NeurIPS 2020 Tutorial

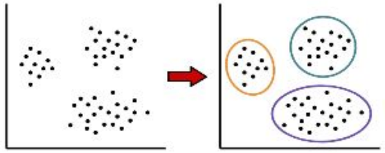
Offline Reinforcement Learning:

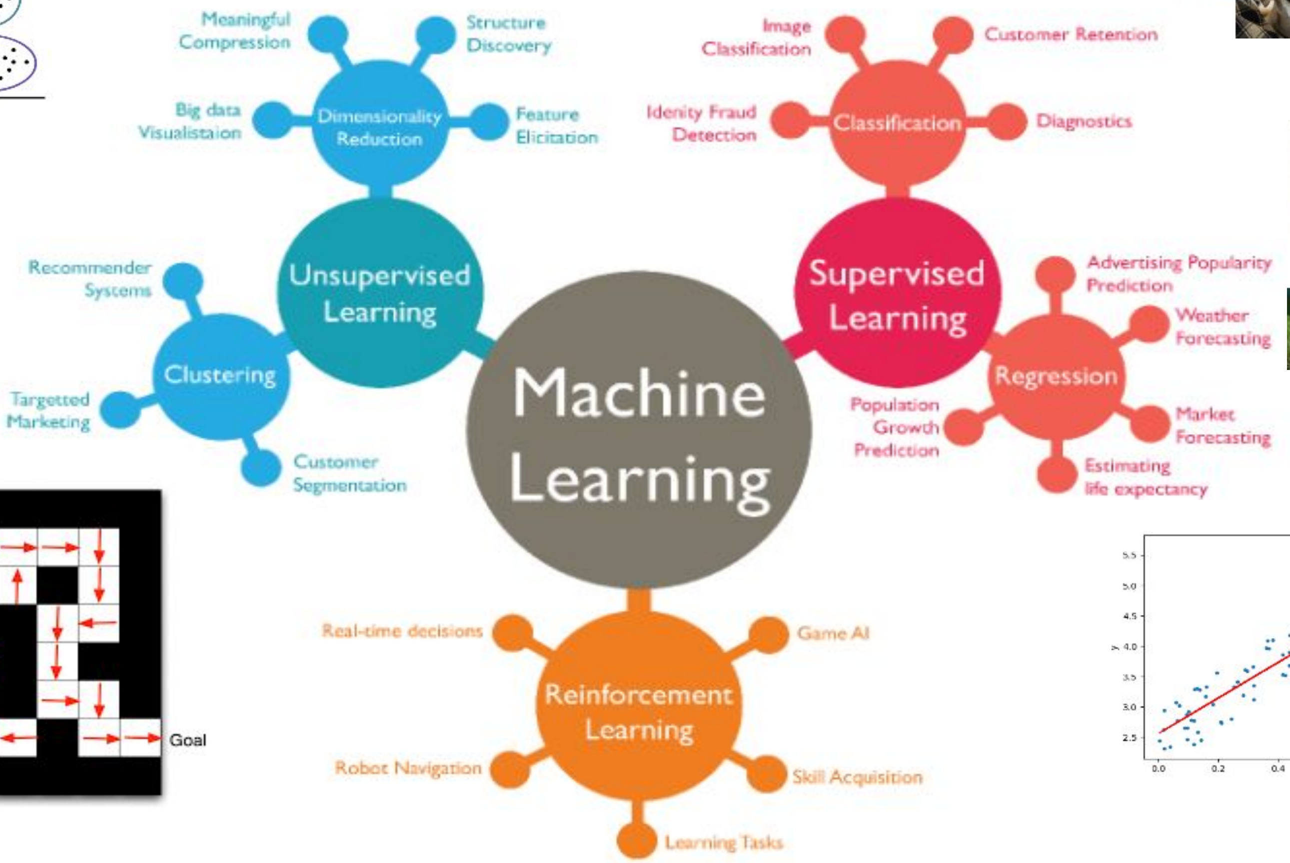
From Algorithms to Practical Challenges

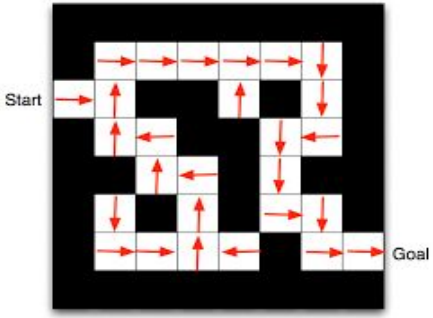
Aviral Kumar, Sergey Levine(UC Berkeley)

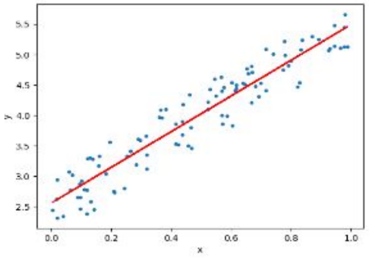
Reviewed by Susang Kim

Reinforcement Learning is a branch of Machine Learning









https://box-world.tistory.com/5

Reinforcement in Computer Vision

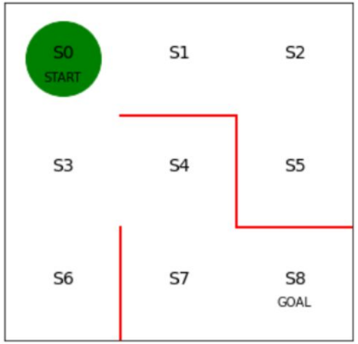
Challenges in **high-dimensional observation.** (e.g., image, unstructured, a large number of pixels)

The standard approach relies **on sensors to obtain information that would be helpful for learning.**

It's hard to use reinforcement learning algorithms to solve tasks using **only low-level observations**, such as learning **robotic control using only unstructured raw image data.** (Robot Vision)
(by explicitly learning latent representations)

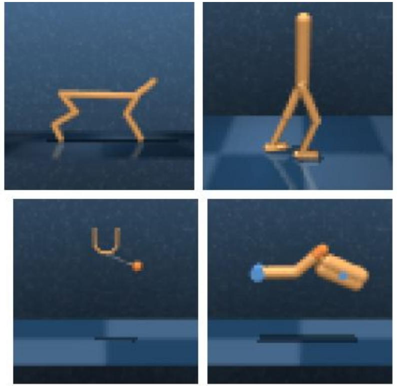
Learning from only image data is hard because the RL algorithm must learn **both a useful representation of the data** and **the task itself.** (+representation learning problem)

The agent (cheetah) didn't have any prior knowledge about movement.



