TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2023

Reinforcement Learning

Definition of Reinforcement Learning

RL is defined by the following properties:

- An environment with **state**.
- State changes are influenced by **sequential decisions**.
- Reward (or loss) depends on **making decisions** that lead to **desirable states**.

Reinforcement Learning Examples

- Board games (chess or go)

- Atari Games (pong)

- Robot control (driving)

- Dialog

- Life

Policies

A policy is a way of behaving.

Formally, a (nondeterministic) policy maps a state to a probability distribution over actions.

$\pi(a_t|s_t)$ probability of action a_t in state s_t

Imitation Learning

Construct a training set of state-action pairs (s, a) from experts.

Define stochastic policy $\pi_\Phi(s)$.

$$\Phi^* = \operatorname*{argmin}_\Phi E_{(s,a)\sim \mathrm{Train}} \; - \ln \pi_\Phi(a \mid s)$$

This is just cross-entropy loss where we think of a as a “label” for s.

Dangers of Imperfect Imitation Learning

Perfect imitation learning would reproduce expert behavior. Imitation learning is **off-policy** — the state distribution in the training data is different from that defined by the policy being learned.

Imitating experts, such as expert fire eaters, can be dangersous. “Don’t try this at home”.

Also, it is difficult to exceed expert performance by imitating experts.

Markov Decision Processes (MDPs)

An MDP consists of a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of allowed actions, a reward function R and a next-state probability function P_T. We will use the following notation.

$s_t \in \mathcal{S}$ is the state at time t

$a_t \in \mathcal{A}$ is the action taken at time t.

$r_t = R(s_t, a_t) \in \mathbb{R}$ is the reward at time t

$P_T(s_{t+1}|s_t, a_t)$ is the probability of s_{t+1} given s_t and a_t.

The function $R(s, a)$ can allow for a cost of the action a.

Optimizing Reward

In RL we maximize reward rather than minimize loss.

$$\pi^* = \underset{\pi}{\mathrm{argmax}}\ R(\pi)$$

$$\begin{array}{lll} R(\pi) = & E_\pi\ \sum_{t=0}^{T} r_t & \text{episodic reward (go)} \\ \text{or} & E_\pi\ \sum_{t=0}^{\infty} \gamma^t r_t & \text{discounted reward (financial planning)} \\ \text{or} & \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} r_t & \text{asymptotic average reward (driving)} \end{array}$$

The Value Function

For discounted reward:

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi} \sum_{t} \gamma^{t} r_{t} \ \mid \pi,\ s_0 = s \\
V^{*}(s) &= \sup_{\pi} \ V^{\pi}(s) \\
\pi^{*}(a|s) &= \operatorname*{argmax}_{a} \ R(s,a) + \gamma E_{s' \sim P_T(s'|s,a)} \, V^{*}(s') \\
V^{*}(s) &= \max_{a} \ R(s,a) + \gamma E_{s' \sim P_T(s'|s,a)} \, V^{*}(s')
\end{aligned}
$$

Value Iteration

Suppose the state space and action space are finite.

In that case we can do value iteration.

$$V_0(s) = 0$$

$$V_{i+1}(s) = \max_a \; R(s,a) + \gamma E_{s' \sim P_T(\cdot|s,a)} \; V_i(s')$$

If all rewards are non-negative then

$$V_{i+1}(s) \geq V_i(s) \quad V_i(s) \leq V^*(s) \quad \text{so} \; \lim_{i \to \infty} V_i(s) \; \text{exists}$$

Value Iteration

Theorem: **For discounted reward**

$$V_\infty(s) \doteq \lim_{i \to \infty} V_i(s) = V^*(s)$$

Proof

$$\begin{aligned}
\Delta &\doteq \max_s \quad V^*(s) - V_\infty(s) \\
&= \max_s \begin{pmatrix} \max_a R(s,a) + E_{s'|a}\gamma V^*(s') \\ -\max_a R(s,a) + E_{s'|a}\gamma V_\infty(s') \end{pmatrix} \\
&\leq \max_s \max_a \begin{pmatrix} R(s,a) + E_{s'|a}\gamma V^*(s') \\ -R(s,a) + E_{s'|a}\gamma V_\infty(s') \end{pmatrix} \\
&= \max_s \max_a \; E_{s'|a}\,\gamma(V^*(s') - V_\infty(s)) \\
&\leq \gamma\Delta
\end{aligned}$$

Summary

- **A Policy** π is a stochastic way of selection an action at a state.

