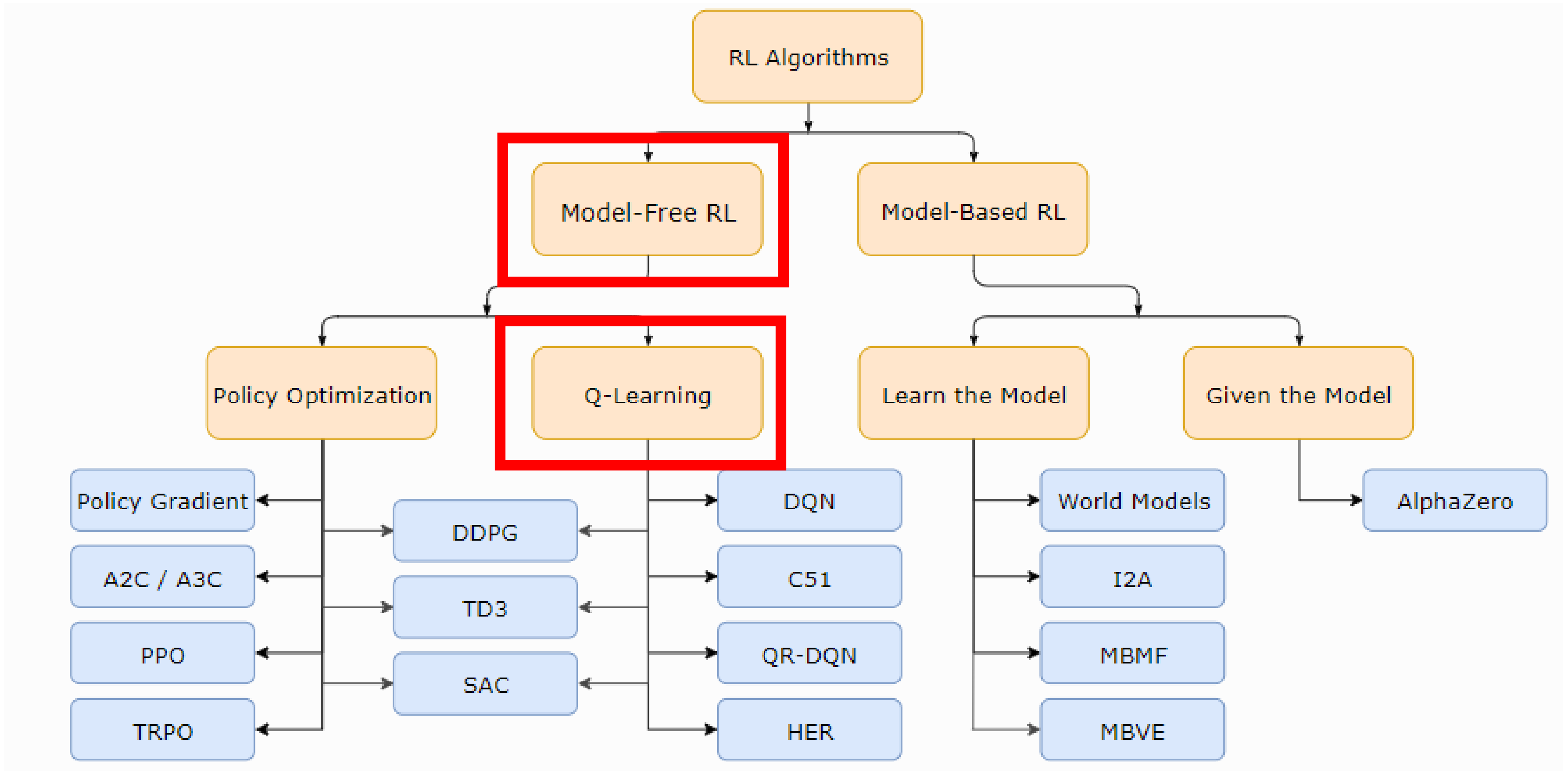
Q-learning based RL

Model free RL algorithms include Policy Optimization and Q-Learning.