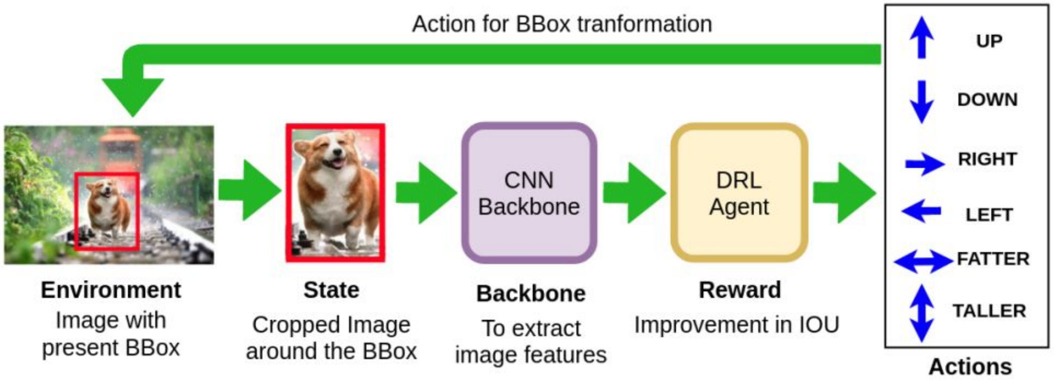

```
state = np.array([[np.nan, 1, 1, np.nan],  # s0
                  [np.nan, 1, np.nan, 1],  # s1
                  [np.nan, np.nan, 1, 1],  # s2
                  [1, 1, 1, np.nan],  # s3
                  [np.nan, np.nan, 1, 1],  # s4
                  [1, np.nan, np.nan, np.nan],  # s5
                  [1, np.nan, np.nan, np.nan],  # s6
                  [1, 1, np.nan, np.nan], # s7, s8
                  ])
```

Sensor numerical data

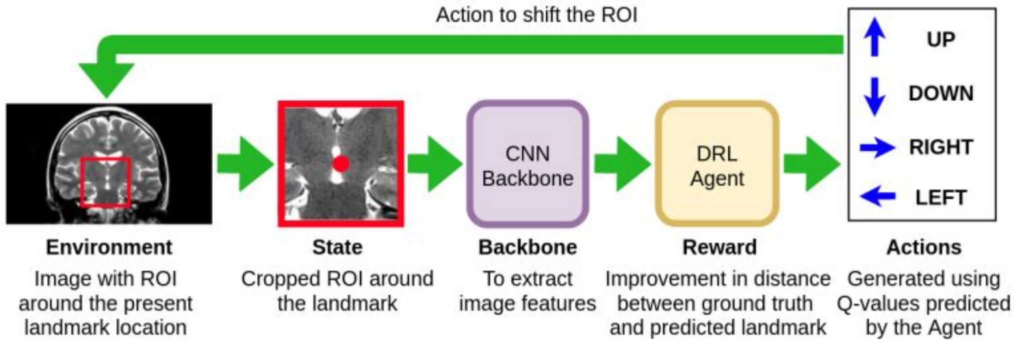


raw image data

Deep Reinforcement Learning in Object Detection



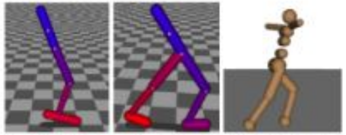
DRL implementation for object detection to maximize the improvement in IOU. DRL used a tuple of feature vector and history of actions for state and change in IOU across actions as a reward.



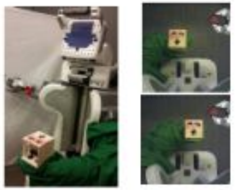
DRL implementation for landmark detection, Red box is the RoI to maximize the reward corresponding to the improvement in distance between the ground truth and predicted landmark location.

