Intro:

A RL agent can optionally consist 3 things:

1) Model { transitional Probability - deterministic or stochastic $P(S'|S,a)$
rewards - deterministic or stochastic $R(S)$ or $R(S,a)$

- The agent may not have access to model (how world works). This is the distinction between model-based and model-free agent.

2) Policy $\pi : S \to A$ - deterministic or stochastic

3) Value function $V_t(S)$: $E[G_t(S)] = E\left[\sum_{i=t}^{H-1} \gamma^{i-t} r_i\right]$

Horizon (H): length of the episode { $H = \infty$ infinite, no subscript t
$H < \infty$ finite, have subscript t.

- For $H < \infty$, P, R, π, V can be stationary or non-stationary (t-dependent)
- For $H = \infty$, P, R, π, V are all stationary.

If V is stationary, $V(S) = V_0(S) = V_1(S) = \ldots = V_t(S)$ for any t.

Model-Based Agents:

Markov process: (S, P) $P = P(S'|S)$ The key is memoryless

Markov Reward process: (S, P, R, γ) $P = P(S'|S)$

$H = \infty$: $V(S) = R(S) + \gamma \sum_{S' \in S} P(S'|S) V(S')$

- Analytical soln: $V = R + \gamma P V \to V = (I - \gamma P)^{-1} R \to O(|S|^3)$
- Iterative soln: $V_{new} = R + \gamma P V$ until $\|V_{new} - V\|_\infty < \varepsilon$

$H < \infty$: $V_t(S) = R(S) + \gamma \sum_{S' \in S} P(S'|S) V_{t+1}(S')$ w/ stationary P, R

- DP soln: $V_H(S) = 0$ for all S, compute $V_t(S)$ backwards.

Markov Decision Process: (S, A, P, R, γ) $P = P(S'|S,a)$

$V_t^\pi(s) = E_\pi[G_t \mid S_t = s]$ for $H < \infty$ $\quad V^\pi(s) = V_0^\pi(s)$ for $H = \infty$.

$Q_t^\pi(s,a) = E_\pi[G_t \mid S_t = s, a_t = a]$ for $H < \infty$ $\quad Q^\pi(s,a) = Q_0^\pi(s,a)$ for $H = \infty$

- $V_t^\pi(s)$ is $E[G_t]$ when taking actions according to π at $i \geq t$
- $Q_t^\pi(s,a)$ is $E[G_t]$ when $a_t = a$ and taking actions according to π at $i \geq t+1$.

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

Two algorithms to find π^* for MDP: (for both deterministic and stochastic)

1) Policy Iteration: Compute optimal value and policy iteratively

- based on monotonic improvement in π $\quad V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$

```
def policy_iteration(P, R, γ, ε):
    while (π* != π).any():
        π = π*
        V^π = policy_eval(P, R, π, γ, ε)
        π* = policy_improve(P, R, V^π, γ)
    return V^π, π
```

O(|S|) `def policy_eval(P, R, π, γ, ε):` * similar to iterative soln for MRP

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

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

```
while ||V - V_old||_∞ > ε:
    V_old = V
    V(s) = R^π(s) + γ Σ_{s'∈S} P^π(s'|s) V(s')   ∀ s ∈ S
return V
```

O(|S|) `def policy_improve(P, R, V^π, γ):`

$$\hat{\pi}(s) = \underset{a \in A}{\operatorname{argmax}} \left[R(s,a) + \gamma \sum_{s \in S} P(s' \mid s,a) V^\pi(s') \right] \quad \forall s \in S$$

```
return π̂(s)
```

- policy_improve() is based on $\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right]$

2) Value iteration: compute value function iteratively, then find π^*

- based on $V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right] = V^{\pi^*}$

```
def value_iteration(P, R, γ, ε):
    while ||V - V_old|| > ε:
        V_old = V
        V(s) = max_{a∈A} [R(s,a) + γ Σ_{s'∈S} P(s'|s,a) V(s')]    ∀ s ∈ S
    π* = policy_improve(P, R, V, γ)
    return V, π*
```

Above function are all for $H = \infty$, thus, V is stationary. For $H < \infty$ and non-stationary V_t, we could use DP instead of iterative case.

