08/24/22 Wednesday.

Reinforcement learning: Mathematical formalism for learning-based decision making.
Approach for learning decision making and control from experience.
→ Data Not i.i.d.
→ No ground truth.

Machine Learning: given $D = \{(x_i, y_i)\}$
learn to predict y from x $f(x) = y$.

Decisions: Action.

Consequence: Observation (State). Reward.

Why DEEP RL? able to Adapt. / handle unstructured environments => formalism for behavior.

End-to-End: manual feature extraction => automatically learned features
(perception) (action)
Recognition + Control separately => Sensorimotor loop.

learning reward functions from examples. (inverse reinforcement learning).
transfer knowledge between domains (Meta-learning, transfer learning).
learn to predict, use prediction to act. (predictive models).
imitation learning.
inferring intentions.

08/29/22 Monday.

s_t - state — underlying true configuration.

o_t - observation.

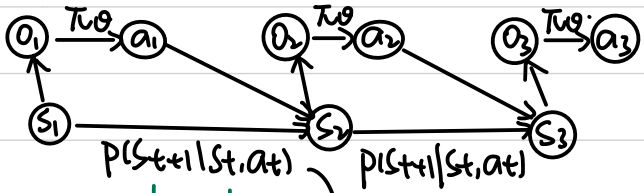
a_t - action.

State results observation.
observation may or may not be enough to deduce state.

distribution of a_t given o_t.

$\pi_\theta(a_t|o_t)$ - policy.

$\pi_\theta(a_t|s_t)$ - policy (fully observed).



future | present — Conditional independent of s_{t-1}. (past) ⇒ Markov property.

Not for $p_{t+1}(s_{t+1}|o_t,a_t)$.
(past observations may give additional info. e.g. car).

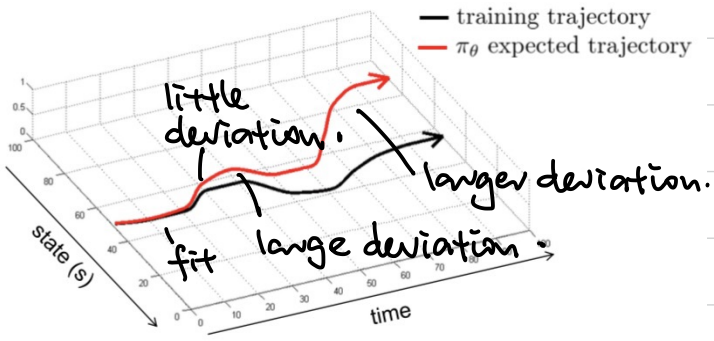
Imitation Learning.

behavior cloning. supervised learning.

o_t - left | center | right camera. → map.

a_t - rotations by steering wheel.

Left camera is supervised with steering angle that is a little bit to the right.



Distribution shift problem.

distribution of training data.

Make $p_{data}(o_t) = p_{\pi_\theta}(o_t)$? π_θ ⇒ different o_t.

distribution of test data.

Make π_θ perfect. Hard. ⇒ change data.

DAgger: Data Aggregation.

collect training data from $p_{\pi_\theta}(o_t)$ instead of $p_{data}(o_t)$.

1) train $\pi_\theta(a_t|o_t)$ from human data $D=\{o_1, a_1, \cdots, o_N, a_N\}$.

2) run $\pi_\theta(a_t|o_t)$ to get dataset $D_\pi=\{o_1, \cdots, o_M\}$.

3) Ask human to label D_π with actions a_t.

4) Aggregate: $D \leftarrow D \cup D_\pi$.

D_π will converge to the same distribution as D

Avoid distribution drift : mimic expert behavior very accurately, but don't overfit.

reasons of failure:

① Non-Markovian behavior.

$\pi_\theta(a_t|o_t)$ behavior depends only on current observation.

if see the same thing twice, do the same thing twice, regardless of what happened before. → unnatural for human beings.

Reality: $\pi_\theta(a_t|o_1, o_2, \cdots, o_t)$. Concatenate → input with various length, too large. X.

RNN, LSTM √.

Causal confusion. e.g. brake is caused by the brake light or pedestrian.

possible solutions: include the history?

DAgger?

② Multimodal behavior.

w/ discrete actions. soft max √.

if continuous, average of left and right ⇒ middle. X.

① output mixture of Gaussians. $\pi(a|o)=\sum_i w_i \mathcal{N}(\mu_i, \Sigma_i)$.

⊖: output parameters +++ ; high dimension data ⇒ challenge.

② latent variable models.

prior distribution.

input: image + latent variable (noise). output: one gaussian.

③ Autoregressive discretization. discrete continuous nD data ⇒ exponential in n.

discretize one dimension at a time. ⇒ sample from softmax ⇒ value 1. ⇒ feed into another NN. ⇒ (repeat...).

⊖ of imitation learning: Human need to provide data (only finite data; hard in some cases)

Human: unlimited data from own experience ; cts. self-improvement.

Cost / Reward function

$\min_\theta E_{a\sim\pi_\theta(a|o),\, s'\sim p(s'|s,a)}[\delta(s'=\text{eaten by tiger})]$. $\delta = \begin{cases} 0 & \text{Not eaten} \\ 1 & \text{eaten.} \end{cases}$

$= \min_\theta E_{s_{1:T}, a_{1:T}}[\sum_t \delta(s_t = \text{eaten by tiger})]$.

$c(s_t, a_t)$ – Cost function. / $\max r(s_t, a_t)$.