Reinforcement learning vs Supervised Learning

Mnih et al. '13

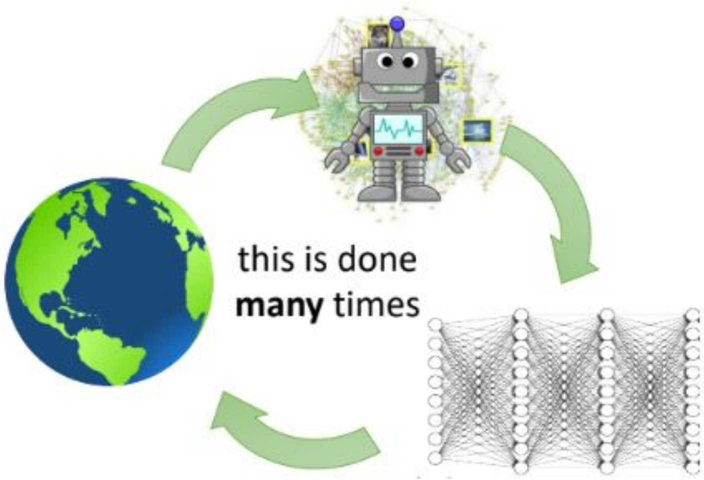


Schulman et al. '14 & '15



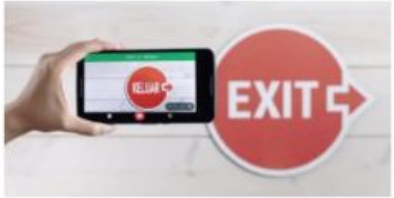
Levine*, Finn*, et al. '16

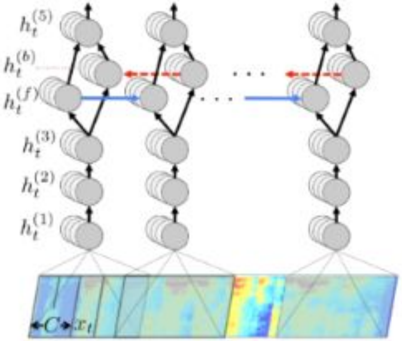
OpenAI



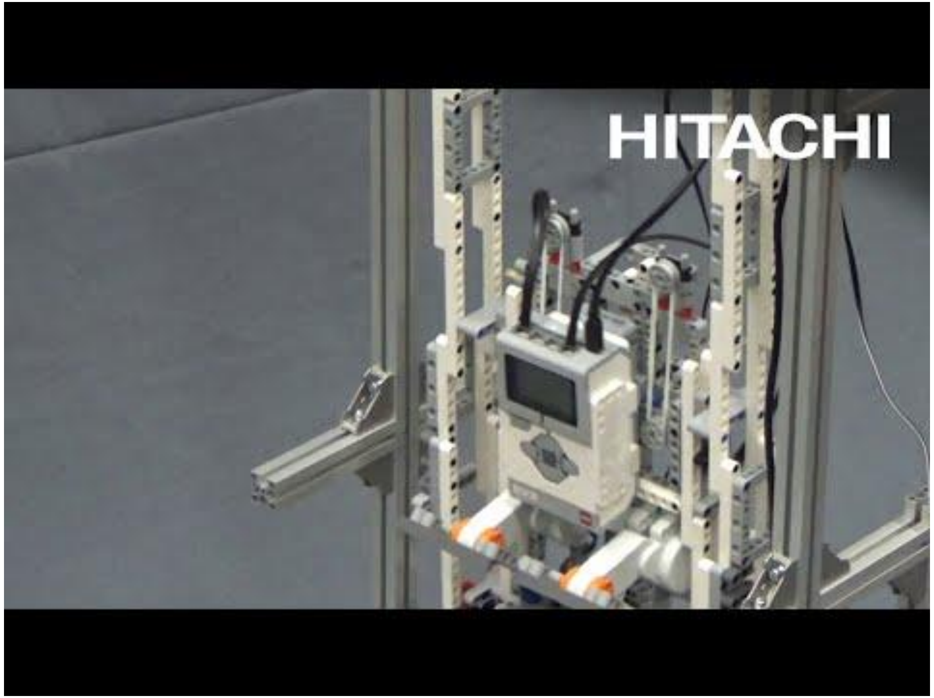
DeepMind

enormous gulf



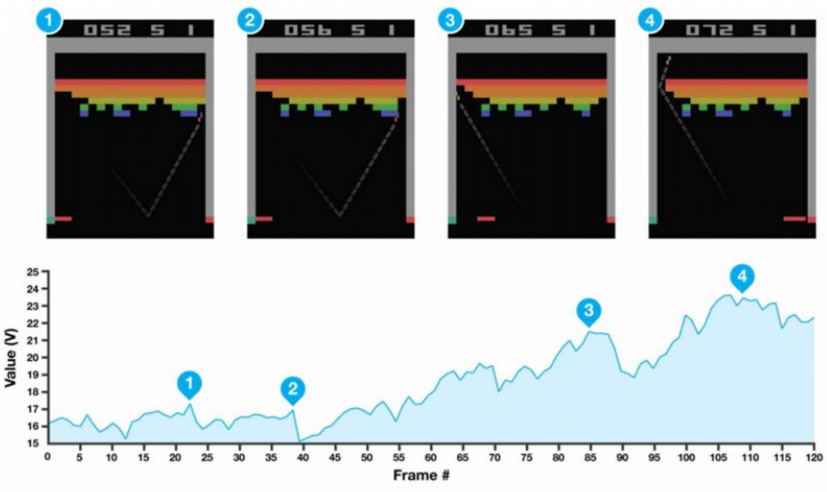


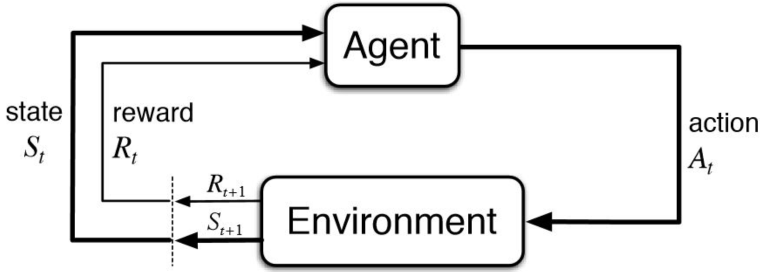
Practical Challenges



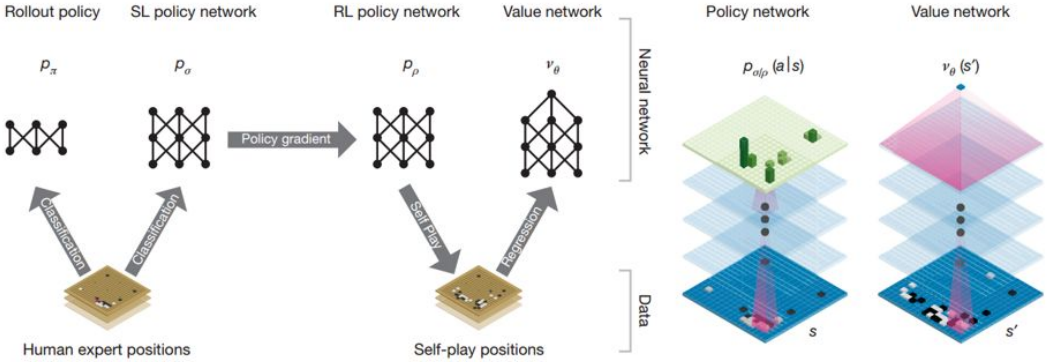
AI × Swing Robot - Hitachi : https://www.youtube.com/watch?v=q8i6wHCefU4
Deep Reinforcement Learning for Robotic Manipulation : https://youtu.be/ZhsEKTo7V04