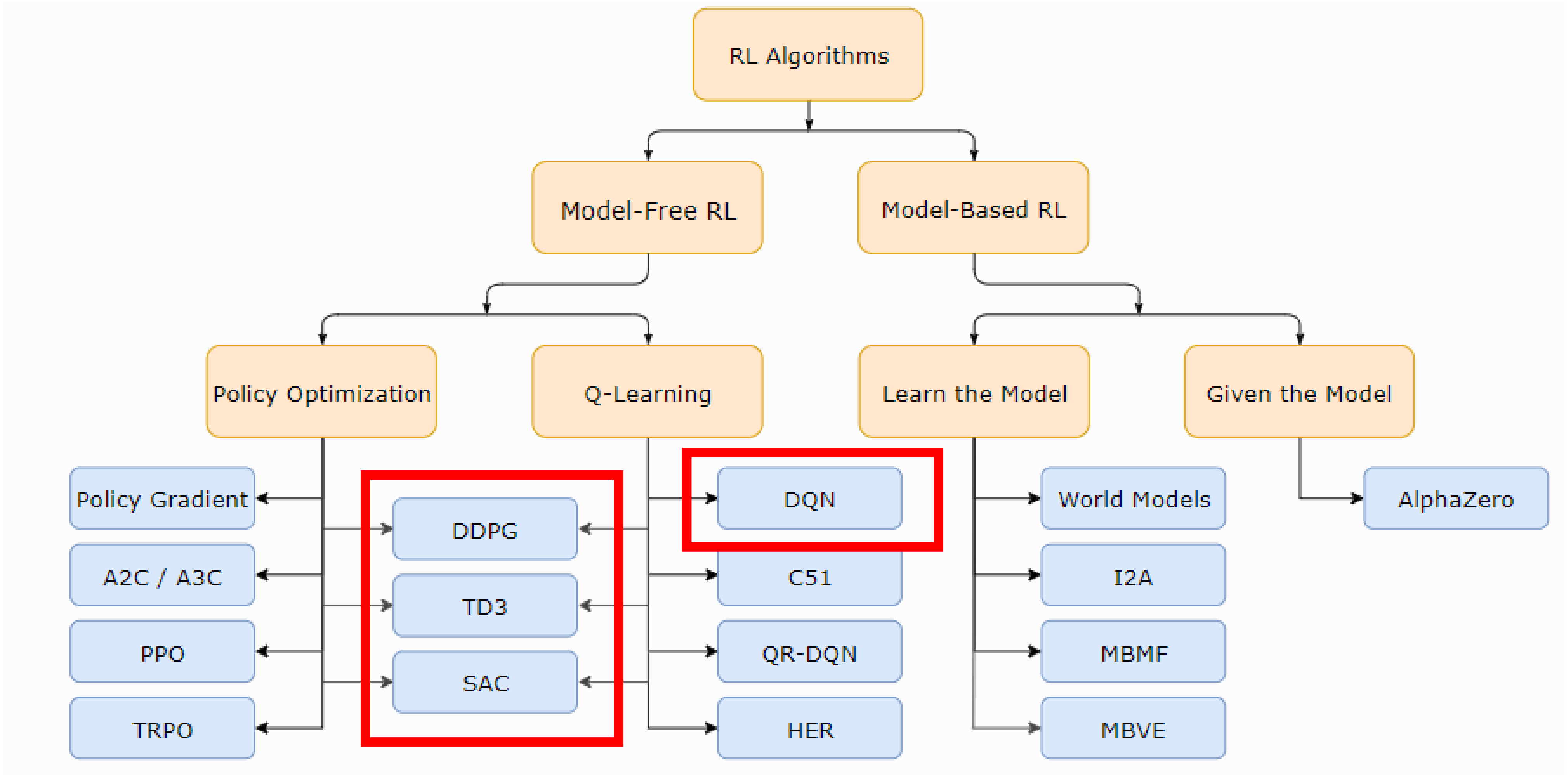


A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

[Welcome to Spinning Up in Deep RL! — Spinning Up documentation (openai.com)]

Q-learning based RL

Popular Q-learning based RL algorithms include DQN, DDPG, TD3 and SAC.



A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

Recap – Key concepts

Policies	$a_t = \mu_\theta(s_t)$ $a_t \sim \pi_\theta(s_t)$
Trajectories	$\tau = (s_0, a_0, s_1, a_1, \ldots)$
Reward	$r_t = R(s_t, a_t, s_{t+1})$
Return	$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

$$V^\pi(s) = E_{\tau\sim\pi}(R(\tau) \mid s_0 = s)$$

$$Q^\pi(s,a) = E_{\tau\sim\pi}(R(\tau) \mid s_0 = s, a_0 = a)$$

Bellman Equation

$$V^\pi(s) = E_{\substack{\tau\sim\pi \\ s'\sim P}}[r(s,a) + \gamma V^\pi(s')]$$

$$Q^\pi(s,a) = E_{s'\sim P}\left[r(s,a) + \gamma E_{a'\sim\pi}[Q^\pi(s',a')]\right]$$

$$V^*(s) = \max_a E_{s'\sim P}[r(s,a) + \gamma V^*(s')]$$

$$Q^*(s,a) = E_{s'\sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

Q-learning algorithm to find $Q^*(s,a)$

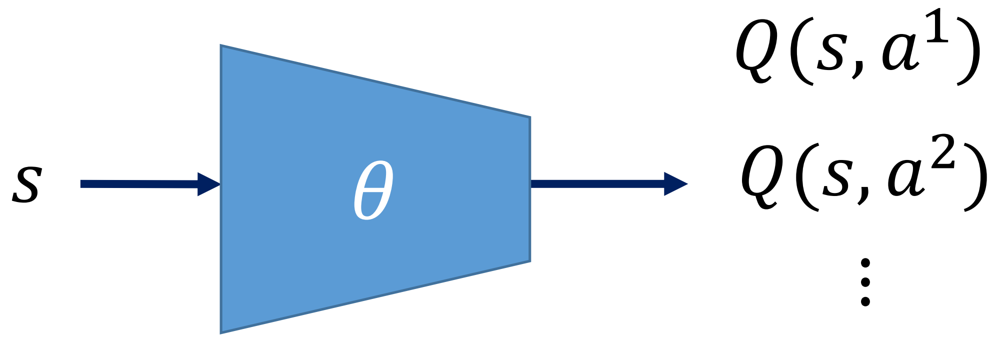
1. Initialize Table Q(S,A) with random values.
2. Take a action (A) with epsilon-greedy policy and move to next state S'
3. Update the Q value of a previous state by following the update equation:

$$Q(s,a) = Q(s,a) + \alpha(R_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

https://markus-x-buchholz.medium.com/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862

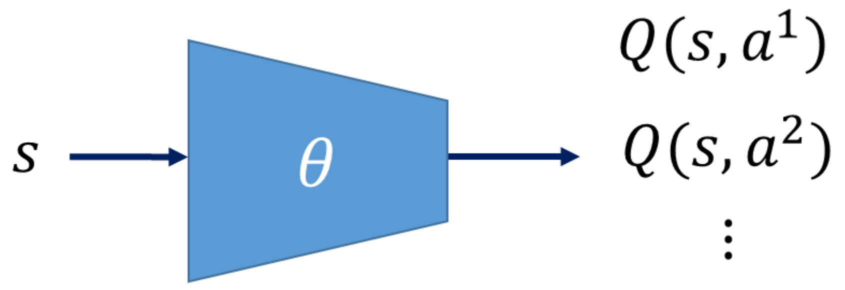
Deep Q-Network (DQN)

Q-table is replaced by Deep Neural Network (Q-Network) which maps (non-linear approximation) environment states to Agent actions.



