PAGE No.
DATE / /

Reinforcement Learning

Introduction to RL

- "learning to make good sequences of decisions"

- RL involves:
① Optimization
② Delayed consequences → challenging!
③ Exploration
④ Generalization.

- Delayed consequences
→ decisions now can impact future
→ 2 challenges:
① decisions have both short term and long term effects
② temporal credit assignment is hard (what caused later high or low rewards?)

- Exploration
→ Agent is like a scientist
→ Censored data: only get a reward for decision made, don't know about other decisions
→ decisions impact what we learn about.

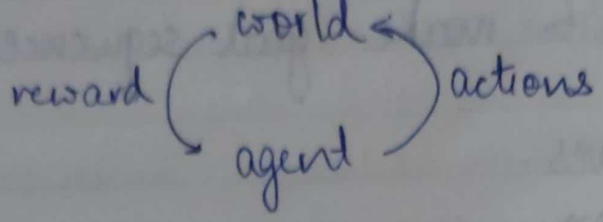
- Generalization
→ "Policy": mapping past experience to action

* Supervised learning: no exploration and delayed consequences
* imitation learning: learning from the experiences... of others

Sequential decision making under uncertainty

- interactive closed-loop process:

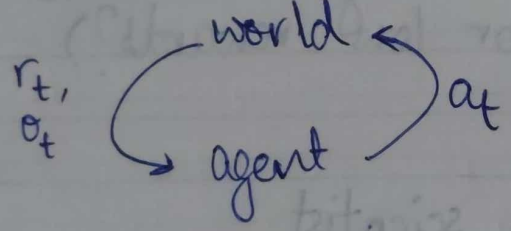
→



→ goal is to select actions to maximize ~~total future~~ expected ~~reward~~ future reward.

- discrete time:

→ at every time step t:
- action a_t is taken by the agent
- world updates action a_t, returns rewards r_t and observation o_t, which agent recieves.



→ history: a sequence of past a_i, o_i, r_i
- $h_t \rightarrow (a_1, o_1, r_1, a_2, o_2, r_2 \cdots a_t, o_t, r_t)$
- agent chooses action based on history

- state

→ world state: the true stat of the world used to determine how the o_t and r_t are generated.

↔ - often hidden to agent.

→ agent state
- the state used by agent to make decisions.
- usually a function of history: $s_t = f(h_t)$

• Markov assumption:

→ assumptions:
① state used by the agent is a sufficient statistic of the history
② state s_t is markov iff:
$$p(s_{t+1} | s_t, a_t) = p(s_{t+1} | h_t, a_t)$$
③ future is independent of the past given present

→ basically, a state follows the Markov assumption if the current state tells us everything we ~~need to~~ know from the whole history.

→ why is it popular?
• can always be satisfied by using $s_t = h_t$

→ in practice we just generally use $s_t = o_t$, & ie, we assume that the most recent obs is a sufficient statistic of the history.

→ state representation has impact on:
① computational complexity
② data required
③ resulting performance.

- Full observability / Markov Decision Process (MDP)
→ world state $s_t = o_t$

- Partial observability / Partially observable MDP (POMDP)
→ agent state is not the same as the world state
→ agent constructs its own state, eg, uses $s_t = h_t$, uses world beliefs, uses RNN, etc.
→ ex: poker player sees only his own cards.

- Types of Sequential Decision Making Processes

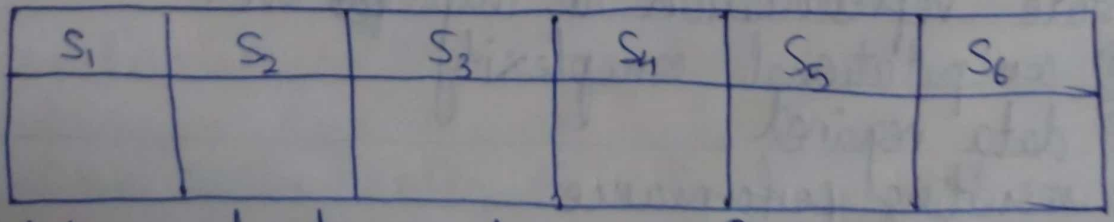
① Bandits: actions have no influence over next observations
: ie, no delayed awards.

② MDPs and POMDPs: actions affect future observations
: credit assignment and strategic actions may be needed.

- How the world changes:
① deterministic: given h_t, a_t; single o_t, r_t.
② stochastic: given h_t, a_t; many possible o_t and r_t

Components of RL

- Mars rover example



→ states: location of rover (s_k)
→ actions: TryLeft or TryRight

PAGE No.
DATE / /

- components:
  1. model
  2. policy
  3. value function

- Model
→ Agent's representation of how the world changes in response to the agent's action.
→ transition / dynamics model: predicts next agent state:

$$p(s_{t+1} = s' \mid s_t = s, a_t = a)$$

→ reward model: predicts immediate reward:

$$r(s_t = s, a_t = a) = E[r_t \mid s_t = s, a_t = a]$$

→ in mars rover example:

- s_1 s_2 s_3 s_4 s_5 s_6 → states
  0 0 0 0 0 0 → agent's reward model
- part of agent's transition model:
  $0.5 = P(s_1 \mid s_1, \text{TryRight}) = P(s_2 \mid s_1, \text{TryRight})$
- model may be wrong.

