MDP and Reinforcement Learning

Large state spaces and approximations

Nicolas Gast

October 23, 2023

Reminder: Tabular MDP

We want to find $Q(s, a) \approx Q^*(s, a)$.

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

Two types of methods:

- MC methods:

$$Q^\pi(s, a) = \frac{1}{K} \sum_{k=1}^{K} G^{(k)}$$

- TD methods (SARSA / Q-learning)

Reminder: Tabular MDP

We want to find $Q(s, a) \approx Q^*(s, a)$.

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

Two types of methods:

- MC methods:

$$Q^\pi(s, a) = \frac{1}{K} \sum_{k=1}^{K} G^{(k)}$$

- TD methods (SARSA / Q-learning)

Does it scale?

The complexity is $\Omega(|\mathcal{S}||\mathcal{A}|)$.

$Q(s,a)$	a_1	a_2	a_3	$\dots$
s_1				
s_2				
s_3				
s_4				
$\vdots$				

What are typical state space sizes?

The curse of dimensionality

Managing a portfolio of 10 types of product, with 100 product each max.

- $|\mathcal{S}| = 100^{10} = 10^{20}$.
- $\mathcal{A}$ = possible orders (=10×100?)

What are typical state space sizes?

The curse of dimensionality

Managing a portfolio of 10 types of product, with 100 product each max.

- $|\mathcal{S}| = 100^{10} = 10^{20}$.
- $\mathcal{A}$ = possible orders (=10×100?)

Game of go

- $|\mathcal{S}| = 3^{19\times 19}$ (19×19 board game).
- $|\mathcal{A}| = 19 \times 19$.

There are $\approx 10^{170}$ Q-values.

What are typical state space sizes?

The curse of dimensionality

Breakout (1976) ▸ Atari games

- $|\mathcal{S}| = 8^{84\times 84}$ (84×84 screen, 8 colors).
- $|\mathcal{A}| = 2$ (left, right).

There are $\approx 10^{2000}$ Q-values.

What are typical state space sizes?

The curse of dimensionality

Breakout (1976) ▸ Atari games

- $|\mathcal{S}| = 8^{84\times 84}$ (84×84 screen, 8 colors).
- $|\mathcal{A}| = 2$ (left, right).

There are $\approx 10^{2000}$ Q-values.

Starcraft ▸ alphastar

- $|\mathcal{S}| \gg |\mathcal{A}| \approx +\infty$??

We need approximations.

Outline

TD-learning and function approximation

The tabular TD-learning or Q-learning algorithm is:

$$V(S_t) := V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right).$$

This does not scale if $|\mathcal{S}|$ (or $|\mathcal{A}|$) are large.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

$$Q(S, A) \approx q_{\mathrm{w}}(S, A),$$

where w is a vector parameter to be found.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

$$Q(S,A) \approx q_{\mathrm{w}}(S,A),$$

where w is a vector parameter to be found.

- (classic): Use a linear approximation. For instance:

  $$Q(S,A) = \mathrm{w}^T \phi(s,a),$$

  where $\phi(s,a)$ is a feature vector.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

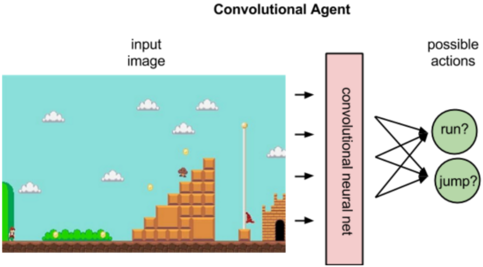
$$Q(S, A) \approx q_{\mathrm{w}}(S, A),$$

where w is a vector parameter to be found.

- (classic): Use a linear approximation. For instance:

  $$Q(S, A) = \mathrm{w}^T \phi(s, a),$$

  where $\phi(s, a)$ is a feature vector.
- ("modern"): q_{w} is a deep neural network.



