Continuous Control with Deep Reinforcement Learning

Timothy P. Lillicrap, et al. · 14 p · 2015

By Flavio Schneider · November 2019 · ETH zürich

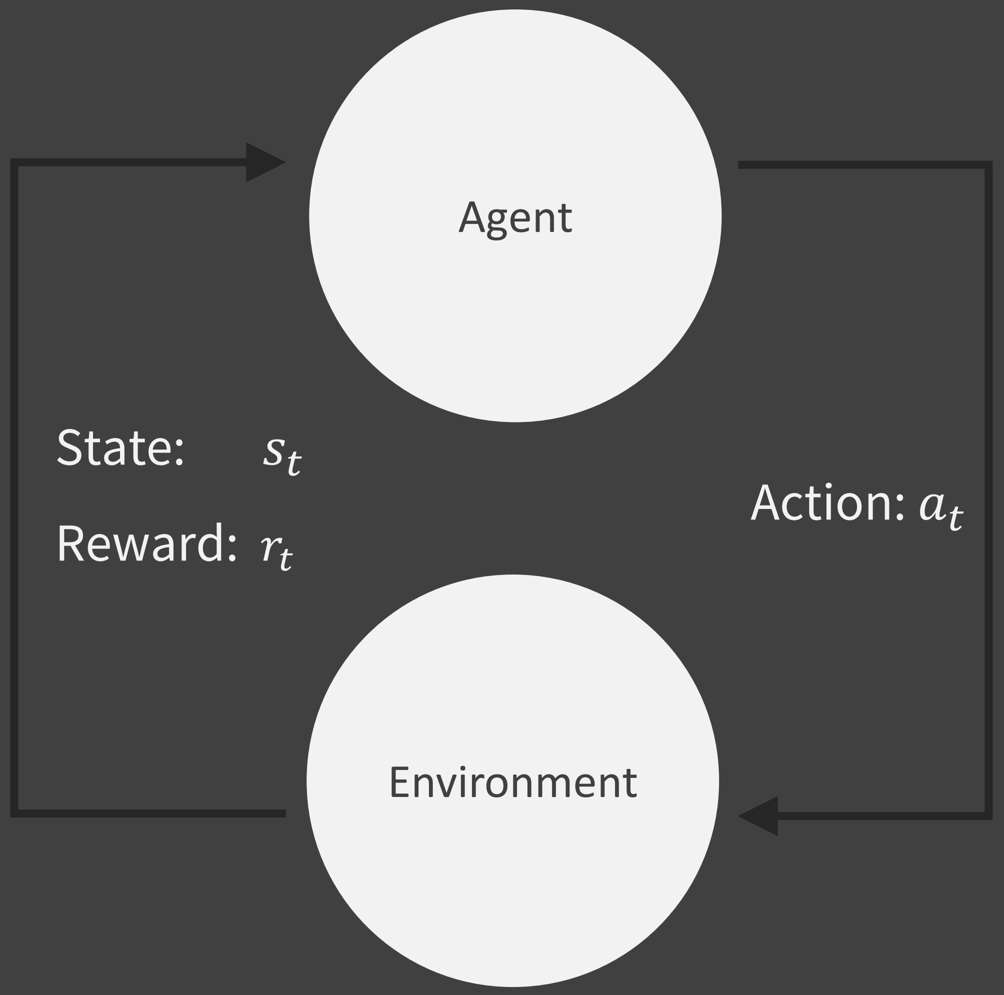
Index

- What is reinforcement learning (RL)?
- How can we describe RL formally?
- How does a RL *algorithm* look like?

What is Reinforcement Learning?

- One of the three basic paradigms of ML
- Learns intelligent behavior from reward
- Many applications
- Exploitation and exploration

Agent-Environment loop



What is Reinforcement Learning?

State · Action · Reward · Policy · Goal

Pendulum Example

What is Reinforcement Learning?

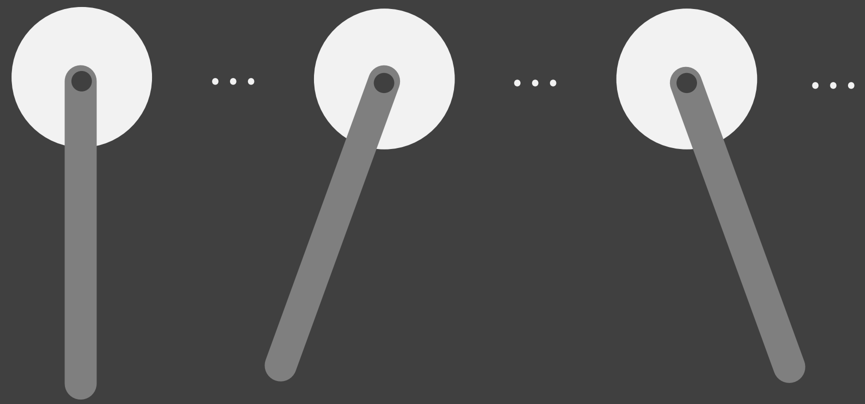
State · Action · Reward · Policy · Goal

State: $s_t \in S = R^{n+1}$ describes to the agent the environment completely at time t.

State Vector

$$s_t = \begin{bmatrix} Pendulum\ angle \\ Pendulum\ speed \end{bmatrix}$$

$t = 0$ $t = 5$ $t = 10$



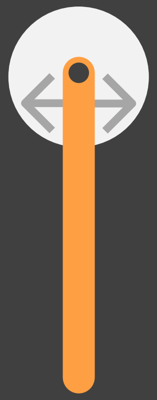
$$s_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad s_5 = \begin{bmatrix} 20 \\ -5 \end{bmatrix} \quad s_{10} = \begin{bmatrix} -20 \\ 4 \end{bmatrix}$$

What is Reinforcement Learning?

State · **Action** · Reward · Policy · Goal

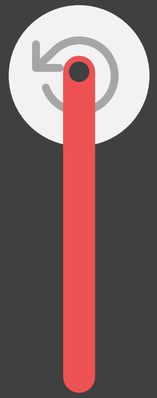
Action: $a_t \in A$ represents what the agent is going to do at time t.

Discrete Action Space



$$a_t \in A = (Left, Right) = (a^{(0)}, a^{(1)})$$

Continuous Action Space



$$a_t \in A = [-1,1]$$

What is Reinforcement Learning?

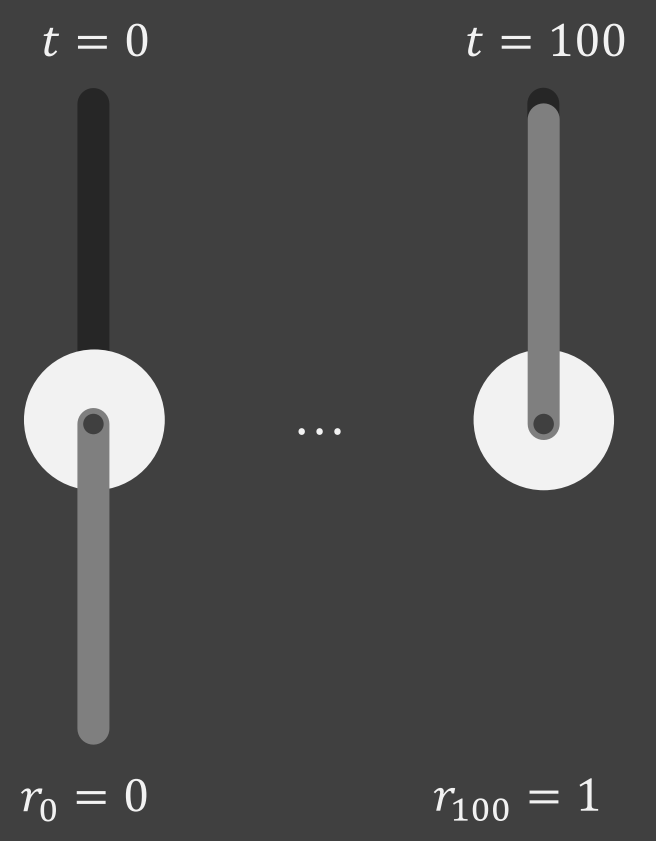
State · Action · **Reward** · Policy · Goal

Reward: $r_t = R(s_t)$ or $r_t = R(s_t, a_t)$ is a score that tells the agent how good was the last move a_t in state s_t.