$c(s,a) = \begin{cases} 0 & \text{if } a = \pi^*(s) \\ 1 & \text{otherwise.} \end{cases}$ Assume $\forall s \in D_{train}.\ \pi_\theta(a \neq \pi^*(s)|s) \le \varepsilon$. $p(\text{mistake}) \le \varepsilon$. ⇒ supervised learning works.

$\frac{E[\sum_t c(s_t,a_t)]}{O(\varepsilon T^2)} \le \frac{\varepsilon T \cdot + (1-\varepsilon)(\varepsilon(T-1) + (1-\varepsilon)(\cdots))}{T \text{ terms, each } O(\varepsilon T)}$

⇒ fall off ⇒ T mistakes. $1-\varepsilon$ is large.

⇓

Assume $\forall s \sim p_{train}(s)\ \pi_\theta(a \neq \pi^*(s)|s) \le \varepsilon$. (enough if $E_{p_{train}(s)}[\pi_\theta(a \neq \pi^*(s)|s)] \le \varepsilon$.) same probability distribution.

w/ DAgger, $p_{train}(s) \to p_\theta(s)$ $E[\sum_t c(s_t,a_t)] \le \varepsilon T$.

if $p_{train}(s) \neq p_\theta(s)$, $p_\theta(s_t) = (1-\varepsilon)^t p_{train}(s_t) + (1-(1-\varepsilon)^t) p_{mistake}(s_t)$.

$|p_\theta(s_t) - p_{train}(s_t)| = (1-(1-\varepsilon)^t)|p_{mistake}(s_t) - p_{train}(s_t)| \le 2(1-(1-\varepsilon)^t)$ ($\max(p) = 1$.)

$\le 2\varepsilon t$ ($(1-\varepsilon)^t \ge 1-\varepsilon t$ if $\varepsilon \in [0,1]$).

$\sum_t E_{p_\theta(s_t)}[c_t] = \sum_t \sum_{s_t} p_\theta(s_t) c_t(s_t) \le \sum_t \sum_{s_t} p_{train}(s_t) c_t(s_t) + |p_\theta(s_t) - p_{train}(s_t)| c_{max}$.

$\le \sum_t \varepsilon + 2\varepsilon t \cdot \le \underline{\varepsilon T + 2\varepsilon T^2} \cdot\ O(\varepsilon T^2)$.

Goal-conditioned behavioral cloning.

$\pi_\theta(a|s)$ policy for reaching p_1.

$\pi_\theta(a|s,p)$ policy for reaching any p.

demo: $\{s_1, a_1, \cdots, s_{T-1}, a_{T-1}, s_T\}$. successfully reach s_T.

$\max \log \pi_\theta(a_t|s_t, g = s_T)$.

Need demonstration? ⇒ start with random policy. ⇒ random data ⇒ label goals.

09/07/22 Wednesday.

reinforcement learning.

reward function. $r(s,a)$.

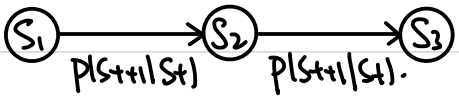
Markov chain. $M=\{S,T\}$.

S - state space. states $s \in S$. (discrete or continuous states).

T - transition operator. $p(s_{t+1}|s_t)$ probability.

⇓ matrix. let $\mu_{t,i} = p(s_t=i)$ $\vec{\mu}_t$ is a vector of probability.

let $T_{i,j} = p(s_{t+1}=i|s_t=j)$, then $\vec{\mu}_{t+1} = T\vec{\mu}_t$.



Markov decision process. $M=\{S,A,T,r\}$.

S - state space. states $s \in S$. (discrete or continuous states).

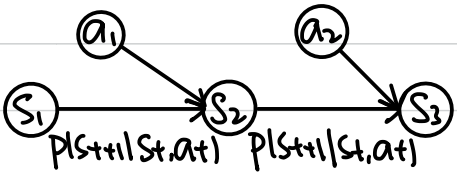
A - Action space. actions $a \in A$ (discrete or continuous actions).

T - transition operator. ⇒ tensor. 3D.

let $\mu_{t,j} = p(s_t=j)$. $\xi_{t,k} = p(a_t=k)$.

$T_{i,j,k} = p(s_{t+1}=i|s_t=j, a_t=k)$.

$\mu_{t+1,i} = \sum_{j,k} T_{i,j,k}\mu_{t,j}\xi_{t,k}$.



r - reward function. $r: S \times A \to \mathbb{R}$.

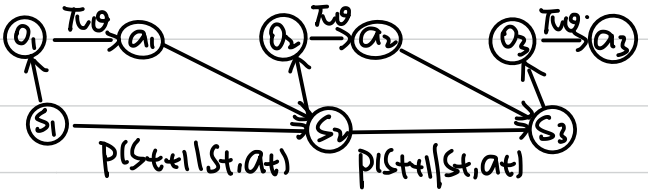
$r(s_t,a_t)$ - reward.

partially observed Markov decision process. POMDP. $M=\{S,A,O,T,\Sigma,r\}$.

O - observation space.

observation $o \in O$ (discrete or continuous observation).

Σ - emission probability $p(o_t|s_t)$.



goal of reinforcement learning.

$p_\theta(s_1,a_1,\cdots,s_T,a_T) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$.

$p_\theta(\tau)$

$p((s_{t+1},a_{t+1})|(s_t,a_t)) = p(s_{t+1}|s_t,a_t)\pi_\theta(a_{t+1}|s_{t+1})$.

$\theta^* = \arg\max_\theta E_{\tau\sim p(\tau)}\left[\sum_t r(s_t,a_t)\right]$.

