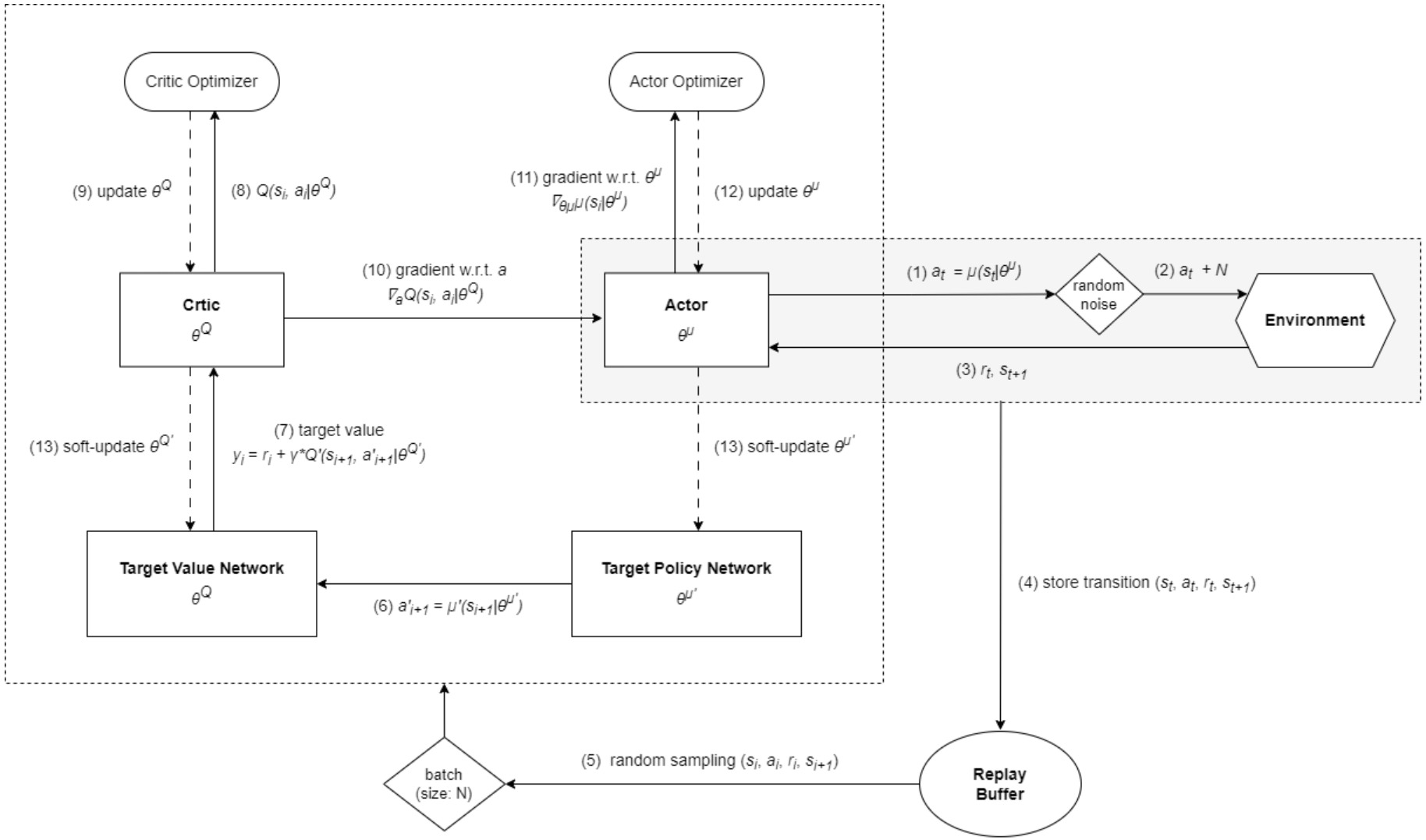
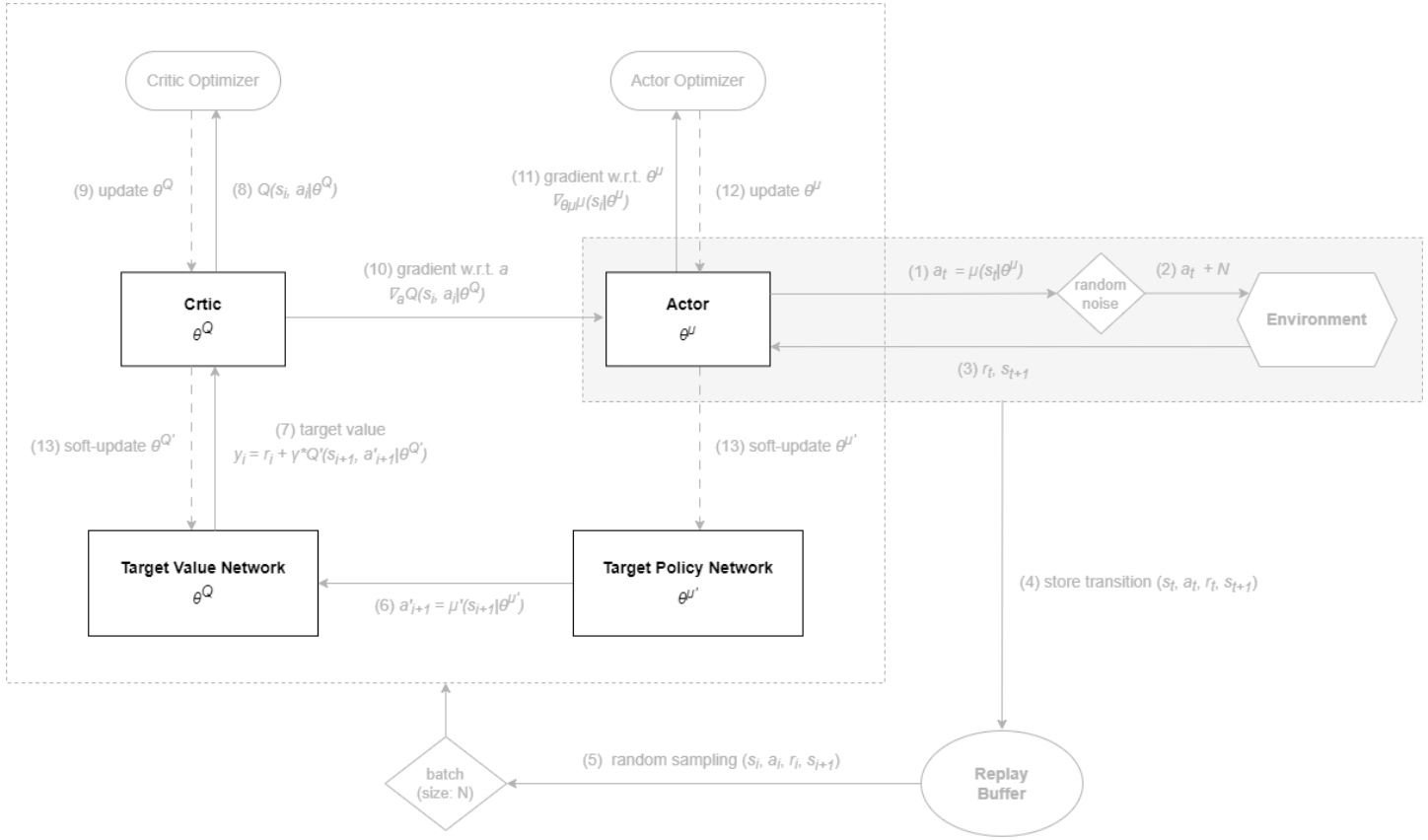
Workflow of DDPG

Overview


Critic Optimizer
Actor Optimizer
(9) update θ^Q
(8) $Q(s_i, a_i|\theta^Q)$
(11) gradient w.r.t. θ^μ
$\nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$
(12) update θ^μ
Crtic
θ^Q
(10) gradient w.r.t. a
$\nabla_a Q(s_i, a_i|\theta^Q)$
Actor
θ^μ
(1) $a_t = \mu(s_t|\theta^\mu)$
random noise
(2) $a_t + N$
Environment
(3) r_t, s_{t+1}
(13) soft-update $\theta^{Q'}$
(7) target value
$y_i = r_i + \gamma * Q'(s_{i+1}, a'_{i+1}|\theta^{Q'})$
(13) soft-update $\theta^{\mu'}$
Target Value Network
θ^Q
Target Policy Network
$\theta^{\mu'}$
(6) $a'_{i+1} = \mu'(s_{i+1}|\theta^{\mu'})$
(4) store transition (s_t, a_t, r_t, s_{t+1})
batch (size: N)
(5) random sampling (s_i, a_i, r_i, s_{i+1})
Replay Buffer

Structure



The structure of DDPG is composed of four networks:

- Actor (policy network) - μ
- Critic (value network) - Q
- Target policy network - μ'
- Target value network - Q'

The Actor and Critic are the networks that DDPG aims to optimize, while the target policy and value network are used to estimate the target values to update/optimize the Actor and Critic.