From Q-learning to deep Q-learning

The original Q-learning uses that:

$$Q(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)\right].$$

We want to find w such that $\underbrace{q_{\mathrm{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathrm{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

From Q-learning to deep Q-learning

The original Q-learning uses that:

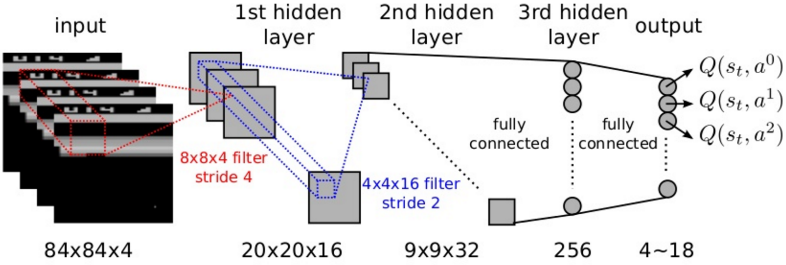
$$Q(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)\right].$$

We want to find w such that $\underbrace{q_{\mathrm{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathrm{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

Deep Q-learning minimizes the L_2 norm and use gradient descent:

$$\mathrm{w} := \mathrm{w} + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathrm{w}}(S_t, a) - q_{\mathrm{w}}(S_t, A_t)\right) \nabla_{\mathrm{w}}(q_{\mathrm{w}}(S_t, A_t)).$$

Example of breakout



Why is vanilla unstable?

We want to find w such that $\underbrace{q_{\mathrm{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a\in\mathcal{A}} q_{\mathrm{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

For that, we do:

$$\mathrm{w} := \mathrm{w} + \alpha \left(R_{t+1} + \gamma \max_{a\in\mathcal{A}} q_{\mathrm{w}}(S_t, a) - q_{\mathrm{w}}(S_t, A_t)\right) \nabla_{\mathrm{w}}(q_{\mathrm{w}}(S_t, A_t)).$$

Problems:

Why is vanilla unstable?

We want to find w such that $\underbrace{q_{\mathrm{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a\in\mathcal{A}} q_{\mathrm{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

For that, we do:

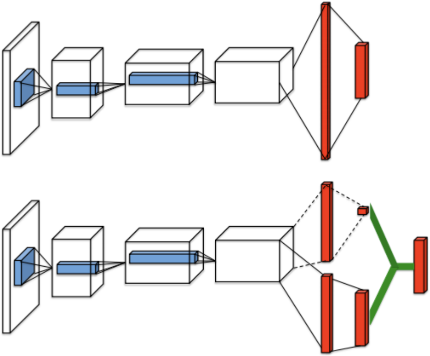
$$\mathrm{w} := \mathrm{w} + \alpha\left(R_{t+1} + \gamma \max_{a\in\mathcal{A}} q_{\mathrm{w}}(S_t, a) - q_{\mathrm{w}}(S_t, A_t)\right)\nabla_{\mathrm{w}}(q_{\mathrm{w}}(S_t, A_t)).$$

Problems:

- Target and sources are highly correlated
- Target changes as we learn.
- Exploration is not guaranteed.

Learning algorithm can be unstable.

Possible solution: replay buffer or separate target network



Vanilla Q-learning uses a single network

DDQN uses a slow learning target network and a fast learning q-network.

Applications of Deep RL

- Resource management (energy)
- Computer vision and robotics
- Finance
- ...

Fundamental idea is simple but making the system stable and fast is an issue. Also, delayed actions or sparse rewards is difficult.

Outline

Policy search

We are given a family of policies π_{w} parametrized by $\mathrm{w} \in \mathcal{W}$. Typically:

$$\pi_{\mathrm{w}}(a \mid s) \propto \exp(\mathrm{w}^T \phi(s, a)),$$

where $\phi(s, a)$ is a feature vector.

Policy search

We are given a family of policies π_{w} parametrized by $\mathrm{w} \in \mathcal{W}$. Typically:

$$\pi_{\mathrm{w}}(a \mid s) \propto \exp(\mathrm{w}^T \phi(s, a)),$$

where $\phi(s, a)$ is a feature vector.

Let $J(\mathrm{w}) := V^{\pi_{\mathrm{w}}}(s_0)$ be its performance. We want to find w that maximizes $J(\mathrm{w})$.