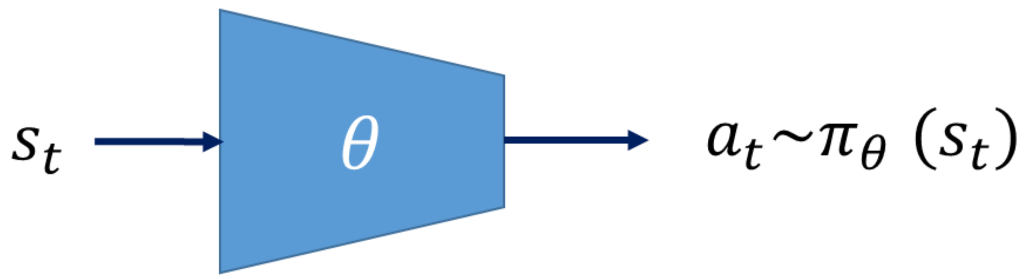
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

Q-learning vs Policy optimization



$$Q(s,a) = Q(s,a) + \alpha(R_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- Learn expected value of long-term reward of discrete actions $Q(s,a)$
- Use Bellman eq. to recursively learn $Q^*(s,a)$ from $Q^*(s',a')$

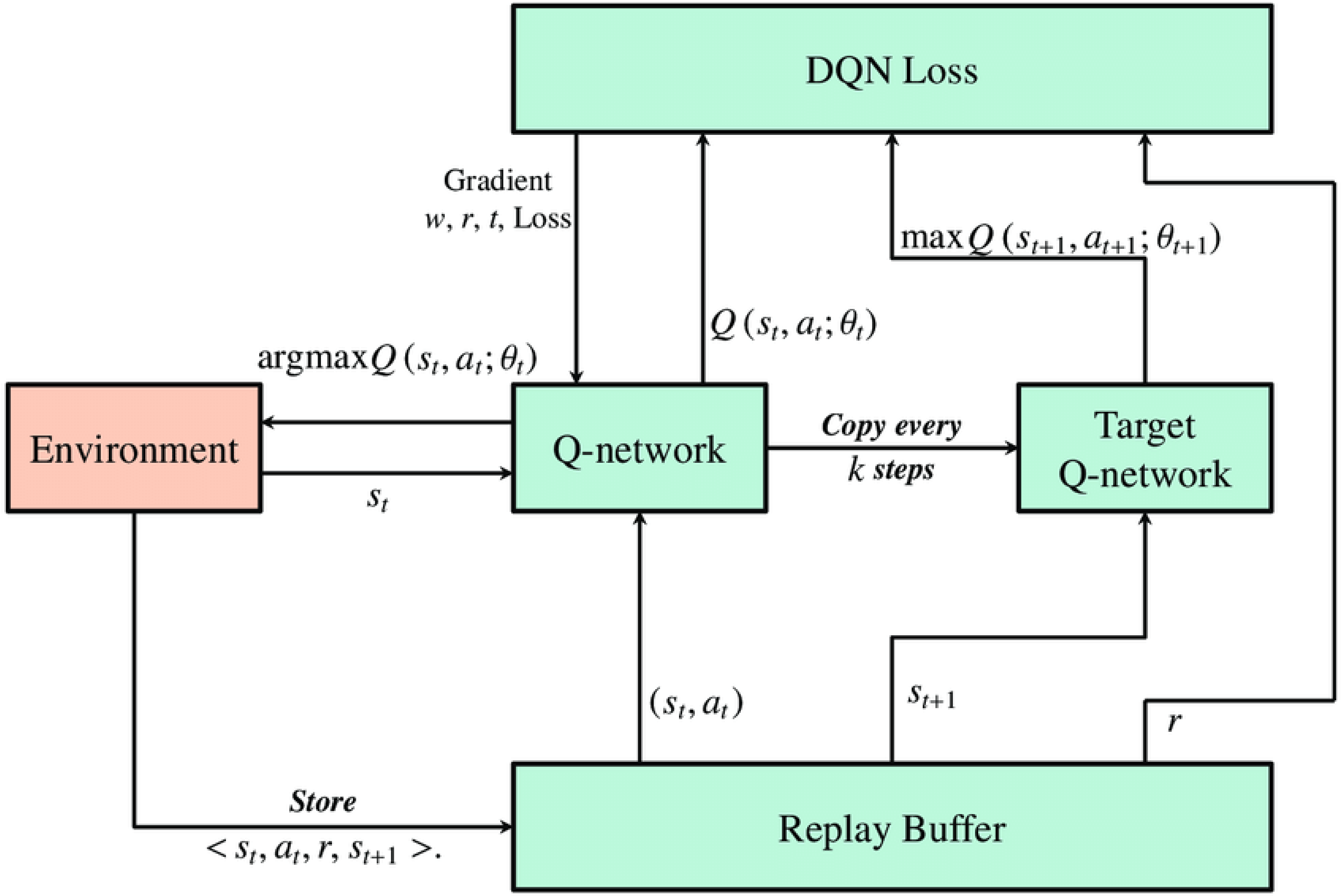


$$\max\ J(\pi) = E_{\tau\sim\pi}\ (R(\tau))$$

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$

- Learn policy of continuous action $p_\Theta\ (a|s)$
- Use expected value of long-term reward to adjust probability $p_\Theta\ (a|s)$