Policy and value networks

- For the Actor and the target policy network,
  - Input: **state**
  - Output: **action** (not a distribution of action probability!)

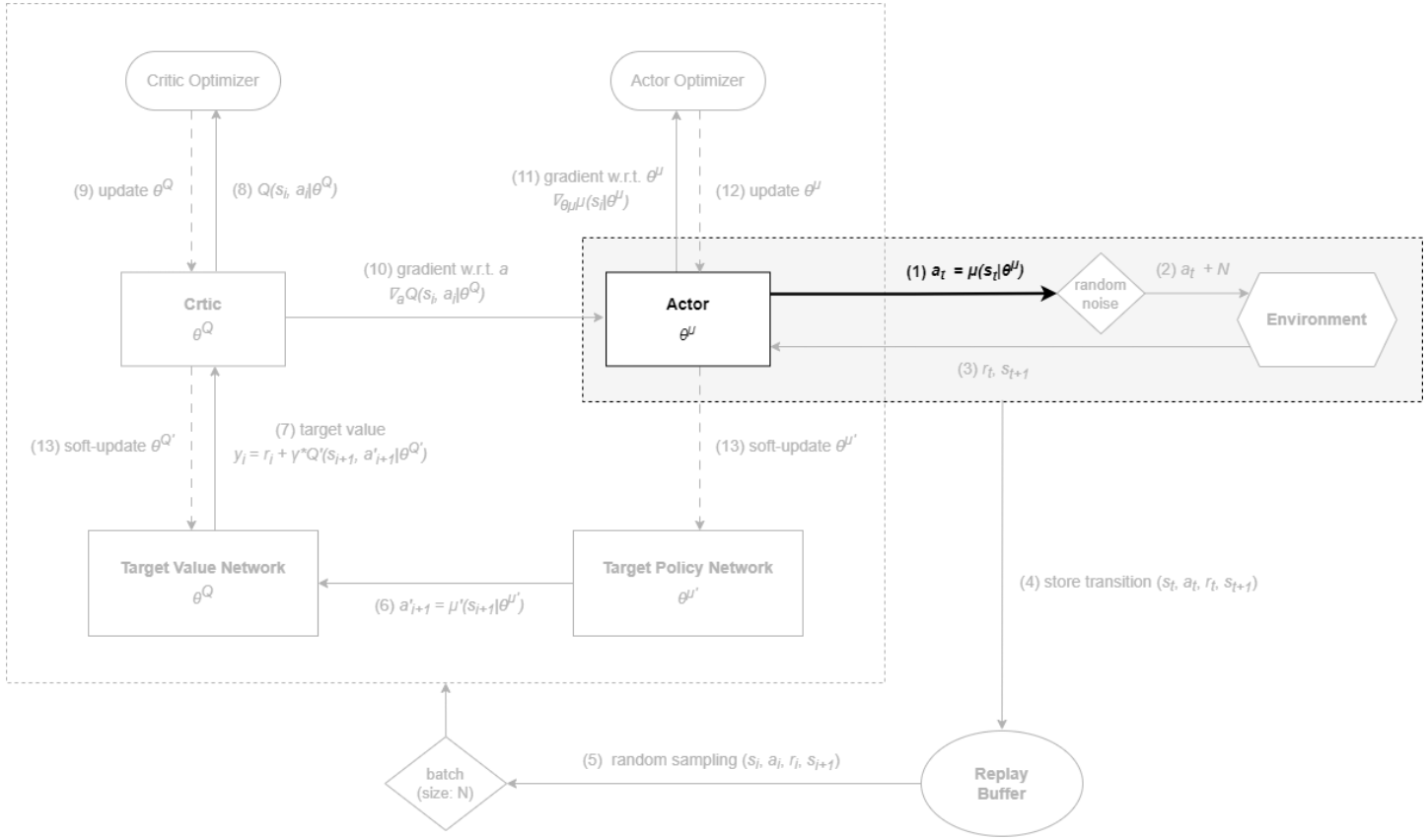
- For the Critic and the target value network,
  - Input: **state-action pair**
  - Output: **Q-value**

Notation

Let $\boldsymbol{s}$ denote the state and $\boldsymbol{a}$ denote the action of the agent,

Network	Parameter	Input	Output
Actor	θ^{μ}	s	$\mu(s\|\theta^{\mu})$
Critic	θ^{Q}	s, a	$Q(s, a\|\theta^{Q})$
Target policy network	$\theta^{\mu'}$	s	$\mu'(s\|\theta^{\mu'})$
Target value network	$\theta^{Q'}$	s, a	$Q'(s, a\|\theta^{Q'})$

Step 1. Select an action

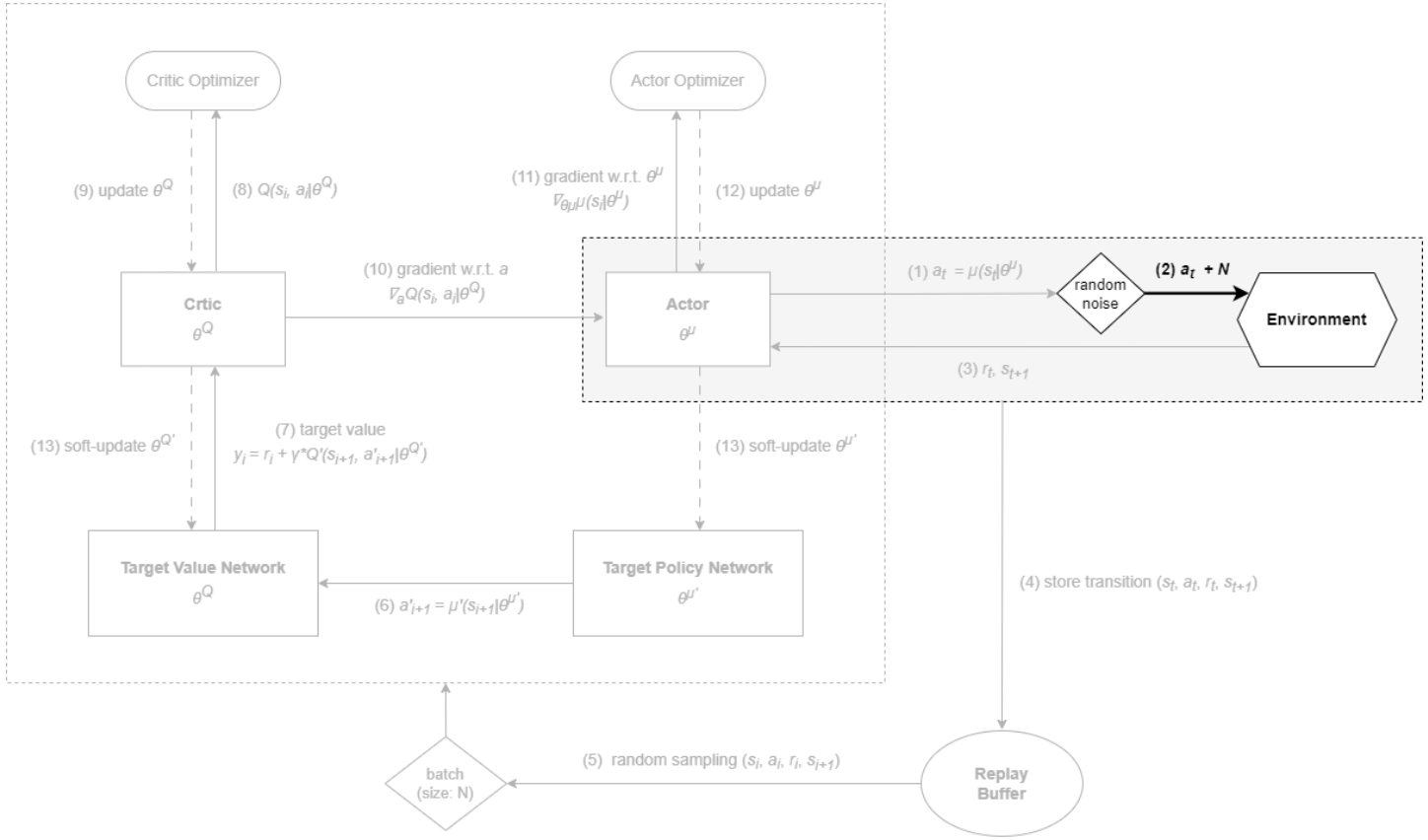


To optimize the Actor and Critic, the first step is to collect training data through the interaction between the Actor and the environment.