Policy search

We are given a family of policies π_w parametrized by $w \in \mathcal{W}$. Typically:

$$\pi_w(a \mid s) \propto \exp(w^T \phi(s, a)),$$

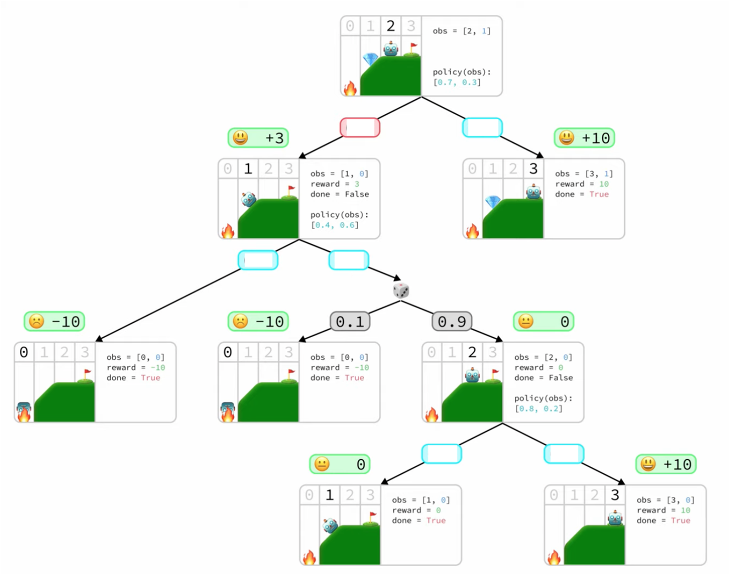
where $\phi(s, a)$ is a feature vector.

Let $J(w) := V^{\pi_w}(s_0)$ be its performance. We want to find w that maximizes $J(w)$.

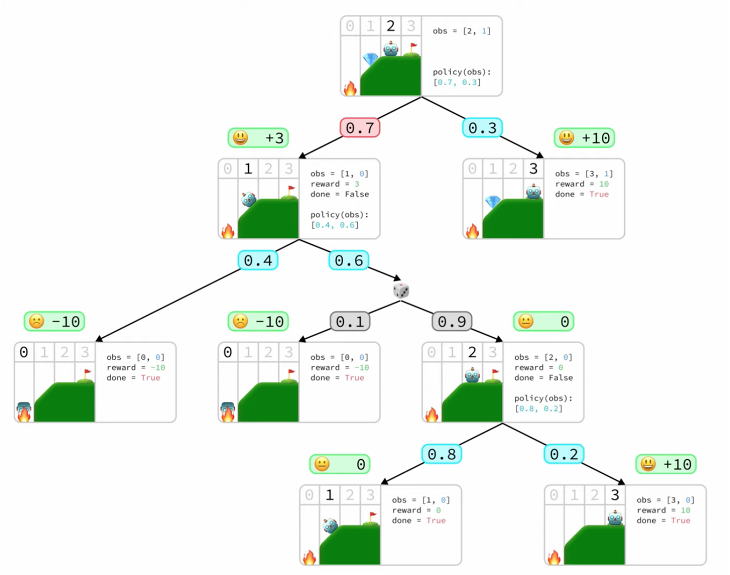
- Sometimes, this works well with direct methods (brute-force)
- We can also use policy gradients:

$$w := w + \alpha \nabla_w J(w).$$

On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

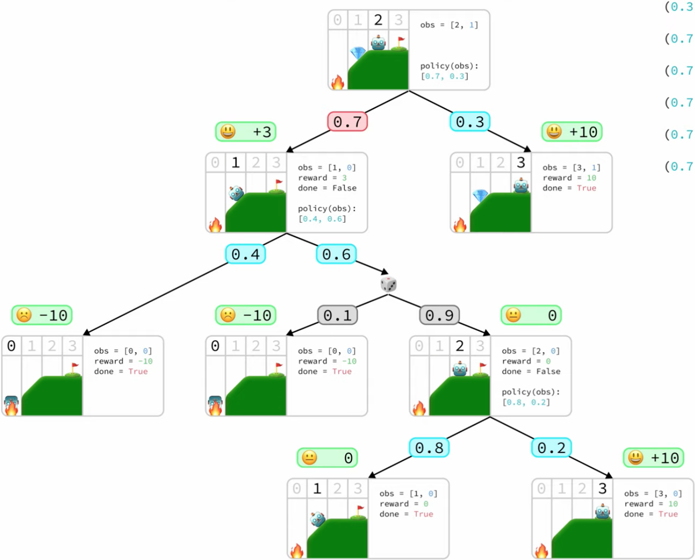


On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

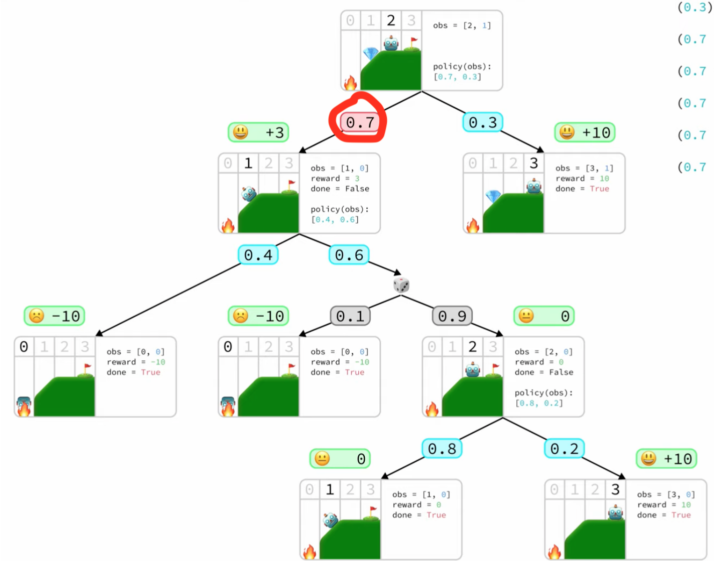


On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

```
Expected Return (G) =

    (0.7) * (3) +

    (0.3) * (10) +

    (0.7 * 0.4) * (-10) +

    (0.7 * 0.6 * 0.1) * (-10) +

    (0.7 * 0.6 * 0.9) * (0) +

    (0.7 * 0.6 * 0.9 * 0.8) * (0) +

    (0.7 * 0.6 * 0.9 * 0.2) * (10)
```



On an example https://www.youtube.com/watch?v=cQfOQcpYRzE



Expected Return (G) =

$$(0.7) \star (3) +$$
$$(0.3) \star (10) +$$
$$(0.7 \star 0.4) \star (-10) +$$
$$(0.7 \star 0.6 \star 0.1) \star (-10) +$$
$$(0.7 \star 0.6 \star 0.9) \star (0) +$$
$$(0.7 \star 0.6 \star 0.9 \star 0.8) \star (0) +$$
$$(0.7 \star 0.6 \star 0.9 \star 0.2) \star (10)$$

On an example https://www.youtube.com/watch?v=cQf0QcpYRzE

Expected Return (G) =

(0.7) * (3) +

(0.3) * (10) +

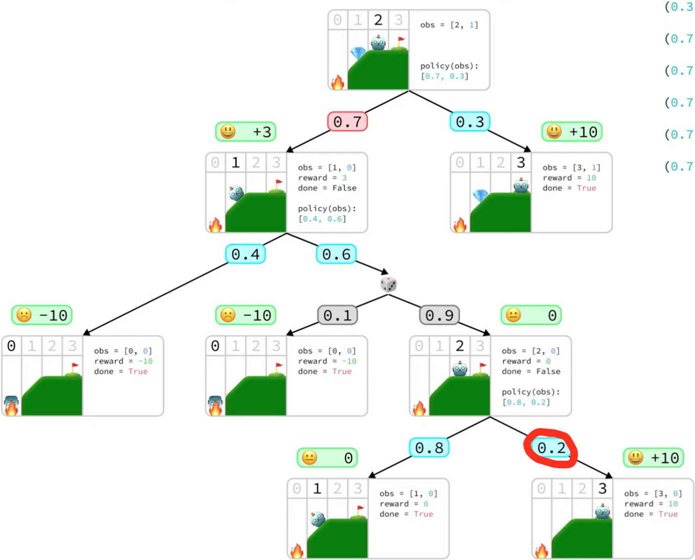
(0.7 * 0.4) * (-10) +

(0.7 * 0.6 * 0.1) * (-10) +

(0.7 * 0.6 * 0.9) * (0) +

(0.7 * 0.6 * 0.9 * 0.8) * (0) +

(0.7 * 0.6 * 0.9 * 0.2) * (10)



How to estimate the gradient with trajectories?