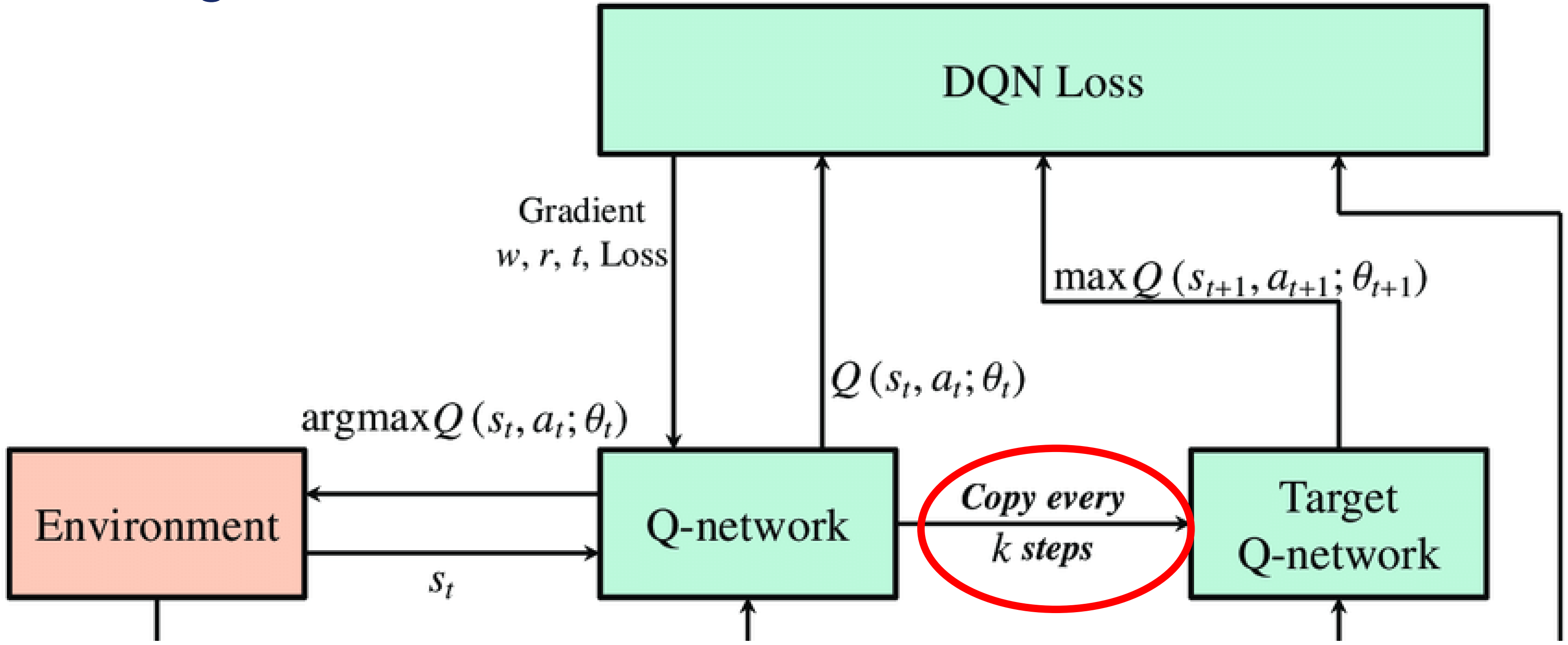
DQN architecture



https://www.researchgate.net/figure/A-Concept-of-DQN-Algorithm7_fig4_348444080

Stabilize DQN training

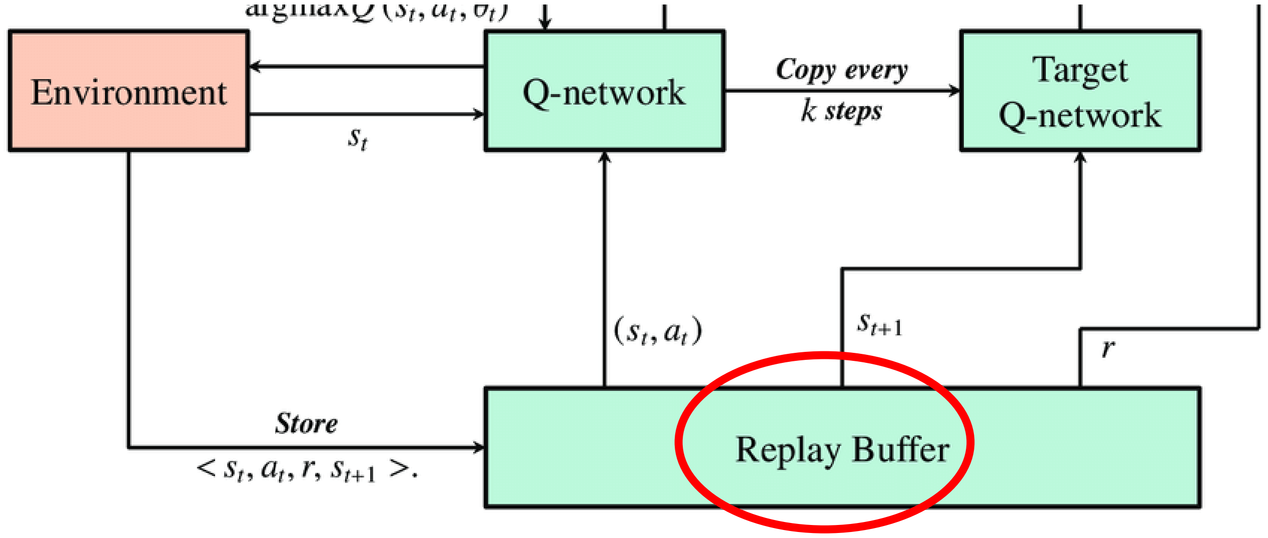
The target network is frozen for several time steps and then the target network weights are updated by copying the weights from the actual Q network. Freezing the target Q-network for a while and then updating its weights with the actual Q network weights stabilizes the training.



https://markus-x-buchholz.medium.com/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862

Stabilize DQN training (2)

In order to make training process more stable (we would like to avoid learning network on data which is relatively correlated, which can happen if we perform learning on consecutive updates - last transition) we apply replay buffer which memorizes experiences of the Agent behavior. Then, training is performed on random samples from the replay buffer (this reduces the correlation between the agent's experience and helps the agent to learn better from a wide range of experiences).



https://markus-x-buchholz.medium.com/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862