$= \arg\max_\theta \sum_{t=1}^{T} E_{(s_t,a_t)\sim p_\theta(s_t,a_t)}[r(s_t,a_t)]$. ↳ state-action marginal.

finite horizon case. T.

↙ state-action transition operator.

$\begin{pmatrix} s_{t+1} \\ a_{t+1} \end{pmatrix} = \mathcal{T}\begin{pmatrix} s_t \\ a_t \end{pmatrix}$. $\Rightarrow \begin{pmatrix} s_{t+k} \\ a_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} s_t \\ a_t \end{pmatrix}$.

Stationary. (same).

$t\to\infty$. if $p(s_t,a_t)$ converges to a stationary distribution μ, we can write $\mu = \mathcal{T}\mu$. ↳ single.

Assumptions: ① ergodicity.

every state can be reached from other states w/ nonzero probability.

② chain being aperiodic. ↘

$(\mathcal{T}-\mathcal{I})\mu = 0$. μ is the eigenvector of $\lambda = 1$. (always exists) $\mu = p_\theta(s,a)$.

Infinite horizon case. $\theta^* = \arg\max_\theta E_{(s,a)\sim p_\theta(s,a)}[r(s,a)]$.

in RL, we always care about expectations.

use smooth optimization algorithms for non-smooth reward function. ↳ differentiable. e.g. GD.

E.g. binary reward function. $r(x) = \begin{cases} 1 & \text{if not fall} \\ -1 & \text{if fall} \end{cases}$ Not smooth.

$\pi_\theta(a=\text{fall}) = \theta$.

$E_{\pi_\theta}[r(x)]$ – smooth in θ. $(-1)(\theta) + (1)(1-\theta)$.

Algorithms. Trial and Error.

A. generate samples. — run the policy to get trajectory distribution.

B. fit a model / estimate the return.

C. improve the policy.

twirtal, fast.

Simple Example: B: $J(\theta) = E_{\tau}\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$. C: $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$.

expensive.

Another Example: backprop: B: learn f_ϕ s.t. $s_{t+1} \approx f_\phi(s_t,a_t)$.

C: backprop through f_ϕ and r to train $\pi_\theta(s_t) = a_t$.

Value Functions. Q-Functions. evaluate π in B $\Rightarrow$ C.

$E_{\tau\sim p(\tau)}\left[\sum_{t=1}^{T} r(s_t,a_t)\right]$.

$= E_{s_1 \sim p(s_1)}[E_{a_1 \sim \pi(a_1|s_1)}[r(s_1,a_1) + \underbrace{E_{s_2 \sim p(s_2|s_1,a_1)}[E_{a_2 \sim \pi(a_2|s_2)}[r(s_2,a_2)+\dots]|s_2]}_{Q(s_1,a_1)=}|s_1,a_1]|s_1]$.

$= E_{s_1 \sim p(s_1)}[E_{a_1 \sim \pi(a_1|s_1)}[Q(s_1,a_1)]|s_1])$.

if we know $Q(s_1,a_1)$, we can easily set $\pi(a_1|s_1)=1$ if $a_1 = \arg\max_{a_1} Q(s_1,a_1)$. ♥

Q-function: $Q^\pi(s_t,a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'},a_{t'})|s_t,a_t]$ (↓ depend on π) total expected future reward | s_t, a_t.

Value-function: $V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'},a_{t'})|s_t] = E_{a_t \sim \pi(a_t|s_t)}[Q^\pi(s_t,a_t)]$.

RL-objective: $E_{s_1 \sim p(s_1)}[V^\pi(s_1)]$.

why useful? ① if we know $Q^\pi(s,a)$, we can improve π. ♥ π' is at least as good as π.

② compute gradient to increase probability of good actions a.

if $Q^\pi(s,a) > V^\pi(s)$, a is better than average. ⇒ modify $\pi(a|s)$ to increase $p(a)$.

Types of RL Algorithms.

$\theta^* = \arg\max_\theta E_{\tau \sim p(\tau)}[\sum_t r(s_t,a_t)]$.

① policy gradient: directly differentiate obj. ⇒ GD.

② Value-Based: estimate Value fn / Q fn of optimal π to improve π. no explicit policy.

③ Actor-Critic: learn Value fn / Q fn of current π to improve π. ① + ②

④ Model-based RL: estimate the transition model $p(s_{t+1}|s_t,a_t)$ for planning directly / improve policy / ...

④: B. $p(s_{t+1}|s_t,a_t)$. ⇒ s_{t+1} or pro. dis. of s_{t+1}.

C. a) use the model of the world to plan. (→ future actions without performing them.) No explicit policy.

e.g. (continuous) trajectory optimization / optimal control.

(discrete) Monte Carlo tree search.

b) backpropagate

c) learn a value function. (dynamic programming).

②: B: fit $V(s)$ or $Q(s,a)$ NN. C: set $\pi(s) = \arg\max_a Q(s,a)$.

①: B: $R_\tau = \sum_t r(s_t,a_t)$. C: $\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(s_t,a_t)]$.