Assume for simplicity that each state is visited only once.
The probability of choosing a in state s is $\pi(a|s)$.

$$\nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right] = \mathbb{P}(\text{attaining } s)Q(s,a)$$
$$= \frac{1}{\pi(a|s)}\mathbb{P}(\text{observing } (s,a))Q(s,a)$$

How to estimate the gradient with trajectories?

Assume for simplicity that each state is visited only once.
The probability of choosing a in state s is $\pi(a|s)$.

$$\begin{aligned}\nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right] &= \mathbb{P}(\text{attaining } s)Q(s,a)\\ &= \frac{1}{\pi(a|s)}\mathbb{P}(\text{observing } (s,a))Q(s,a)\end{aligned}$$

Algorithm: We want to compute $\text{gradient}(S,A) = \nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right]$.

- Run a trajectory and observe S_t, A_t.
- For each t:

$$\widehat{gradient}(S_t, A_t) = \frac{1}{\pi(A_t|S_t)}G_t.$$

Theorem. For all s, a: $\mathbb{E}\left[\widehat{gradient}(s,a)\right] = \nabla_{\pi(a|s)}\mathbb{E}\left[G\right]$.

The policy gradient theorem

Assume that $\pi(a|s) = f_w(s,a)$. We have:

$$\nabla_{\mathrm{w}} \mathbb{E}\left[G_0\right] = \sum_{s,a} \nabla_w \pi(a|s) \nabla_{\pi(a|s)} \mathbb{E}\left[G_0\right]$$

The policy gradient theorem

Assume that $\pi(a|s) = f_w(s,a)$. We have:

$$\nabla_{\mathrm{w}}\mathbb{E}\left[G_0\right] = \sum_{s,a} \nabla_w \pi(a|s) \nabla_{\pi(a|s)} \mathbb{E}\left[G_0\right]$$

Hence, an unbiased estimate of the gradient $\nabla_{\mathrm{w}}\mathbb{E}\left[G_0\right]$ is

$$\sum_t \frac{(\nabla_w \pi(A_t|S_t))}{\pi(A_t|S_t)} G_t.$$

By using that $\nabla log(y) = \nabla(y)/y$, we get:

An unbiased estimate of the gradient is:

$$\nabla_{\mathrm{w}}\mathbb{E}\left[G_0\right] = \mathbb{E}\left[\sum_t (\nabla_w \log \pi(A_t|S_t)) G_t\right].$$

Why is $\nabla \log \pi(a|s)$ easy to compute?

Reminder: if $p_i = e^{u_i} / \sum e^{u_j}$, then

$$\frac{\partial}{\partial u_j} \log p_i = 1_{\{i=j\}} - p_j.$$

Why is $\nabla \log \pi(a|s)$ easy to compute?

Reminder: if $p_i = e^{u_i} / \sum e^{u_j}$, then

$$\frac{\partial}{\partial u_j} \log p_i = 1_{\{i=j\}} - p_j.$$

If $\pi(a|s) \propto \exp(w^T \phi(s, a))$, then it means that $\pi(a|s) = \frac{\exp(w^T \phi(s,a))}{\sum_{a'} \exp(w^T \phi(s,a'))}$.

As a consequence:

$$\nabla_w \pi_w(a|s) = \phi(a, s) - \sum_{a'} \phi(a'|s) \pi_w(a'|s).$$

The REINFORCE algorithm

REINFORCE

Initialize w.
while True **do**
  Simulate a trajectory (from $t = 1$ to T)
  for $t = T$ to $t = 1$ **do**
    $G_t := \sum_{t'=t}^{T} R_{t'}$.
    $\nabla J := G_t \nabla \log \pi(A_t | S_t)$.
    $\mathrm{w} := \mathrm{w} + \alpha \nabla J$.
  end for
end while

Recall that $\nabla \log \pi(a|s)$ is easy to compute when $\pi(a|s) \propto w^T \phi(s, a)$.