Reinforcement Learning





$$r(s, a) = E[R_{t+1} \mid S_t = s, A_t = a]$$



Policy and Value Networks

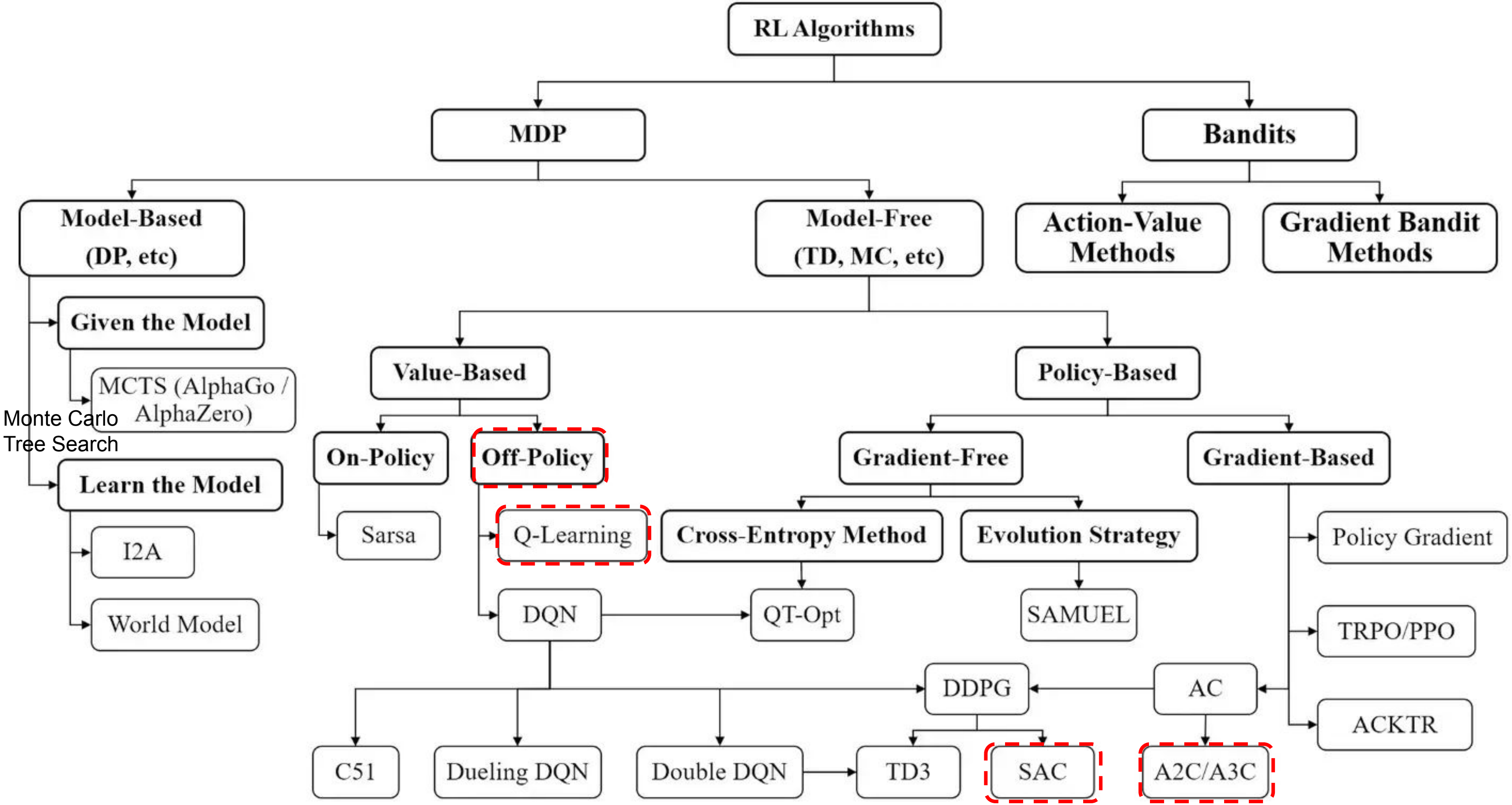
$$V^*(x_t) = \max_\pi V^\pi(x_t)$$

$$Q^*(x_t, u_t) = \max_\pi Q^\pi(x_t, u_t)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_{t=0}^{T} \gamma^t r(x_t, u_t)\right]$$

https://opendatascience.com/deep-learning-research-review-week-2-reinforcement-learning/

RL Algorithms Overview



https://adabhishekdabas.medium.com/rl-world-3fc4dc38a73d

RL Notations

$$P(Y|X) = \frac{P(Y \cap X)}{P(X)}$$

$$\mathbb{E}[X] = \sum_i p(x_i)x_i$$

$$\mathbb{E}[Y|X = x] = \sum_i p(Y = y_i|X = x)y_i$$

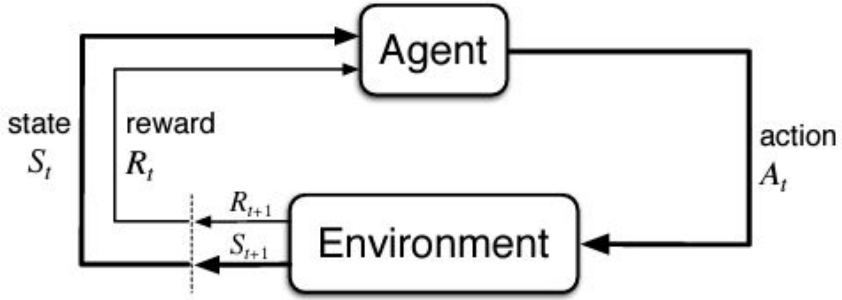
1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

Symbol	Meaning
$s \in \mathcal{S}$	States.
$a \in \mathcal{A}$	Actions.
$r \in \mathcal{R}$	Rewards.
S_t, A_t, R_t	State, action, and reward at time step t of one trajectory. I may occasionally use s_t, a_t, r_t as well.
γ	Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.
G_t	Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
$P(s', r\|s, a)$	Transition probability of getting to the next state s' from the current state s with action a and reward r.
$\pi(a\|s)$	Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by θ.
$\mu(s)$	Deterministic policy; we can also label this as $\pi(s)$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either π or μ is what a reinforcement learning algorithm aims to learn.
$V(s)$	State-value function measures the expected return of state s; $V_w(.)$ is a value function parameterized by w.
$V^\pi(s)$	The value of state s when we follow a policy π; $V^\pi(s) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s]$.
$Q(s, a)$	Action-value function is similar to $V(s)$, but it assesses the expected return of a pair of state and action (s, a); $Q_w(.)$ is a action value function parameterized by w.
$Q^\pi(s, a)$	Similar to $V^\pi(.)$, the value of (state, action) pair when we follow a policy π; $Q^\pi(s, a) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s, A_t = a]$.
$A(s, a)$	Advantage function, $A(s, a) = Q(s, a) - V(s)$; it can be considered as another version of Q-value with lower variance by taking the state-value off as the baseline.