Return: $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ where $\tau = (s_0, a_0, s_1, a_1, \dots)$ is a sequence of state/action pairs and $\gamma \in (0,1)$.

Reward Function

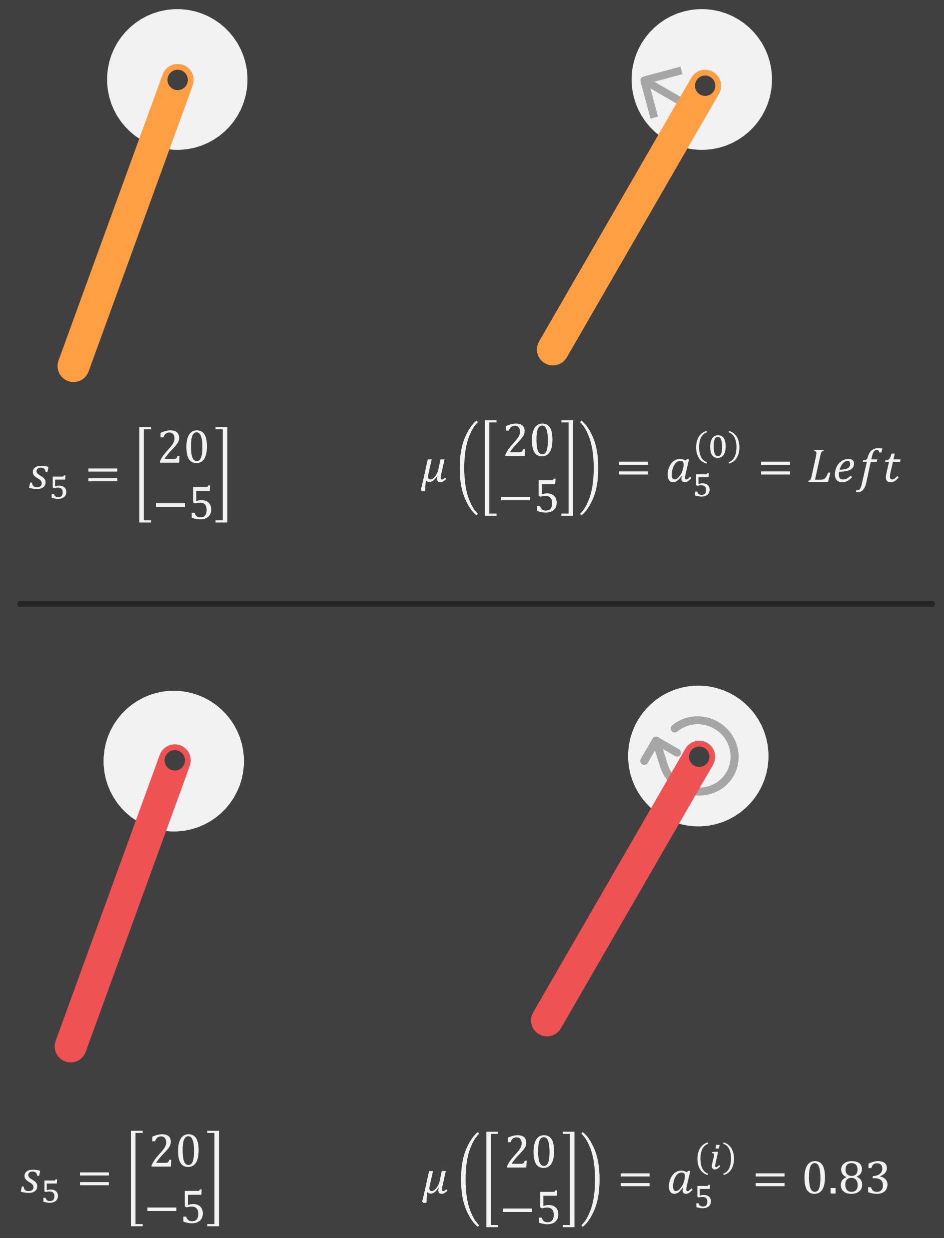
$$r_t = R(s_t) = \begin{cases} 1 & if\ s_t[0] == 180 \\ 0 & otherwise \end{cases}$$



What is Reinforcement Learning?

State · Action · Reward · **Policy** · Goal

Policy: $\mu(s_t) = a_t$ is the brain of the agent, a function that maps states to actions.



What is Reinforcement Learning?

State · Action · Reward · Policy · **Goal**

Trajectory Probability:

$$P(\tau \mid \mu) \coloneqq \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t) P(\mu(s_t) = a_t)$$

Expected Return:

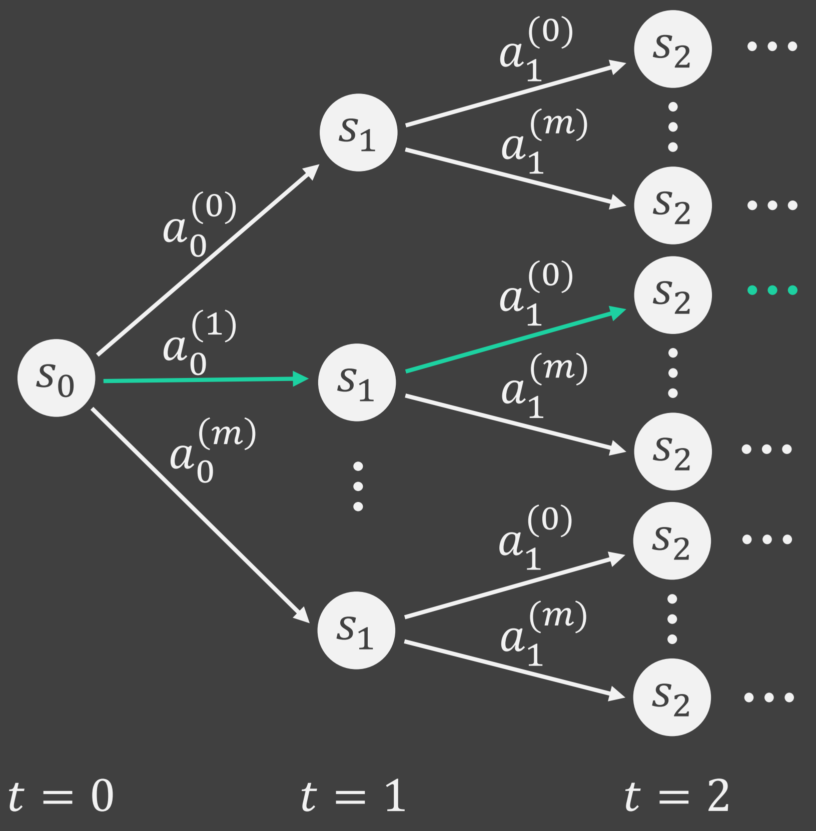
$$\mathbb{E}_{a_t=\mu(s_t)}[R(\tau)] = \int_\tau P(\tau \mid \mu) R(\tau)$$