e.x.

```
def value_iteration(P, R, γ, H):
    V*_H(s) = 0    ∀ s ∈ S
    for t in range(H, -1, -1):
        V*_t(s) = max_{a∈A} [R(s,a) + γ Σ_{s'∈S} P(s'|s,a) V*_{t+1}(s)]  ∀ s ∈ S
        π*_t(s) = argmax_{a∈A} [R(s,a) + γ Σ_{s'∈S} P(s'|s,a) V*_{t+1}(s)]  ∀ s ∈ S
    return V*_t and π*_t    ∀ t = 0..H and s ∈ S
```

Model-free Agents:

Policy Evaluation: 2 methods: Monte-Carlo (MC) or Temporal Difference (TD(0))

1) MC: first-visit (update only for 1st visit to s in each episode) vs. every-visit

$V^{\pi}(s_{j,t}) = V^{\pi}(s_{j,t}) + \alpha \left(G_{j,t} - V^{\pi}(s_{j,t}) \right)$ $\quad s_{j,t}$ = j^{th} episode, t^{th} time step.

$\alpha = \frac{1}{N(s_{j,t})}$ normally, $\alpha > \frac{1}{N(s_{j,t})}$ if more weight for recent occurance.

Properties: episodic only, unbiased (first-visit) and biased (every-visit), data efficient

high variance, need to wait for the end of episode to update.

2) TD(0): Combination of MC and DP (bootstrapping)

Instead of G_t, we use $r_t + \gamma V^{\pi}(S_{t+1})$ as the approximation of G_t

properties: works for non-episodic environment too, biased, Markov assumption,

low variance, can update as soon as a state is visited, obviously smaller

update compared to MC since it perform many updates in 1 episodes.

- Batch MC and TD(0): we can think of this as mini-batch whereas normal MC and TD(0) as SGD. we accumulate several episode of history, find the avg then update.

$$\hat{P}(s' \mid s, a) = \frac{1}{N(s,a)} \sum_{j=1}^{n} \sum_{t=1}^{L_j - 1} 1(s_{j,t} = s, a_{j,t} = a, s_{j,t+1} = s')$$

$$\hat{r}(s,a) = \frac{1}{N(s,a)} \sum_{j=1}^{n} \sum_{t=1}^{L_j - 1} 1(s_{j,t} = s, a_{j,t} = a)\, r_{j,t}$$

Control: (Policy optimization)

- In control, we work w/ $Q(s,a)$ instead of $V(s)$, we can update $Q(s,a)$ in the exact same ways as updating $V(s)$ above (in policy evaluation).

- MC: $Q^{\pi}(s_t, a_t)$ = avg of G_t $(s = s_t, a = a_t)$

TD(0):

- SARSA: $Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha_t [r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t)]$
- Q-Learning: $Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q^{\pi}(s_{t+1}, a') - Q^{\pi}(s_t, a_t)]$

- Since we don't know the model, we can't simply do exploitation every time, we need to do exploration → ε-greedy algorithm: $\pi(s) = \begin{cases} \text{random } a \text{ w/ prob } \frac{\varepsilon}{|A|} \\ \underset{a \in A}{\text{argmax}}\, Q^{\pi}(s,a) \text{ w/ prob } 1-\varepsilon \end{cases}$

ε ↓ as time goes, typically $\varepsilon = \frac{1}{t}$.

- ε-greedy algorithm is used as the policy Improvement ✌

Off-policy Evaluation & Control: when it's expensive to apply a policy.

- on-policy = apply π_1, update Q^{π_1} - off-policy = apply π_2, update Q^{π_1}

- Importance Sampling : say we want $E_{x\sim q}[f(x)]$ but we can only get $E_{x\sim p}[f(x)]$

in RL setting, $q = \pi_1$, $p = \pi_2$, x = state, $f(x) = V^{\pi_1}(s)$ or Q^{π_1}

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

MC: $V^{\pi_1}(s) \approx \frac{1}{n}\sum_{j=1}^{n} \frac{P(h_j|\pi_1, s)}{P(h_j|\pi_2, s)} G(h_j) = \frac{1}{n}\sum_{j=1}^{n} G(h_j) \prod_{t=1}^{L_j-1} \frac{\pi_1(a_{j,t}|s_{j,t})}{\pi_2(a_{j,t}|s_{j,t})}$

where $P(h_j|\pi, s) = \prod_{t=1}^{L_j-1} \pi(a_{j,t}|s_{j,t}) P(r_{j,t}|s_{j,t}, a_{j,t}) P(s_{j,t+1}|s_{j,t}, a_{j,t})$