~~model will be learned~~

- Policy (π)

→ determines how the agent chooses actions.
→ $\pi : S \rightarrow A$ (mapping from states to actions)

→ deterministic policy: $\pi(s) = a$
stochastic policy: $\pi(a|s) = P(a_t = a \mid s_t = s)$

→ in mars rover example
- policy could be: $\pi(s_1) = \pi(s_2) \cdots \pi(s_6) = \text{TryRight}$

• Value function (V^π)

→ It is the expected discounted ~~function~~ sum of future rewards under a given policy π.

$$V^\pi(s_t = s) = E_\pi\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s\right)$$

→ γ weighs immediate vs future rewards (discount factor)

→ V^π can be used to quantify goodness / badness of states and actions, and decide how to act by comparing policies.

→ in mars rover example
- same policy (TryRight always)
- $\gamma = 0.9$
- S_1 S_2 S_3 S_4 S_5 S_6 $\longrightarrow$ S_k
  +1 +6.561 +7.29 +8.1 +9 +10 $\longrightarrow$ $V^\pi(S_k)$

• Key challenges

→ Planning: (Agent's internal computation)
- Given ~~the~~ how the world works (dynamics and rewards model), algorithm ~~to~~ computes how to act in order to maximize

no interaction with the real environment

→ RL
- agent doesn't know how the world works
- interacts with the world to learn this
- then agent improves policy (may involve planning)

→ ex: planning may involve dynamic programming, tree search, etc.

• Exploration vs Exploitation

→ Exploration → trying new stuff, that might help the agent make better decisions in the future.

Exploitation → given past experience, agent chooses actions that are expected to give good rewards.

→ often there is an exploration-exploitation tradeoff

• Evaluation and Control

→ Evaluation: estimate / predict the expected rewards from a following a given policy

→ Control: optimization (find the best policy)

Given a model of the world

- MDP (full observability)

→ satisfies markov assumption

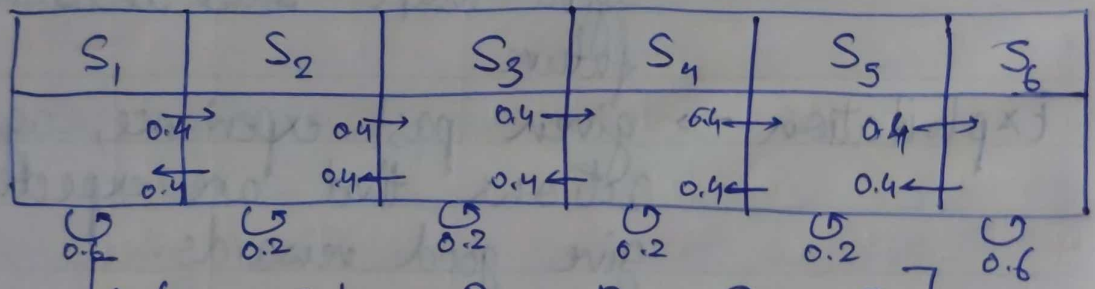
- Markov Process or Markov Chain

→ definition: S is a finite set of states ($s \in S$), P is a dynamics / transition model that specifies: $p(S_{t+1} = s' \mid S_t = s)$

* no rewards / actions are involved yet.

→ P can be represented as a matrix if finite states.

→ Ex:



$$P = \begin{bmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{bmatrix}$$

$$P_{ij} = p\left(S_{t+1} = S_j \mid S_t = S_i\right)$$

$$p\left(S_{t+1} \mid S_t = S_2\right) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} P$$

→ we can sample starting from a certain state:
ex: $s_4, s_3, s_4, s_5, s_6, s_5 \dots$
$s_4, s_3, s_2, s_1, s_2, s_1 \dots$
etc.

• Markov Reward Process (MRP)

→ it is basically: Markov process + reward
* still no actions involved.
→ ~~it is the expected return~~

→ definition:
- S is a (finite) set of states S ($s \in S$)
- P is a dynamics/transition model which specifies $p(s_{t+1} = s' | s_t = s)$
- R is a reward function $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
- γ is the discount factor ($\gamma \in [0, 1]$)

→ R can be represented as a vector if finite states

→ Ex: $R = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 10 \end{bmatrix}$

→ return and value function
- horizon: # steps in each episode
: can be infinite
: otherwise is called finite MRP.

→ Return (G_t) (for a MRP)
- discounted sum of rewards from time step t to horizon:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots \gamma^n r_{t+n} \cdots$$

→ state value function ($V(s)$) for a MRP
- expected return starting from state s.

$$V(s) = E(G_t | s_t = s)$$

$$= E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots \gamma^n r_{t+n} \cdots \middle| s_t = s\right]$$

* if process is deterministic: $G_t = V(s)$

→ Ex: $R = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 10 \end{bmatrix}$, $P = \begin{bmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{bmatrix}$

for example episode s_4, s_5, s_6; $\gamma = 1/2$

$$G = 0 + 0\left(\frac{1}{2}\right) + (10)\left(\frac{1}{4}\right) = 2.5.$$

→ Computing the value of a MRP:
- simulation
- average returns
- requires no assumption of markov structure.

→ thus: (Bellman Equation)

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)\, V(s')$$

⇒

⇒ $V = R + \gamma P V$

⇒ $(I - \gamma P)V = R$

⇒ $\boxed{V = (I - \gamma P)^{-1} R}$ ⇔ for MRPs

* computing this takes $O(n^3)$ time ($n \rightarrow$ # states)

→ we use a dynamic programming based approach:

let $V_0(s) = 0$ for all s
for $k=1$ until convergence {
for all s in S {

$$V_k(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)\, V_{k-1}(s')$$

}
}

* $O(n^2)$ time
* convergance: $|V_k - V_{k-1}| < \epsilon$

- Markov Decision Processes (MDP)

→ it is basically MRP + action

→ definition:
- S is a (finite) set of Markov states ($s \in S$)
- P is a dynamics/transition model for <u>each action</u> that specifies:

$$P(S_{t+1} = s' \mid S_t = s, a_t = a)$$

- A is a (finite) set of actions ($a \in A$)
- R is a reward function

$$R(S_t = s, a_t = a) = E[r_t \mid S_t = s, a_t = a]$$

- discount factor $\gamma \in [0, 1]$.