Markov Decision(Reward) Process

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \ldots, s_0)$$

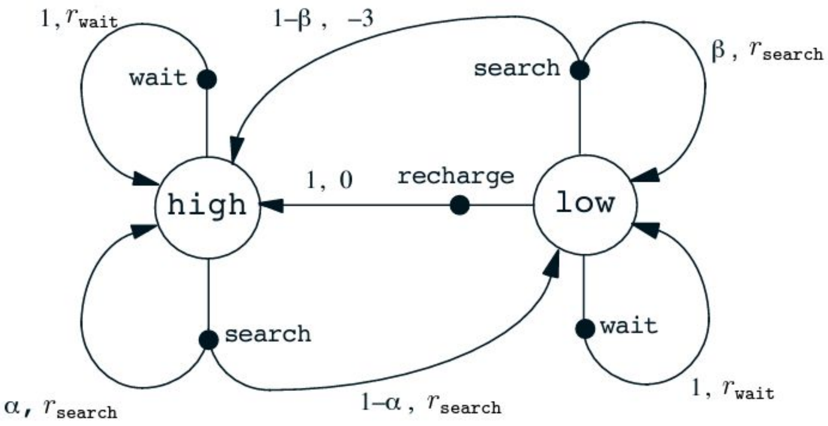


The agent–environment interaction in a Markov decision process.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$



Recycling robot example.

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0.

Bellman (Optimal) Equation

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

$$V(s_t) \boxed{= \max_{a_t}} \left(R(a_t, s_t) + \gamma V(s_{t+1}) \right)$$

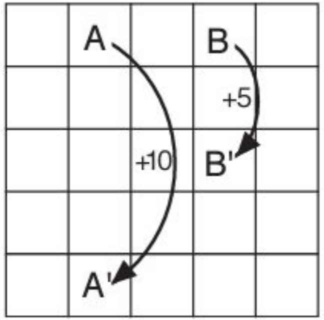
optimal

$$v_*(s) \doteq \max_{\pi} v_\pi(s),$$

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma v_*(s') \big]. \end{aligned}$$

$$q_*(s,a) \doteq \boxed{\max_{\pi}} q_\pi(s,a),$$

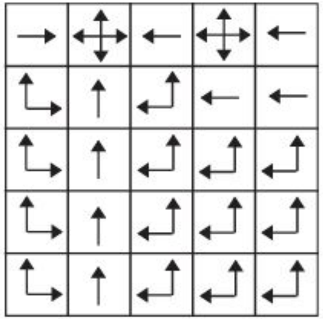
$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \Bigm| S_t = s, A_t = a \Big] \\ &= \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big]. \end{aligned}$$



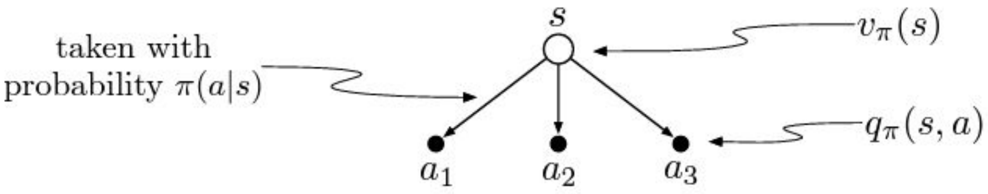
Gridworld

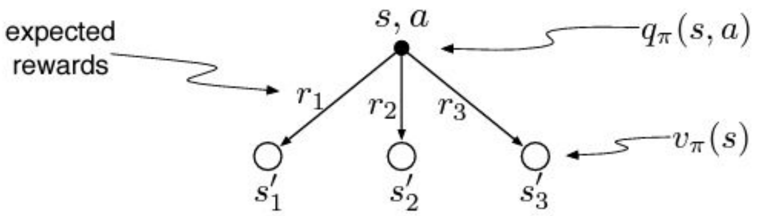
22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*

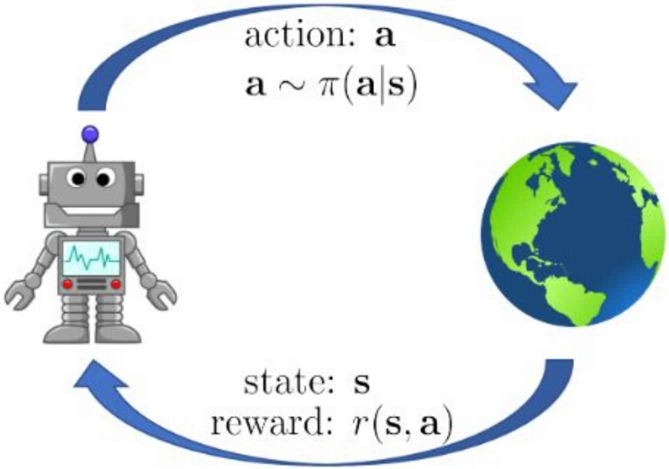


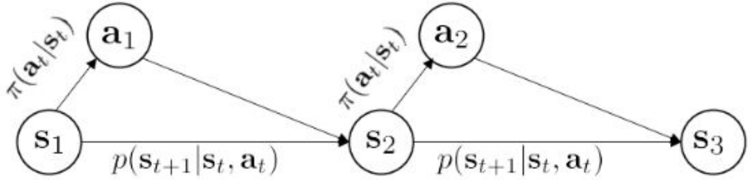
π_*





Policies and Objectives





could be ∞

discount factor

$$\text{RL objective:} \quad \max_\pi \sum_{t=0}^{T} E_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})} [\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$$

state distribution under π

some definitions:

$\mathbf{s} \in \mathcal{S}$ – discrete or continuous state

$\mathbf{a} \in \mathcal{A}$ – discrete or continuous action

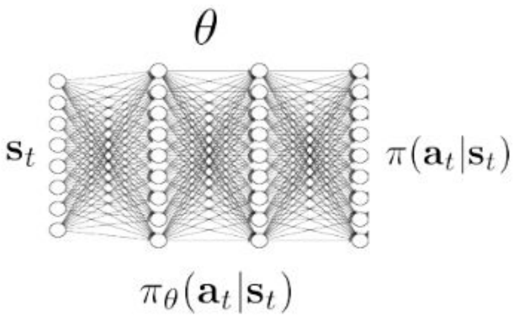
$\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T, \mathbf{a}_T\}$ – *trajectory*