Let s_t be the state at time t, the action a_t selected by the Actor at time t can be expressed in terms of its parameter θ^μ as

$$a_t = \mu(s_t|\theta^\mu)$$

Step 2. Add random noise

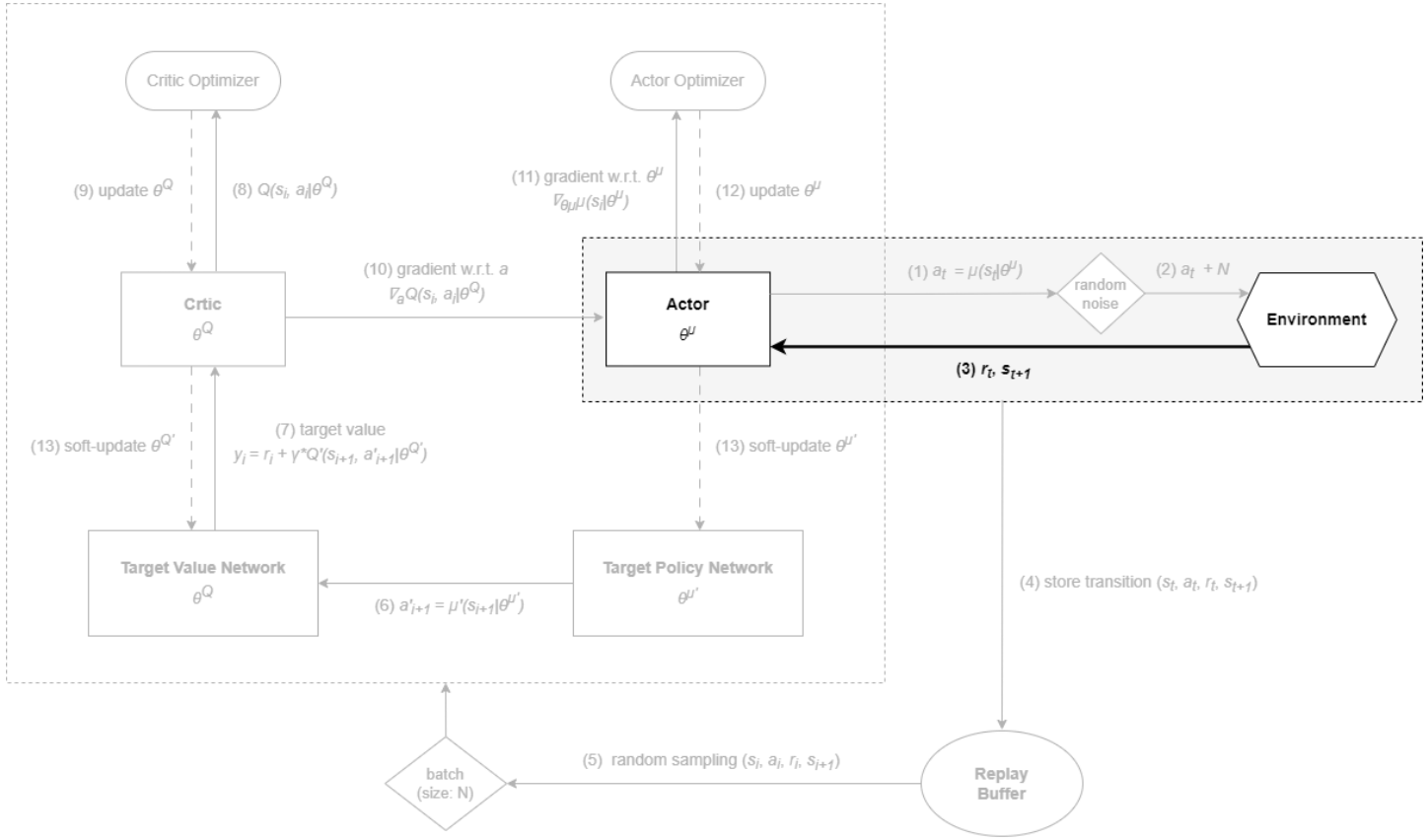


However, this action won't be directly applied to the environment. Instead, a random noise N' will be added to it, i.e.,

$$a_t = a_t + N'$$

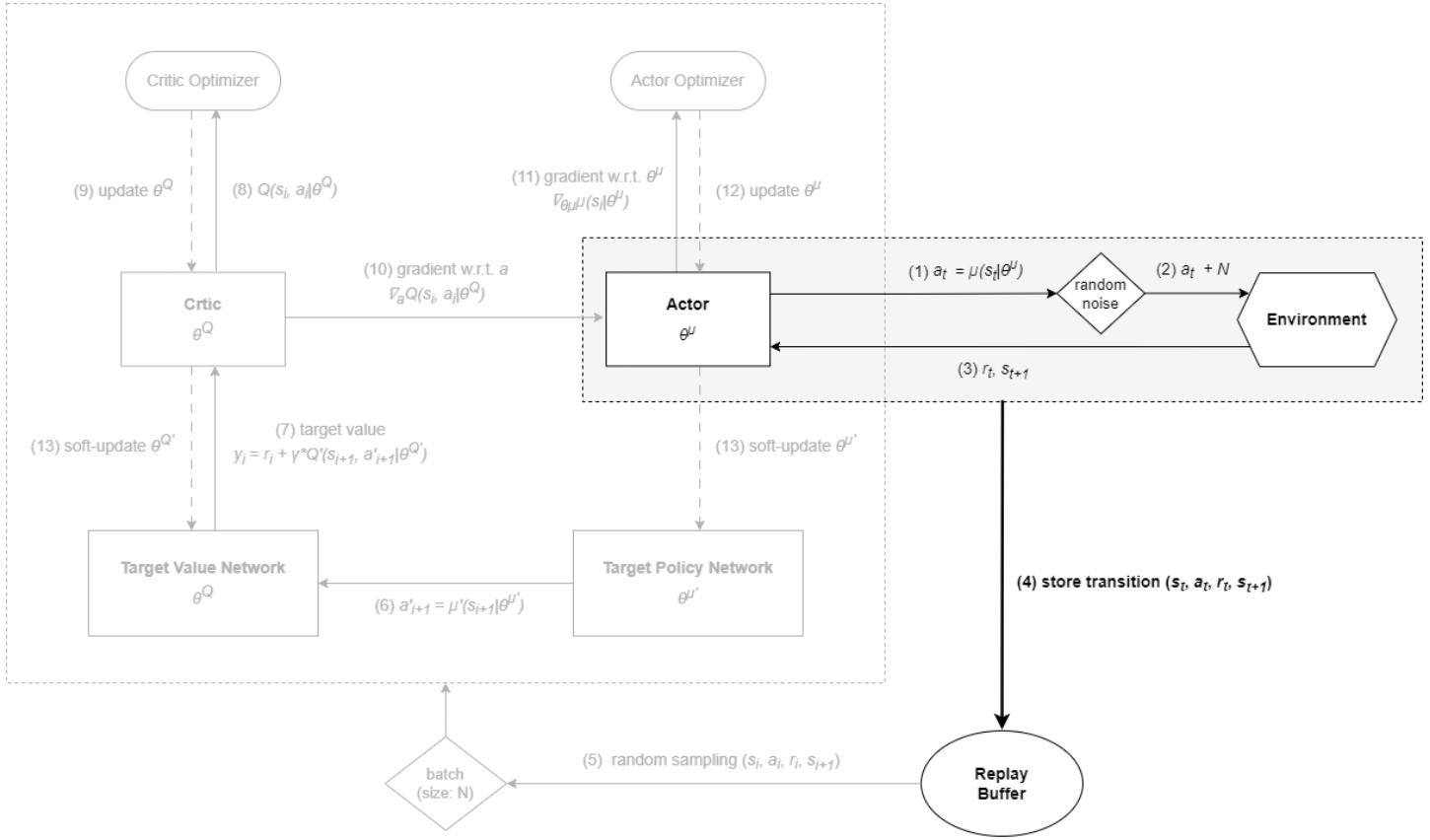
This process introduces a variation to the action chosen by the Actor even when the input state is the same, forcing the Actor to explore the action space to find the optimal policy (like ϵ- greedy used in most RL algorithms.)

