
17             Set $\phi_T^1 \leftarrow \phi_P^1$ and $\phi_T^2 \leftarrow \phi_P^2$.
18         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

- Most of the lines of the pseudocode should be self explanatory (provided you have looked into the psuedocodes discussed before in these slides). Lines 8, 13, 14, and 15 needs some explanation.

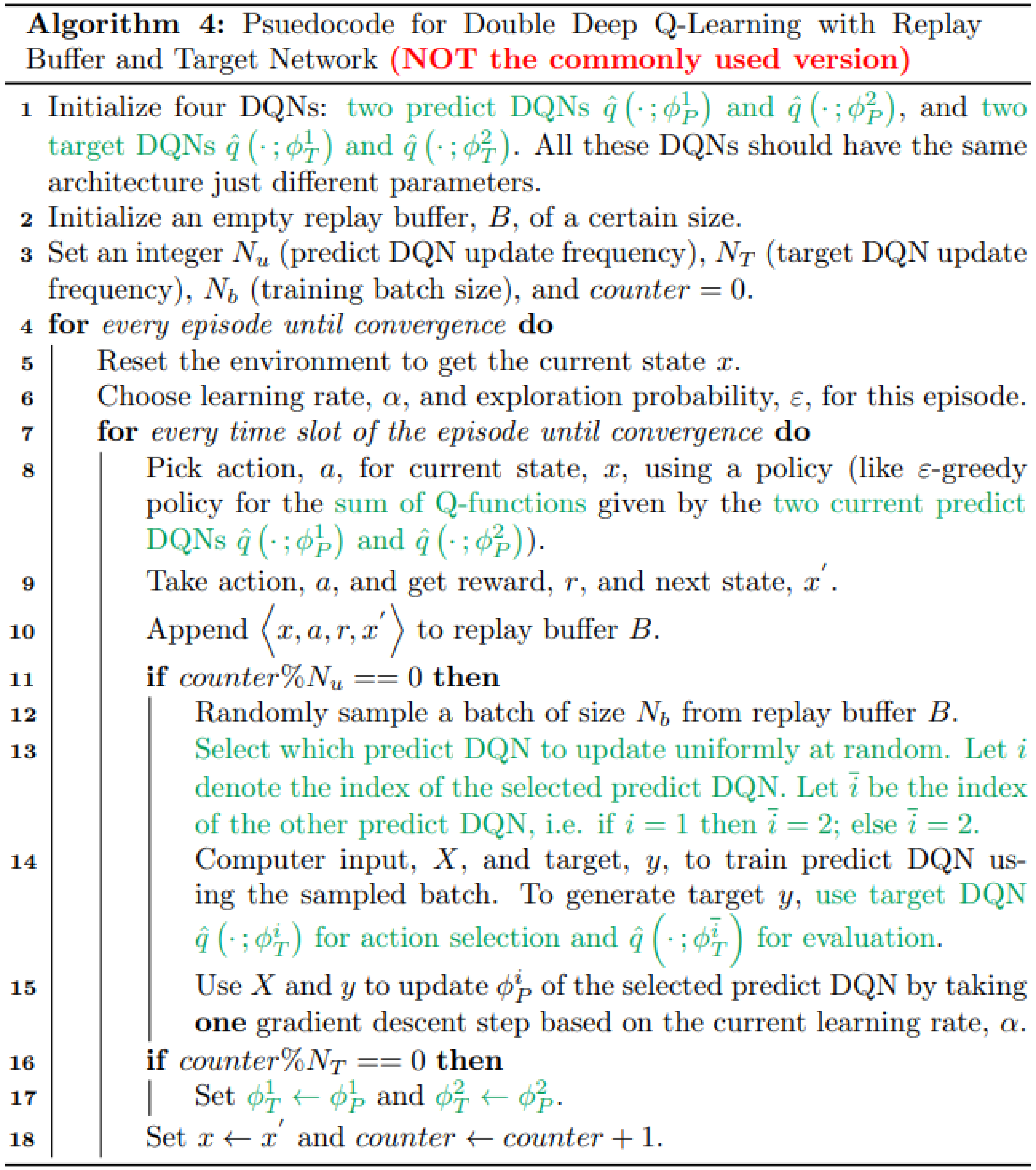
- **Line 8**: Recall that in double Q-learning, action is selected based on the sum of the two Q-functions. Similarly, in double DQN, action is selected based on the sum of the prediction of the two predict networks as follows,