TD(0): $V^{\pi_1}(s) \approx V^{\pi_1}(s) + \alpha\left[\frac{\pi_1(a|s)}{\pi_2(a|s)}(r + \gamma V^{\pi_1}(s') - V^{\pi_1}(s))\right]$

- Q-learning is an off-policy control because $a_{t+1} = \underset{a'}{\text{argmax}}\, Q^{\pi}(s_{t+1}, a')$ instead of $a_{t+1} = \pi(s_{t+1})$

Maximization Bias: when policy is not biased but we are systematically biased (+ve) wrt $V^*(s)$ $\quad \hat{V}(s) \geq V^*(s)$

- Use Double Q-learning (significant speed up during training)

Have 2 Q functions, during training update one of Q using the $\underset{a'}{\text{argmax}}$ of the other Q.

take ε-greedy wrt $Q_1 + Q_2$.

Value Function Approximation (VFA): $\hat{v}(s, w) \approx V(s)$ $\quad \hat{q}(s, a, w) \approx Q(s, a, w)$

- For large / infinite state space, impossible to use tabular approach.
- Typical functions : linear, decision tree, nearest neighbors, neural network ...
- Typically a regression problem, loss function $J(w) = E_{\pi}[(q_{\pi}(s,a) - \hat{q}(s,a,w))^2]$

optimize through gradient descent, $w = w + \alpha \Delta w$

- MC: $\Delta w = (G_t(s) - \hat{q}_t(s, w)) \nabla_w \hat{q}(s,a,w)$
- SARSA: $\Delta w = (r + \gamma \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s,a,w)$
- Q-learning: $\Delta w = (r + \gamma \max_{a'} \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s,a,w)$

Linear Feature Representation: $\hat{q}(s,a,w) = w^T x(s) = \sum_{j=1}^{\infty} w_j x_j(s) \rightarrow \nabla_w \hat{q}(s,a,w) = x(s)$

Neural Network : $\nabla_w \hat{q}(s,a,w)$ can come from back propagation

For linear VFA, MC and SARSA are guaranteed to converge to the min MSE.

For non-linear VFA, we don't have guarantee for any of the 3 methods.

Deep Q-learning:

- Deep Q-Network (DQN): similar to q-learning except w/ the added CNN feature exactor
  - In Atari paper, besides the preprocessing steps, 2 additional tricks are proposed.
    - Experience Replay: randomly pick a batch of (s_t, a_t, r_t, s_{t+1}) from memory
      - Greater data efficiency, less correlation, less oscillation / divergence.
      - Can use prioritized replay to more often update problematic (s, a, s').

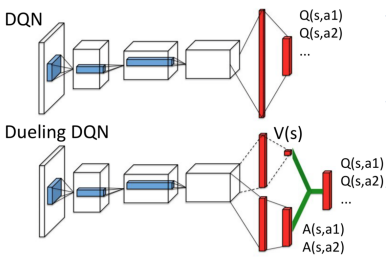
$$P_i = |r + \gamma \max_{a'} \hat{q}'(s_{i+1}, a'; w^-) - Q(s_i, a_i; w)| \qquad p(i) = \frac{P_i^\alpha}{\sum_k P_k^\alpha}$$ ← probability of being chosen.

  - Target Network: keep 2 sets of weights. w^- is the target, update very C steps to w. w is update at each step.
    - Avoid "chasing a moving target" → hyperparameter C.