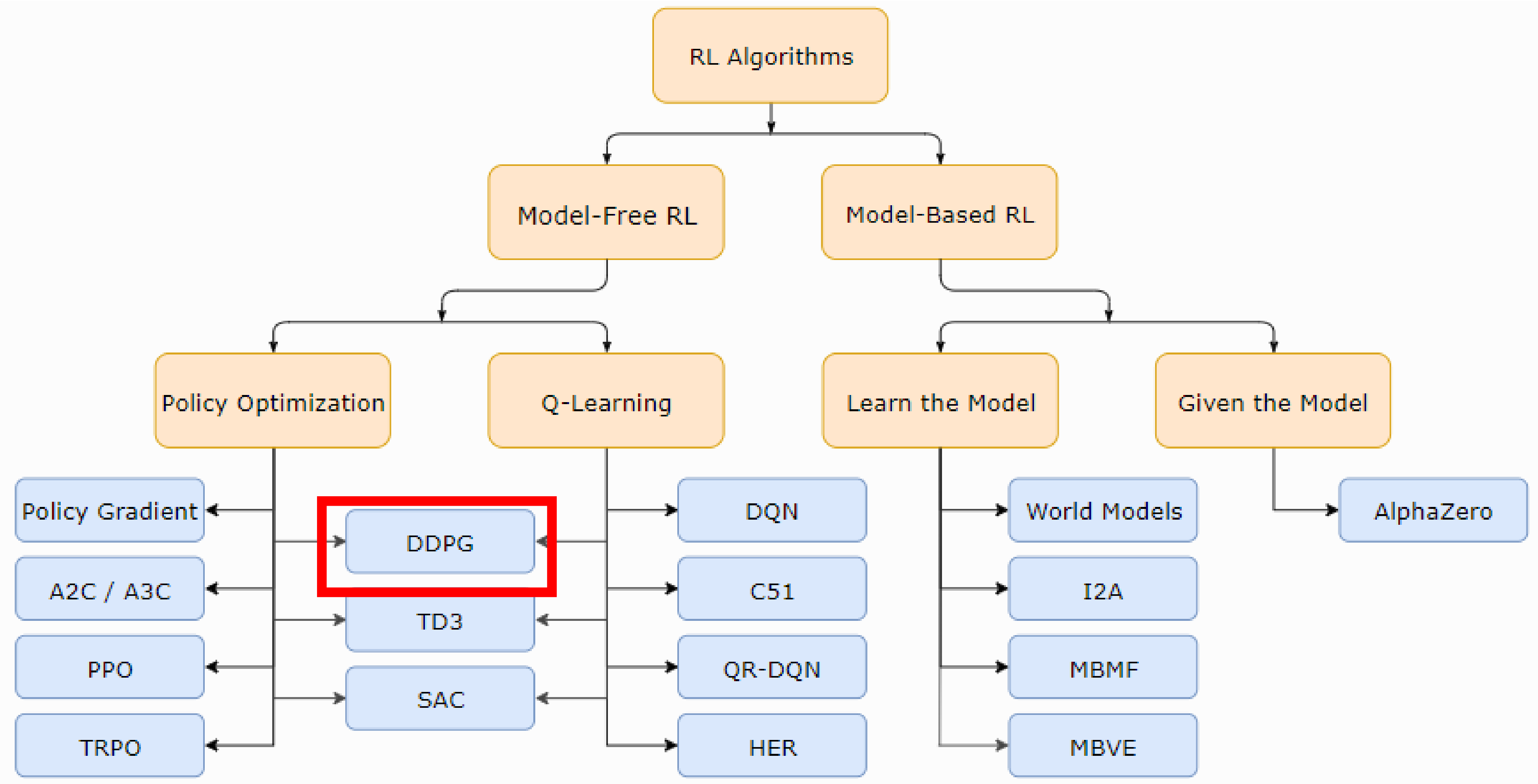
DQN algorithm

1. Initialize replay buffer.
2. Send s to DQN and select an action using the epsilon-greedy policy. Select an action that has a maximum Q value.
3. Agent performs chosen action and move to a new state S' and receive a reward R.
4. Store transition in replay buffer.
5. Sample batches of transitions from the replay buffer and calculate the loss.
6. Perform gradient descent to minimize the loss.
8. After every k steps, copy our actual network weights to the target network weights.
9. Repeat these steps for M number of episodes.

$$Loss = \left(R_s^a + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2$$

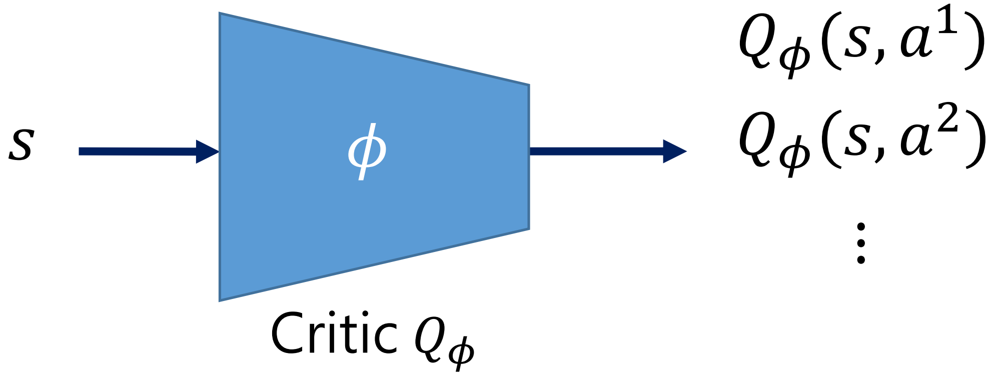
https://markus-x-buchholz.medium.com/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862

DDPG



A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

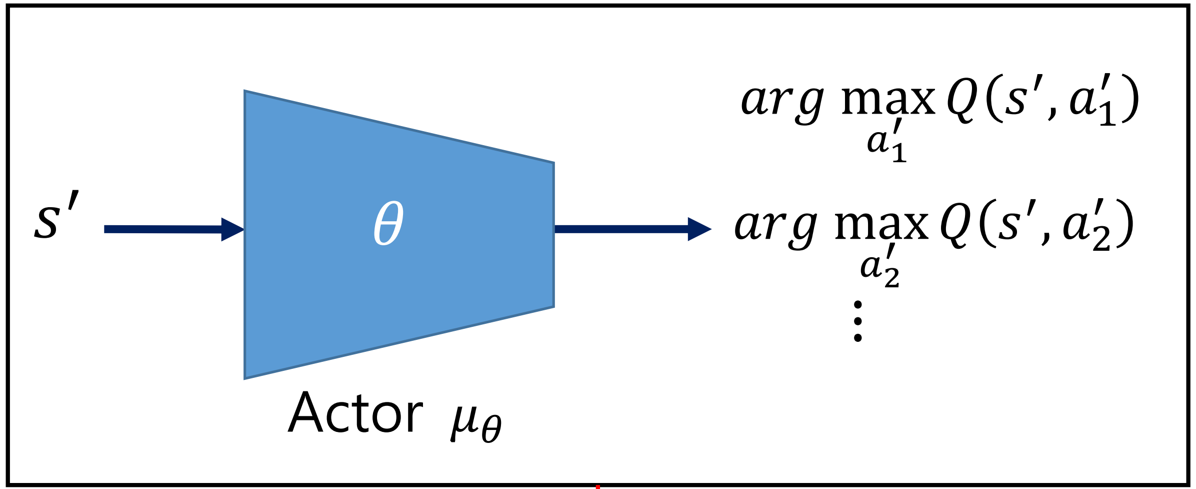
Can we use Q-learning to learn continuous actions?