$$a = \begin{cases} \underset{\bar{a} \in \mathcal{A}(x)}{\arg\max} \quad \hat{q}\left(x, \bar{a}; \phi_P^1\right) + \hat{q}\left(x, \bar{a}; \phi_P^2\right) & , w.p.\ 1 - \varepsilon \\ \\ \text{select uniformly at random from } \mathcal{A}(x) & , w.p.\ \varepsilon \end{cases}$$

w.p. means “with probability”.

- **Lines 13 and 15**: Line 13 chooses (uniformly at random) which of the two predict DQN to update. Let the index of selected predict DQN be $i \in \{1,2\}$. Then ϕ_P^i is updated in line 15 using gradient descent.

Training DQN: Double DQN



Algorithm 4: Psuedocode for Double Deep Q-Learning with Replay Buffer and Target Network **(NOT the commonly used version)**

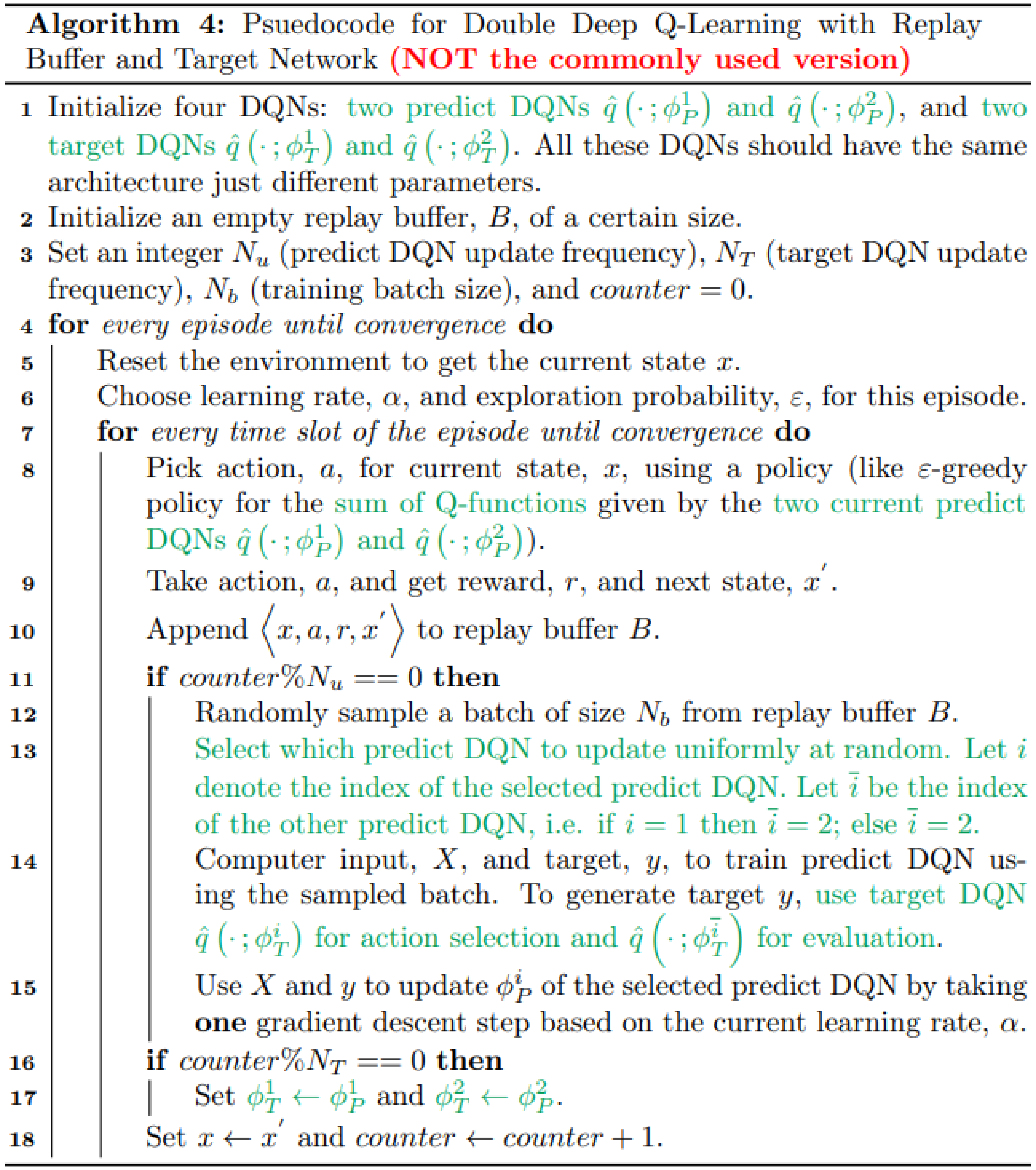
```
1  Initialize four DQNs: two predict DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$, and two
   target DQNs $\hat{q}(\cdot;\phi_T^1)$ and $\hat{q}(\cdot;\phi_T^2)$. All these DQNs should have the same
   architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
   frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the sum of Q-functions given by the two current predict
           DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Select which predict DQN to update uniformly at random. Let $i$
               denote the index of the selected predict DQN. Let $\bar{i}$ be the index
               of the other predict DQN, i.e. if $i = 1$ then $\bar{i} = 2$; else $\bar{i} = 2$.