Goal · Find Optimal Policy:

$$\mu^* \coloneqq \underset{\mu}{\operatorname{argmax}}\, \mathbb{E}_{a_t=\mu(s_t)}[R(\tau)]$$

Trajectory

$$\tau = \left(s_0, a_0^{(1)}, s_1, a_1^{(0)}, \ldots\right)$$



Important Functions

Quality · Bellmann

Important Functions

Quality · Bellmann

Quality Function:

$$\mathbb{E}_{a_t=\mu(s_t)}[R(\tau)]$$

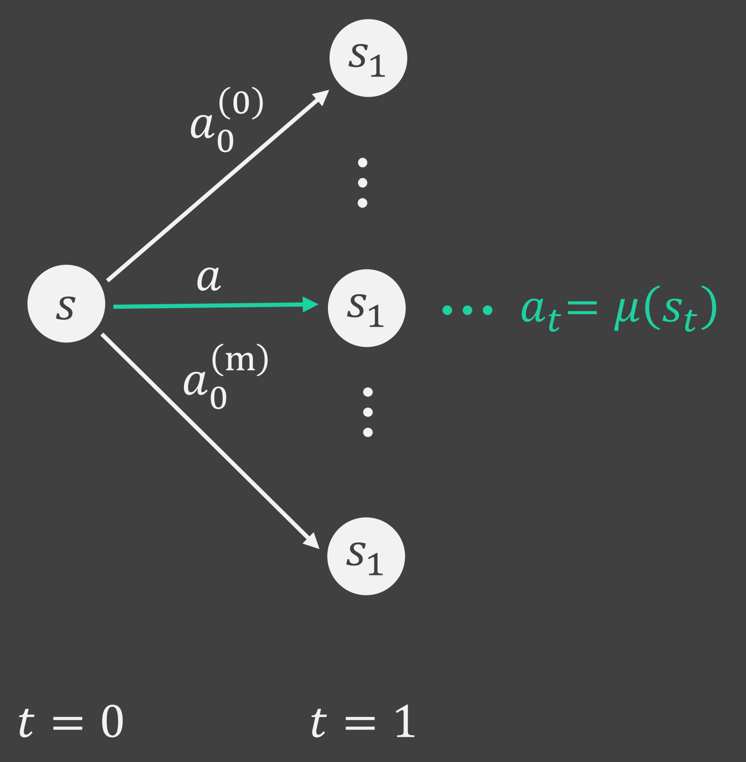
$$Q^\mu(s,a) \coloneqq \mathbb{E}_{s_0=s,a_0=a,\, a_t=\mu(s_t)}[R(\tau)]$$

Optimal Quality Function:

$$Q^*(s,a) \coloneqq \max_\mu Q^\mu(s,a)$$

Optimal Action:

$$\mu^*(s) = \underset{a}{argmax}\, Q^*(s,a) = a^*$$



Important Functions

Quality · Bellman

Bellman Equation:

$$Q^*(s,a) = \mathbb{E}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

Algorithms Evolution

DQN · DPG · DDPG

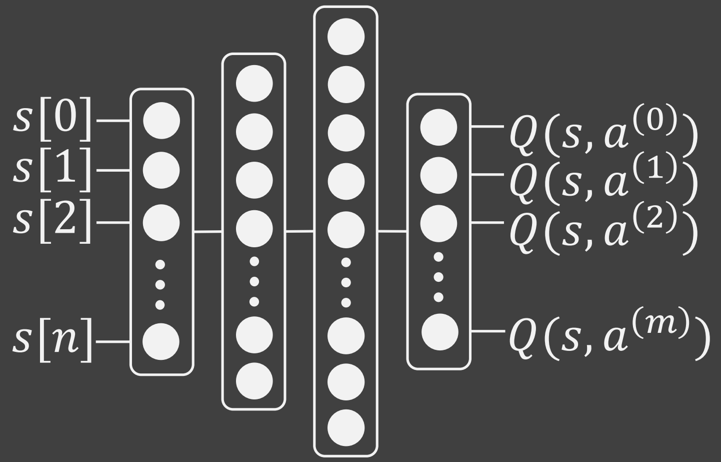
Question:

How can we solve any (discrete and then continuous) reinforcement learning problem?

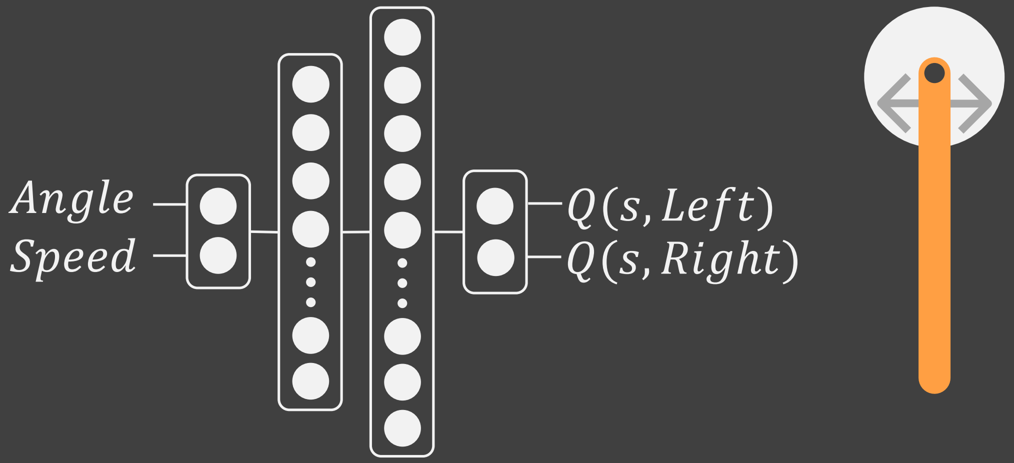
Idea:

Approximate $Q^*(s, a)$ using a neural network.

"Critic" Network · $Q^{\mu}(s, a \mid \theta^{Q})$:



Pendulum



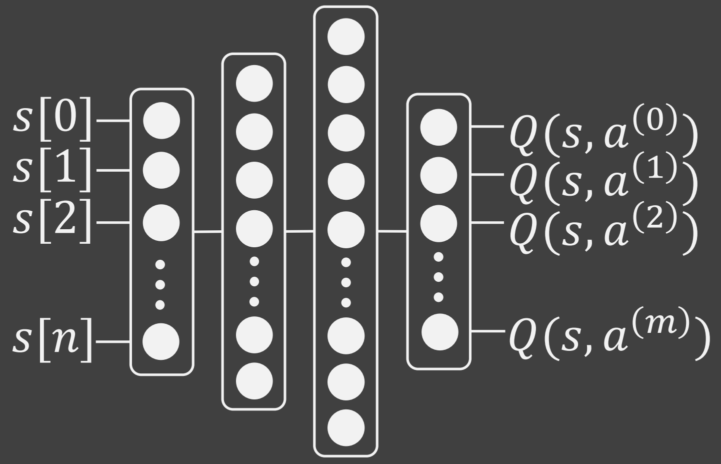
Algorithms Evolution

DQN · DPG · DDPG

Deep Q Network

1. *Initialize randomly* $Q^{\mu}(s, a \mid \theta^{Q})$
2. *Get initial state* s
3. *Repeat:*
   a. $a = \underset{a'}{argmax}\, Q^{\mu}(s, a' \mid \theta^{Q})$
   b. $a = a^{(i)}$ *for* $i \in \{0, \dots, m\}$ *random with probability* ϵ
   c. *Execute* a *and observe* $r = R(s, a)$ *and* s'
   d. $Q^{T}(s, a, r, s') \coloneqq r + \gamma \underset{a'}{max}\, Q^{\mu}(s', a' \mid \theta^{Q})$
   e. $\theta^{Q} \leftarrow \theta^{Q} - \alpha \nabla_{\theta^{Q}} \left[\left(Q^{T}(s, a, r, s') - Q^{\mu}(s, a \mid \theta^{Q}) \right)^{2} \right]$
   f. $s \leftarrow s'$