$$Loss = \mathrm{E}\left[\left(r_s^a + \gamma \max_{a'} Q_\phi(s', a') - Q_\phi(s, a)\right)^2\right]$$

For continuous actions, it is impossible to calculate $Q_\phi(s', a')$ for every possible a' value

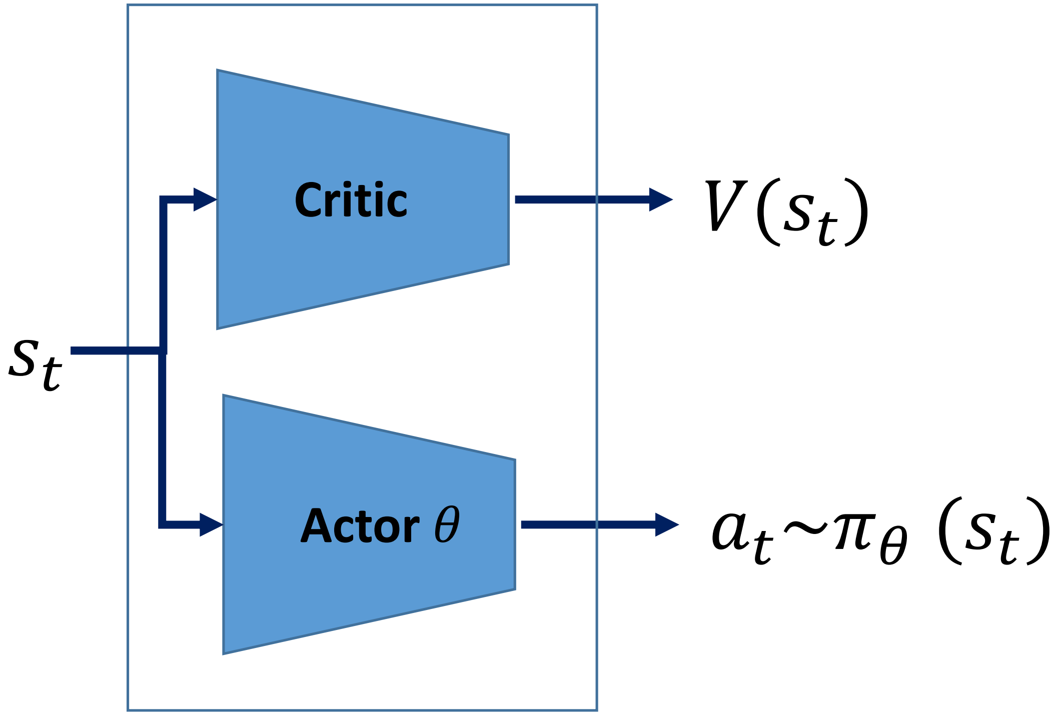
Solution: train a target policy network to learn $\max_{a'} Q(s', a')$



$$Loss = \mathrm{E}\left[\left(\left(r_s^a + \gamma \boxed{Q_\phi(s', \mu_\theta(s'))}\right) - Q_\phi(s, a)\right)^2\right]$$

Recap: Actor-critic for policy optimization

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)(r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\,)\right]$$



$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)$$

- The “Critic” estimates the value function.
- The “Actor” updates the policy distribution in the direction suggested by the Critic.

Introduction to DDPG

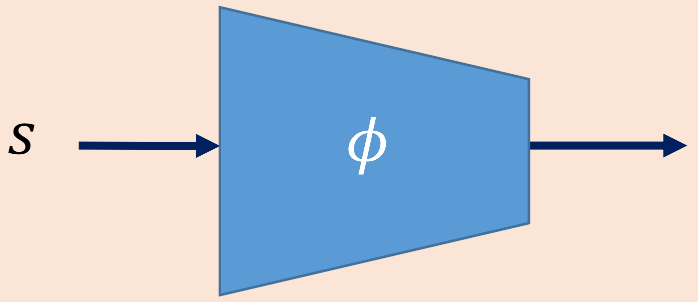
Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

If you know the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving

$$a^*(s) = arg\max_a Q^*(s,a)$$

https://spinningup.openai.com/en/latest/algorithms/ddpg.html#deep-deterministic-policy-gradient

The Q-learning side of DDPG



Critic Q_ϕ

$Q_\phi(s, a^1)$

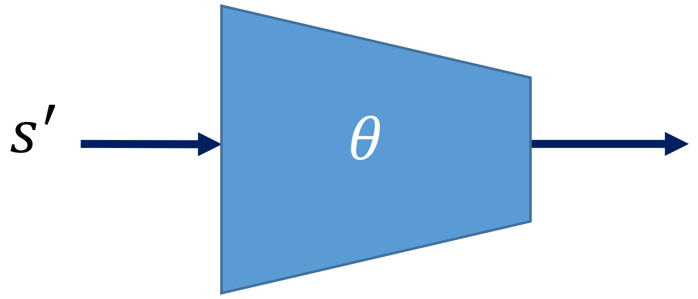
$Q_\phi(s, a^2)$

$\vdots$

$$Loss = \mathrm{E}\left[\left(r_s^a + \gamma \max_{a'} Q_\phi(s', a') - Q_\phi(s, a)\right)^2\right]$$

For continuous actions, it is impossible to calculate $Q_\phi(s', a')$ for every possible a' value