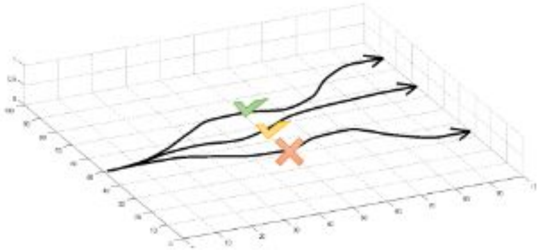
$$\underbrace{\pi(\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi(\tau)} = p(\mathbf{s}_1) \prod_{t=0}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

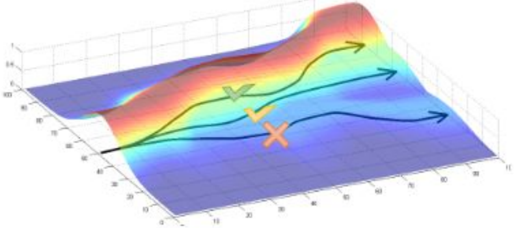
$d_t^\pi(\mathbf{s}_t)$ – state marginal of $\pi(\tau)$ at t

$d^\pi(\mathbf{s}) = \frac{1}{1-\gamma} \sum_{t=0}^{T} \gamma^t d_t^\pi(\mathbf{s}_t)$ – "visitation frequency"

Policy Gradient in Practice



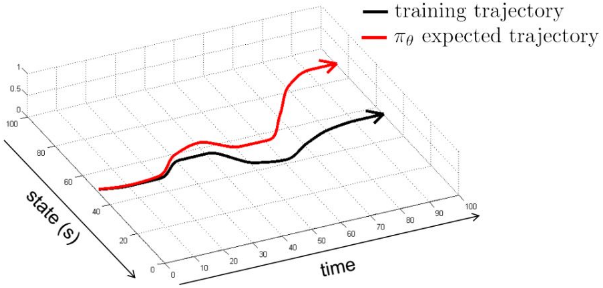




RL objective: $\max_\pi \sum_{t=0}^{T} E_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})}[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$

exactly the same thing!

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s)]
\end{aligned}$$

On-policy vs Off-Policy

On-policy : a single policy, require observations(state, action, reward, next state,) to generate that policy.
Off-policy : two policies target policy and policy generates the observations(called the behaviour policy).

	Value Based	Policy Based	Actor-Critic
On-Policy	• Monte Carlo Learning (MC) • TD(0) • SARSA • Expected SARSA • n-Step TD/SARSA • TD(λ)	• REINFORCE • REINFORCE with Advantage	• A3C • A2C • TRPO • PPO
Off-Policy	• Q-Learning • DQN • Double DQN • Dueling DQN		• DDPG • TD3 • SAC • IMPALA

SARSA $$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot (r_s + \gamma \cdot \boxed{Q(a',s')} - Q(a,s))$$

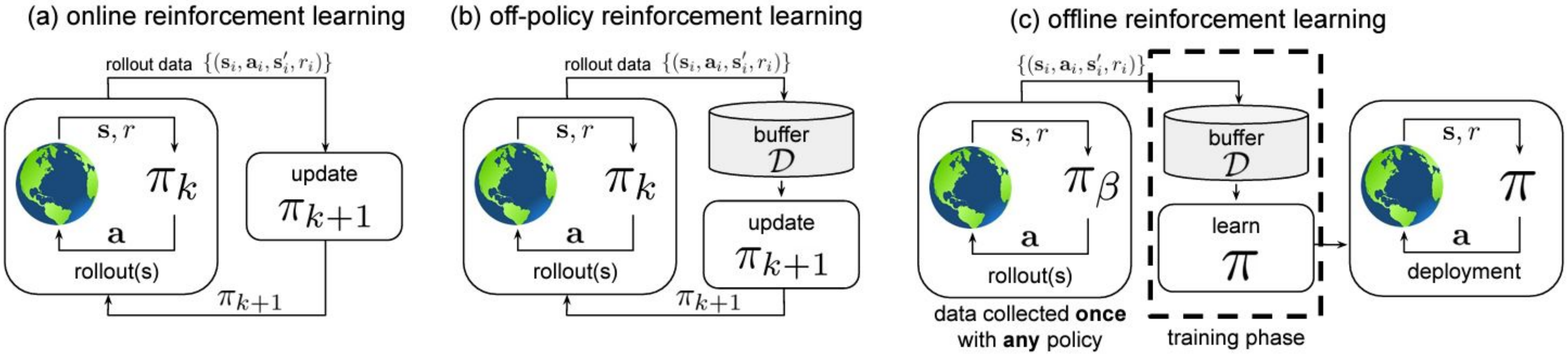
the action(a') was taken on policy

Q-Learning $$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \boxed{\max_{a'} Q(a',s')} - Q(a,s)\right)$$

all actions(a') were probed in state

https://data-newbie.tistory.com/543

What does offline RL mean?



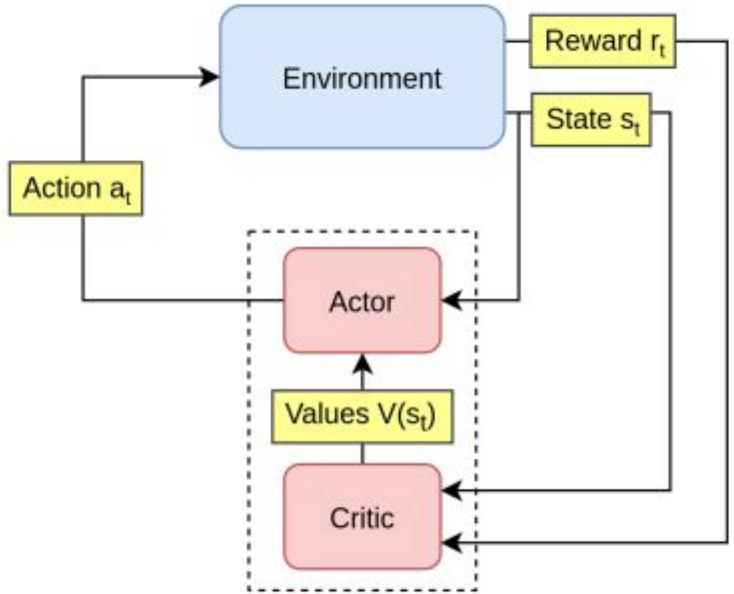
Online RL : Agent collects data each time it is trained.(modified), either uses narrow datasets (e.g., collected in one environment) or or manually designed simulators (using its own (partially trained) policy). Generalization can be poor due to small, narrow datasets, or simulators that differ from reality

Off-policy RL : Old data is retained, and new data is still collected periodically as the policy changes.