"Critic" Network · $Q^{\mu}(s, a \mid \theta^{Q})$:



Bellman Equation

$$Q^{*}(s, a) = \mathbb{E}\left[R(s, a) + \gamma \max_{a'} Q^{*}(s', a') \right]$$

Algorithms Evolution

DQN · DPG · DDPG

Deep Q Network

1. *Initialize randomly* $Q^{\mu}(s, a \mid \theta^{Q})$
2. *Get initial state* s
3. *Repeat:*

   a. $a = \underset{a'}{argmax}\, Q^{\mu}(s, a' \mid \theta^{Q})$

   b. $a = a^{(i)}$ *for* $i \in \{0, \dots, m\}$ *random with probability* ϵ

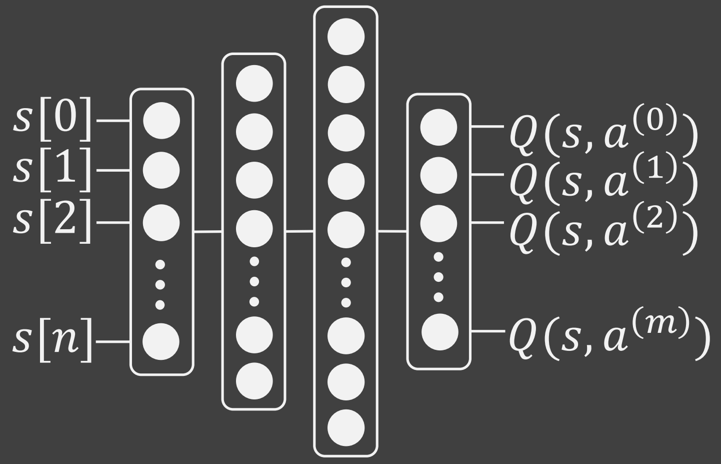
   c. *Execute* a *and observe* $r = R(s, a)$ *and* s'

   d. $Q^{T}(s, a, r, s') \coloneqq r + \gamma \underset{a'}{max}\, Q^{\mu}(s', a' \mid \theta^{Q})$

   e. $\theta^{Q} \leftarrow \theta^{Q} - \alpha \nabla_{\theta^{Q}} \left[\left(Q^{T}(s, a, r, s') - Q^{\mu}(s, a \mid \theta^{Q}) \right)^{2} \right]$

   f. $s \leftarrow s'$

"Critic" Network · $Q^{\mu}(s, a \mid \theta^{Q})$:



Bellman Equation

$$Q^{*}(s, a) = \mathbb{E}\left[R(s, a) + \gamma \max_{a'} Q^{*}(s', a') \right]$$

Algorithms Evolution

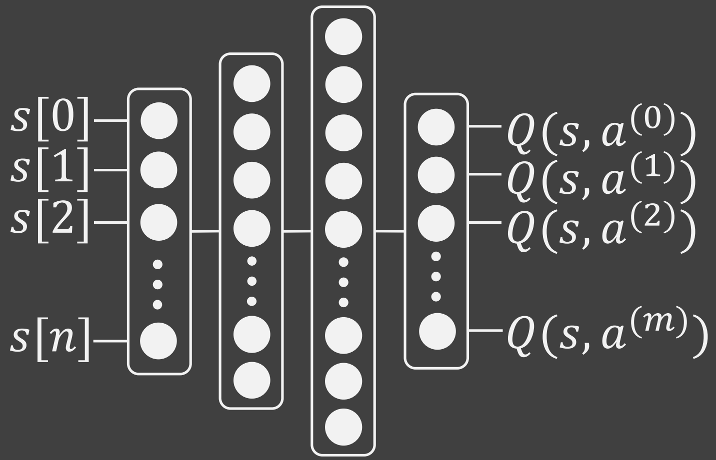
DQN · DPG · DDPG

Deep Q Network

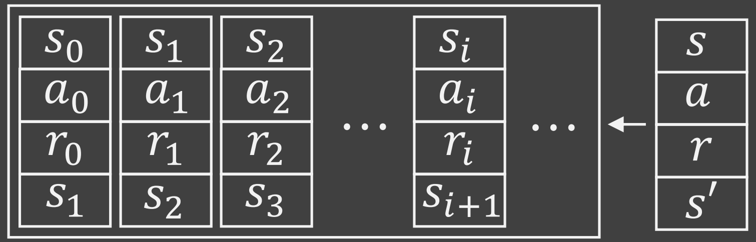
Code Structure

1. *Initialize randomly* $Q^{\mu}(s, a \mid \theta^{Q})$ *and* $Q^{\mu'}(s, a \mid \theta^{Q'})$
2. *Initialize replay buffer* $\mathcal{R} = S \times A \times R \times S$
3. *Get initial state* s
4. *Repeat:*
   a. · *Sample*
   b. · *Train*

Target “Critic” Network · $Q^{\mu'}(s, a \mid \theta^{Q'})$:



Replay Buffer · $\mathcal{R}$:



Algorithms Evolution

DQN · DPG · DDPG

Deep Q Network

· *Sample*

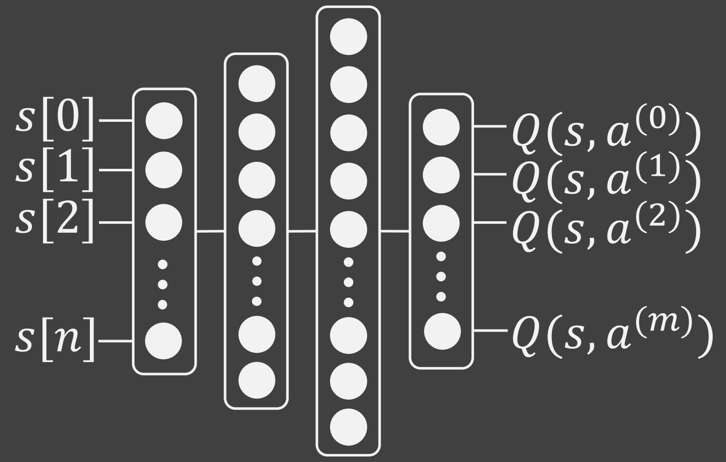
i. $a = \underset{a'}{argmax}\, Q^{\mu}(s, a' \mid \theta^{Q})$

ii. $a = a^{(i)}$ *for* $i \in \{0, \dots, m\}$ *random with probability* ϵ

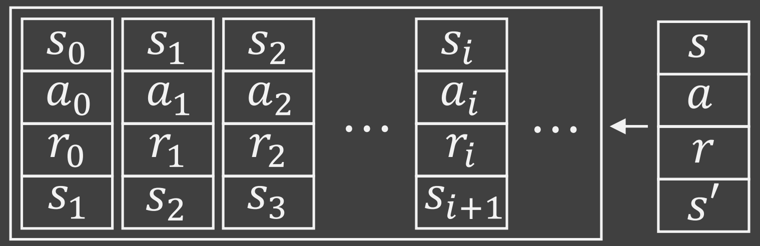
iii. *Execute* a *and observe* r *and* s'

iv. *Store* $\mathcal{R} \leftarrow \mathcal{R} \cup (s, a, r, s')$

Target "Critic" Network · $Q^{\mu'}(s, a \mid \theta^{Q'})$:



Replay Buffer · $\mathcal{R}$:



Algorithms Evolution

DQN · DPG · DDPG

Deep Q Network

· *Train*

i. *Sample random batch* $\mathcal{B} \subseteq \mathcal{R}$

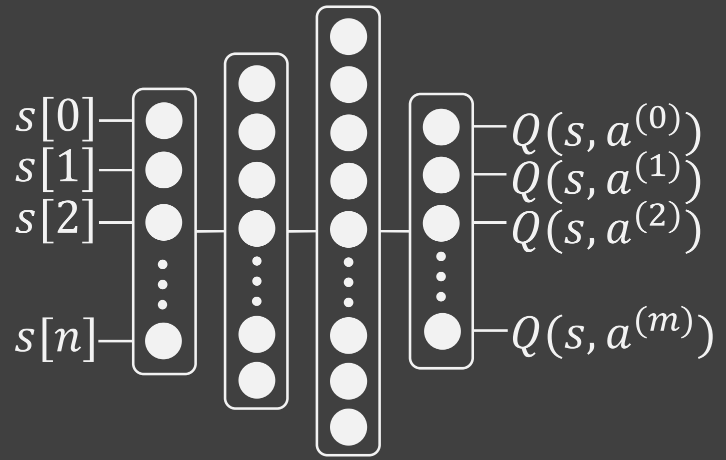
ii. *For each* $(s, a, r, s') \in \mathcal{B}$

I. $$Q^T(s, a, r, s') \coloneqq r + \gamma \max_{a'} Q^{\mu'}(s, a' \mid \theta^{Q'})$$

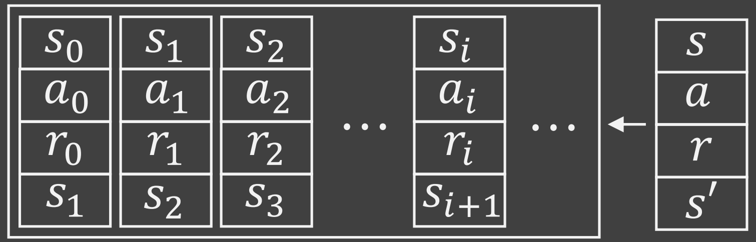
II. $$\theta^Q \leftarrow \theta^Q - \alpha \nabla_{\theta^Q} \left[\left(Q^T(s, a, r, s') - Q^{\mu}(s, a \mid \theta^Q) \right)^2 \right]$$

iii. *Every* C *steps reset* $\theta^{Q'} \leftarrow \theta^Q$

Target "Critic" Network · $Q^{\mu'}(s, a \mid \theta^{Q'})$:



Replay Buffer · $\mathcal{R}$:



Algorithms Evolution

DQN · DPG · DDPG

Deep Q Network

· *Train*

i. *Sample random batch* $\mathcal{B} \subseteq \mathcal{R}$

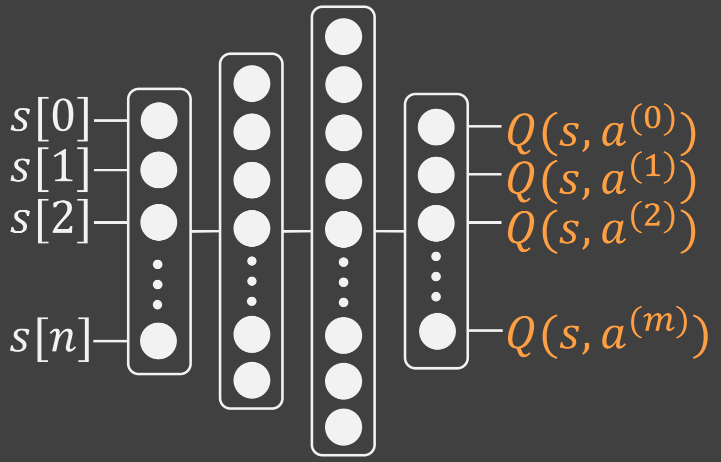
ii. *For each* $(s, a, r, s') \in \mathcal{B}$

I. $Q^T(s, a, r, s') \coloneqq r + \gamma \max_{a'} Q^{\mu'}(s, a' \mid \theta^{Q'})$

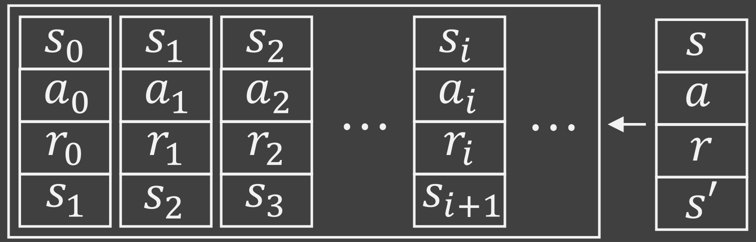
II. $\theta^Q \leftarrow \theta^Q - \alpha \nabla_{\theta^Q} \left[\left(Q^T(s, a, r, s') - Q^{\mu}(s, a \mid \theta^Q) \right)^2 \right]$

iii. *Every* C *steps reset* $\theta^{Q'} \leftarrow \theta^Q$

Target "Critic" Network · $Q^{\mu'}(s, a \mid \theta^{Q'})$:



Replay Buffer · $\mathcal{R}$:



Algorithms Evolution

DQN · **DPG** · DDPG *Deterministic Policy Gradient*

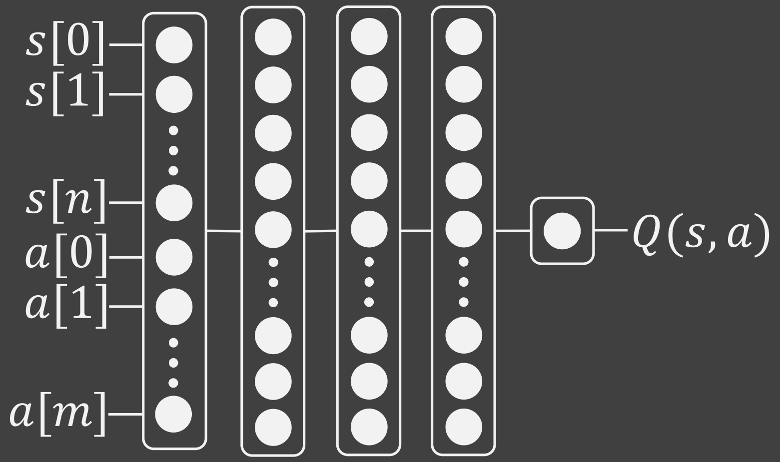
Expected Return:

$$J(\mu_\theta) \coloneqq \mathbb{E}_{a_t=\mu_\theta(s_t)}[R(\tau)]$$

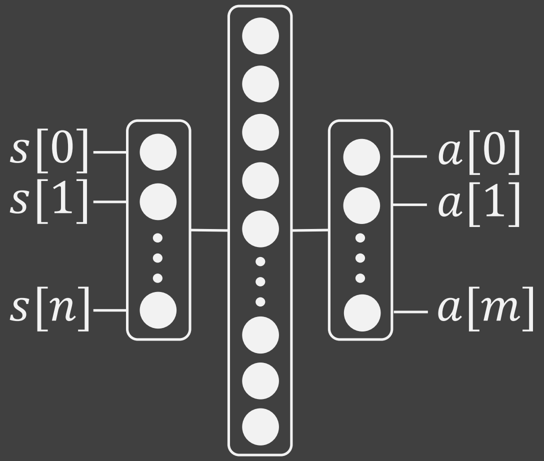
Deterministic Policy Gradient Theorem:

$$\nabla_{\theta^\mu} J(\mu_{\theta^\mu}) = \mathbb{E}\left[\nabla_a Q^\mu(s, a \mid \theta^Q)\nabla_{\theta_\mu}\mu(s \mid \theta^\mu)\right]$$