Solution: train a target policy network to learn $\max_{a'} Q(s', a')$



Actor μ_θ

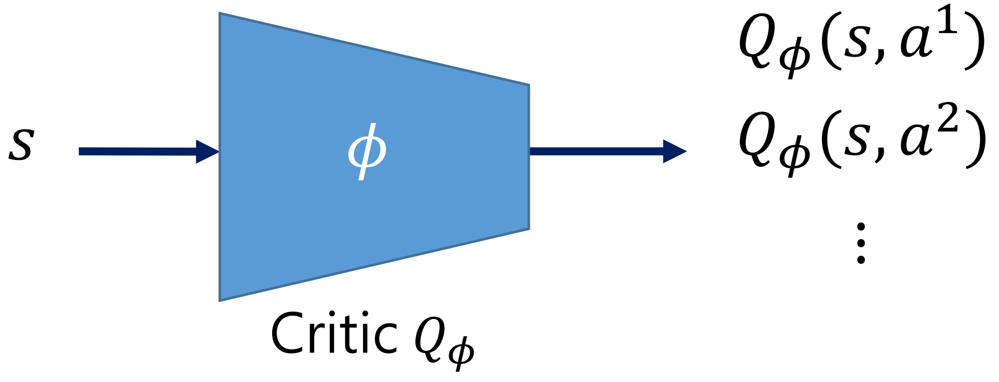
$arg \max_{a_1'} Q(s', a_1')$

$arg \max_{a_2'} Q(s', a_2')$

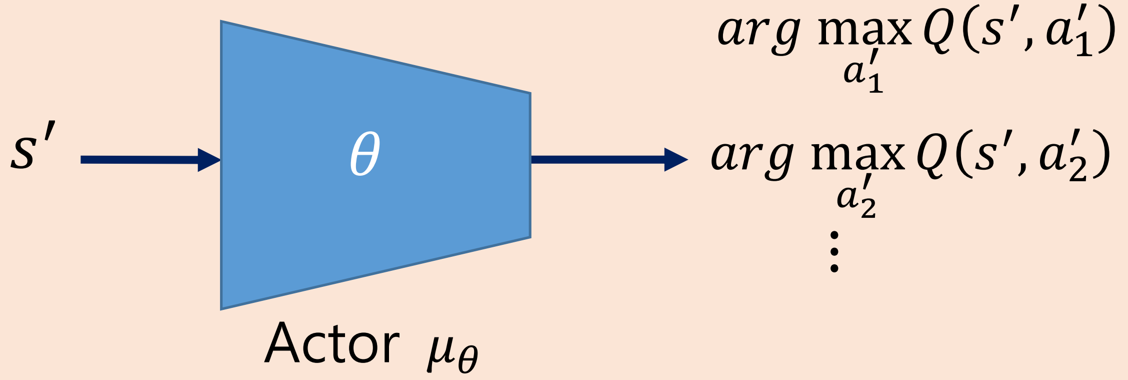
$\vdots$

$$Loss = \mathrm{E}\left[\left((r_s^a + \gamma \boxed{Q_\phi(s', \mu_\theta(s'))} - Q_\phi(s, a)\right)^2\right]$$

The policy learning side of DDPG

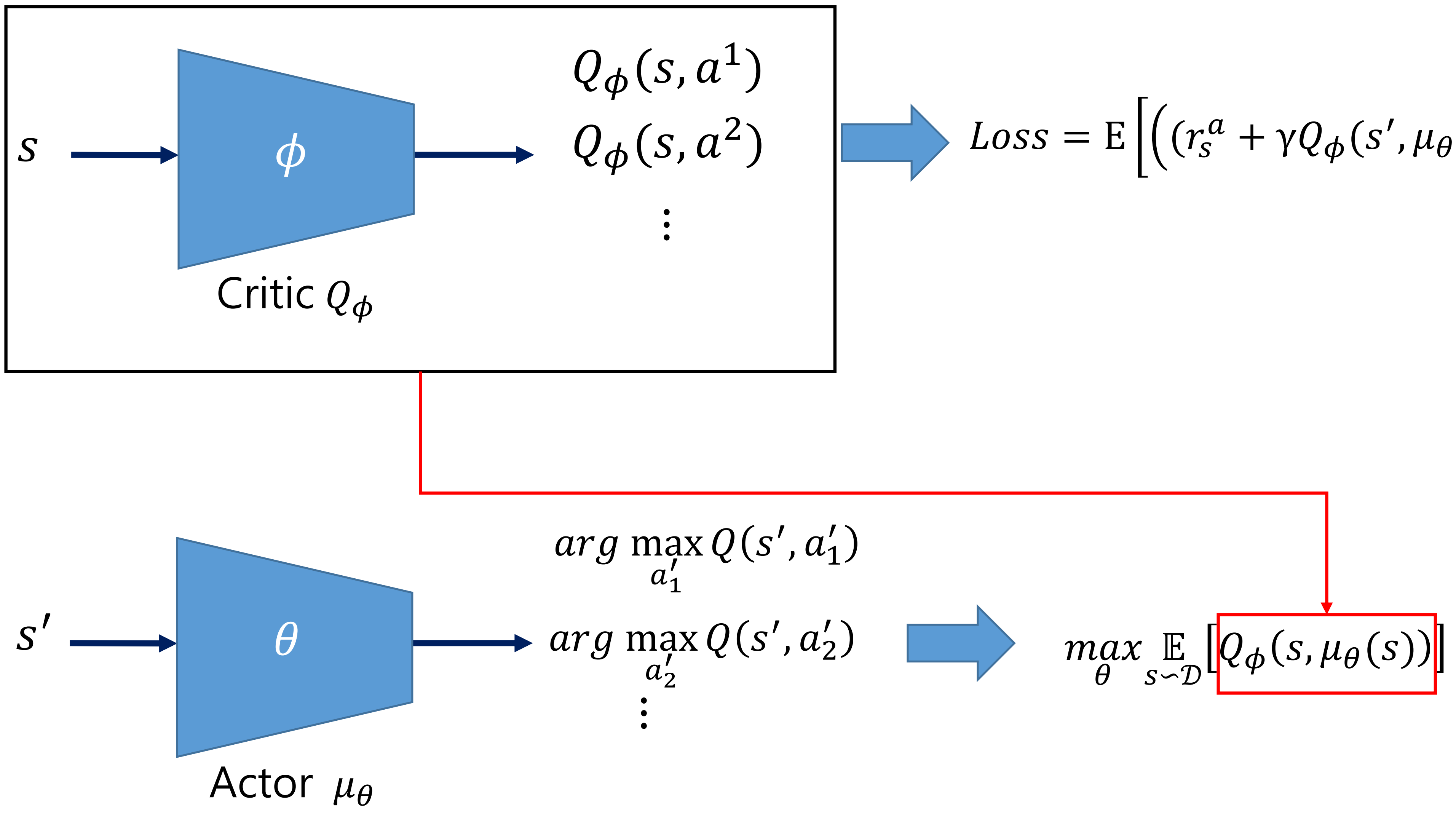


$$Loss = \mathrm{E}\left[\left(r_s^a + \gamma \max_{a'} Q_\phi(s',a') - Q_\phi(s,a)\right)^2\right]$$



$$\max_\theta \mathop{\mathbb{E}}_{s\backsim\mathcal{D}}\left[Q_\phi(s,\mu_\theta(s))\right]$$