→ MDP is a tuple (S, A, P, R, γ)

Ex: 2 deterministic actions: $A_1 \longrightarrow$ always left
$A_2 \longrightarrow$ always right

$$P(s' \mid s, a_1) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P(s' \mid s, a_2) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

→ MDP policy:
- determines what action to take in each state : (can be deterministic / stochastic)
- $\pi(a|s) = P(a_t = a | s_t = s)$.

→ MDP + policy = MRP
- MRP $(S, R^{\pi}, P^{\pi}, \gamma)$

$$R^{\pi}(s) = \sum_{a \in A} \pi(a|s) R(s,a)$$

$$P^{\pi}(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s,a)$$

→ now we can perform the MRP iterative process for a policy, to get the value function for that policy!

let $V_0^{\pi}(s) = 0$ for all s.
for k=1 till convergance {
  for all s in S {

$$V_k^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{s' \in S} P^{\pi}(s'|s) V_{k-1}^{\pi}(s')$$

  }
}

→ MDP control

- Compute optimal policy for state s:

$$\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\ V^{\pi}(s)$$

- There exists a unique optimal value function
- The optimal policy for a MDP in an infinite horizon problem is deterministic.

→ Ex: 6 discrete states: s_1 s_2 s_3 s_4 s_5 s_6
2 actions possible: tryLeft, tryRight

⇒ 2^6 deterministic policies possible

* an optimal policy is not necessarily unique; ex:

s_1	s_2	s_3	← states
+10	0	+10	← rewards

⇒ $\pi^*(s_2)$ is not unique.

• finding the optimal policy.

→ one way could be to calculate the value function for every possible policy, but #policies is large.

→ <u>policy iteration</u> is generally better than policy enumeration

→ MDP Policy Iteration

set $i = 0$
initialize $\pi_0(s)$ randomly for all states s.
while ($i == 0$ or $\|\pi_i - \pi_{i-1}\|_1 > 0$) {

$V^{\pi_i} \leftarrow$ policy evaluation of π_i
$\pi_{i+1} \leftarrow$ policy improvement
$i++$

}

→ ~~Policy improvement step:~~

→ State - action value (Q)

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s')$$

- take action 'a', THEN follow policy π.

→ policy improvement step:

- $Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s')$
- compute π_{i+1} as:

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a) \quad \forall s \in S$$

→ proof that $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$, and inequality holds only if π_i is suboptimal.

Proof: by definition:

$$V^{\pi_{i+1}}(s) = \arg\max_a Q^{\pi_i}(s,a) \geq V^{\pi_i}(s)$$

→ Proof that π_{i+1} is a better policy than π_i, or equal if π_i is optimal.

Proof: by definition:

$$\max_a Q^{\pi_i}(s,a) \geq R(s,\pi_i(s)) + \gamma \sum_{s' \in S} P(s' | s, \pi_i(s)) V^{\pi_i}(s')$$

$$= V^{\pi_i}(s)$$

$$\therefore \pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

[but this is saying that, take $\pi_{i+1}(s)$ then take $\pi_i(s')$. we cannot yet comment if always taking $\pi_{i+1}(s)$ is a better policy]

to prove: $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s) \quad \forall s \in S$

proof: $V^{\pi_i}(s) \leq \max Q^{\pi_i}(s,a)$

$$= \max_a R(s,a) + \gamma \sum_{s' \in S} P(s' | s, a) V^{\pi_i}(s')$$

$$\Rightarrow V^{\pi_i}(s) \leq \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s') \right]$$

- but $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$

$$\therefore V^{\pi_i}(s) \leq R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s')$$

$$\leq R(s, \pi_{i+1}(s)) + \gamma \sum P(s'|s, \pi_{i+1}(s)) \left(\max_a Q^{\pi_i}(s,a) \right)$$

$$\Rightarrow V^{\pi_i}(s) \leq V^{\pi_{i+1}}(s) \Rightarrow$$ policy can only get better!

and equality holds only when $\pi_i(s) = \pi_{i+1}(s) \; \forall s_i$

- if $\pi_i == \pi_{i+1}$, then policy will never change again:

$$\pi_{i+2}(s) = \arg\max_a Q^{\pi_{i+1}}(s,a) = \arg\max_a Q^{\pi_i}(s,a)$$

$$= \pi_{i+1}(s).$$

- Value Iteration

→ It is an alternative method to find optimal policy in a MDP.

→ Idea: maintain optimal value of starting in a state s if have a finite number of steps k left in the episode. Iterate to consider longer & longer episodes

→ Bellman Equation and Bellman Backup Opera

- BE: $$V^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{s' \in S} P(s'|s) V^{\pi}(s')$$

- Bellman backup operator:
  - applied to a value fn
  - returns a new value fn
  - improves the value if possible

$$BV(s) = \max_{a} R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)$$

- BV yields a value function over all states s.

Value iteration algorithm:

let $k = 0$
let $V_0(s) = 0$ for all states
loop until convergance / horizon {
  for each state s {

$$V_{k+1}(s) = \max_{a} R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a$$

  }

$$\pi_{k+1}(s) = \arg\max_{a} R(s,a) + \sum_{s' \in S} P(s'|s,a$$

}

→ Bellman operation for a policy

- $B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V(s)$
- policy evaluation ⇒ computing the fixed point of B^π.
- meaning, to evaluate a policy apply B^π till V stops changing:

$$V^\pi = B^\pi B^\pi B^\pi \cdots B^\pi V$$

→ will value iteration converge?