“Critic” Network · $Q^\mu(s, a \mid \theta^Q)$:



“Actor” Network · $\mu(s \mid \theta^\mu)$:

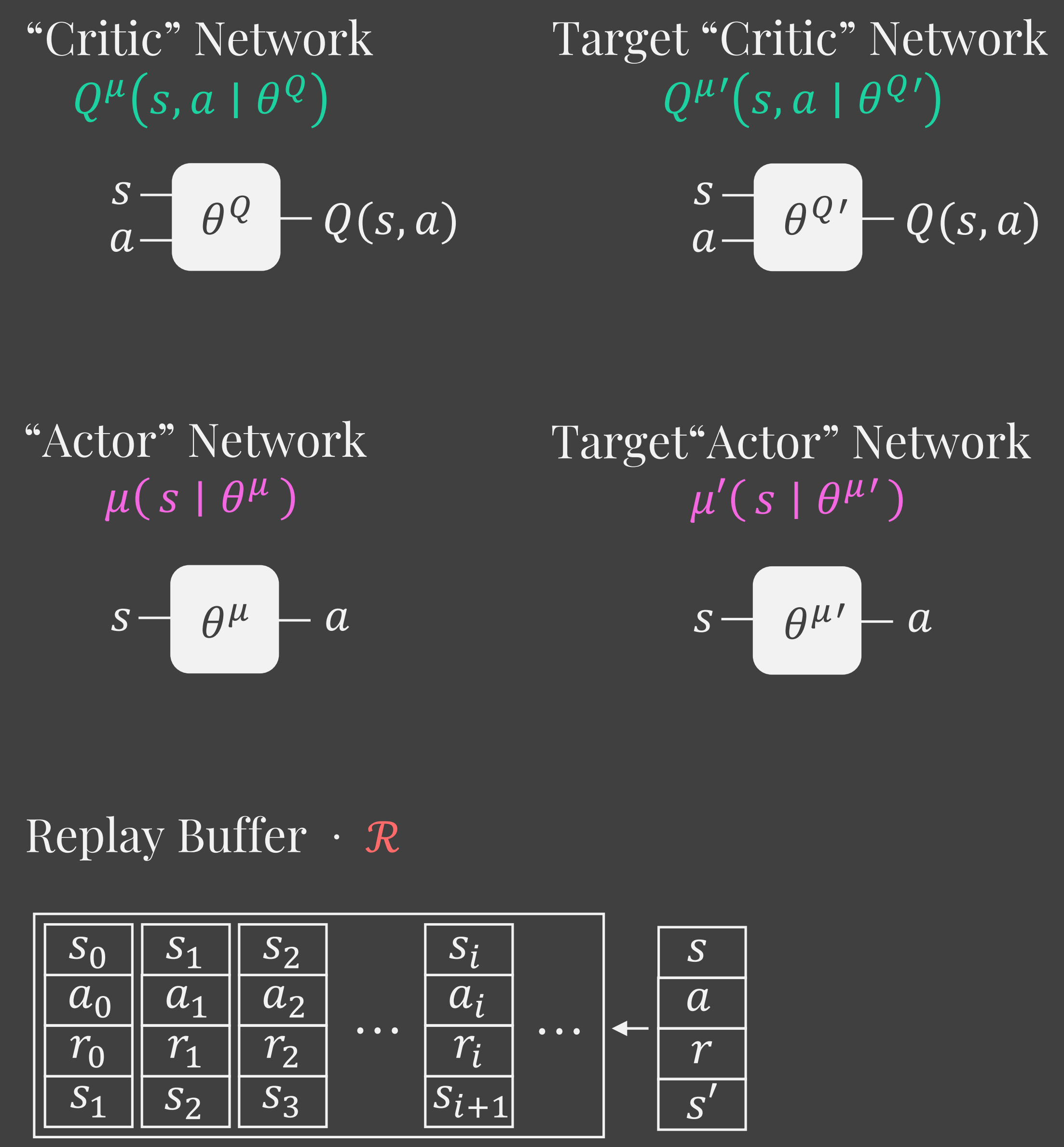


Algorithms Evolution

DQN · DPG · **DDPG** *Deep DPG*

Code Structure

1. *Initialize randomly* $Q^{\mu}(s, a \mid \theta^{Q})$ *and* $\mu(s \mid \theta^{\mu})$
2. *Initialize* $Q^{\mu'}(s, a \mid \theta^{Q'} \leftarrow \theta^{Q})$ *and* $\mu'(s \mid \theta^{\mu'} \leftarrow \theta^{\mu})$
3. *Initialize replay buffer* $\mathcal{R} = S \times A \times R \times S$
4. *Observe initial state* s
5. *Repeat:*
   a. · *Sample*
   b. · *Train*