Algorithm 1 deep Q-learning

Initialize replay memory D with a fixed capacity
Initialize action-value function $\hat{q}$ with random weights $\mathbf{w}$
Initialize target action-value function $\hat{q}$ with weights $\mathbf{w}^- = \mathbf{w}$
for episode $m = 1, \ldots, M$ **do**
  Observe initial frame x_1 and preprocess frame to get state s_1
  for time step $t = 1, \ldots, T$ **do**
    Select action $a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a \hat{q}(s_t, a, \mathbf{w}) & \text{otherwise} \end{cases}$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Preprocess s_t, x_{t+1} to get s_{t+1}, and store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample uniformly a random minibatch of N transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = r_j$ if episode ends at step $j+1$;
    otherwise set $y_j = r_j + \gamma \max_{a'} \hat{q}(s_{j+1}, a', \mathbf{w}^-)$ ← wrt w^-
    Perform a stochastic gradient descent step on $J(\mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{q}(s_j, a_j, \mathbf{w}))^2$ w.r.t. parameters $\mathbf{w}$
    Every C steps reset $\mathbf{w}^- = \mathbf{w}$

- Double DQN: similar to double q-learning
  - DQN target: $r + \gamma \max_{a'} \hat{q}(s', a', w^-)$
  - DDQN target: $r + \gamma \hat{q}(s', \underbrace{\arg\max_{a'} \hat{q}(s', a', w)}_{\text{for picking policy}}, \underbrace{w^-}_{\text{for evaluating policy}})$
- Dual DQN: $\hat{q}(s, a, w, w_v, w_A) = \hat{v}(s, w, w_v) + \hat{A}(s, a, w, w_A)$
  - Advantage function: $A^{\pi}(s, a) = V^{\pi}(s) - Q^{\pi}(s, a)$
    - Measure each action's effect on the value function.



- $A(s,a^*) = 0 \qquad A(s,a) \geq 0 \quad \text{for } a \neq a^*$

- To make Q-function identifiable:

$$\hat{q} = \hat{V}(s,w,w_v) + \left(\hat{A}(s,a,w,w_A) - \max_{a' \in \mathcal{A}} A(s,a',w,w_A)\right) \quad \text{or}$$

$$\hat{q} = \hat{V}(s,w,w_v) + \left(\hat{A}(s,a,w,w_A) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a',w,w_A)\right)$$

Imitation Learning:

- we have an human to give a few demonstrations to the RL agent $\rightarrow \pi^*$ for limited states.
- $P(s'|s,a)$ may / may not be known, reward function is not known.

3 types of Imitation Learning:

1) Behavioural Cloning: like supervised learning $\rightarrow$ Given s, predict a.

drawbacks:
- In supervised learning, we assume data is i.i.d. In RL, data is highly correlated.
- Agent doesn't know what to do when entered in an unseen state
- Error is compounding in RL, unlike SL.

2) Inverse RL: recover the reward function from human's demonstrations.

- use function approximation, say $R(s) = w^T x(s)$ $\quad V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s\right] = w^T \mu(\pi)$

find w s.t. $\mathbb{E}_{\pi^*}\left[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) | s_0 = s\right] \geq \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) | s_0 = s\right] \quad \forall \pi$

a.k.a. $w^{*T} \mu(\pi^* | s_0 = s) \geq w^{*T} \mu(\pi | s_0 = s) \quad \forall \pi, \forall s$

3) Apprenticeship learning: Generate a good policy from the recovered reward

$\|\mu(\pi|s_0 = s) - \mu(\pi^*|s_0 = s)\|_1 \leq \varepsilon$ complicated, see notes

---

↑ Value-based Approaches: find optimal value at each state, policy = picking a that produced max V

↓ Policy-based Approaches: find optimal policy directly by optimizing V wrt π.

- Policy-based approaches can be applied to stochastic policy. It's guaranteed to converge to local min. Effective in high-dimensional / continuous state. Not sample-efficient because once policy is updated, we can't use the previous episodes $\rightarrow$ can be solved by Importance Sampling.

Policy-Based Approaches:

Given a trajectory $\tau : (s_1, a_1, \dots, s_T, a_T)$, probability of it under policy π_θ is

$$\pi_\theta(\tau) = P(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)\, P(s_{t+1}|s_t, a_t) \quad \text{where} \quad \pi_\theta(a_t|s_t) = P(a_t|s_t;\theta)$$

Objective function is the value function: $\theta^* = \underset{\theta}{\operatorname{argmax}}\, E_{\tau\sim\pi_\theta(\tau)}[V^{\pi_\theta}(s_1)]$

- Finite horizon: $J(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}[s_1]$ (fixed initial state)