- Bellman operator is a "contraction operator", ie:

$$\|BV_i - BV_j\| \leq \|V_i - V_j\| \quad \text{for some } \|\cdot\|_\infty$$

- proof:

$$\|BV_i - BV_j\|_\infty = \max_s |V_i(s) - V_j(s)|$$

$$\|BV_i - BV_j\| = \left\| \max_a \left(R(s,a) + \gamma \sum P(s'|s,a) V_i(s') \right) - \max_{a'} \left(R(s,a') + \gamma \sum P(s'|s,a') V_j(s') \right) \right\|$$

$$\leq \left\| \max_a \left[R(s,a) + \gamma \sum P(s'|s,a) V_i(s') - R(s,a) - \gamma \sum P(s'|s,a) V_j(s') \right] \right\|$$

$$= \left\| \max_a \left[\gamma \sum P(s'|s,a) \left(V_i(s') - V_j(s') \right) \right] \right\|$$

$$\leq \left\| \max_a \left[\gamma \sum P(s'|s,a) \|V_i - V_j\| \right] \right\|$$

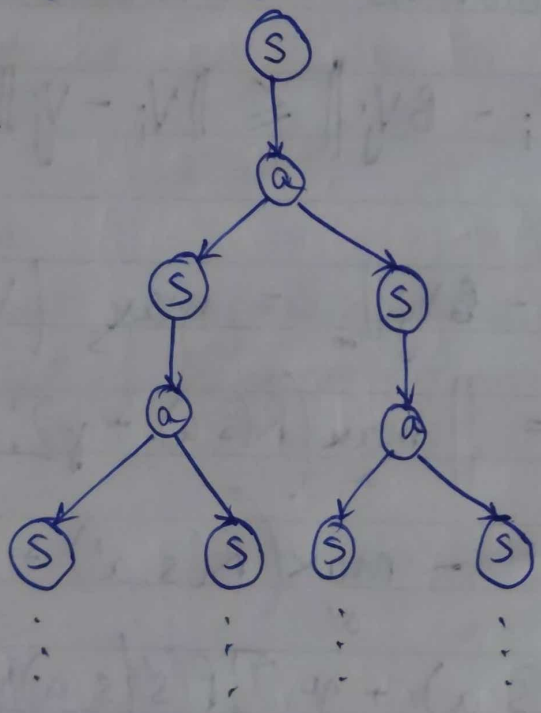
$$= \gamma \|V_i - V_j\|.$$

- since the Bellman operator is a contraction operator, the distance between BV_i & BV_j is atleast less than that between V_i & V_j. (if $\gamma < 1$). thus, it must converge.

Model free policy evaluation

- Till now, we have considered that we have access to $R(s)$, $P(s'|s,a)$. But this might not always be the case.
- Dynamic Programming (requires model)

→



$$V^{\pi}_{k+1}(s) = R^{\pi}(s) + \sum_{s' \in S} P(s'|s,a)\, V^{\pi}_{k}(s)$$

→ ie, we do not go down the whole tree, we just bootstrap the value of a state

substitute for value.

- Monte Carlo Policy Evaluation

→ can be applied even if model is not known

→ $G_t^\pi = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots$ in MDP M under policy π.

then: $V^\pi(s) = \mathbb{E}_{T\sim\pi}[G_t \mid s_t = s]$

ie, expected return over trajectories generated by following π.

→ If all trajectories are finite, sample a set of trajectories and average the return.

→ Does not require bootstrapping dynamics model, rewards model, does not assume Markov.

→ can only be applied to episodic MDPs (ie, we must be able to sample many times)

- First Visit MC on policy evaluation

→ it is an algorithm.

→ let $N(s) = 0$, $G(s) = 0$ $\forall s \in S$

loop {

sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots s_{i,T_i}$

define $G_{i,t} = r_{i,t} + \gamma r_{i,t+1} \cdots \gamma^{T_i - 1} r_{i,T_i}$ as returns from time step t onwards in the i^{th} episode.

for each state s visited in i {

for the first time t that s is visited in i {

$N(s) ++;$

$G(s) = G(s) + G_{i,t}$

$V^{\pi}(s) = \frac{G(s)}{N(s)}$

}

}

}

→ this is an unbiased estimator of true $E_{\pi}[G_t | s_t = s]$

→ How good is an estimate?

params: θ → true, $\hat{\theta}$ → estimate prob distribution over observed data $P(x/\theta)$

⇒ Bias $= E_{x/\theta}[\hat{\theta}] - \theta$

Variance $= E[(\hat{\theta} - E[\hat{\theta}])^2]$

- MSE of an estimator $\hat{\theta}$ is:

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias_\theta(\hat{\theta})^2$$

- Every Visit MC on policy evaluation

→ same algorithm, but for all times t that s is visited

→ biased estimator [other times when s is visited are dependent on the first time]

→ but better MSE, as it has low variance

- Incremental MC policy evaluation

→ dont keep track of $G(s)$, instead update value estimate as:

$$V^\pi(s) \leftarrow V^\pi(s)\left(\frac{N(s)-1}{N(s)}\right) + \frac{G_{i,t}}{N(s)}$$

ie:

$$V^\pi = V^\pi(s) + \frac{1}{N(s)}\left(G_{i,t} - V^\pi(s)\right)$$

→ same algo; but:

for each state s visited in i {

$N(s) = N(s) + 1$

$$V^\pi(s) = V^\pi(s) + \alpha\left(G_{i,t} - V^\pi(s)\right)$$

}

→ if $\alpha = \frac{1}{N(s)}$ ⟶ every visit MC

$\alpha > \frac{1}{N(s)}$ ⟶ weighs recent data higher

- MC policy evaluation key limitations

→ Generally high variance, thus will require a lot of data
→ Requires episodic settings

Temporal Difference (TD) learning

- Introduction

→ combination of MC & DP methods
→ can be used to find the value of model free
→ can be used both ⇒ in episodic & infinite horizon non-episodic settings cases.