- **Imitation Learning** (cross entropy imitation of action given state).

- Imitation Learning is **off-policy**.

- The **value function** $V^\pi(s)$.

- **Value Iteration** $V_{i+1}(s) = \mathrm{argmax}_a\ R(s,a) + \gamma E_{s'}\ \gamma V_i(s')$

The Q Function

For discounted reward:

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi \sum_t \gamma^t r_t \mid \pi,\ s_0 = s,\ a_0 = a \\
Q^*(s,a) &= \sup_\pi Q^\pi(s,a) \\
\pi^*(a|s) &= \operatorname*{argmax}_a Q^*(s,a) \\
Q^*(s,a) &= R(s,a) + \gamma E_{s' \sim P_T(\cdot|s,a)} \max_{a'} Q^*(s',a')
\end{aligned}
$$

Q Function Iteration

It is possible to define Q-iteration by analogy with value iteration, but this is generally not discussed.

Value iteration is typically done for finite state spaces. Let S be the number of states and A be the number of actions.

One update of a Q table takes $O(S^2A^2)$ time while one update of value iteration is $O(S^2A)$.

Q-Learning

When learning by updating the Q function we typically assume a parameterized Q function $Q_\Phi(s,a)$.

Bellman Error:

$$\mathrm{Bell}_\Phi(s,a) \doteq \left(Q_\Phi(s,a) - \left(R(s,a) + \gamma\ E_{s' \sim P_T(s'|s,a)}\ \max_{a'}\ Q_\Phi(s',a')\right)\right)^2$$

Theorem: If $\mathrm{Bell}_\Phi(s,a) = 0$ for all (s,a) then the induced policy is optimal.

Algorithm: Generate pairs (s,a) from the policy $\mathrm{argmax}_a\ Q_\Phi(s_t,a)$ and repeat

$$\Phi \mathrel{-}= \eta \nabla_\Phi\ \mathrm{Bell}_\Phi(s,a)$$

Issues with Q-Learning

Problem 1: Nearby states in the same run are highly correlated. This increases the variance of the cumulative gradient updates.

Problem 2: SGD on Bellman error tends to be unstable. Failure of Q_Φ to model unused actions leads to policy change (exploration). But this causes Q_Φ to stop modeling the previous actions which causes the policy to change back ...

To address these problems we can use a **replay buffer**.

Using a Replay Buffer

We use a replay buffer of tuples (s_t, a_t, r_t, s_{t+1}).

Repeat:

1. Run the policy $\mathrm{argmax}_a\, Q_\Phi(s, a)$ to add tuples to the replay buffer. Remove oldest tuples to maintain a maximum buffer size.
2. $\Psi = \Phi$
3. for N times select a random element of the replay buffer and do

$$\Phi \mathrel{-}= \eta \nabla_\Phi \left(Q_\Phi(s_t, a_t) - (r_t + \gamma \max_a Q_\Psi(s_{t+1}, a)\right)^2$$

Replay is Off-Policy

Note that the replay buffer is from a **mixture of policies** and is **off-policy** for $\mathrm{argmax}_a\ Q_\Phi(s,a)$. This seems to be important for stability.

This seems related to the issue of stochastic vs. deterministic policies. More on this later.

Multi-Step Q-learning

$$\Phi \mathrel{-}= \sum_t \nabla_\Phi \left(Q_\Phi(s_t, a_t) - \sum_{\delta=0}^{D} \gamma^\delta r_{(t+\delta)} \right)^2$$

Asynchronous Q-Learning (Simplified)

No replay buffer. Many asynchronous threads each repeating:

$\tilde{\Phi} = \Phi$ (retrieve Φ)

using policy $\mathrm{argmax}_a\, Q_{\tilde{\Phi}}(s,a)$ compute

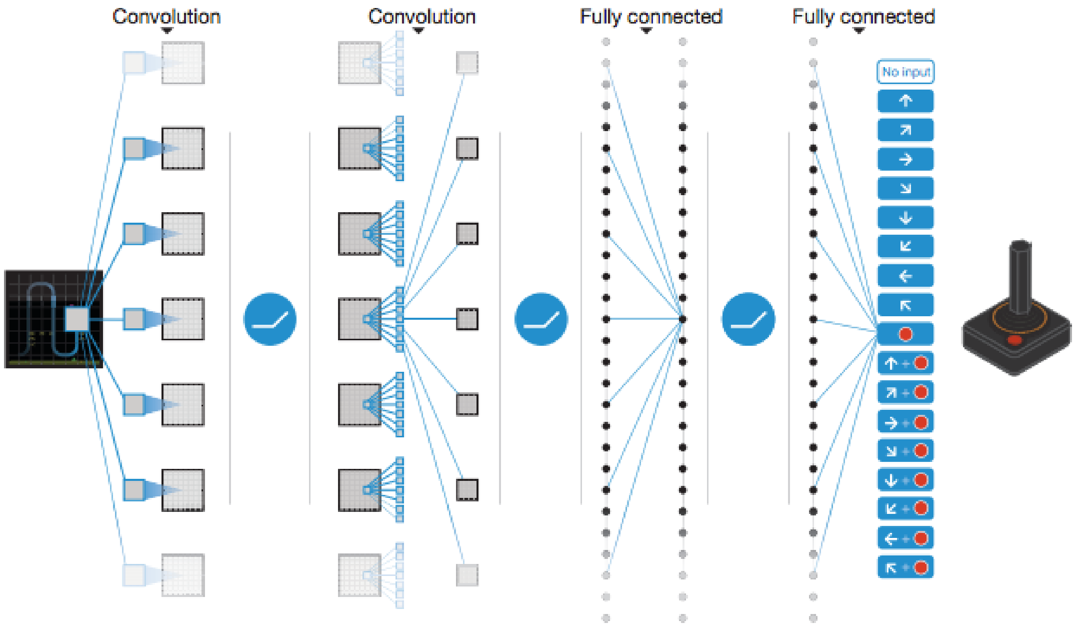
$$s_t, a_t, r_t, \ldots, s_{t+K}, a_{t+K}, r_{t+K}$$

Φ `-=` $\eta \sum_{i=t}^{t+K-D} \nabla_{\tilde{\Phi}} \left(Q_{\tilde{\Phi}}(s_i,a_i) - \sum_{\delta=0}^{D} \gamma^\delta r_{i+\delta}\right)^2$ (update Φ)

Human-level control through deep RL (DQN)

Mnih et al., Nature, 2015. (Deep Mind)

We consider a CNN $Q_\Phi(s, a)$.



Watch The Video

https://www.youtube.com/watch?v=V1eYniJ0Rnk

The REINFORCE Algorithm

Williams, 1992

REINFORCE is a Policy Gradient Algorithm

We assume a parameterized policy $\pi_\Phi(a|s)$.

$\pi_\Phi(a|s)$ is normalized while $Q_\Phi(s,a)$ is not.

Policy Gradient Theorem (Episodic Case)

$$\Phi^* = \operatorname*{argmax}_{\Phi} \; E_{\pi_\Phi} \; R$$

$$\nabla_\Phi \; E_{\pi_\Phi} \; R \;=\; \sum_{s_0,a_0,s_1,a_1,\ldots,s_T,a_T} \nabla_\Phi P(s_0, a_0, s_1, a_1, \ldots, s_T, a_T) \; R$$

$$\begin{aligned}
\nabla_\Phi \; P(\ldots)R \;=\; & P(S_0)\nabla_\Phi \; \pi(a_0)P(s_1)\pi(a_1)\cdots P(s_T)\pi(a_T) \; R \\
& + P(S_0)\pi(a_0)P(s_1)\nabla_\Phi \; \pi(a_1)\cdots P(s_T)\pi(a_T) \; R \\
& \vdots \\
& + P(S_0)\pi(a_0)P(s_1)\pi(a_1)\cdots P(s_T)\nabla_\Phi \; \pi(a_T) \; R \\
\\
\;=\; & P(\ldots)\left(\sum_t \frac{\nabla_\Phi \; \pi_\Phi(a_t)}{\pi_\Phi(a_t)}\right) R
\end{aligned}$$