Offline RL : RL algorithms that can learn from prior data (the data is collected once - supervised learning) and is then used to train optimal policies without any additional online data collection. the policy is deployed to collect additional data to improve online. utilize large and diverse datasets only practical to collect once

The scale of ImageNet or MS-COCO, which capture a wide slice of real-world situations.

Advantage Actor Critic (A2C) algorithm



Actor-critic methods consist of two models, which may optionally share parameters:

Critic updates the value function parameters w and depending on the algorithm it could be action-value Q(a|s) or state-value V(s).

Actor updates the policy parameters θ for piθ(a|s) in the direction suggested by the critic.

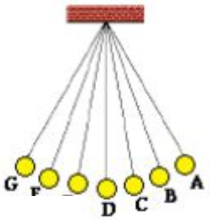
Asynchronous Advantage Actor-Critic (Mnih et al., 2016), short for A3C, is a classic policy gradient method with a special focus on parallel training.

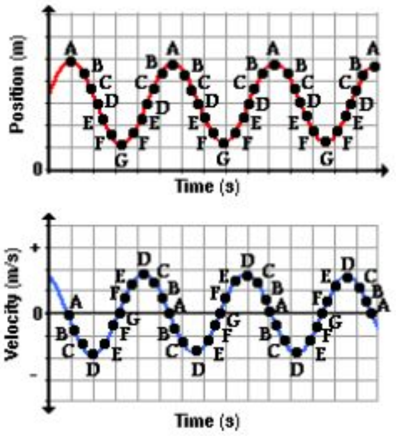
$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \qquad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right]$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad \text{(Advantage function).}$$

Open AI Gym (Pendulum)

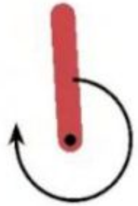
Action Space	Box(-2.0, 2.0, (1,), float32)
Observation Shape	(3,)
Observation High	[1. 1. 8.]
Observation Low	[-1. -1. -8.]
Import	`gym.make("Pendulum-v1")`





Action Space

Num	Action	Min	Max
0	Torque	-2.0	2.0



Observation Space

Num	Observation	Min	Max
0	x = cos(theta)	-1.0	1.0
1	y = sin(angle)	-1.0	1.0
2	Angular Velocity	-8.0	8.0

Position D is zero position

Rewards :

$$r = -(theta^2 + 0.1 * theta\_dt^2 + 0.001 * torque^2)$$

Theta is normalized between -pi and pi. Therefore, the lowest cost is -(pi^2 + 0.18^2 + 0.0012^2) = -16.2736044, and the highest cost is 0. In essence, the goal is to remain at zero angle (vertical), with the least rotational velocity, and the least effort.

https://ai-mrkogao.github.io/openai/pendulum/

Actor-Critic Code

```python
## 액터 신경망
class Actor(Model):

    def __init__(self, action_dim, action_bound):
        super(Actor, self).__init__()
        self.action_bound = action_bound

        self.h1 = Dense(64, activation='relu')
        self.h2 = Dense(32, activation='relu')
        self.h3 = Dense(16, activation='relu')
        self.mu = Dense(action_dim, activation='tanh')
        self.std = Dense(action_dim, activation='softplus')

    def call(self, state):
        x = self.h1(state)
        x = self.h2(x)
        x = self.h3(x)
        mu = self.mu(x)
        std = self.std(x)

        # 평균값을 [-action_bound, action_bound] 범위로 조정
        mu = Lambda(lambda x: x*self.action_bound)(mu)

        return [mu, std]
```

```python
## 크리틱 신경망
class Critic(Model):

    def __init__(self):
        super(Critic, self).__init__()

        self.h1 = Dense(64, activation='relu')
        self.h2 = Dense(32, activation='relu')
        self.h3 = Dense(16, activation='relu')
        self.v = Dense(1, activation='linear')

    def call(self, state):
        x = self.h1(state)
        x = self.h2(x)
        x = self.h3(x)
        v = self.v(x)
        return v
```

Actor-Critic Code

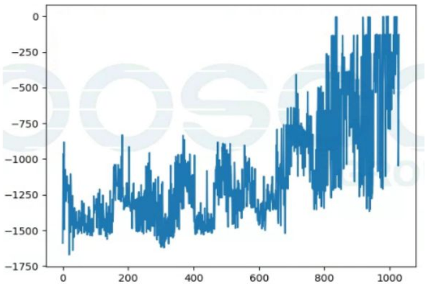
```python
## 액터 신경망에서 행동 샘플링
def get_action(self, state):
    mu_a, std_a = self.actor(state)
    mu_a = mu_a.numpy()[0]
    std_a = std_a.numpy()[0]
    std_a = np.clip(std_a, self.std_bound[0], self.std_bound[1])
    action = np.random.normal(mu_a, std_a, size=self.action_dim)
    return action
```

```python
## 액터 신경망 학습
def actor_learn(self, states, actions, advantages):
    with tf.GradientTape() as tape:
        # 정책 확률밀도함수
        mu_a, std_a = self.actor(states, training=True)
        log_policy_pdf = self.log_pdf(mu_a, std_a, actions)
        # 손실함수
        loss_policy = log_policy_pdf * advantages
        loss = tf.reduce_sum(-loss_policy)

    # 그래디언트
    grads = tape.gradient(loss, self.actor.trainable_variables)
    self.actor_opt.apply_gradients(zip(grads, self.actor.trainable_variables))
```