→ IMMEDIATELY updates estimate of V after each (s, a, r, s') tuple.

- TD for estimating V.

→ Aim: estimate $V^{\pi}(s)$ given episodes generated under policy π.

→ $G_t^\pi = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots$ in MDP M, policy π.

⇒ $V^\pi(s) = E_\pi [G_t^\pi | s_t = s]$

→ if MDP model were known:

$$B^\pi V(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi(s)) V(s')$$

→ in incremental every visit MC, update estimate using 1 sample of return (for i^{th} episode):

$$V^\pi(s) = V^\pi(s) + \alpha (G_{i,t} - V^\pi(s))$$

→ insight for TDL: we have an estimat of V^π, use it to estimate expected return

- so we would be able to update at every step.

$$V^\pi(s) = V^\pi(s) + \alpha \left[[r_t + \gamma V^\pi(s_{t+1})] - V^\pi(s) \right]$$

• TD(0) learning

→ exactly what we have seen above:

$$\boxed{V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha [(r_t + \gamma V^\pi(s_{t+1})) - V^\pi(s_t)]}$$

→ TD error: $\boxed{\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)}$

→ algorithm:

let $\alpha = \cdots$, $\gamma = \cdots$

$V^{\pi}(s) = 0 \quad \forall s \in S$

loop {

sample tuple (s_t, a_t, r_t, s_{t+1})

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha\left[(r_t + \gamma V^{\pi}(s_{t+1})) - V^{\pi}(s_t)\right]$$

}

Model free control

• Introduction

→ optimization ⟶ identify best policy

→ delayed consequences ⟶ may take many time steps to evaluate whether an earlier decision was good or bad.

→ exploration ⟶ necessary to try diff actions to learn which actions lead to highest rewards

→ $\pi^{\beta}_{i+1}(s) = \underset{a}{\text{argmax}}\ Q^{\pi_i}(s,a)$

so if we could estimate $Q^{\pi}(s,a)$, we would be done.

- MC for on policy Q - evaluation

→ Let $N(s,a) = 0$, $G(s,a) = 0$, $Q^{\pi}(s,a) = 0 \ \forall s,a$.

Loop {

episode i (following π) $= s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2} \cdots s_{T_i}$

$$G_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} \cdots \gamma^{T_i - 1} r_{i,T_i}$$

for each (s,a) visited in episode i {

for first/every time t that (s,a) is visited {

$N(s,a) \pm 1$, $G(s,a) = G(s,a) + G_{i,t}$

$Q^{\pi}(s,a) \leftarrow G(s,a) / N(s,a)$

}

}

}

- Model free generalized policy improvement

→ Given an estimate $Q^{\pi_i}(s,a) \ \forall s,a$;

Update new policy:

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

- Model free policy iteration

initalize π
loop {
compute $Q^{\pi}(s,a)$
$\pi(s) \leftarrow \underset{a}{\text{argmax}}\,(Q(s,a))$
}

→ we will have to try all (s,a) for $Q^{\pi}(s,a)$.
to ensure this, we introduced randomly:

- ϵ-greedy policies
→ balance exploration-exploitations.
→ $|A| \Rightarrow$ # actions.

then:

$$\pi(a|s) = \begin{cases} \text{prob } (1-\epsilon) : \underset{a}{\text{argmax}}\, Q^{\pi}(s,a) \\ \text{prob } \frac{\epsilon}{|A|} : \text{other actions.} \end{cases}$$

what action we will take

→ proof that ϵ-greedy policy improvement is monotonic:

$$V^{\pi_{i+1}} > V^{\pi_i} \Rightarrow \text{(to prove)}$$

proof:

$$Q^{\pi_i}(s, \pi_{i+1}(s)) = \sum_{a \in A} \pi_{i+1}(a|s)\, Q^{\pi_i}(s,a)$$

$$= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + (1-\epsilon) \max_a Q^{\pi_i}(s,a)$$

$$= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + \left[\sum_{a \in A} \pi_i(a|s) - \epsilon\right] \max_a Q^{\pi_i}(s,a)$$

$$\Rightarrow Q^{\pi_i}(s, \pi_{i+1}(s)) = \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s,a) + \sum_{a \in A} \pi_i(a|s)\, Q^{\pi_i}(s,a) - \sum_{a \in A} \frac{\epsilon}{|A|} \max_a Q^{\pi_i}(s,a)$$

$$\geq \sum_{a \in A} \pi_i(a|s)\, Q^{\pi_i}(s,a)$$

$$\therefore Q^{\pi_i}(s, \pi_{i+1}(s)) \geq V^{\pi_i}$$

$$\Rightarrow \boxed{V^{\pi_{i+1}} \geq V^{\pi_i}}$$

→ we want ϵ to decay so that ~~greedy policy converges to optimal~~ behaviour policy converges to greedy policy.

- a simple strategy : $\epsilon_i = \frac{1}{i}$

- MC online control / on policy improvement.

initialize $Q(s,a) = 0$
$N(s,a) = 0$
$\epsilon = 1$
$k = 1$
$\pi_k = \epsilon\text{-greedy}(Q)$
loop {

sample k^{th} episode given π_k.