Policy Gradient Theorem (Episodic Case)

$$\nabla_\Phi \; P(\ldots)R \;=\; P(\ldots)\left(\sum_t \frac{\nabla_\Phi \; \pi_\Phi(a_t|s_t)}{\pi_\Phi(a_t|s_t)}\right) R$$

$$\nabla_\Phi \; E_{\pi_\Phi} \; R \;=\; E_{\pi_\Phi} \; \left(\sum_t \nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) R$$

Policy Gradient Theorem

$$\nabla_\Phi \; E_{\pi_\Phi} \; R$$

$$= \; E_{\pi_\Phi} \; \left(\sum_t \; \nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) \quad R$$

$$= \; E_{\pi_\Phi} \; \left(\sum_t \; \nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) \left(\sum_t r_t\right)$$

$$= \; E_{\pi_\Phi} \; \sum_{t,t'} \nabla_\Phi \; \ln \pi_\Phi(a_t|s_t) \;\; r_{t'}$$

Policy Gradient Theorem

$$\nabla_\Phi \; E_{\pi_\Phi} \; R = \sum_{t,t'} \; E_{s_t,a_t,r_{t'}} \;\; \nabla_\Phi \;\; \ln \pi_\Phi(a_t|s_t) \;\; r_{t'}$$

For $t' < t$ we have

$$\begin{aligned} E_{r_{t'},s_t,a_t} \;\; r_{t'} \nabla_\Phi \;\; \ln \pi_\Phi(a_t|s_t) &= E_{r_{t'},s_t} \, r_{t'} \; \sum_{a_t} \; \pi_\Phi(a_t|s_t) \; \nabla_\Phi \;\; \ln \pi_\Phi(a_t|s_t) \\ &= E_{r_{t'},s_t} \, r_{t'} \; \sum_{a_t} \; \nabla_\Phi \; \pi_\Phi(a_t|s_t) \\ &= E_{r_{t'},s_t} \, r_{t'} \; \nabla_\Phi \; \sum_{a_t} \; \pi_\Phi(a_t|s_t) \\ &= 0 \end{aligned}$$

REINFORCE

$$\nabla_\Phi \; E_{\pi_\Phi} \; R \quad = \quad E_{\pi_\Phi} \; \sum_{t,\; t' \geq t} \; (\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)) \; r_{t'}$$

Sampling runs and computing the above sum over t and t' is Williams' REINFORCE algorithm.

Optimizing Discrete Decisions with Non-Differentiable Loss

The REINFORCE algorithm is (or was) used generally for non-differentiable loss functions.

For example error rate and BLEU score are non-differentiable — they are defined on the result of discrete decisions.

$$\Phi^* = \operatorname*{argmax}_{\Phi} \; E_{w_1,\ldots,w_n \sim P_\Phi} \; \mathrm{BLEU}$$

REINFORCE

$$\nabla_\Phi \; E_{\pi_\Phi} \; R \quad = \quad E_{\pi_\Phi} \sum_{t,\; t' \geq t} \left(\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) \; r_{t'}$$

Sampling runs and computing the above sum over t and t' is Williams' REINFORCE algorithm.

The Variance Issue

REINFORCE typically suffers from high variance of the gradient samples requiring very small learning rates and very long convergence times.

$$\nabla_\Phi\, E_{\pi_\Phi}\, R = \sum_{t,\, t' \geq t} E_{s_t, a_t, r_{t'}}\ (\nabla_\Phi\ \ln \pi_\Phi(a_t|s_t))\ \ r_{t'}$$

We will consider

- reducing variance due to $r_{t'}$ with Actor-Critic methods.
- reducing variance due to s_t with Advantage Actor-Critic methods.
- finally Asynchronous Advantge Actor-Critic Methods (A3C).

Actor-Critic Algorithms

"Policy Gradient Methods for Reinforcement Learning with Function Approximation" Mansour, Sutton, McAllester, Singh, 2000, 560 citations in 2018, 3200 citations in 2022.

"Actor-Critic Algorithms", Konda and Tsitsilas, 2000, 240 citatins in 2018, 4100 citations in 2022.