③: B: fit $V(s)$ or $Q(s,a)$. C: $\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(s,a)]$. (OR $V(s)$)

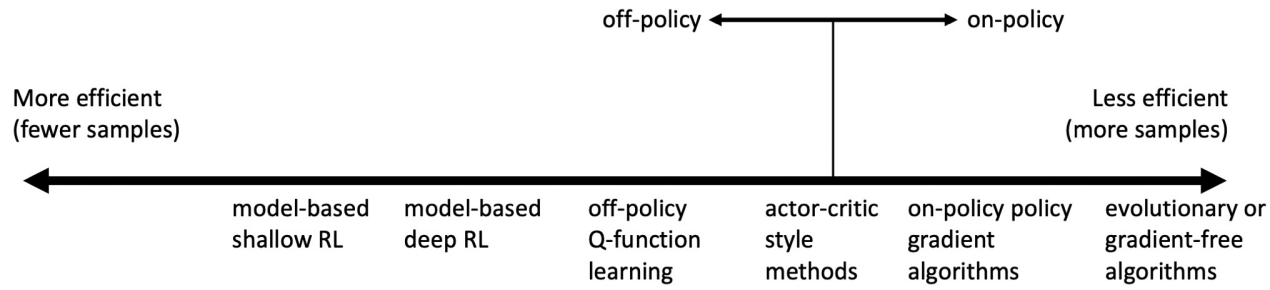
Different tradeoffs: A. Sample efficiency.

B. Stability and Ease of use.

A. # samples we need to get a good policy. BUT computation efficiency varies.

off policy algorithm: able to improve policy without generating new samples from that policy.

on policy algorithm: generate samples each time the policy is changed.



B. converge? converge to what? converge every time?

Though RL is often Not GD.

> Model-based RL: optimized to be accurate model instead of obj. minimize error of fit.

> policy gradient: GD. least efficient.

> Value function fitting: At best, minimize "Bellman Error" error of fit.

At worst, nothing.

fixed point iteration.

Different assumptions: #0 stochastic OR deterministic environment?

#1. full observability. (Value fitting).

#2. episodic learning. (policy gradient, model-based RL). infinite horizon. some.

#3. Continuity/smoothness. (value fitting, model-based RL). discrete. continuous. some.

easier to represent the policy OR model.

a_{t+1} p_{t+1}

①: REINFORCE, Natural policy gradient. Trust region policy optimization.

②: Q-learning, Deep Q-Network (DQN), temporal difference learning. Fitted value iteration.

③: Asynchronous advantage actor-critic (A3C), Soft actor-critic (SAC)

④: Dyna. Guided policy search.

09/12/2022 Monday.

Policy Gradient

$\theta^* = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}[\sum_t r(s_t,a_t)]}_{J(\theta)}$ $\quad J(\theta) \approx \frac{1}{N}\sum_i \sum_t r(s_{i,t}, a_{i,t})$. ($i$: ith sample.)

$J(\theta) = E_{\tau\sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau$. $\quad \nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau$.

$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$.

$= E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$.

By obj. $\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^T \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t,a_t)$.

$\nabla_\theta \log p_\theta(\tau) = \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t|s_t)$

$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}[(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^T r(s_t,a_t))] \approx \frac{1}{N}\sum_{i=1}^N (\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_{t=1}^T r(s_{i,t},a_{i,t}))$.

($\tau\sim p_\theta(\tau)$: unknown: $p(s_1)$, $p(s_{t+1}|s_t,a_t)$. known. known.)

REINFORCE Algorithm:
1. sample $\{\tau^i\}$ from $\pi_\theta(a_t|s_t)$
2. $\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i))(\sum_t r(s_t^i, a_t^i))$.
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

→ formalize Trial and Error as GD.

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N (\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_{t=1}^T r(s_{i,t},a_{i,t}))$.

a_i might be good/bad (since it's generated by our policy)

⇒ increase/decrease acc. to reward. → weight.

Maximum likelihood: $\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^N (\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))$

(Supervised learning). think a_i is good ⇒ Maximize/increase probability of a_i.

Conti. case: $\pi_\theta(a_t|s_t) = \mathcal{N}(f_{NN}(s_t); \Sigma)$ (f_{NN}: mean; Σ: Gaussian.)

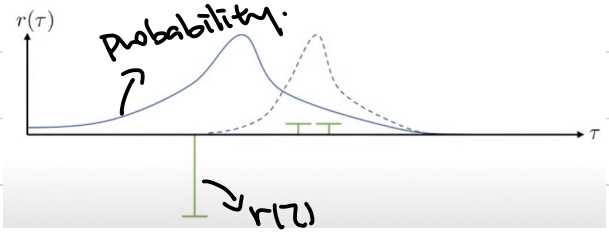
$\log \pi_\theta(a_t|s_t) = -\frac{1}{2}\|f(s_t) - a_t\|_\Sigma^2 + \text{const}$.

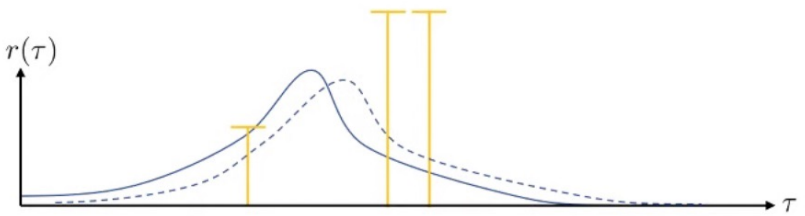
$\nabla_\theta \log \pi_\theta(a_t|s_t) = -\frac{1}{2}\Sigma^{-1}(f(s_t) - a_t)\frac{df}{d\theta}$.

Partial observability.
- s_t: Markov
- o_t: Non-Markov.

$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N (\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|o_{i,t}))(\sum_{t=1}^T r(s_{i,t},a_{i,t}))$. (doesn't use Markov property)

Add a constant term to reward fn.