Step 3. Obtain the reward and next state



After the environment receives the action selected by the Actor, it will return a corresponding reward r_t and a new state s_{t+1}.

Step 4. Store transitions

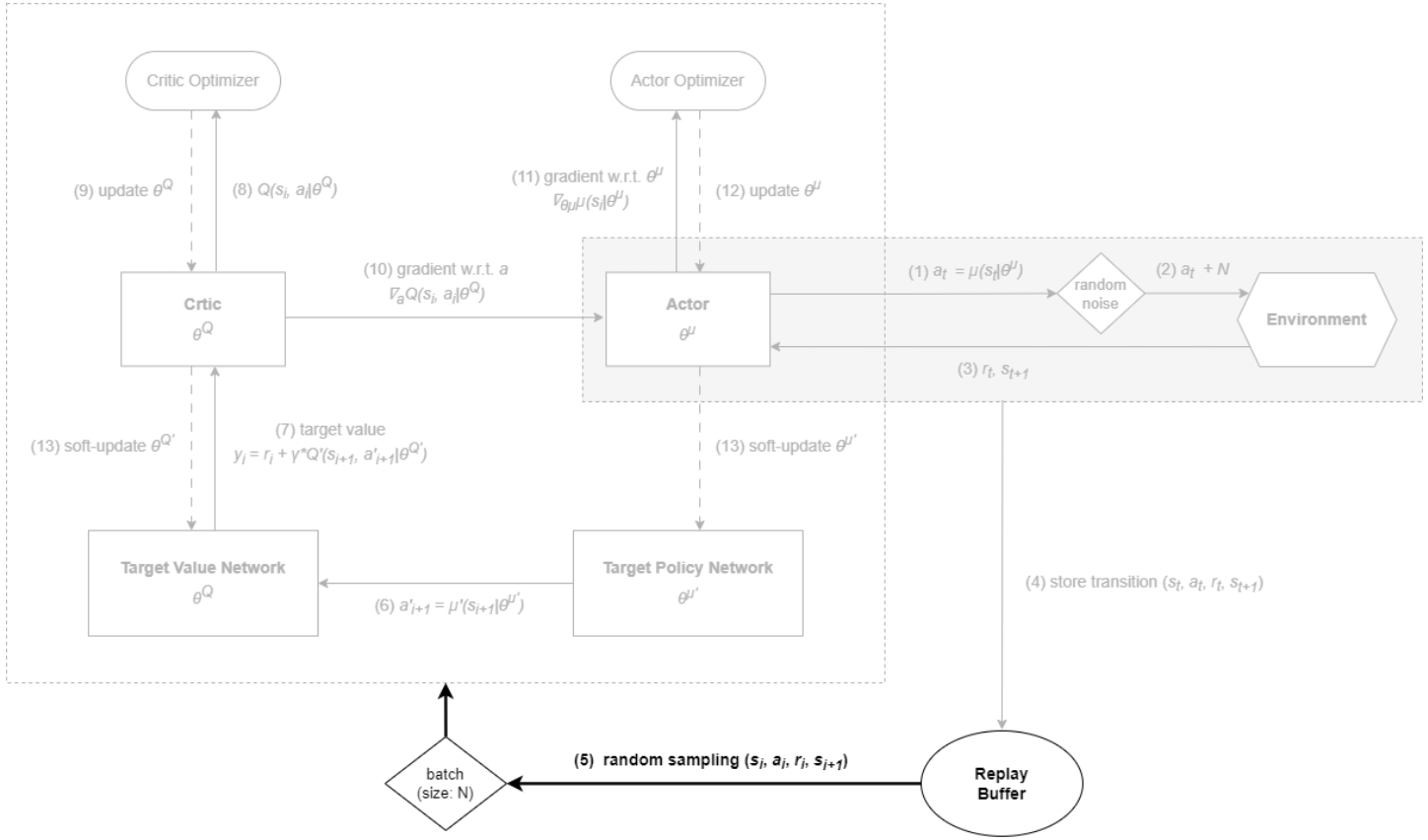


The process that the Actor selects an action and the environment returns a reward and new state based on its choice is called transition, which is denoted by

$$(s_i, a_i, r_i, s_{i+1})$$

For each time step, the transition is stored into a replay buffer as a collection of training data.

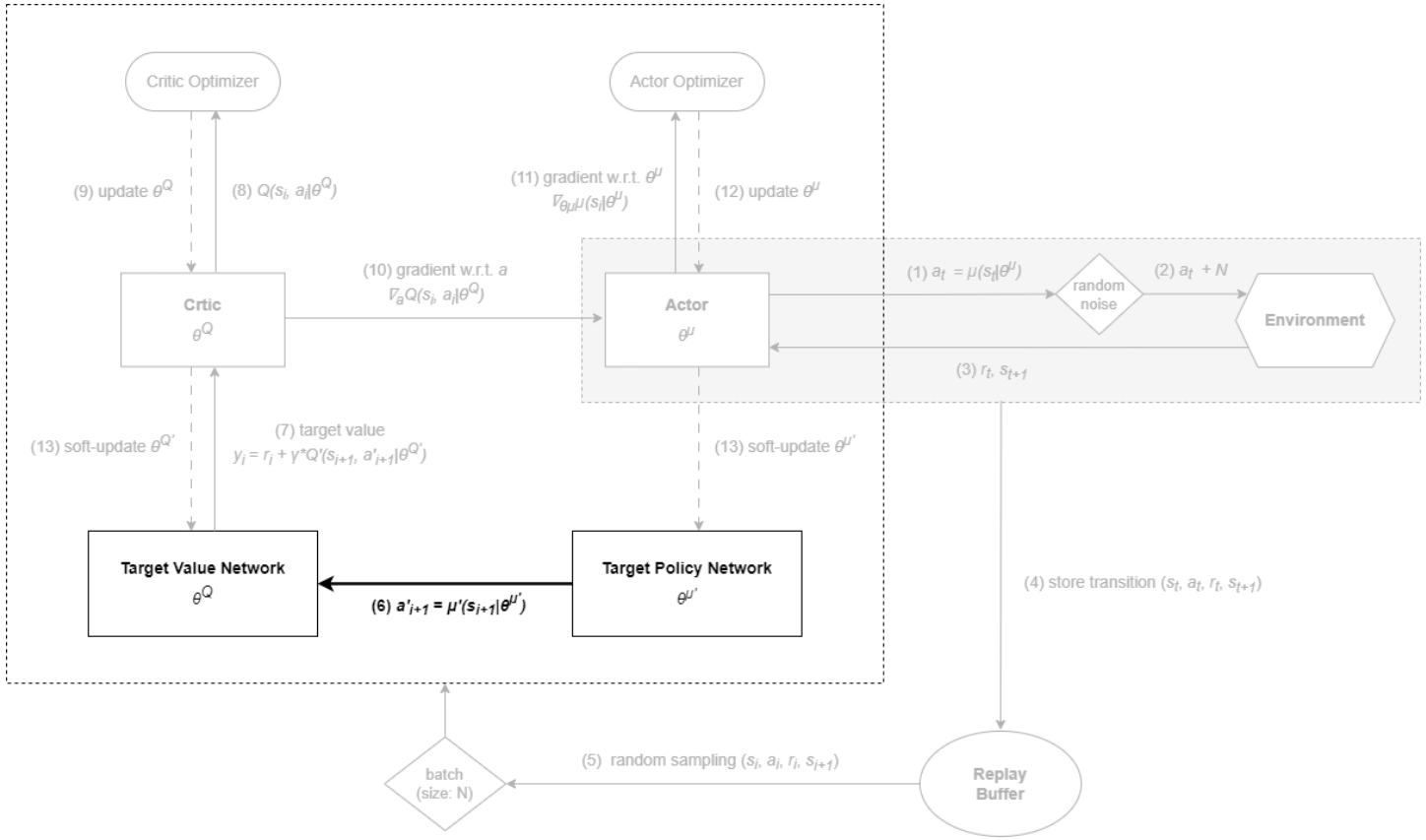
Step 5. Create sample batch



However, in order to ensure the independencies of data (which is a desired property that can make the update of the networks simpler), transitions obtained in a sequential decision process cannot be directly used as the action chosen at each time step always depends on the one at the previous step.

As a result, a sample batch of size N is created by random sampling from the replay buffer instead.