  $J(\theta) = \sum_s d(s)\, E_{\pi_\theta}[s_1]$ (stochastic initial state)

- Infinite horizon: $J(\theta) = \sum_s d(s)\, E_{\pi_\theta}[s]$

$$V(\theta) = \sum_\tau P(\tau;\theta) R(\tau) = \sum_\tau P(\tau;\theta) \sum_{t=1}^{T} \gamma^t r(s_t, a_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^t r(s_t^{(i)}, a_t^{(i)})$$

$$\nabla_\theta V(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) \sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}^{(i)}, a_{t'}^{(i)}) \right] \rightarrow \text{high variance}$$

Reinforce Algorithm: For $\tau^{(i)}$ in $i = 1 \dots N$, For $t = 1 \dots T$, $\theta = \theta + \alpha \nabla_\theta V(\theta)$

Baseline: $\nabla_\theta V(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\, A_t^{(i)} \right] \rightarrow$ lower variance.

$$A_t^{(i)} = \sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}^{(i)}, a_{t'}^{(i)}) - b(s_t) \leftarrow \text{doesn't affect result as long as } b = f(s_t) \text{ only.}$$

A natural choice for $b(s_t)$ is $V(s_t) \approx \hat{V}(s_t, w)$

Actor-Critic Algorithm: developed from baseline intuition → learn θ and w simultaneously

- Learn w by minimizing $b(s_t, w) - G_t \rightarrow \min_w \sum_{i=1}^{N} \sum_{t=1}^{T} (b(s_t, w) - G_t)^2$ → critic
- Learn θ by $\theta = \theta + \alpha \nabla V_\theta$ as shown above → actor

off-policy Policy Gradient: as we update the policy, we are under a different distribution.

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] \rightarrow J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} r(\tau)\right]$$

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

This allows us to use the replay memories.

N-Step Estimator: The $J(\theta)$ and $\nabla_\theta J(\theta)$ above are based on MC, we can use TD as well.

$\hat{G}_t^{(1)} = r_t + \gamma V(s_{t+1})$ → high bias, low variance (TD(0))

$\hat{G}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$ n-step is a hyperparameter

$\hat{G}_t^{(\infty)} = G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \rightarrow$ low bias, high variance. (MC)

[Discrete action space: $\pi_\theta(a|s) = \frac{\exp[\theta^T \phi(s,a)]}{\sum_{a'} \exp[\theta^T \phi(s,a')]}$ $\nabla_\theta \log \pi(a|s) = \phi(s,a) - \sum_{a'} \pi(a'|s)\phi(s,a')$

Continuous action space : $a \sim N(\mu(s), \sigma^2)$ where $\mu(s) = \theta^T \phi(s)$ $\nabla_\theta \log \pi_\theta(a|s) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$

Trust Region Policy Optimization : Solves the problem that it's hard to pick appropriate learning rate.

- Complicated, see note for details.

Exploration vs. Exploitation :

- Pure exploitation : stuck in suboptimal policy e.x. Greedy
- Pure exploration : never use learned information to make good decision.

- Computationally Efficient : Compute a decision in a reasonable amount of time e.x. A.V.
- Sample Efficient : Compute a decision w/ a reasonable amount of data e.x. medical

- Greedy, ε-Greedy are not good. Decay ε-greedy, being optimistic are good algorithms

Optimistic : choose action that may have the best upper bound $P(Q(a) > \hat{Q}(a) + u) < \delta$

- actions w/ high uncertainty (haven't been visited much) are often selected.
- Intuition : if action is the best action → perfect, else → we gained info and adjust mean and uncertainty.
- Optimistic initialization : assign all $Q(s,a)$ to be $\frac{R_{max}}{1-\gamma}$ (best it could be)

Multi-Armed Bandit (MAB) : Simpler than MDP, only 1 state so everything is i.i.d.