$G_{k,t} = r_{k,t} + \gamma r_{k,t+1} + \gamma^2 r_{k,t+2} \cdots$

for $(t = 1 \cdots T)$ {

$N(s,a) = N(s,a) + 1$

$Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)} (G_{k,t} - Q(s_t, a_t))$

}

$k = k+1, \quad \epsilon = 1/k$

$\pi_k = \epsilon\text{-greedy}(Q)$

}

PAGE No.
DATE / /

- Model free policy iteration with T-D- methods

→ basic algorithm:

initialize π
loop {

compute Q^{π} using T-D learning (policy evaluation)

same as MC, $\pi_0 = \epsilon$-greedy(Q^{π}) (policy improvement)

}

- Sarsa algorithm

→ $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$

→ algo:

let $s_t = s_0$, $t=0$, initialize π
take $a_t \sim \pi(s_t)$
observe (r_t, s_{t+1})

loop {
take $a_{t+1} \sim \pi(s_{t+1})$
observe (r_{t+1}, s_{t+2})

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$$

$\pi_t(s_t) = \underset{a}{\text{argmax}}\, Q(s_t, a)$ w.prob $1-\epsilon$, else random
$t \leftarrow t+1$

}

→ SARSA converges under the following conditions:

① $\pi_t(a|s)$ satisfies GLIE

② α_t satisfy Robbins-Munro sequence:

$$\sum_1^\infty \alpha_t = \infty$$
$$\sum_1^\infty \alpha_t^2 < \infty$$

} but in practical uses we use something like $\alpha_t = 1/t$

• Q-Learning

→ estimate value of optimal policy π^* without knowing π^*!

→ key insight:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left((r_t + \gamma \max_{a'} Q(s_{t+1}, a')\right. \\ \left. - Q(s_t, a_t)\right)$$

"use previously computed Q estimate to bootstrap and use the value of the best future action."

→ behaviour policy = ϵ-greedy $(Q^*(s,a))$

→ algo:

initialize $Q(s, a)$, $t = 0$, $s_t = s_0$
set π_b to be ϵ-greedy wrt Q.

loop {

take $a_t \sim \pi_b(s_t)$

observe (r_t, s_{t+1})

update Q given (s_t, a_t, r_t, s_{t+1})

$\pi_b = \epsilon$-greedy (Q)

$t = t + 1$

}

→ update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a')\right) - Q(s_t, a_t)\right)$$

→ condition for Q^*: visit each (s, a) ∞ times
$\alpha_t \rightarrow$ RM sequence

→ condition for π^*: GLIE

- Maximization bias

→ even with an unbiased $\hat{Q}(s,a)$, ~~we can get a biased $\hat{V}$~~ the estimate of $\hat{\pi}$'s value $V^{\hat{\pi}}$ can be biased.

$$\hat{\pi} = \underset{a}{\text{argmax}}\ \hat{Q}(s,a)$$

Lets say $\hat{Q}(s,a_1) = \hat{Q}(s,a_2) = 0 = V(s)$

$$\hat{V}^{\hat{\pi}} = E(\max(\hat{Q}(a_1), \hat{Q}(a_2)))$$

$$\geq \max[E[\hat{Q}(a_1)], E[\hat{Q}(a_2)]]$$

$$\Rightarrow \hat{V}^{\hat{\pi}} \geq 0 = V^{\pi} \Rightarrow \therefore \hat{V}^{\hat{\pi}} \text{ is biased.}$$

→ Q learning is succeptible to this

- Double Q-learning

→ use one Q for policy & one Q for estimate $\hat{Q}(s,a)$

→ algo:

$Q_1(s,a)$, $Q_2(s,a)$, $t=0$, $s_t = s_0$.

loop {

$a_t \sim \epsilon\text{-greedy}(\pi(s)) = \underset{a}{\text{argmax}}\ Q_1(s_t,a) + Q_2(s_t,a)$

observe (r_t, s_{t+1})

with 50% chance:

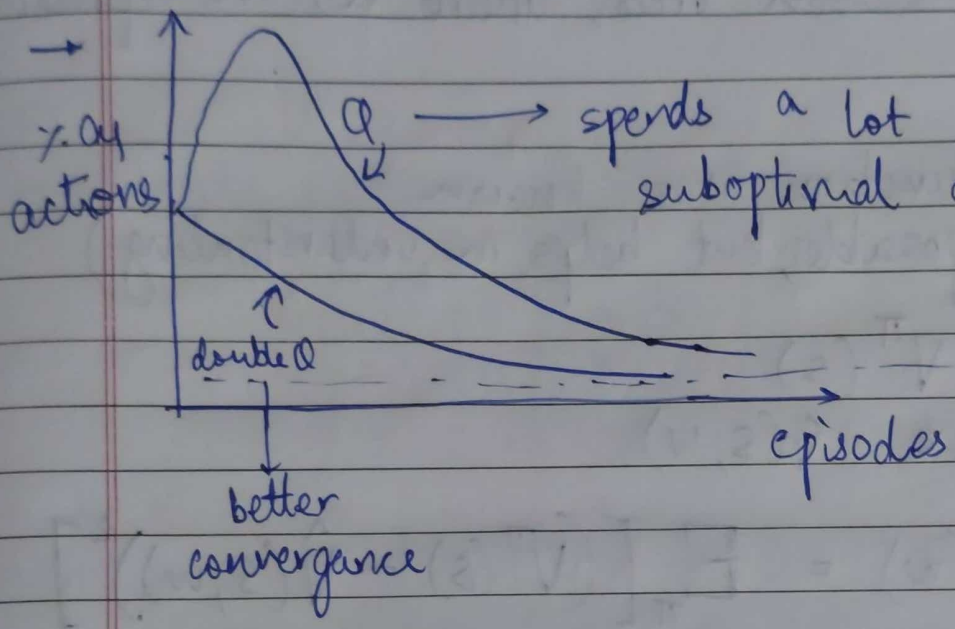
$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(\cdots)$$

else:

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(\cdots)$$

$t = t + 1$

}



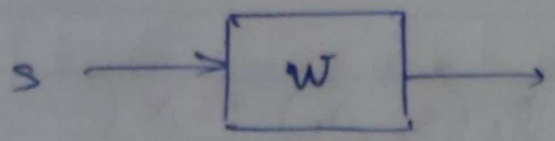
→ Q ⟶ spends a lot of time making suboptimal decisions.

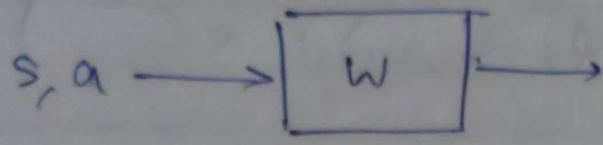
Value function approximation

- Motivation

→ usually we have very large state/action spaces, so saving the $Q(s,a)$ values is not feasible. Also, we cannot generalize in such cases as whenever we see a new state, we have no idea about what action to take

→ we want to represent the value function as a ~~function~~ parameterized function of





→ function approximation can be done using deep NNs, decision trees, linear feature representations, KNN, etc.

- If the true function was known (not actually possible, but helps in understanding)

→ true : $V^{\pi}(s)$

our approx : $\hat{v}(s;w)$

MSE : $J(w) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s) - \hat{V}(s;w)\right)^2\right]$

→ use gradient descent :

$$w \leftarrow w - \frac{1}{2}\alpha \nabla_w (J(w))$$

→ SGD : (samples states)

$$\nabla_w J(w) = -\mathbb{E}\left[2\left(V^{\pi}(s) - \hat{v}(s;w)\right)\right] \nabla_w \hat{v}$$

then $w \leftarrow w + \alpha\left(V^{\pi}(s) - \hat{v}^{\pi}(s;w)\right) \nabla_w V$

evaluated at a

- Model free value function approximation.

→ similar to previously seen.

→ ~~instead of~~

→ change the estimate update step to include fitting the function approximator.

→ state feature representation:

$$x(s) = \begin{bmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_n(s) \end{bmatrix}$$

- linear value function approximation (with an oracle)

→ $\hat{V}(s; w) = x(s)^T w$

→ objective fⁿ : $J(w) = E_\pi \left[(V^\pi(s) - \hat{V}(s; w))^2 \right]$

→ weight update : $w \leftarrow w - \frac{1}{2} \alpha \nabla_w J(w)$

ie; $\boxed{w \leftarrow w + \alpha \left(V^\pi(s) - \hat{V}(s; w) \right) x(s)}$

→ but again, we do not have access to an oracle.

M.C

- linear value function approximation (without an oracle)

→ G_t is an unbiased but noisy sample of the true expected return $V^{\pi}(s_t)$

→ algo :

initialize $w = 0, \; k = 1$

loop {

sample k^{th} episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2} \cdots s_{k,L_k})$
↳ given π

for all ● states s {

● for first (or every) visit to (s) in ep k {

$$G_t(s) = \sum_{j=t}^{L_k} r_{k,j} \gamma^{j-t}$$

$$w = w + \alpha \left(G_t(s) - \hat{V}(s; w)\right) x(s)$$

}

}

}

$k = k+1$

}

→ update rule : $\boxed{w \leftarrow w + \alpha \left[G_t(s) - \hat{V}(s; w)\right] \nabla_w \hat{V}}$

- Convergance guarantees for L.V.F.A for policy eval:

→ The MDP will eventually converge to a prob distribution over states $d(s)$ .. (given π)

→ $d(s) \Rightarrow$ stationary distribution over states of π

→ $d(s)$ satisfies:

$$d(s') = \sum_{s} \sum_{a} \pi(a|s) p(s'|s,a) d(s)$$

$$\boxed{d(s') = \sum_{s} \sum_{a} \pi(a|s) p(s'|s,a) d(s)}$$

→ $$MSVE(w) = \sum_{s \in S} d(s) \left[V^{\pi}(s) - \hat{V}^{\pi}(s; w) \right]^2$$

→ MC VFA for a policy converges to the weights w_{mc} which has the min MSVE(w):

$$MSVE(w_{mc}) = \min_{w} \sum_{s \in S} d(s) \left(V^{\pi}(s) - \hat{V}^{\pi}(s; w) \right)^2$$

- Batch Monte Carlo Value function approximation for policy evaluation.

→ May have a set of episodes from π.