* High Variance.

if $r(\tau)=0$, $\nabla \log \pi_\theta(\tau)$ doesn't matter at all.

w/ finite samples, largely depend on these random samples. ⇒ Need to reduce Variance.

w/ infinite samples, always correct. No

Reduce Variance

① **Causality**: policy at time t' cannot affect reward at time t if $t<t'$.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right)}_{\text{"reward to go" } \hat{Q}_{i,t}}$$

unbiased (expectation) ⇒ smaller variance. smaller value

② **Baseline** unbiased expectation, smaller variance.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla \log p_\theta(\tau)\,[r(\tau)-b]. \quad b=\frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

cannot increase p for all τ with positive reward instead, higher than average.

$$E[\nabla \log p_\theta(\tau) b] = \int p_\theta(\tau)\nabla \log p_\theta(\tau) b\, d\tau$$
$$= \int \nabla p_\theta(\tau) b\, d\tau = b\nabla \int p_\theta(\tau) d\tau = b\nabla \cdot 1 = 0.$$

Optimal baseline to minimize variance:

$$\mathrm{Var}[X] = E[X^2] - E[X]^2$$

$$\mathrm{Var} = E_{\tau\sim p(\tau)}\left[\left(\nabla \log p_\theta(\tau)\,(r(\tau)-b)\right)^2\right] - E_{\tau\sim p(\tau)}\left[\nabla \log p_\theta(\tau)\,(r(\tau)-b)\right]^2$$

$g(\tau)$

$$\frac{d\,\mathrm{Var}}{db} = \frac{d}{db} E_{\tau\sim p(\tau)}\left[g(\tau)^2 (r(\tau)-b)^2\right] = \frac{d}{db}\left(E[g(\tau)^2 r(\tau)^2] - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right) = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$ expected reward weighted by gradient magnitudes.

Policy gradient is on-policy since need generate data everytime we update θ.

NN only change a little bit each time ⇒ inefficient.

Important sampling. unknown known

$$E_{x\sim p(x)}[f(x)] = \int p(x) f(x) dx = \int q(x) \frac{p(x)}{q(x)} f(x) dx = E_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right].$$

$$J(\theta) = E_{\tau\sim \bar{p}(\tau)}\left[\frac{p(\tau)}{\bar{p}(\tau)} r(\tau)\right]$$

$$\frac{p_\theta(\tau)}{p(\tau)} = \frac{p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)}{p(s_1)\prod_{t=1}^{T}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)}$$

unknown, unknown

estimate θ'

$$\nabla_{\theta'}J(\theta') = E_{\tau\sim p(\tau)}\left[\frac{\nabla_{\theta'}p_{\theta'}(\tau)}{p_\theta(\tau)}r(\tau)\right] = E_{\tau\sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)}\nabla_{\theta'}\log p_{\theta'}(\tau)r(\tau)\right].$$

$\theta=\theta'$: $= E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$.

$\theta\neq\theta'$: $= E_{\tau\sim p_\theta(\tau)}\left[\left(\prod_{t=1}^{T}\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}\right)\left(\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(a_t|s_t)\right)\left(\sum_{t=1}^{T}r(s_t,a_t)\right)\right]$.

by causality

$$= E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(a_t|s_t)\left(\prod_{t'=1}^{t}\frac{\pi_{\theta'}(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})}\right)\left(\sum_{t'=t}^{T}r(s_{t'},a_{t'})\left(\prod_{t''=t}^{t'}\frac{\pi_{\theta'}(a_{t''}|s_{t''})}{\pi_\theta(a_{t''}|s_{t''})}\right)\right)\right].$$

<1, T finely => expon. to 1

Variance expon. to infinite

w/o this => policy iteration algorithm.

off-policy policy gradient: $\nabla_{\theta'}J(\theta') \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\frac{\pi_{\theta'}(s_{i,t},a_{i,t})}{\pi_\theta(s_{i,t},a_{i,t})}\right)\nabla_{\theta'}\log\pi_{\theta'}(a_{i,t}|s_{i,t})\hat{Q}_{i,t}$.

ignore

works if $\theta'\to\theta$

$$= \frac{\pi_{\theta'}(s_{i,t})}{\pi_\theta(s_{i,t})}\frac{\pi_{\theta'}(s_{i,t},a_{i,t})}{\pi_\theta(s_{i,t},a_{i,t})}$$

implement policy gradient

Automatic differentiation.

"pseudo-loss": $\tilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\log\pi_\theta(a_{i,t}|s_{i,t})\hat{Q}_{i,t}$. => cross entropy (discrete) OR squared error (Gaussian).

=> $\nabla_\theta\tilde{J}(\theta)$ is policy gradient.

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

* Consider large batches.
* tuning will be hard (Adam OK-ish)

Advanced Policy Gradient.

$\theta' \leftarrow \arg\max_{\theta'}(\theta'-\theta)^T\nabla_\theta J(\theta)$ s.t. $D(\pi_{\theta'},\pi_\theta)\leq\varepsilon$.

parameter-independent divergence measure (e.g. KL-divergence)

$D_{KL}(\pi_{\theta'}\|\pi_\theta) = E_{\pi_{\theta'}}[\log\pi_\theta - \log\pi_{\theta'}]$

$D_{KL}(\pi_{\theta'}\|\pi_\theta) \approx (\theta'-\theta)^T F(\theta'-\theta)$ Fisher-information matrix. $F = E_{\pi_\theta}[\nabla_\theta\log\pi_\theta(a|s)\nabla_\theta\log\pi_\theta(a|s)^T]$

$\theta' \leftarrow \arg\max_{\theta'}(\theta'-\theta)^T\nabla_\theta J(\theta)$ s.t. $\|\theta'-\theta\|_F^2\leq\varepsilon$. $\theta\leftarrow\theta+\alpha F^{-1}\nabla_\theta J(\theta)$

natural policy gradient: pick α.