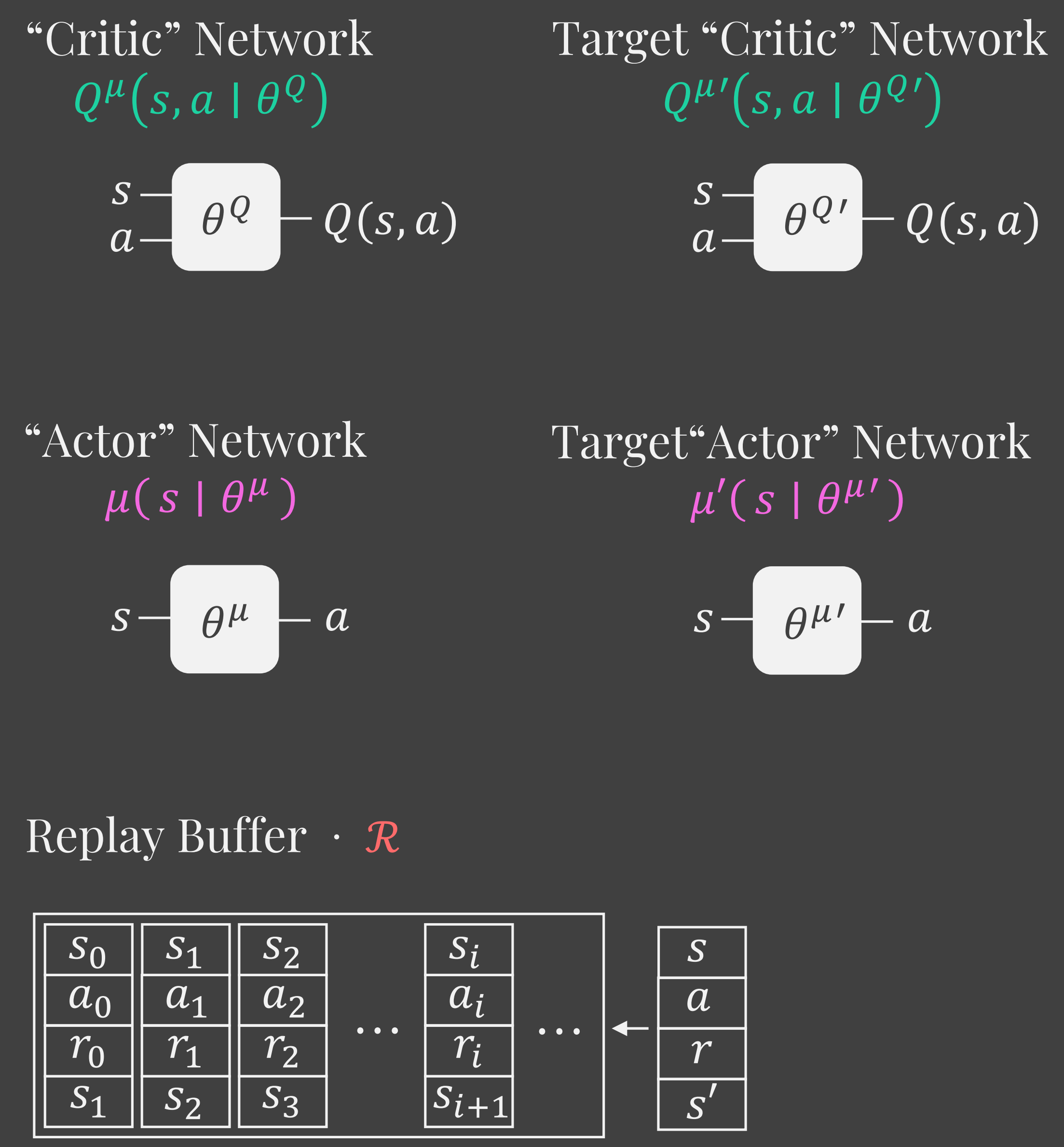
Algorithms Evolution

DQN · DPG · **DDPG** *Deep DPG*

· *Sample:*

i. *Execute* $a = \mu(s \mid \theta^{\mu}) + \mathcal{N}$ *and observe* r *and* s'

ii. *Store* $\mathcal{R} \leftarrow \mathcal{R} \cup (s, a, r, s')$



Algorithms Evolution

DQN · DPG · **DDPG** *Deep DPG*

· *Train*

i. *Sample random batch* $\mathcal{B} \subseteq \mathcal{R}$

ii. $Q^T(s,a,r,s') \coloneqq r + \gamma Q^{\mu'}\left(s', \overbrace{\mu'\left(s' \mid \theta^{\mu'}\right)}^{a'} \mid \theta^{Q'}\right)$

iii. $L^Q(\theta^Q) = \frac{1}{|\mathcal{B}|}\sum_{(s,a,r,s')\in\mathcal{B}}\left[Q^T(s,a,r,s') - Q^\mu\left(s,a \mid \theta^Q\right)\right]^2$

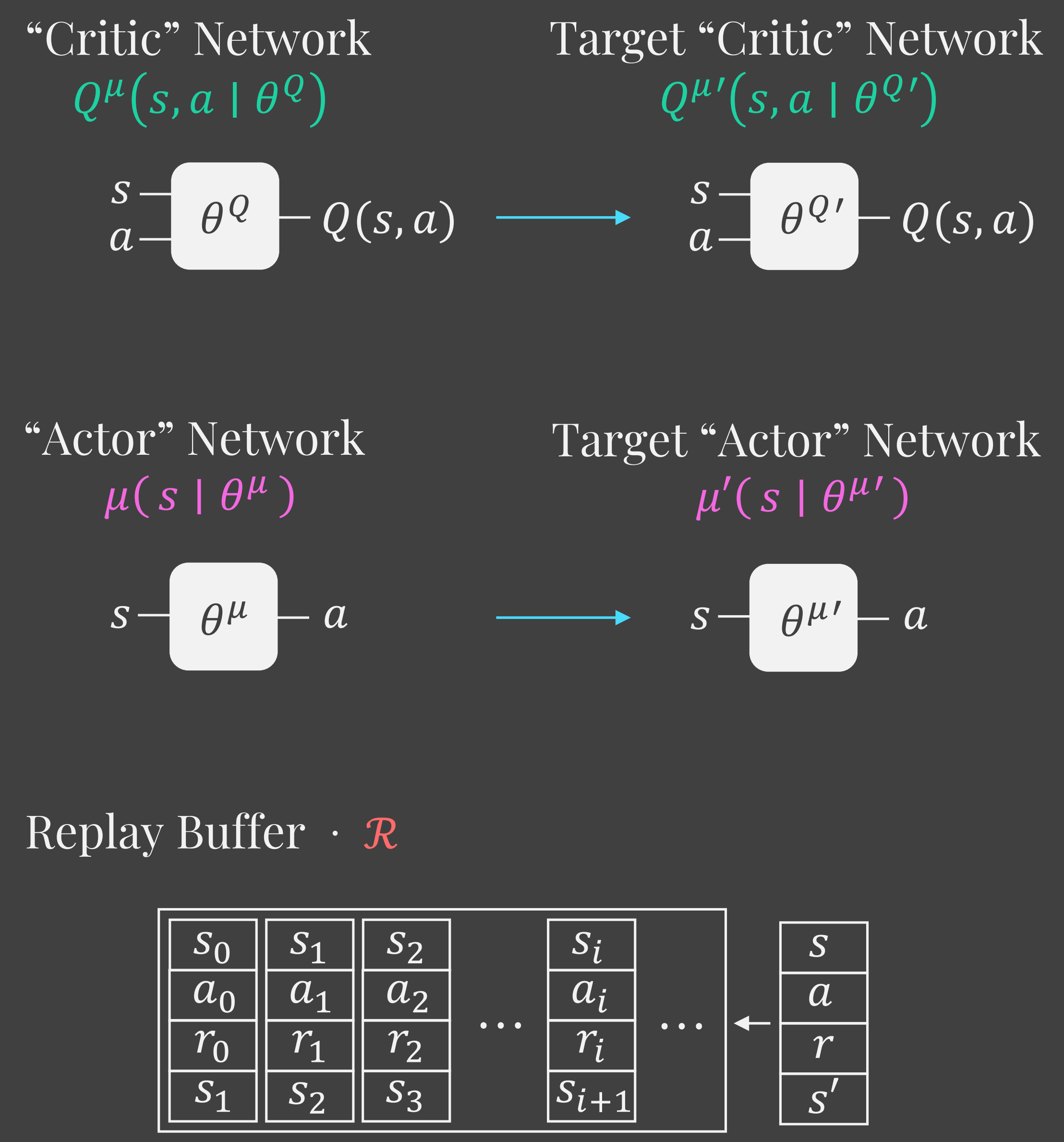
iv. $\theta^Q \leftarrow \theta^Q - \alpha \nabla_{\theta^Q} L(\theta^Q)$

v. $L^\mu(\theta^\mu) = \frac{1}{|\mathcal{B}|}\sum_{(s,a,r,s')\in\mathcal{B}} \nabla_a Q^\mu\left(s,a \mid \theta^Q\right)\nabla_{\theta^\mu}\mu\left(s \mid \theta^\mu\right)$