- Evaluation metric : regret
  - Action value : $Q(a) = E[r|a]$ - optimal : $V^* = Q(a^*) = \max_{a \in A} Q(a)$
  - Gap $\Delta_a = V^* - Q(a)$ - Regret : $L_t = E[V^* - Q(a)]$
  - Objective : minimize total regret : $L_t = E[\sum_{t=1}^{T} V^* - Q(a_t)] = TV^* - E[\sum_{t=1}^{T} Q(a_t)]$

Upper Confidence Bound (UCB) :

- pick $a_t = \underset{a \in A}{\operatorname{argmax}} \{\hat{Q}_t(a) + \hat{U}_t(a)\}$ where $\hat{u}_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}} = \sqrt{\frac{2\log t}{N_t(a)}}$ $p = t^{-4}$

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t \mathbb{1}(a_t = a)$$

- $\hat{U}_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$ is based on Hoeffding's Inequality $P(E[X] > \hat{X}_t + u) \leq \exp(-2Nu^2)$
- The only assumption is that reward is bounded.

Bayesian Bandit :

- Assumes rewards follows a known distribution, e.x. beta distribution for bernoulli event
- we could use UCB approach. e.x. For Gaussian, $a_t = \underset{a \in A}{\operatorname{argmax}} \left\{ \mu_a + c\frac{\sigma_a}{\sqrt{N(a)}} \right\}$

  $P[\mu_a, \sigma_a^2 | h_t] \propto P[\mu_a, \sigma_a^2] \prod_{t|a_t=a} N(r_t; \mu_a, \sigma_a^2)$
- we could use sampling instead of the upper bound.
  - Thompson sampling: sample the reward for each action, pick the max reward and update distribution. For beta (α, β), α ++ if reward =1 or β ++ if reward =0

Provably Approximately Correct (PAC):

- Minimizing regret doesn't help us to determine whether we make infrequent large mistakes or frequent small mistakes. The former is worse.
- Defn: An algorithm is PAC if $P[Q(a) \geq Q(a^*) - \varepsilon] \geq 1 - \delta$ on all but a polynomial number, usually in terms of ε, δ, N.
- UCB, TS are PAC algorithms, greed is not.

- Generalization from MAB to MDP involves model approximation, but most is similar.
  - There are different states, so $P(s'|s,a)$ needs to be approximated
  - Instead of approximating $Q(a)$, we needs to approximate $Q(s,a)$.

Safe RL: Combination of off-policy Evaluation / Control and sample-efficient algorithms

- an application of Safe RL is the Medical Industry, we can't cheaply change treatment on patients (off-policy methods) and we want figure out optimal solution quickly (sample-efficient algorithms).
- Defn: Given historical data $D = \{s_1, a_1, r_1, \ldots\}_{i=1}^{101}$ and current policy π_b, can we find new policy π_e s.t. $P(V^{\pi_e} > V^{\pi_b}) \geq 1 - \delta$ where $V^{\pi} = E\left[\sum_{t=1}^{L} \gamma^t R_t | \pi\right]$

Importance Sampling:

- Recall: $V^{\pi_e}(s) \approx \frac{1}{n} \sum_{i=1}^{n} \frac{P(h_i|\pi_e, s)}{P(h_i|\pi_b, s)} G(h_j) = \frac{1}{n} \sum_{i=1}^{n} \left(\prod_{t=1}^{L_i} \frac{\pi_e(a_t^{(i)}|s_t^{(i)})}{\pi_b(a_t^{(i)}|s_t^{(i)})} \right) \left(\sum_{t=1}^{L_i} \gamma^t R_t^{(i)} \right)$
- Variations that achieves better results:
  - per-Decision IS: $V^{\pi_e}(s) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=1}^{L_i} \left(\gamma^t R_t^{(i)} \prod_{t=1}^{t} \frac{\pi_e(a_t^{(i)}|s_t^{(i)})}{\pi_b(a_t^{(i)}|s_t^{(i)})} \right)$ intuition: future action can't affect past rewards.