Trust Region Policy optimization (TRPO): pick ε. solve for optimal α while solving $F^{-1}\nabla_\theta J(\theta)$.

09/14/2022 Wednesday.

Actor-Critic

(π_θ, $V^{\pi_\theta}(s_t)$)

policy-gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) - b\right)$.

unbiased, high variance, single-sample estimate.

$\hat{Q}_{i,t}$: estimate of expected future reward if we take $a_{i,t}$ in state $s_{i,t}$.

randomness $\Rightarrow \hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$. (better estimate).

$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left(Q(s_{i,t}, a_{i,t}) - b\right)$. how much better $a_{i,t}$ is (avg) over all $a_{i,t}$ given $s_{i,t}$. (avg)

↳ decrease variance. ↳ decrease variance. ⇓ increase probability on these $a_{i,t}$.

$Q = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$ the expected reward-to-go.

$b = \frac{1}{N}\sum_i Q(s_{i,t}, a_{i,t}) \Rightarrow b = V(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)}[Q(s_t, a_t)]$.

↳ decrease variance. all possible a_t given s_t.

$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$.

Actor-Critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) A^{\pi_\theta}(s_{i,t}, a_{i,t})$.

much lower variance, little bias. → fit Q^{π_θ}, V^{π_θ} or A^{π_θ} in B.

fitting Q^{π_θ}, V^{π_θ} OR A^{π_θ}.

$Q^{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \sum_{t'=t+1}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t] = r(s_t, a_t) + E_{s_{t+1}\sim p(s_{t+1}|s_t, a_t)}[V^{\pi_\theta}(s_{t+1})]$.

↑ ↑ Not Random

$\approx r(s_t, a_t) + V^{\pi_\theta}(s_{t+1})$. single-sample estimate at t+1 only.

$A^{\pi_\theta}(s_t, a_t) \approx r(s_t, a_t) + V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)$. V is easy to learn (depend only on state). ⇒ fit $V^{\pi_\theta}(s)$.

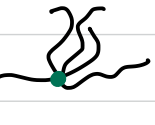
Monte Carlo policy evaluation (fit $V^{\pi_\theta}(s)$).

$V^{\pi_\theta}(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t]$. $J(\theta) = E_{s_1\sim p(s_1)}[V^{\pi_\theta}(s_1)]$.

model-free: $V^{\pi_\theta}(s_t) \approx \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$ → GP.

Want $V^{\pi_\theta}(s_t) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(s_{t'}, a_{t'})$

↑ trajectory.



only at the beginning.

but we can't run multiple trails at a particular state.

model-based:

NN will learn to make it similar. ⇒ decrease variance.

training data: $\{(s_{i,t}, \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}))\}$. → $y_{i,t}$. ↑ prevent overfit.

supervised: $L(\phi) = \frac{1}{2}\sum_i \|\hat{V}_\phi^{\pi_\theta}(s_i) - y_i\|^2$.

ideal $y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_{i,t}] \approx r(s_{i,t}, a_{i,t}) + V^{\pi_\theta}(s_{i,t+1}) \approx r(s_{i,t}, a_{i,t}) + \hat{V}_\phi^{\pi_\theta}(s_{i,t+1})$

↳ previous fitted.

bootstrapped estimate

high bias, low variance.

Actor-Critic Algorithm (batch)

1. sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$
2. fit $\hat{V}^\pi_\phi(s)$ to sampled reward sums
3. evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \hat{V}^\pi_\phi(s_i') - \hat{V}^\pi_\phi(s_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$.
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

Discount factor. (reduce variance, introduce bias).

↳ Not counting all rewards.

if $T \to \infty$, $\hat{V}^\pi_\phi \to \infty$.

better to get rewards sooner than later.

$y_{i,t} \approx \underbrace{r(s_{i,t}, a_{i,t})}_{\text{1st step}} + \underbrace{\gamma \hat{V}^\pi_\phi(s_{i,t+1})}_{\text{future.}} + (1-\gamma)0 \quad \gamma \in [0,1]$. 0.9 ~ 0.999.

to Monte Carlo GP: option 1: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right)$

option 2: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t=1}^{T} \gamma^{t-1} r(s_{i,t}, a_{i,t}) \right)$.

$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-1} r(s_{i,t'}, a_{i,t'}) \right)$

$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^{t-1} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right)$

↑ discount future policy as well. ⇒ decide case by case.

(different)

option 1 > option 2: actually want infinite movements.

to actor-critic: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t}|s_{i,t}) \left(r(s_{i,t}, a_{i,t}) + \gamma \hat{V}^\pi_\phi(s_{i,t+1}) - \hat{V}^\pi_\phi(s_{i,t}) \right)$.

online actor-critic algorithm.

1. take action $a \sim \pi_\theta(a|s)$, get (s, a, s', r)
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(s')$
3. evaluate $\hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}^\pi_\phi(s') - \hat{V}^\pi_\phi(s)$
4. $\nabla_\theta J(\theta) \approx \nabla \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

Actor-Critic Design Decisions.

- two network design.
- shared network design.

synchronized parallel: update θ using synchronized info. batch size = # parallel workers.

asynchronized parallel: update θ separately.