→ minimize MSE on this dataset

→ $$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \left[G(s_i) - x(s_i)^T w \right]^2$$

→ take derivative and set to zero:

$$\boxed{w^* = (X^T X)^{-1} X^T G}$$

$X \Rightarrow$ feature matrix: $\begin{bmatrix} x_1(s_1) & x_2(s_1) & \cdots & x_m(s_1) \\ x_1(s_2) & x_2(s_2) & \cdots & x_m(s_2) \\ \vdots & & & \\ x_1(s_N) & x_2(s_N) & \cdots & x_m(s_N) \end{bmatrix}$

$G \Rightarrow$ return vector: $\begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_N \end{bmatrix}$

• TD learning (TD-0) with VFA:

→ target is $r + \gamma \hat{V}^\pi(s'; w)$, a biased approximate estimate of the true value $V^\pi(s)$

→ supervised learning on:

$(s_1, r_1 + \gamma \hat{V}^\pi(s_2; w)), (s_2, r_2 + \gamma \hat{V}^\pi(s_3; w)) \cdots$

→ in linear TD(0):

$$\Delta w = \alpha \left(r + \gamma \hat{V}^\pi(s'; w) - \hat{V}^\pi(s; w) \right) \nabla_w \hat{V}^\pi(s; w)$$

$$\Rightarrow w \leftarrow w + \alpha \left(r + \gamma \hat{V}^{\pi}(s'; w) - \hat{V}^{\pi}(s; w) \right) \cdot x(s)$$

$$\Rightarrow \boxed{w \leftarrow w + \alpha \left(r + \gamma\, x(s')^T w - x(s)^T w \right) x(s)}$$

→ algorithm:

$w = 0$, $k = 1$
loop {
  sample tuple $(s_k, a_k, r_k, s_{k+1}) \sim \pi$
  update weights:
    $w \leftarrow w + \alpha \left(r + \gamma\, x(s')^T w - x(s)^T w \right) x(s)$
  $k = k + 1$
}

→ convergance:
TD(0) policy evaluation with value func approxim converges to weights w_{TD} which is within a constant factor of the min MSVE possible:

$$MSVE(w_{TD}) \leq \frac{1}{1-\gamma} \min_{w} \sum_{s \in S} d(s) \left(\hat{V}^{\pi}(s) - \hat{V}^{\pi}(s; w) \right)$$

→ TD converges faster, but may give higher errors

∴ Control using LVFA

→ can be unstable ∴,

→ $\hat{Q}(s, a; w) = x(s, a)^T w$ •

$$x(s,a) = \begin{bmatrix} x_1(s,a) \\ x_2(s,a) \\ \vdots \\ x_n(s,a) \end{bmatrix}$$

• Incremental MC approach:

$$\Delta w = \alpha (G_t - \hat{Q}(s_t, a_t; w)) \nabla_w \hat{Q}(s_t, a_t; w)$$

• Q- learning:

→ TD target : $r + \gamma \max_{a'} \hat{Q}(s', a'; w)$

$$\Delta w = \alpha (r + \gamma \max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}$$

• SARSA

→ TD target : $r + \gamma \hat{Q}(s', a'; w)$

$$\Delta w = \alpha (r + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}$$

- Convergence of TD methods with VFA

→ bellman operators were contractions, so convergence was guaranteed
however value function approximation fitting can be an expansion.

→ 3 problems: function approximation
bootstrapping
of policy learning

Deep Q-learning

- Similar to previous section, but $\hat{Q}$ & $\nabla_w \hat{Q}$ are computed by DNNs

- Convergance problems

→ 2 issues: correlations b/w samples
: non-stationary targets

→ correlations b/w samples:

$s, a, r, s', a', r', s'' \cdots$

highly correlated
may cause bias

→ non stationary targets: we do not know V^π true, it is constantly changing.

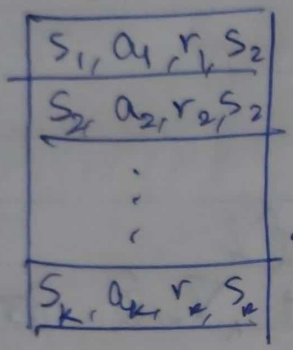
- Address these challenges by:

→ Experience replay & fixed Q-targets

- Experience replay → VERY IMPORTANT!

→ to remove correlations, store a dataset (replay buffer)

→ sample randomly from this



sample randomly each time while updating the weights

- Fixed Q-targets

→ fix the weights for multiple updates (improves stability)

→ let $w^{\ominus}$ denote set of weights used in the target, and w be the weights being updated

PAGE No.
DATE / /

- Combined algo:

initialize w^{Θ}, w, $Q(s,a;w)$, $t=0$, $s_t = s_0$
π_b = ϵ-greedy (Q)
dataset creation
loop {

take $a_t \sim \pi_b(s_t)$

add to dataset (s_t, a_t, r_t, s_{t+1})

choose randomly from dataset (s, a, r, s'):

$$\Delta w = \alpha \left[r + \gamma \max_{a'} \hat{Q}(s', \hat{a}, w^{\Theta}) - \hat{Q}(s', a; w) \right] \cdot \nabla_w \hat{Q}(s, a; w)$$

~~each~~ each 'n' loops { $\bar{w} \leftarrow w$ }

$t = t+1$
π_b = ϵ-greedy (Q)

}

- Double DQN

→ save 2 Q-networks
current Q-net to select actions
old Q-net to evaluate actions

→ $\Delta w = \alpha \left[r + \gamma \hat{Q}\left(s', \underset{a'}{\text{argmax}}\ \hat{Q}(s', a'; w); w^{\Theta}\right) - \hat{Q}(s, a; w) \right] \nabla_w Q$

→ loop {

$a_t \sim$ ϵ-greedy $(Q_1(s_t, a) + Q_2(s_t, a))$

50% chance {

$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha (r_t + Q_1(s_{t+1}, \underset{a'}{\text{argmax}}\ Q_2(s_{t+1}, \cdot) - Q_1(s_t, a_t))$

} else {

}

$t = t + 1$

}

- Priority Replay

→ don't select randomly from buffer.

→ $p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; w^{\Theta}) - Q(s_i, a_i; w) \right|$

choose tuple with ~~largest~~ higher p_i

→ update p_i every update.

→ maybe choose proportional to:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

↓
prob of choosing ith tuple