14             Computer input, $X$, and target, $y$, to train predict DQN us-
               ing the sampled batch. To generate target $y$, use target DQN
               $\hat{q}(\cdot;\phi_T^i)$ for action selection and $\hat{q}(\cdot;\phi_T^{\bar{i}})$ for evaluation.
15             Use $X$ and $y$ to update $\phi_P^i$ of the selected predict DQN by taking
               one gradient descent step based on the current learning rate, $\alpha$.
16         if $counter \% N_T == 0$ then
17             Set $\phi_T^1 \leftarrow \phi_P^1$ and $\phi_T^2 \leftarrow \phi_P^2$.
18         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

- **Line 14**: Generates the training data used to update ϕ_P^i in line 15. Example: Say that one of the sample in the batch is $\langle x, a, r, x' \rangle$.
  - **Architecture 1:** Input $X = (x, a)$ and the target,

$$y = r + \beta\,\widehat{q}\left(x', \underbrace{\underbrace{\arg\max_{a'\in\mathcal{A}(x')} \widehat{q}\left(x', a', \phi_T^{i}\right)}_{\text{Action selection}} ; \phi_T^{\bar{i}}}_{\text{Evaluation}}\right)$$

  - IMPORTANT: Highlight which parameters is used for action selection and evaluation and its significance.

Training DQN: Double DQN



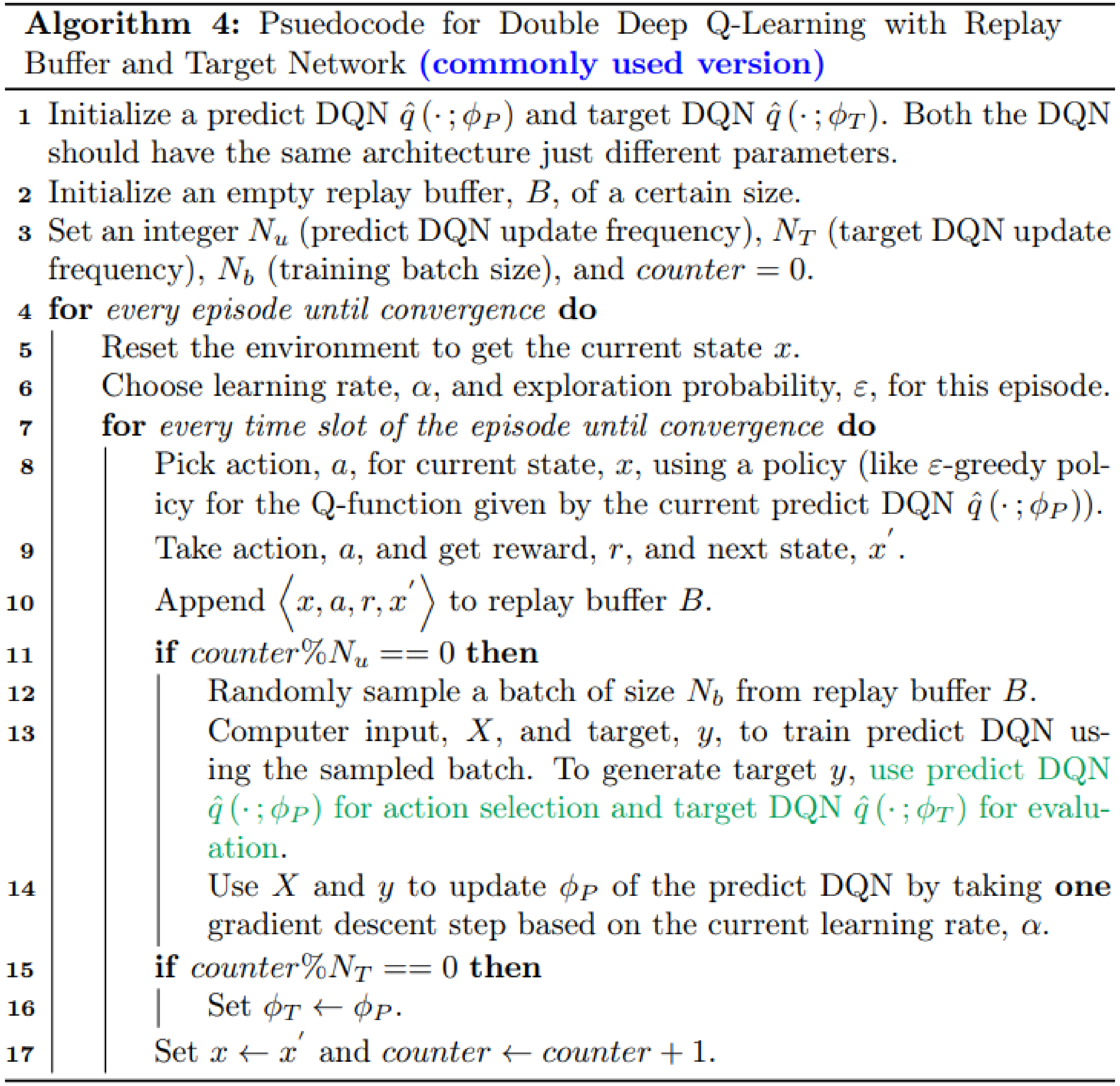
Algorithm 4: Psuedocode for Double Deep Q-Learning with Replay Buffer and Target Network **(NOT the commonly used version)**