- weighted IS: Let $w_i = \frac{L_i}{\prod_{t=1}} \frac{\pi_e(a_t^{(i)}|s_t^{(i)})}{\pi_b(a_t^{(i)}|s_t^{(i)})}$, $V^{\pi_e}(s) = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i \left(\sum_{t=1}^{L_i} \gamma^t R_t^{(i)} \right)$

- Doubly Robust IS: Combined raw return $R_t^{(i)}$ w/ $\hat{Q}^{\pi_e}(S_t^{(i)}, A_t^{(i)})$

  - IS is not biased but high variance. Model based estimate ($Q(s,a)$) is biased but low var

  - Let $X = R$, $Y = \hat{Q}^{\pi_e}$ and $E[Y] = \hat{V}^{\pi_e}$, $E[X - Y + E[Y]] = E[X]$

$$V^{\pi_e}(s) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{L_i} \gamma^t w_t^{(i)} \left(R_t^{(i)} - \hat{Q}^{\pi_e}(S_t^{(i)}, a_t^{(i)}) \right) + \gamma^t \rho_{t-1}^{(i)} \hat{V}^{\pi_e}(S_t^{(i)}) \quad \text{where } w_t^{(i)} = \frac{t}{\prod_{t=1}} \frac{\pi_e(a_t^{(i)}|s_t^{(i)})}{\pi_b(a_t^{(i)}|s_t^{(i)})}$$

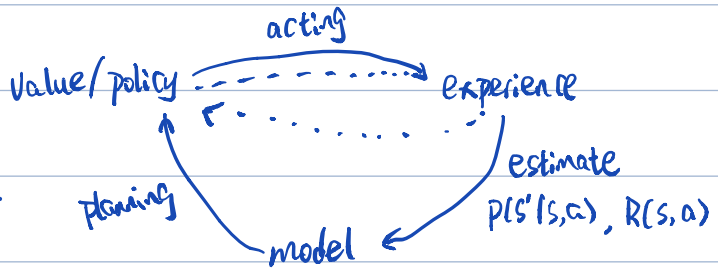
- Limitation: only works when ($\pi_b(a|s) > 0$ when $\pi_e(a|s) > 0 \quad \forall s, a$)

Hoeffding's Inequality:

$$E[X_i] \geq \frac{1}{n} \sum_{i=1}^{n} X_i - b\sqrt{\frac{\ln(1/\delta)}{2n}} \qquad X_i = w_i \sum_{t=1}^{L} \gamma^t R_t^{(i)} \qquad w_i = \frac{L_i}{\prod_{t=1}} \frac{\pi_e(a_t^{(i)}|s_t^{(i)})}{\pi_b(a_t^{(i)}|s_t^{(i)})}$$

$b = \max_i (G_i w_i)$ ← Could explode as w_i could be very large if $\pi_e(a_t^{(i)}|s_t^{(i)}) >> \pi_b(a_t^{(i)}|s_t^{(i)})$

Model-Based RL:



model free ---

model based —

model-based is usually more sample efficient

- Just like estimate $\hat{Q}(s,a,w)$ and $\hat{\pi}_\theta(s)$, we need to first define a function approximation (tabular, linear, NN ...) and a loss function (MSE, KL Divergence...).

MC Tree Search: simulation-based approach for the planning stage.

- To avoid exponential growing # of nodes, we used simulation and we always use the current state as the root, so each time we are solving a sub MDP starting from the current state.

Algorithm 1 General MCTS algorithm

```
function MCTS(s_0)
    Create root node v_0 corresponding to s_o
    while within computational budget do
        v_k ← TreePolicy(v_o)
        Δ ← Simuation(v_k)
        Backprop(v_k, Δ)
    return arg max_a Q(s_o, a)
```

Tree policy: pick the immediate leaf from current state

- e.x. $a = \underset{a \in A}{\text{argmax}}\, Q(v_0, a) + \sqrt{\frac{2 \log N(v)}{N(v,a)}}$

Simulation / Rollout policy: simulate until termination.

- e.x. random, follow the current policy, use a NN ...

Backprop step updates $N(s)$, $N(s,a)$, $Q(s,a)$

- Advantage of MC TR: 1) parallerization 2) partial MDP 3) no exponential growth due to sampling

4) no domain-specific engineering as we are just sampling.

- For multiplayer games (e.x. GO), we can have self-play. Say $\pi = (\pi_B, \pi_W)$ (black, white) and $V^{\pi}(s) = P[\text{Black wins} \mid s]$, then $V^*(s) = \max_{\pi_B} \min_{\pi_W} V^{\pi}(s)$. when picking tree policy, we use UCB for black moves and LCB for white moves ($a = \arg\max_{a \in A} Q(v_0, a) - \sqrt{\frac{2 \log N(v)}{N(v,a)}}$)