Off-policy actor-critic: sample batch from replay buffer. (all old transitions (s,a,s',r))

problem: π_θ is not the latest (wrong a_i, s_i') $\Rightarrow$ step 3. wrong y_i.

step 5. wrong $\nabla_\theta \log \pi_\theta(a_i|s_i)$

Step 3: $V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) | s_t]$.

$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) | s_t, a_t]$. → Not necessarily follow π_θ.

3. update $\hat{Q}_\phi^\pi$ using targets $y_i = r_i + \gamma \hat{Q}_\phi^\pi(s_i', a_i')$ $a_i' \sim \pi_\theta(a_i'|s_i')$ ← Not Replay Buffer. $\mathcal{L}(\phi) = \frac{1}{N}\sum_i \| \hat{V}_\phi^\pi(s_i) - y_i \|^2$.

Step 5: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(a_i^\pi|s_i) \hat{Q}_\phi^\pi(s_i, a_i^\pi)$ $a_i^\pi \sim \pi_\theta(a|s_i)$. ← from RB: ✗

↓ convenience | high variance ✓

state-dependent baseline.

$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right) - \hat{V}_\phi^\pi(s_{i,t}) \right)$. ← critics. low variance. unbiased.

action-dependent baseline.

Control variates: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) (\hat{Q}_{i,t} - \hat{Q}_\phi^\pi(s_{i,t}, a_{i,t})) + \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta E_{a_t \sim \pi_\theta}[\log \pi_\theta(a_t|s_{i,t}) \hat{Q}_\phi^\pi(s_{i,t}, a_t)]$.

eligibility traces / n-step returns: $\hat{A}_n^\pi(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t) + \gamma^n \hat{V}_\phi^\pi(s_{t+n})$.

(cut before variance gets too big). contribute variance (usually $n > 1$; $n \uparrow$, variance $\uparrow$, bias $\downarrow$) contribute bias.

Generalized Advantage Estimation (GAE).

$\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(s_t, a_t)$.

$w_n \propto \lambda^{n-1}$. $\hat{A}_{GAE}^\pi(s_t, a_t) = r(s_t, a_t) + \gamma((1-\lambda)\hat{V}_\phi^\pi(s_{t+1}) + \lambda(r(s_{t+1}, a_{t+1}) + \gamma((1-\lambda)\hat{V}_\phi^\pi(s_{t+2}) + \lambda(r(s_{t+2}, a_{t+2}) + \dots)$

prefer cut earlier. $= \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}$ $\delta_{t'} = r(s_{t'}, a_{t'}) + \gamma \hat{V}_\phi^\pi(s_{t+1}) - \hat{V}_\phi^\pi(s_{t'})$.

09/19/2022 Monday.

Value Function Methods.

$\arg\max_{a_t} A^\pi(s_t, a_t)$ → best action from s_t if we follow π regardless of $\pi(a_t|s_t)$.

c. policy iteration algorithm. → 1. evaluate $A^\pi(s,a) = \underbrace{r(s,a) + \gamma E[V^\pi(s')]}_{Q^\pi(s,a)} - V^\pi(s)$ ⭐

2. set $\pi \leftarrow \pi'$

greedy policy. $\pi'(a_t|s_t) = \begin{cases} 1 \text{ if } a_t = \arg\max_{a_t} A^\pi(s_t, a_t) = \arg\max_{a_t} Q^\pi(s_t, a_t) \\ 0 \text{ otherwise.} \end{cases}$ ♥

> S and a both discrete and small "tabular" → guaranteed to converge.

$T = |S| \times |S| \times |A|$ tensor. ★ $V^\pi \leftarrow E_{a \sim \pi(a|s)}[r(s,a) + \gamma E_{s' \sim p(s'|s,a)}[V^\pi(s')]]$

$\pi(s)$ $\pi(s)$

Value iteration algorithm:
1. set $Q(s,a) \leftarrow r(s,a) + \gamma E[V(s')]$.
2. set $V(s) \leftarrow \max_a Q(s,a)$ ← from argmax.

> Continuous "curse of dimensionality" $|S|$ too big.

fitted value iteration algorithm:
1. set $y_i \leftarrow \max_{a_i} (r(s_i,a_i) + \gamma E[V_\phi(s_i')])$.
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|V_\phi(s_i) - y_i\|^2$. $\leftarrow L(\phi)$

If we don't know the transition dynamics: ★ $Q^\pi \leftarrow r(s,a) + \gamma E_{s' \sim p(s'|s,a)}[Q^\pi(s', \pi(s'))]$.

↑ using samples (s,a,s') instead of π.

↓

fitted Q iteration algorithm: (transition model) Model-free. Batch Mode.

parameters:
1. collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy. — dataset size N, collection policy.
2. set $y_i \leftarrow r(s_i,a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$. — approx. the value of π at s' — iteration k.
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(s_i,a_i) - y_i\|^2$ — gradient steps S.

(k x)

+: works for off-policy; one NN — −: no guarantee to converge.

iterate value instead of policy. ↳ input: s, a output: $Q_\phi(s,a)$

Batch Mode.

Online Q learning algorithm — off-policy.

data available in sequential order.

1. take some action a_i and observe (s_i, a_i, s_i', r_i)
2. $y_i = r(s_i,a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i,a_i)(Q_\phi(s_i,a_i) - y_i)$ ★Not GD.

About the initial policy: not ♥ (fail to find high-reward states), need randomness.

> $\pi(a_t|s_t) = \begin{cases} 1-\varepsilon & \text{if } a_t = \arg\max_{a_t} Q_\phi(s_t,a_t) \\ \frac{\varepsilon}{|A|-1} & \text{otherwise.} \end{cases}$ ♥ epsilon-greedy. Varying ε.