Variance reduction

Problem: Monte-Carlo sampling can have a large variance.
Ex: if $Q(s, a_1) = 8 \pm 1$ and $Q(s, a_2) = 8.5 \pm 1$, is a_2 better than a_1?

Variance reduction

Problem: Monte-Carlo sampling can have a large variance.
Ex: if $Q(s, a_1) = 8 \pm 1$ and $Q(s, a_2) = 8.5 \pm 1$, is a_2 better than a_1?

Solution: add a baseline $h : \mathcal{S} \to \mathbb{R}$. Indeed, using the same log-trick:

$$\begin{aligned} \mathbb{E}\left[h(s_t)\nabla \log \pi(a_t|s_t)\right] &= \mathbb{E}\left[\sum_{a\in\mathcal{A}} h(s_t)\nabla\pi(a|s_t)\right] \\ &= 0 \end{aligned}$$

This shows that for any function h, one has:

$$\nabla_{\mathrm{w}} J(s_0) \propto \sum_t \mathbb{E}\left[(G_t - h(s_t))\nabla \log \pi(a_t|s_t)\right]\}.$$

Choosing a h close to G_t reduces the variance of the estimator.

Outline

Classes of learning algorithms

We have seen two classes of RL methods:

- Value-based (SARSA, Q-learning, Deep QL)
- Policy-based (Policy gradient, REINFORCE)

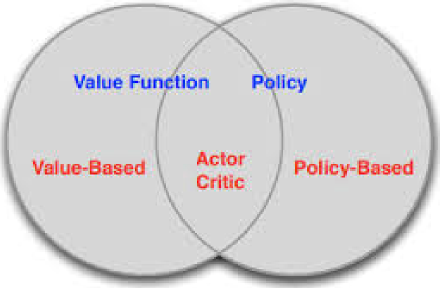
- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.

Classes of learning algorithms

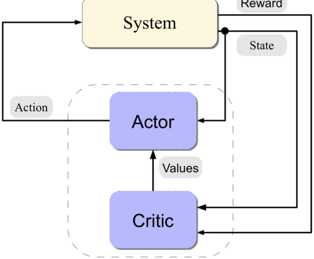
We have seen two classes of RL methods:

- Value-based (SARSA, Q-learning, Deep QL) =Critic
- Policy-based (Policy gradient, REINFORCE) =Actor

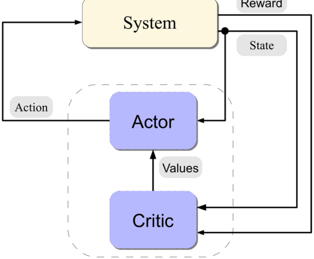
- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.



Actor Critic method



Actor Critic method



Basic Actor Critic

Initialize parameters $\mathrm{w}^{(a)}$ (Actor) and $\mathrm{w}^{(c)}$ (Critic)
while True **do**
  Initialize S
  for $t = 1$ to $t = T$ **do**
    $A_t \sim \pi_{\mathrm{w}}(S)$ and simulate R, S'
    $\mathrm{w}^{(c)} := \mathrm{w}^{(c)} + \alpha^{(c)}(R + \gamma v_{\mathrm{w}^{(c)}}(S') - v_{\mathrm{w}^{(c)}}(S))$ *# TD-update*
    $\mathrm{w}^{(a)} := \mathrm{w}^{(a)} + \alpha^{(a)} v_{\mathrm{w}^{(c)}}(S) \nabla \log \pi(a_t | s_t)$ *# Policy-gradient*
    S:=S'.
  end for
end while

Going further

Extra-reading:

- Introduction to Reinforcement Learning (Sutton-Barto, 2018 last ed.)
- Algorithms for Reinforcement Learning (Szepesvari, 2010)
- Deep Reinforcement learning: hands on (Maxim Lapan, 2020)