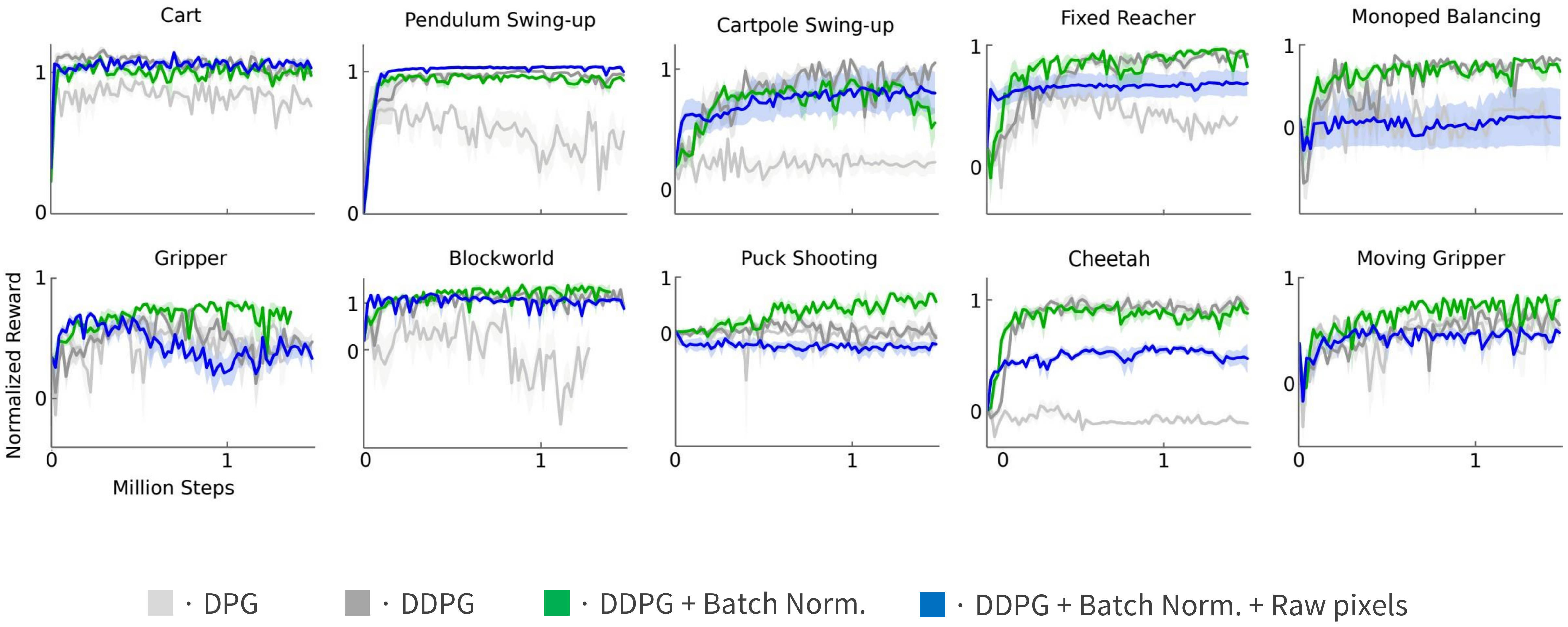
vi. $\theta^\mu \leftarrow \theta^\mu - \alpha \nabla_{\theta^\mu} L(\theta^\mu)$

vii. $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ *where* $\tau \ll 1$

viii. $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$



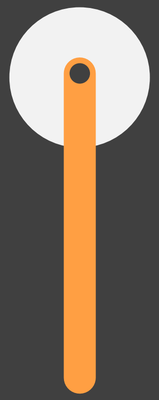
DDPG Performance

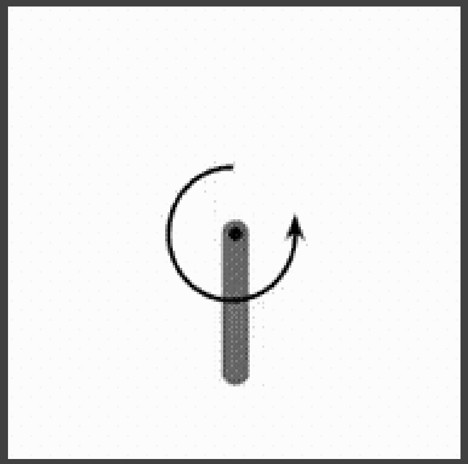


Conclusion

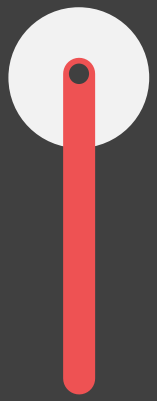
+ *Continuous control is much more stable.*
+ *Same algorithm/tuning solves many (20+) problems.*
+ *Pseudo-Code and hyperparameters are provided.*
- *Computationally expensive.*
- *Many hyperparameters must be tuned correctly.*
- *Many decisions of the paper require lot of pre-knowledge.*

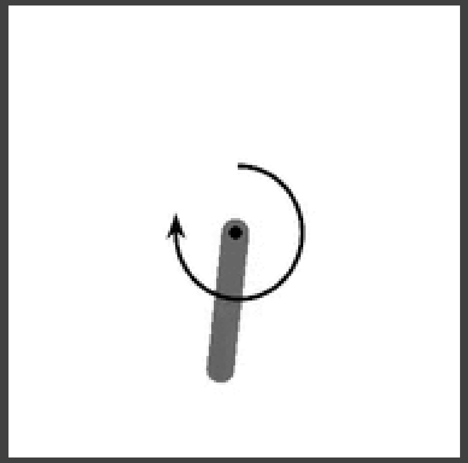
DQN · Discrete



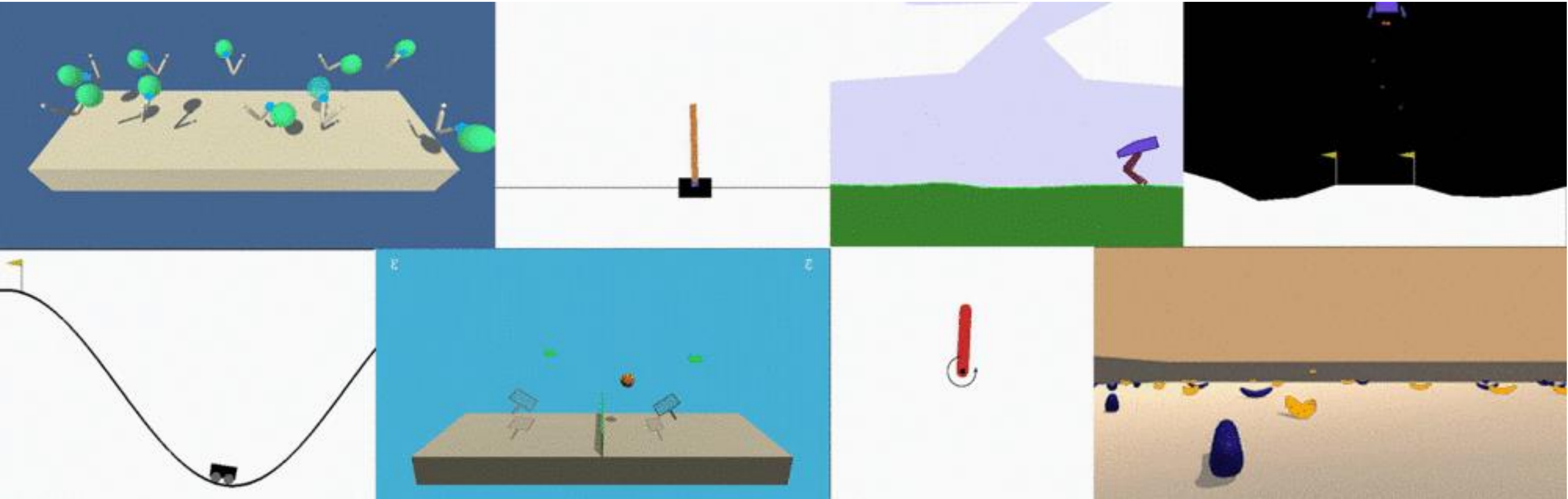


DDPG · Continuous





Any Questions?



References

Presented Paper:
Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

DQN Paper:
Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

DPG Paper:
Silver, David, et al. "Deterministic policy gradient algorithms." 2014.

Useful Websites:
- https://spinningup.openai.com/en/latest/spinningup/rl_intro.html
- https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b
- https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4

Hyperparameters

Learning Rates	$\alpha_{actor} = 10^{-4}$
	$\alpha_{critic} = 10^{-3}$
Discount Factor	$\gamma = 0.99$
Target update	$\tau = 0.001$
Neural network	*ReLU for all except last layer of the actor that uses* $tanh$, 2 hidden layers of 300 respectively 400 neurons.
Buffer, Batch Size	$10^6, 64$

Improvements after 2015

- *Paper uses batch normalization which has been found to improve only certain problems.*
- *An architecture with several actors can explore more of the environment.*