```
1  Initialize four DQNs: two predict DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$, and two
   target DQNs $\hat{q}(\cdot;\phi_T^1)$ and $\hat{q}(\cdot;\phi_T^2)$. All these DQNs should have the same
   architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update
   frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy
           policy for the sum of Q-functions given by the two current predict
           DQNs $\hat{q}(\cdot;\phi_P^1)$ and $\hat{q}(\cdot;\phi_P^2)$).
9          Take action, $a$, and get reward, $r$, and next state, $x'$.
10         Append $\langle x, a, r, x' \rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Select which predict DQN to update uniformly at random. Let $i$
               denote the index of the selected predict DQN. Let $\bar{i}$ be the index
               of the other predict DQN, i.e. if $i = 1$ then $\bar{i} = 2$; else $\bar{i} = 2$.
14             Computer input, $X$, and target, $y$, to train predict DQN us-
               ing the sampled batch. To generate target $y$, use target DQN
               $\hat{q}(\cdot;\phi_T^i)$ for action selection and $\hat{q}(\cdot;\phi_T^{\bar{i}})$ for evaluation.
15             Use $X$ and $y$ to update $\phi_P^i$ of the selected predict DQN by taking
               one gradient descent step based on the current learning rate, $\alpha$.
16         if $counter \% N_T == 0$ then
17             Set $\phi_T^1 \leftarrow \phi_P^1$ and $\phi_T^2 \leftarrow \phi_P^2$.
18         Set $x \leftarrow x'$ and $counter \leftarrow counter + 1$.
```

- **Line 14**: Generates the training data used to update ϕ_P^i in line 15. Example: Say that one of the sample in the batch is $\langle x, a, r, x' \rangle$.
  - **Architecture 2:** Input $X = x$ and the target,

$$y_i = \begin{cases} r + \beta\, \hat{q}\left(x', \underset{a' \in \mathcal{A}(x')}{\arg\max}\, \hat{q}\left(x', a', \phi_T^i\right); \phi_T^{\bar{i}}\right) & , i = a \\ \hat{q}\left(x, i; \phi_P^i\right) & , i \neq a \end{cases}$$

Training DQN: Double DQN



Algorithm 4: Psuedocode for Double Deep Q-Learning with Replay Buffer and Target Network **(commonly used version)**