Step 6. Predict next action a'_{i+1}



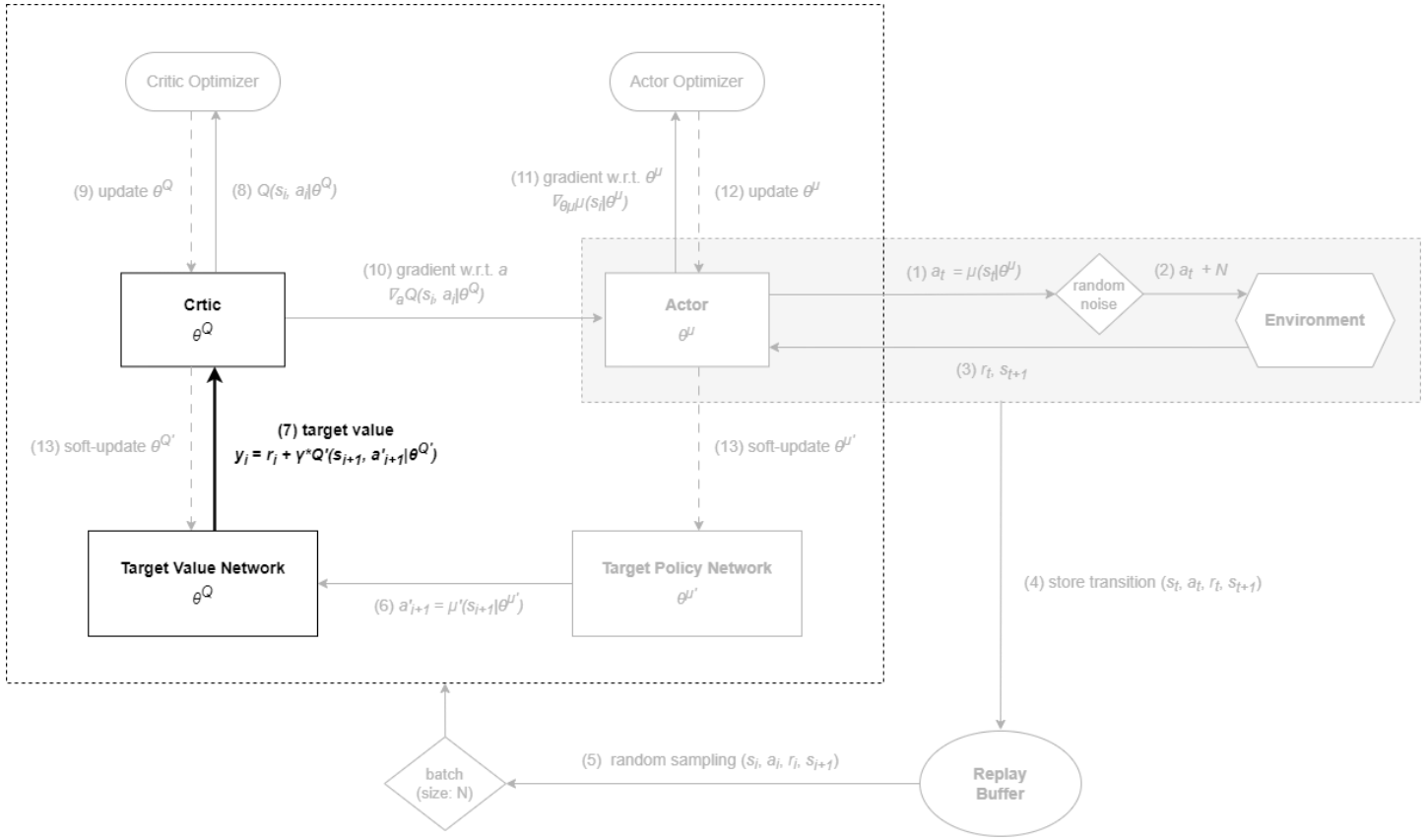
Given a batch of sample, the next step is to use it to update the four networks.

The first network to update is the Critic. To update it, target Q-value of each state-action pair (s_i, a_i) in the sample batch is required to calculate the loss. In DDPG, it is estimated by the target value and policy network.

First, the target policy network will predict the action at time $i+1$ for each sample i in the batch, i.e.,

$$a'_{i+1} = \mu'(s_{i+1}|\theta^{\mu'})$$

Step 7. Estimate target Q-values y_i

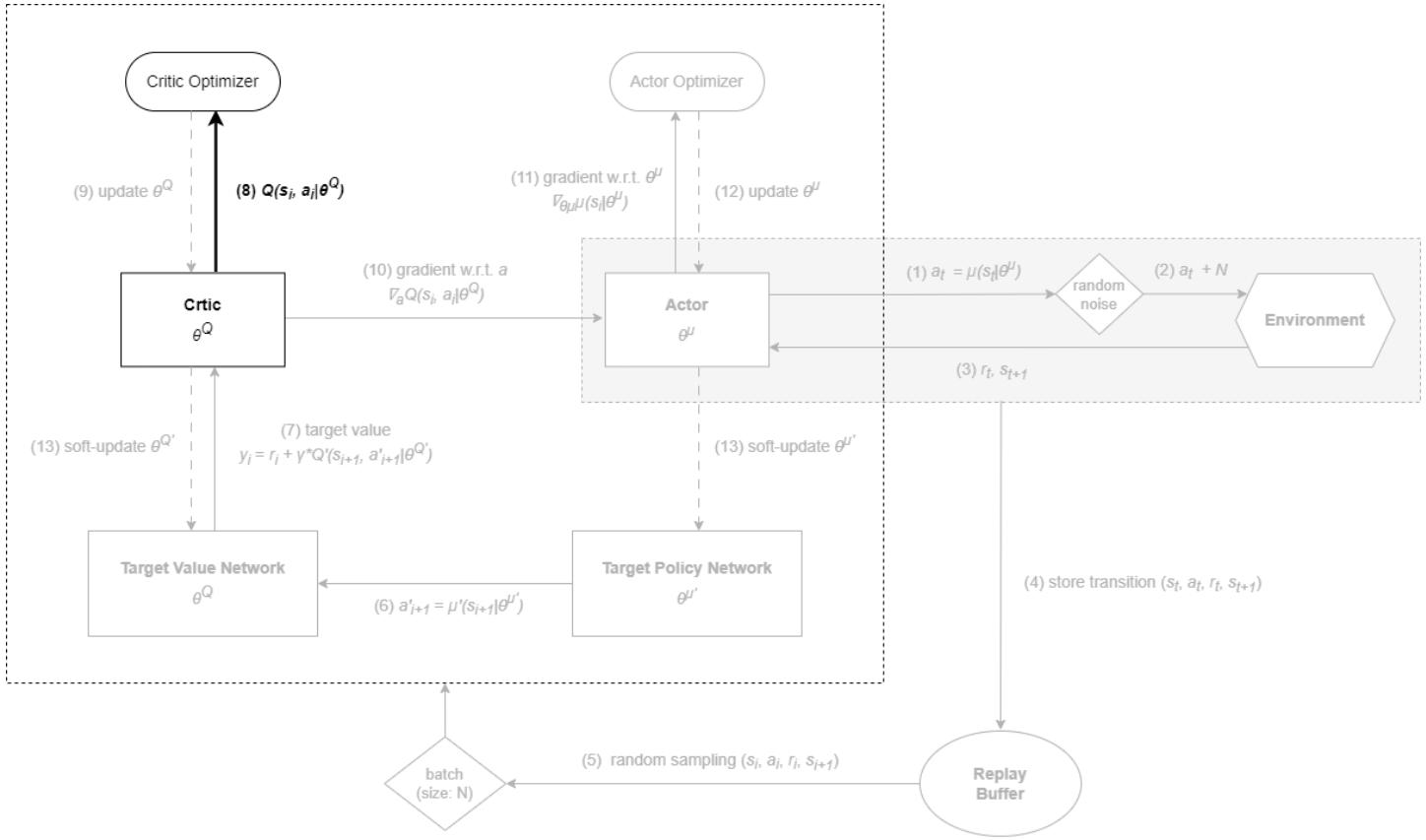


The action a'_{i+1} predicted by the target policy network will then be passed to the target value network to estimate the target value y_i based on the Bellman equation:

$$y_i = r_i + \gamma Q'(s_{i+1}, a'_{i+1}|\theta^{Q'})$$

where γ is the discounted factor and $Q'(s_{i+1}, a'_{i+1}|\theta^{Q'})$ is the Q-value of the state-action pair (s_{i+1}, a'_{i+1}) output by the target value network.