> $\pi(a_t|s_t) \propto \exp(Q_\phi(s_t,a_t))$ Boltzmann Exploration.

+: if 2 a_t share the highest Q, can keep both.

won't waste time on a_t w/ extremely low Q.

Consider operator B: $BV = \max_a (r_a + \gamma T_a V)$ (Backup; columnwise; vector of rewards at all s for a; $|S| \times |S| : P(S'=i|S=j,a)$)

$V^*(s) = \max_a r(s,a) + \gamma E[V^*(s')] \Rightarrow V^* = BV^*$. a fixed point of B. always exist | unique.

Value Iteration Algorithm converges to V^* since B is a contraction w.r.t. ∞-norm.

(Habanlaw) $\|BV - B\bar{V}\|_\infty \le \gamma \|V - \bar{V}\|_\infty \Rightarrow \|BV - V^*\|_\infty \le \gamma \|V - V^*\|_\infty$.

↪ 1. $V \leftarrow BV$

fitted Value Iteration Algorithm:

↪ 1. $V \leftarrow \Pi B V$ $\Pi: \Pi V = \arg\min_{V' \in \Omega} \frac{1}{2} \sum \|V'(s) - V(s)\|^2$. Ω: all value fn represented by (e.NN).

↓ projection.

Contraction w.r.t. ℓ_2-norm. $\|\Pi V - \Pi \bar{V}\|^2 \le \|V - \bar{V}\|^2$

ΠB is Not a contraction. $\Rightarrow$ Not Converge (general).

fitted Q iteration Algorithm: Also Online Q-learning.

↪ 1. $Q \leftarrow \Pi B Q$. $B: BQ = r + \gamma T \max_a Q$ $\Pi: \Pi Q = \arg\min_{Q' \in \Omega} \frac{1}{2} \sum \|Q'(s,a) - Q(s,a)\|^2$.

↓ Contraction ∞-norm. ↓ Contraction ℓ_2-norm.

ΠB is Not a contraction. $\Rightarrow$ Not Converge

Similarly, Actor-Critic using bootstrap target values. (i.e. $y = r + \gamma V(s')$) Not converge.

all algorithms use NN and bootstrap Not Converge

9/12/2022 Wednesday.

Deep RL with Q-Functions.

$\ominus$: sequential states are strongly correlated.

Solution 1: parallel. < synchronized. / asynchronized.

Solution 2: full Q-learning with replay buffer. and target network.

↪ 0. save target network parameters: $\phi' \leftarrow \phi$.

↪ 1. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy. add it to B.

Supervised regression SGD. N×

↪ 2. sample a batch (s_i, a_i, s_i', r_i) from B uniformly.

k×

↪ 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')])$.

$\ominus$. moving target. $\Rightarrow$ doesn't change in the inner loop.

$k=1 \Rightarrow$ classic deep Q-learning Algorithm. (DQN.)

Alternative Target Network: update ϕ' each single step: $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$.

(instead of every N steps). e.g. $\tau = 0.999$.

General View: process 1: $(s,a,s',r) \leftrightarrow$ replay buffer (finite) $\leftrightarrow \pi(a|s)$ ↳ evict old data.

process 2: $\phi \to \phi'$

process 3: Q-function regression.

Vanila Q-learning: evict immediately (↳ replay buffer = 1). process 1|2|3 run at the same speed (↳ all one step).

DQN: process 1|3 at the same speed. process 2 slower.

fitted Q-iteration: process 3 in the inner loop of process 2. process 2 in the inner loop of process 1.

overestimate in Q-learning.

$y_i = r_i + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ $\quad E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$.

$= Q_{\phi'}(s', \operatorname{argmax}_{a'} Q_{\phi'}(s', a'))$

Solution: $\begin{cases} Q_{\phi_A}(s,a) \leftarrow r + \gamma Q_{\phi_B}(s', \operatorname{argmax}_{a'} Q_{\phi_A}(s', a')) \\ Q_{\phi_B}(s,a) \leftarrow r + \gamma Q_{\phi_A}(s', \operatorname{argmax}_{a'} Q_{\phi_B}(s', a')). \end{cases}$

Double Q-learning. $y = r + \gamma Q_{\phi'}(s', \operatorname{argmax}_{a'} Q_{\phi}(s', a'))$.

N-step return estimator. $y_{i,t} = \left(\sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{j,t'}\right) + \gamma^N \max_{a_{j,t+N}} Q_{\phi'}(s_{j,t+N}, a_{j,t+N})$.

Lower bias, higher variance.

* on policy only.

Q-learning with continuous actions

option A: stochastic optimization: cross-entropy Method (CEM)

CMA-ES

option B: easily maximized Q-functions: Normalized Advantage functions. (NAF).

$$Q_\phi(s,a) = -\frac{1}{2}(a - \mu_\phi(s))^T P_\phi(s)(a - \mu_\phi(s)) + V_\phi(s).$$

$$\operatorname{argmax}_a Q_\phi(s,a) = \mu_\phi(s). \qquad \max_a Q_\phi(s,a) = V_\phi(s).$$

option C: learn (NN) an approximate maximizer: DDPG.

Compute $y_i = r_i + \gamma Q_{\phi'}(s'_j, \mu_{\theta'}(s'_j))$ using target nets $Q_{\phi'}$ and $\mu_{\theta'}$.

$$\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)(Q_\phi(s_j, a_j) - y_j). \qquad \theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(s_j) \frac{dQ_\phi}{da}(s_j, \mu(s_j)).$$