```
1  Initialize a predict DQN $\hat{q}(\cdot;\phi_P)$ and target DQN $\hat{q}(\cdot;\phi_T)$. Both the DQN should have the same architecture just different parameters.
2  Initialize an empty replay buffer, $B$, of a certain size.
3  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size), and $counter = 0$.
4  for every episode until convergence do
5      Reset the environment to get the current state $x$.
6      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
7      for every time slot of the episode until convergence do
8          Pick action, $a$, for current state, $x$, using a policy (like $\varepsilon$-greedy policy for the Q-function given by the current predict DQN $\hat{q}(\cdot;\phi_P)$).
9          Take action, $a$, and get reward, $r$, and next state, $x^{'}$.
10         Append $\left\langle x, a, r, x^{'} \right\rangle$ to replay buffer $B$.
11         if $counter \% N_u == 0$ then
12             Randomly sample a batch of size $N_b$ from replay buffer $B$.
13             Computer input, $X$, and target, $y$, to train predict DQN using the sampled batch. To generate target $y$, use predict DQN $\hat{q}(\cdot;\phi_P)$ for action selection and target DQN $\hat{q}(\cdot;\phi_T)$ for evaluation.
14             Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current learning rate, $\alpha$.
15         if $counter \% N_T == 0$ then
16             Set $\phi_T \leftarrow \phi_P$.
17         Set $x \leftarrow x^{'}$ and $counter \leftarrow counter + 1$.
```

➢ The commonly used version of double DQN uses the exact same pseudocode as the one with replay buffer and target network with difference coming only in **line 13** which is used to generate the training data.

$$y = r + \beta\, \widehat{q}\left(x^{'}, \underset{a^{'} \in \mathcal{A}(x^{'})}{\arg\max}\, \widehat{q}\left(x^{'}, a^{'}, \phi_P\right); \phi_T\right)$$

$$y_i = \begin{cases} r + \beta\, \widehat{q}\left(x^{'}, \underset{a^{'} \in \mathcal{A}(x^{'})}{\arg\max}\, \widehat{q}\left(x^{'}, a^{'}, \phi_P\right); \phi_T\right) & , i = a \\ \widehat{q}\left(x, i; \phi_P\right) & , i \neq a \end{cases}$$

Thank you