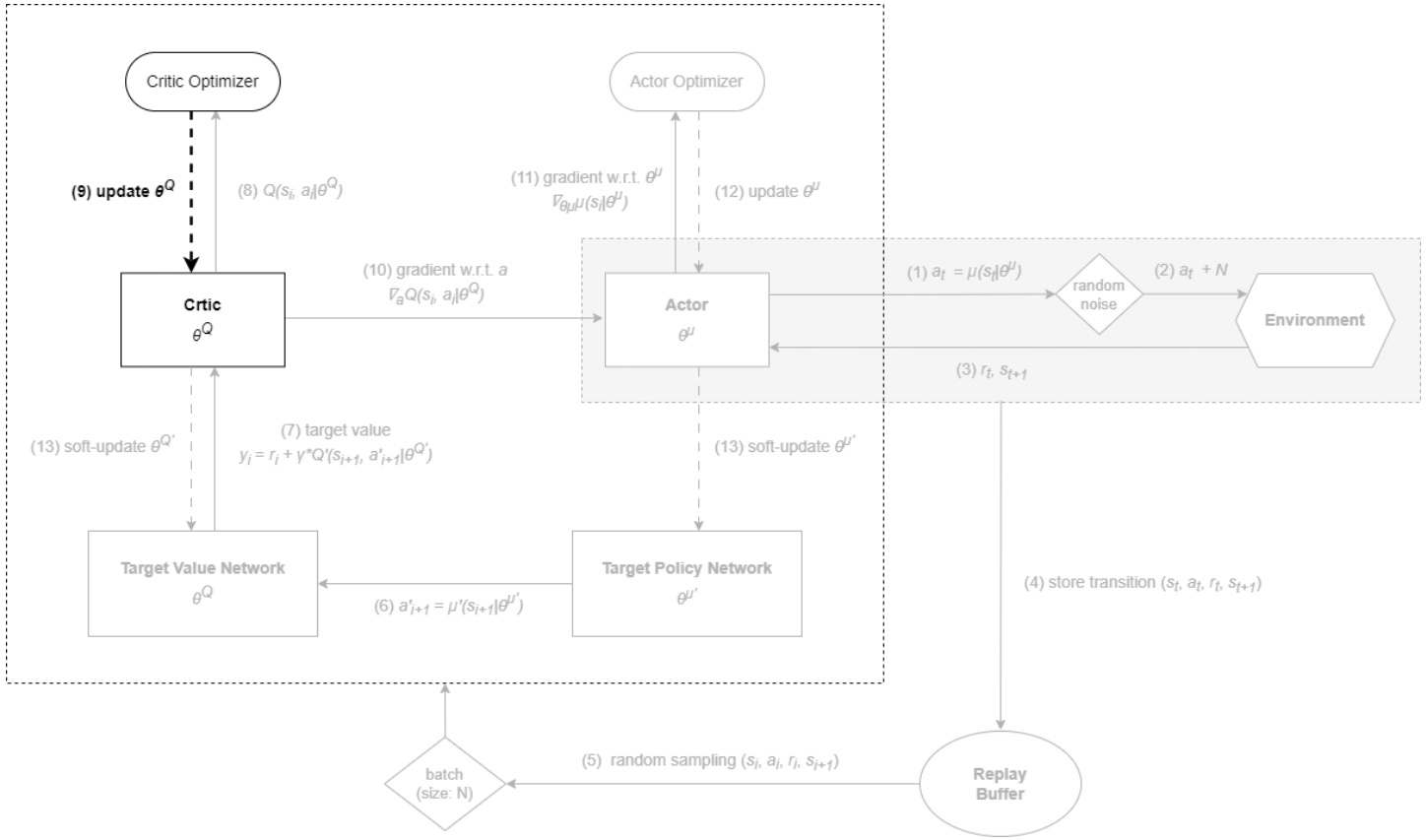
Step 8. Calculate predicted Q-values



On the other hand, the Q-values of each state-action pair (s_i, a_i) predicted by the Critic is

$$Q(s_i, a_i|\theta^Q)$$

Step 9. Update Critic (θ^Q)

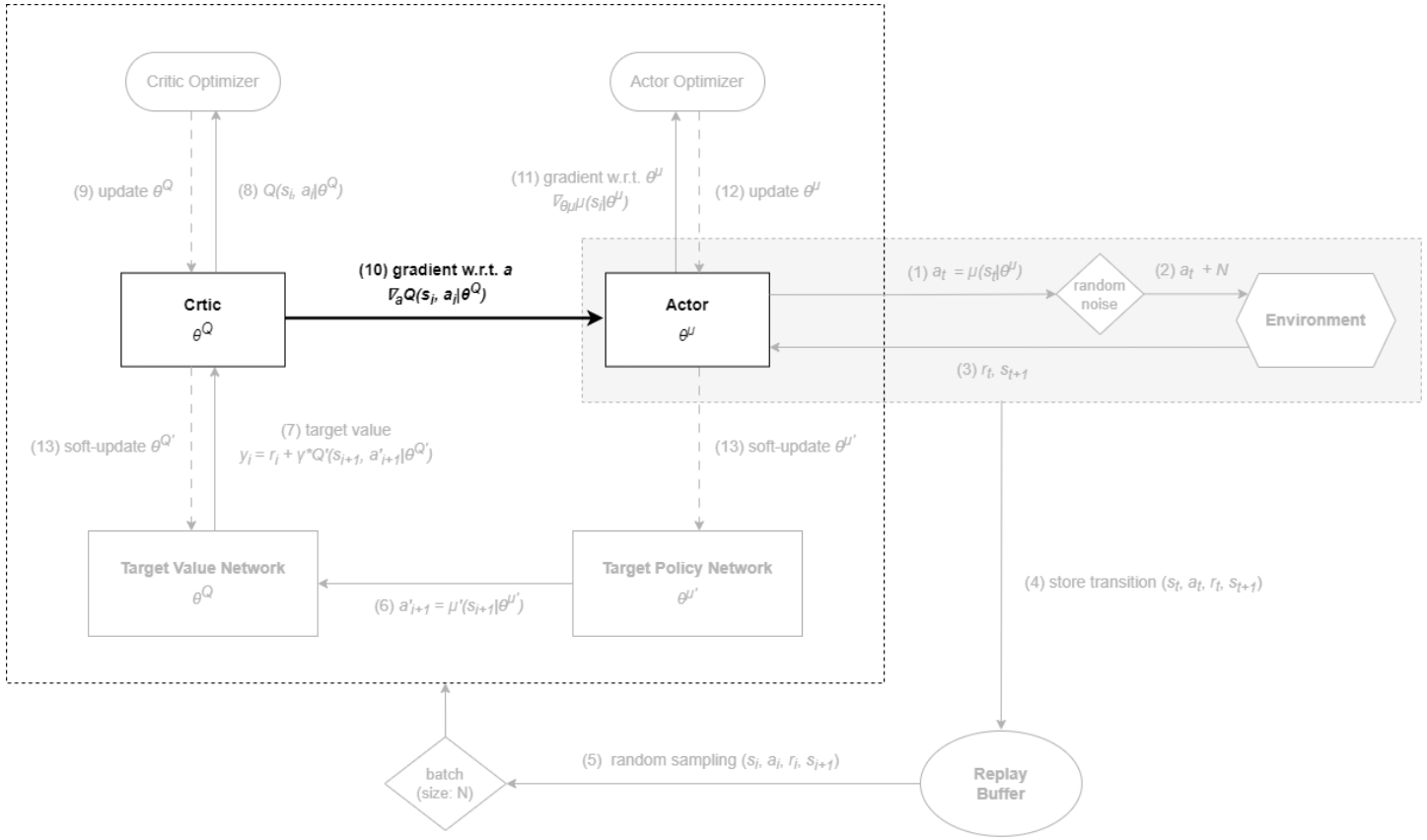


After both the predicted values $Q(s_i, a_i|\theta^Q)$ and target values y_i are known, the parameter of the Critic θ^Q can then be update by the loss defined as

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

which is the mean squared error of the target and predicted Q-values.