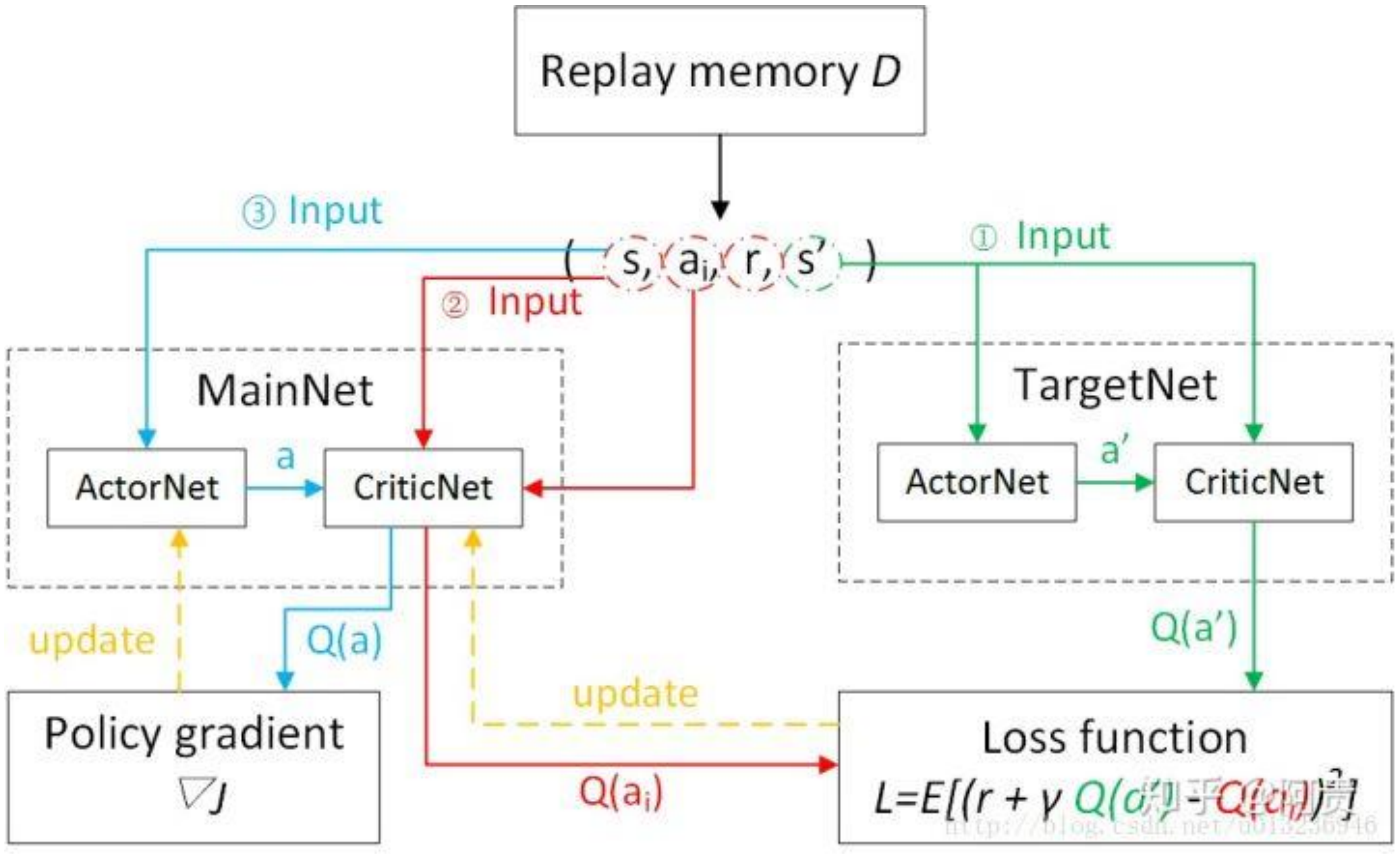
Use Q to update policy network



$$Loss = \mathrm{E}\left[\left(\left(r_s^a + \gamma Q_\phi(s', \mu_\theta(s'))\right) - Q_\phi(s, a)\right)^2\right]$$

$$\underset{\theta}{max} \underset{s \sim \mathcal{D}}{\mathbb{E}}\left[\boxed{Q_\phi(s, \mu_\theta(s))}\right]$$

Deep deterministic policy gradient



圖片來源: https://zhuanlan.zhihu.com/p/47873624

Trick one – replay buffer $\mathcal{D}$

$$L(\phi,\mathcal{D}) = \mathbb{E}_{(s,a,r,s',a')\boxed{\sim\mathcal{D}}}\left[\left(Q_\phi(s,a) - \ (r_s^a + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)^2\right])$$

In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it may not always be good to keep everything. If you only use the very-most recent data, you will overfit to that and things will break; if you use too much experience, you may slow down your learning.

Trick two – Target Network ϕ_{targ}

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',a')\backsim\mathcal{D}}{\mathbb{E}}\left[\left(Q_\phi(s,a) - \;(r_s^a + \boxed{\gamma(1-d)\max_{a'} Q_{\phi_{targ}}(s',a')}\right)^2\right])$$

Eval network ϕ Target network ϕ_{targ}

Target policy network

Computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this by using a target policy network to compute an action which approximately maximizes $Q_{\phi_{targ}}$. The target policy network is found the same way as the target Q-function: by polyak averaging the policy parameters over the course of training.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',a') \sim \mathcal{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - \ (r_s^a + \gamma(1-d) Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s') \right)^2 \right])$$

$\mu_{\theta_{targ}}$: target policy network

Exploration vs. Exploitation

Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make DDPG policies explore better, we add noise to their actions at training time.