These two papers both appeared at NeurIPS 2000 and are almost identical. The first is easier to read. The second has additional insights.

Actor-Critic Algorithms

$$\nabla_\Phi\ E_{\pi_\Phi}\ R\ =\ \sum_{t,t'}\ E_{s_t,a_t,r_{t'}}\ \nabla_\Phi\ \ln\pi_\Phi(a_t|s_t)\ \ r_{t'}$$

$$=\ \sum_{t}\ E_{s_t,a_t}\ \nabla_\Phi\ \ln\pi_\Phi(a_t|s_t)\ \ \sum_{t'\geq t} E_{r_{t'}\ |\ s_t,a_t}\ r_{t'}$$

$$=\ E_{\pi_\Phi}\ \sum_{t}\ (\nabla_\Phi\ \ln\pi_\Phi(a_t|s_t))\ \ \ Q^{\pi_\Phi}(s_t,a_t)$$

$$Q^\pi(s,a)\ =\ E_\pi\ \sum_t\ r_t\ |\ s_0=s,\ \ a_0=a$$

An Actor-Critic Theorem

$$\nabla_\Phi\; E_{\pi_\Phi}\; R = \sum_t\; E_{s_t,a_t}\; (\nabla_\Phi\;\; \ln \pi_\Phi(a_t|s_t))\, Q^{\pi_\Phi}(s_t, a_t)$$

The point is that we can now approximate Q^{π_Φ} with neural network Q_Θ.

We reduced the variance at the cost of approximating the expected future reward.

An Actor-Critic Algorithm

$$\nabla_\Phi \; E_{\pi_\Phi} \; R \approx E_{\pi_\Phi} \sum_t \left(\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) \;\; Q_\Theta(s_t, a_t)$$

$\pi_\Phi(a|s)$ is the “actor” and $Q_\Theta(s,a)$ is the “critic”

An Actor-Critic Algorithm

To get a theorem for following the loss gradient we need

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\,[R \mid \pi_\Phi]$$

$$\nabla_\Phi\ E_{\pi_\Phi}\ R = E\left[\sum_t\ (\nabla_\Phi\ \ln \pi_\Phi(a_t|s_t))\ \ Q_{\Theta^*(\Phi)}(s_t, a_t) \mid \pi_\Phi\right] \qquad (1)$$

$$\Theta^*(\Phi) = \underset{\Theta}{\operatorname{argmin}}\ E\left[\sum_t\ \left(Q_\Theta(s_t, a_t) - \sum_{t' \geq t} r_{t'}\right)^2 \mid \pi_\Phi\right] \quad (2)$$

A stationary point is now a Nash equilibrium of a two-player game defined by (1) and (2).

An Actor-Critic Theorem

Thoerem:

$$\nabla_\Phi \; E_{\pi_\Phi} \; R = \sum_t \; E_{s_t,a_t} \; \left(\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)\right) \left(Q^{\pi_\Phi}(s_t, a_t) - V^{\pi_\Phi}(s_t)\right)$$

$$V^{\pi_\Phi}(s) = E_{a \sim \pi_\Phi(a|s)} \; Q^{\pi_\Phi}(s, a)$$

$Q^{\pi_\Phi}(s, a) - V^{\pi_\Phi}(s)$ is the “advantage” of deterministically using a rather than sampling an action.

Proof

We have the following for any function $V(s)$ of states.

$$
\begin{aligned}
& E_{s_t,a_t} \;\; (\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t))\, V(s_t) \\
= \;& E_{s_t} \sum_{a_t} (\pi_\Phi(a_t|s_t) \; \nabla_\Phi \; \ln \pi_\Phi(a_t|s_t))\, V(s_t) \\
= \;& E_{s_t} \sum_{a_t} (\nabla_\Phi \pi_\Phi(a_t|s_t))\, V(s_t) \\
= \;& E_{s_t}\, V(s_t) \nabla_\Phi \sum_{a_t} \; \pi_\Phi(a_t|s_t) \; = \; 0
\end{aligned}
$$