Step 10. Calculate gradient of Q-function w.r.t action



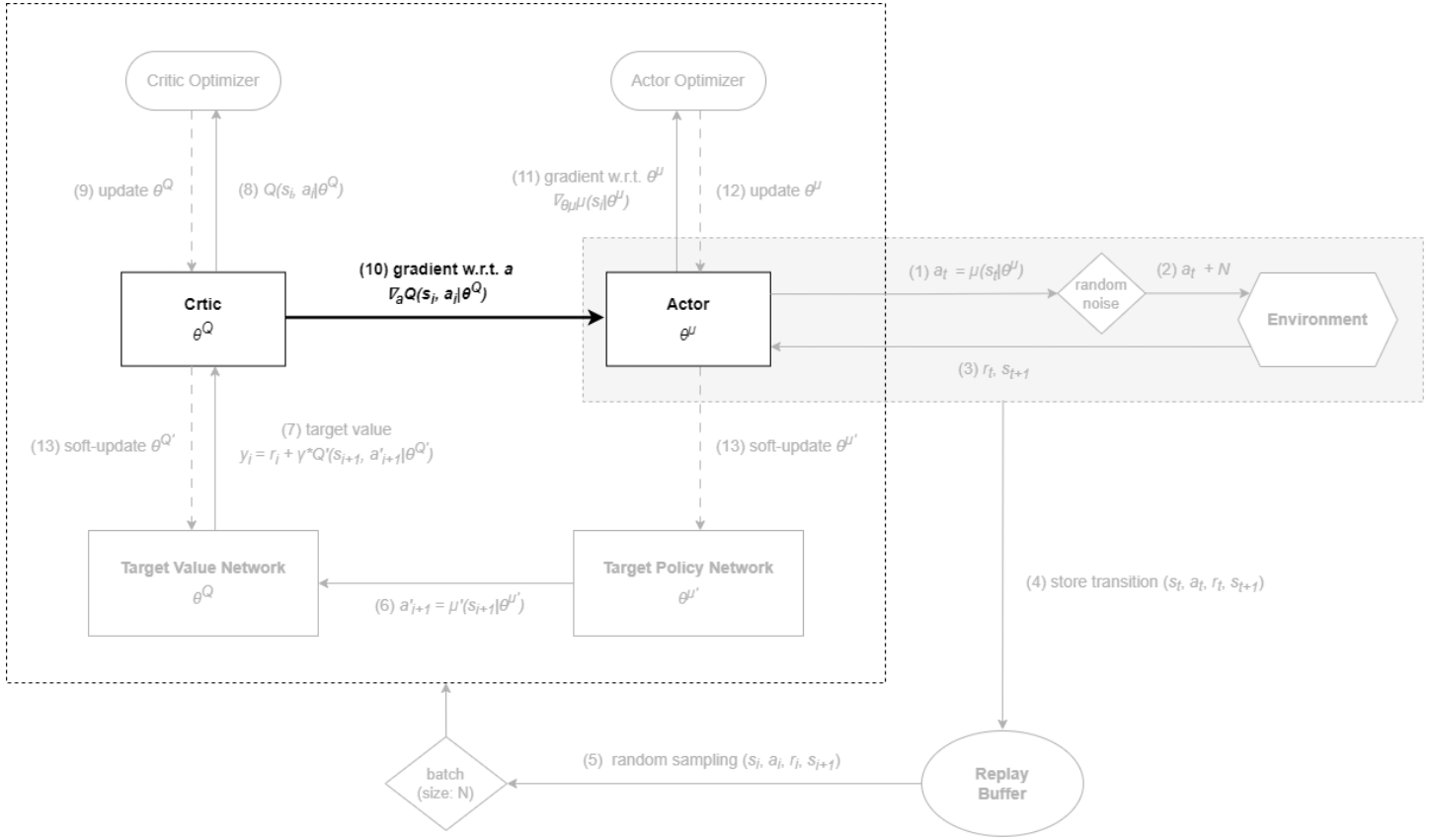
The loss of the Actor, on the other hand, is to maximize the expected return $J(\theta)$ defined as the expectation of the Q-value, i.e.,

$$J(\theta) = E[Q(s,a)]$$
$$= \frac{1}{N}\sum_i Q(s_i, a_i|\theta^Q)$$

To maximize this term, the derivative with respect to θ^μ is necessary, which can be expressed as

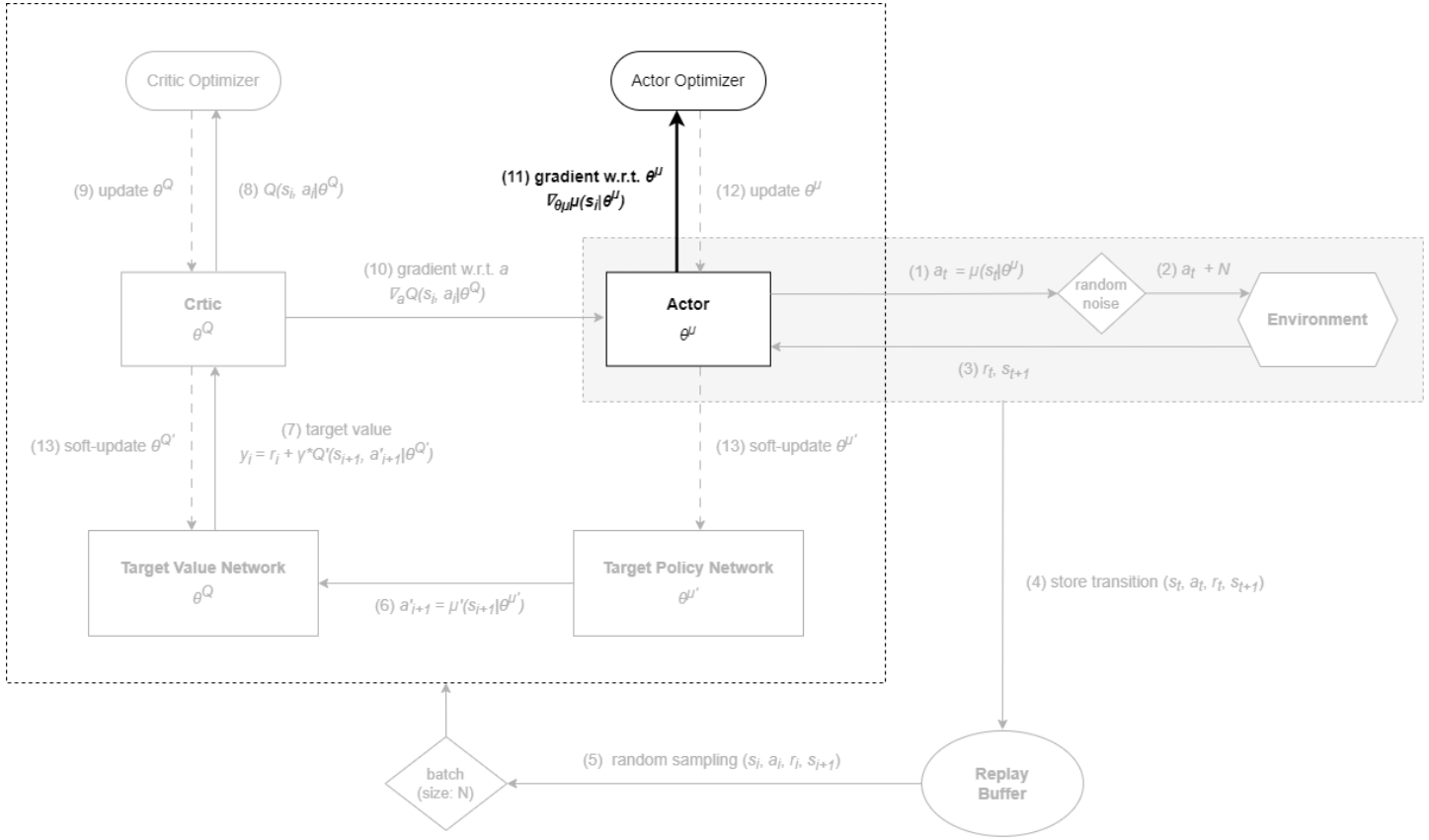
$$\nabla_{\theta^\mu} J(\theta)$$
$$= \frac{1}{N}\sum_i [\nabla_a Q\ (s_i, a_i|\theta^Q)\nabla_{\theta^\mu}\mu(s|\theta^\mu)]$$

Step 10. Calculate gradient of Q-function w.r.t action



The first term of $\nabla_{\theta^\mu} J(\theta)$, i.e., $\nabla_a Q\,(s_i, a_i|\theta^Q)$, is the gradient of the Q-function with respect to action. It can be calculated in the Critic and passed to the Actor.