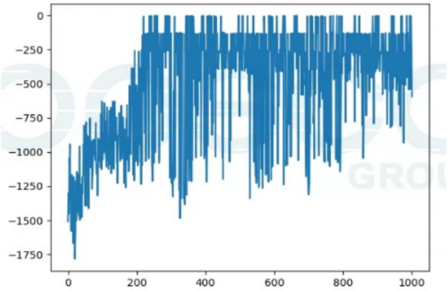
```python
## 크리틱 신경망 학습
def critic_learn(self, states, td_targets):
    with tf.GradientTape() as tape:
        td_hat = self.critic(states, training=True)
        loss = tf.reduce_mean(tf.square(td_targets-td_hat))
```

```python
# 시간차 타깃 계산
next_v_values = self.critic(tf.convert_to_tensor(next_states, dtype=tf.float32))
td_targets = self.td_target(train_rewards, next_v_values.numpy(), dones)

# 크리틱 신경망 업데이트
self.critic_learn(tf.convert_to_tensor(states, dtype=tf.float32),
                  tf.convert_to_tensor(td_targets, dtype=tf.float32))

# 어드밴티지 계산
v_values = self.critic(tf.convert_to_tensor(states, dtype=tf.float32))
next_v_values = self.critic(tf.convert_to_tensor(next_states, dtype=tf.float32))
advantages = train_rewards + self.GAMMA * next_v_values - v_values

# 액터 신경망 업데이트
self.actor_learn(tf.convert_to_tensor(states, dtype=tf.float32),
                 tf.convert_to_tensor(actions, dtype=tf.float32),
                 tf.convert_to_tensor(advantages, dtype=tf.float32))

# 상태 업데이트
state = next_state[0]
episode_reward += reward[0]
time += 1
```

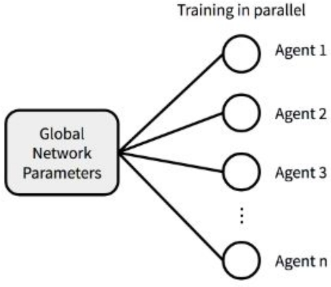
Training Asynchronous Advantage Actor-Critic (A3C)



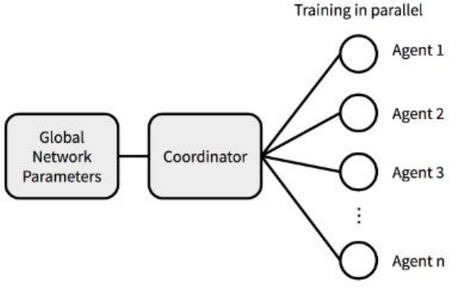
A3C : Worker 1
(MLP : Sharing Parameter)



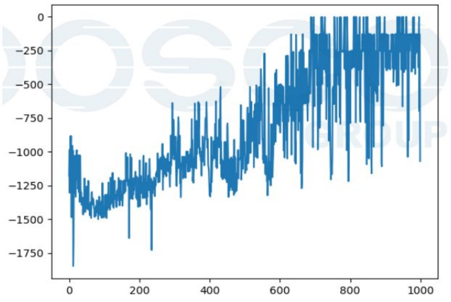
A3C : Worker 8
(MLP : 64,32,16)



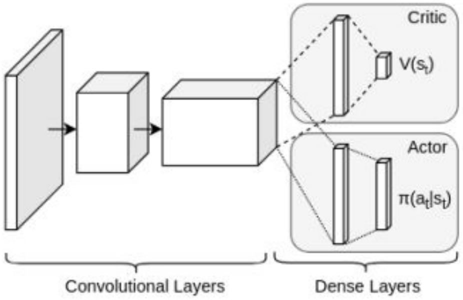
A3C (Async)

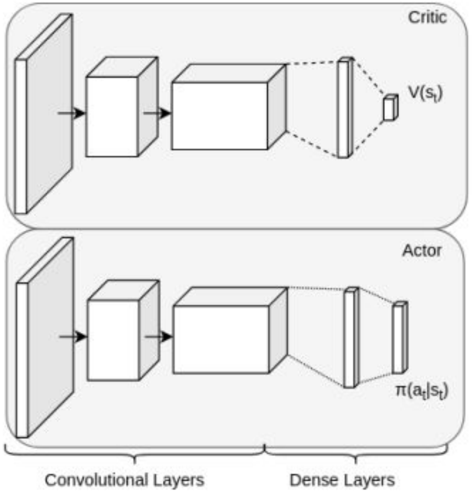


A2C (Sync)



A3C : Worker 1
(MLP : Not Sharing Parameter)





Soft Actor Critic(SAC)

SAC learns two Q-networks, a V-network, and a policy network. Two Q-networks are used to mitigate overestimation bias.
A V-network is used to stabilize training. Taking gradients through the expectations is done using the reparameterization trick
Off-Policy(DDPG: ICLR 2016)+Soft Bellman(Soft Q-Learning: ICML 2017) + Stable Actor-Critic(TD3:ICML 2018)

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

$$u_i^j \sim \mathcal{N}(\mu_\theta(x_i),\ \sigma_\theta^2(x_i))$$

$$u_i^j = f_\theta(x_i,\ \eta_j) = \mu_\theta(x_i) + \sigma_\theta(x_i)\eta_j,\ \ \eta_j \sim \mathcal{N}(0,\ I)$$

$$L_\pi(\theta) = \mathbb{E}_{x_i \sim \mathcal{D}}[\mathbb{E}_{\eta \sim \mathcal{N}}[\alpha \log \pi_\theta(u_i|x_i) - Q_\phi(x_i,\ u_i)] \mid x_i]$$

References

Offline RL Tutorial - NeurIPS 2020 : https://sites.google.com/view/offlinerltutorial-neurips2020/home

A3C Code : GitHub - pasus/Reinforcement-Learning-Book-Revision

Pendulum https://www.gymlibrary.dev/environments/classic_control/pendulum/

Reinforcement Learning: An Introduction : http://incompleteideas.net/book/bookdraft2017nov5.pdf

Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey https://arxiv.org/abs/2108.11510

Policy Gradient Algorithms https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

Decisions from Data: How Offline Reinforcement Learning Will Change How We Use Machine Learning
https://medium.com/@sergey.levine/decisions-from-data-how-offline-reinforcement-learning-will-change-how-we-use-ml-24d98cb069b0

Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." International conference on machine learning. PMLR, 2018.

Thanks
Any Questions?

You can send mail to
Susang Kim(healess1@gmail.com)