The Advantage Actor-Critic Algorithm

$$\Phi^* = \operatorname*{argmin}_{\Phi} \, [R \mid \pi_\Phi] \tag{3}$$

$$\nabla_\Phi \; E_{\pi_\Phi} \; R = E \left[\sum_t \; (\nabla_\Phi \; \ln \pi_\Phi(a_t|s_t)) \;\; (Q_{\Theta^*(\Phi)}(s_t,a_t) - V_{\Psi^*(\Phi)}(s_t)) \mid \pi_\Phi \right]$$

$$\Theta^*(\Phi) = \operatorname*{argmin}_{\Theta} \; E \left[\sum_t \; \left(Q_\Theta(s_t,a_t) - \textstyle\sum_{t' \geq t} r_{t'} \right)^2 \; \mid \pi_\Phi \right] \tag{4}$$

$$\Psi^*(\Phi) = \operatorname*{argmin}_{\Psi} \; E \left[\sum_t \; \left(Q_{\Theta^*(\Phi)}(s_t,a_t) - V_\Psi(s_t) \right)^2 \; \mid \pi_\Phi \right] \tag{5}$$

We now have a three player game defined by (3), (4) and (5).

Advantage-Actor-Critic Algorithm

$$\nabla_\Phi\, E_{\pi_\Phi}\, R \approx E_{\pi_\Phi} \sum_t \left(\nabla_\Phi\ \ln \pi_\Phi(a_t|s_t)\right) \ \ \left(Q_\Phi(s_t, a_t) - V_\Phi(s_t)\right)$$

We can sample an episode and then do

$$\Phi \mathrel{+}= \sum_t \eta_1 \left(\nabla_\Phi\ \ln \pi_\Phi(a_i|s_i)\right) \left(Q_\Phi(s_t, a_t) - V_\Phi(s_t)\right)$$

$$\Phi \mathrel{-}= \sum_t \eta_2\, \nabla_\Phi\ \left(Q_\Phi(s_t, a_t) - \textstyle\sum_{t' \geq t} r_{t'}\right)^2$$

$$\Phi \mathrel{-}= \sum_t \eta_3\, \nabla_\Phi\ \left(V_\Phi(s_t) - Q_\Phi(s_t, a)\right)^2$$

Asynchronous Methods for Deep RL (A3C)

Mnih et al., Arxiv, 2016 (Deep Mind)

$\tilde{\Phi} = \Phi$ (retrieve global Φ)

using policy $\pi_{\tilde{\Phi}}$ compute $s_t, a_t, r_t, \ldots, s_{t+K}, a_{t+K}, r_{t+K}$

$$R_i = \sum_{\delta=0}^{D} \gamma^{i+\delta} r_{(i+\delta)}$$

$$\Phi \;\texttt{+=}\; \eta \sum_{i=t}^{t+K-D} \left(\nabla_{\tilde{\Phi}} \ln \pi_{\tilde{\Phi}}(a_i|s_i)\right) \left(R_i - V_{\tilde{\Phi}}(s_i)\right)$$

$$\Phi \;\texttt{-=}\; \eta \sum_{i=t}^{t+K-D} \nabla_{\tilde{\Phi}} \left(V_{\tilde{\Phi}}(s_i) - R_i\right)^2$$

Issue: Policies must be Exploratory

The optimal policy is deterministic — $a(s) = \mathrm{argmax}_a\, Q(s,a)$.

However, a deterministic policy never samples alternative actions.

Typically one forces a random action some small fraction of the time.

Issue: Discounted Reward

DQN and A3C use discounted reward on episodic or long term problems.

Presumably this is because actions have near term consequences.

This should be properly handled in the mathematics, perhaps in terms of the mixing time of the Markov process defined by the policy.

Issue: Discounted Reward

DQN and A3C use discounted reward on episodic or long term problems.

Presumably this is because actions have near term consequences.

This should be properly handled in the mathematics, perhaps in terms of the mixing time of the Markov process defined by the policy.

Observation: Continuous Actions are Differentiable

In problems like controlling an inverted pendulum, or robot control generally, a continuous loss can be defined and the gradient of loss of with respect to a deterministic policy exists.

More Videos

https://www.youtube.com/watch?v=g59nSURxYgk
https://www.youtube.com/watch?v=rAai4QzcYbs

END