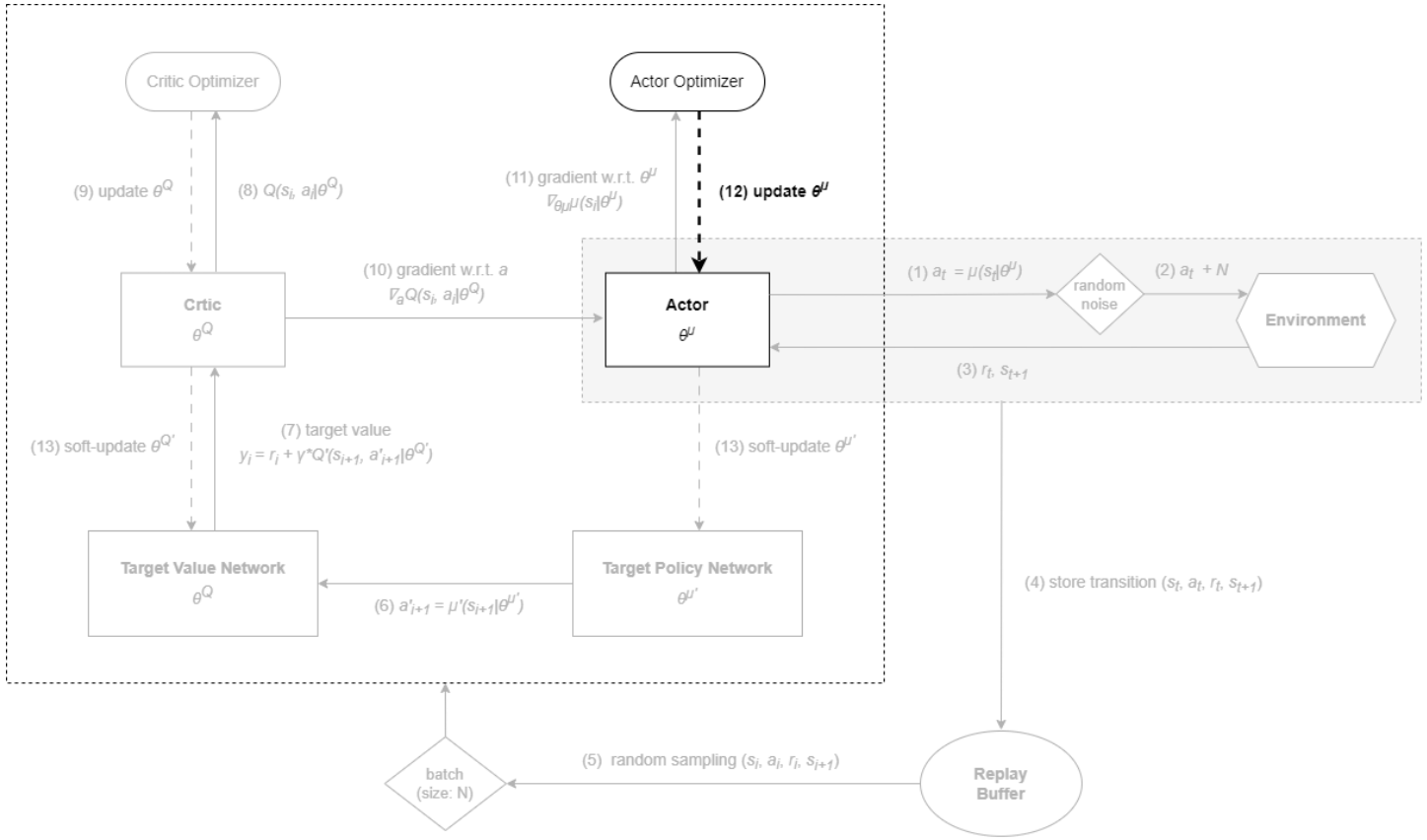
Step 11. Calculate gradient of policy function w.r.t. θ^{μ}



The second term of $\nabla_{\theta^\mu} J(\theta)$, i.e., $\nabla_{\theta^\mu}\mu(s|\theta^\mu)$, is the gradient of the policy function with respect to θ^μ and can be calculated by the Actor itself.

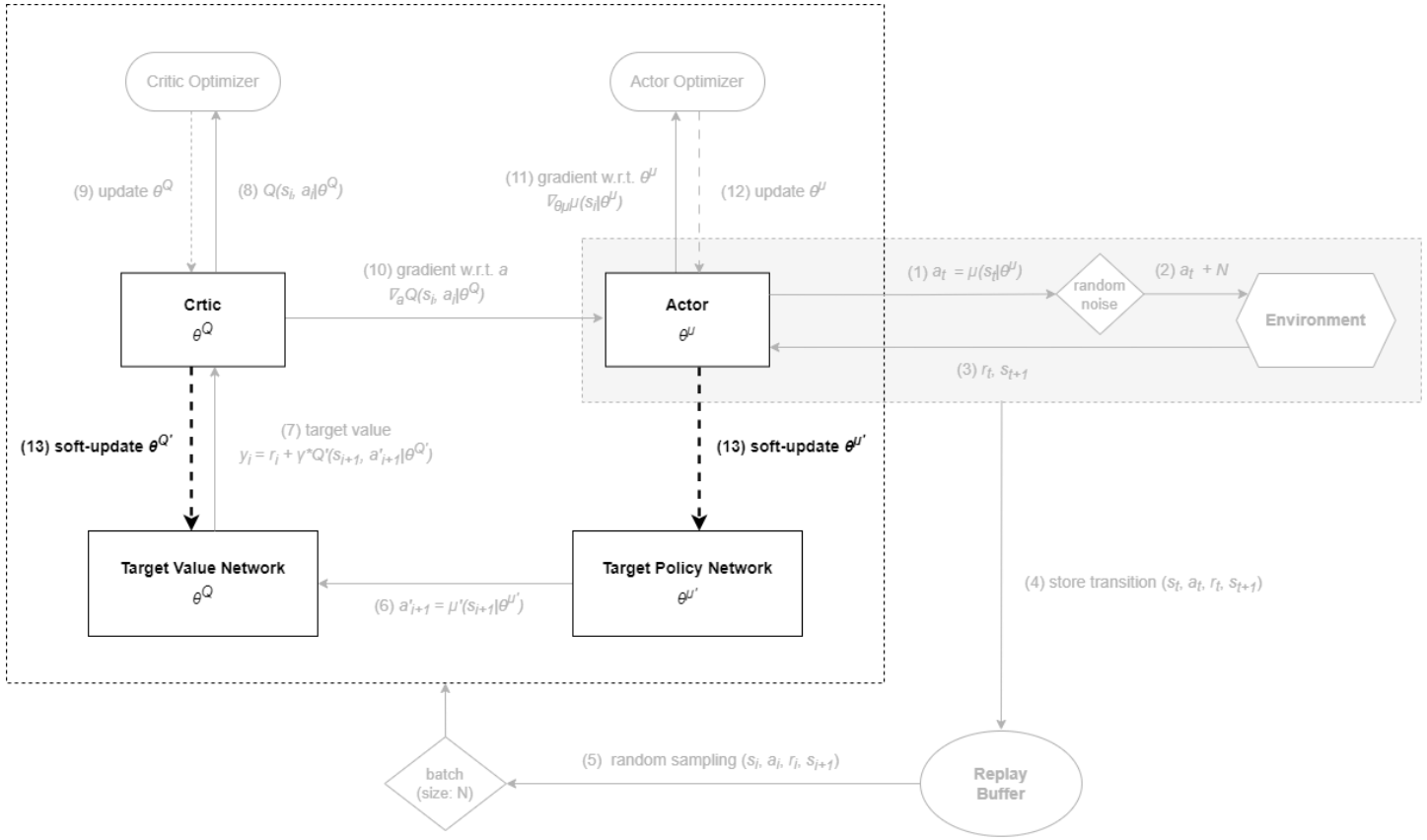
* Note that the term $\nabla_{\theta^\mu}\mu(s|\theta^\mu)$, shows in $\nabla_{\theta^\mu} J(\theta)$ because a_i is function of θ^μ, i.e., $a_i = \mu(s_i|\theta^\mu)$. As a result, based on the chain rule, we should take $\mu(s_i|\theta^\mu)$ out from the Q-function and take its derivative as well.

Step 12. Update Actor (θ^{μ})



By combining the two gradient terms, the parameter of Actor θ^{μ} can then be updated based on $\nabla_{\theta^{\mu}} J(\theta)$ by gradient ascent method.

Step 13. Update target value ($\theta^{\mu'}$) & policy network (θ^{Q})



Finally, the parameters of the target value and policy network are updated by

$$\theta^{Q\prime} \leftarrow \theta^{Q\prime} + \tau(\theta^{Q} - \theta^{Q\prime})$$
$$\theta^{\mu\prime} \leftarrow \theta^{\mu\prime} + \tau(\theta^{\mu} - \theta^{\mu\prime})$$

where τ is the learning rate and typically has a value much smaller than 1.