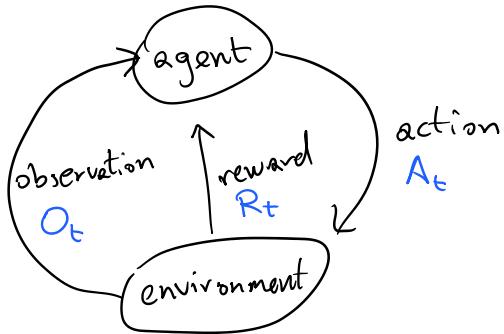


► Intro



- Markov property → the next state is only dependent on the current state, not past states

$$P(S_{t+1} | S_t) = P(S_{t+1} | S_1, \dots, S_t)$$

- fully observable: $O_t = S_t$

- POMDPs → partially observable states

$$S_t = \sigma(S_{t-1} w_s + O_t w_o)$$

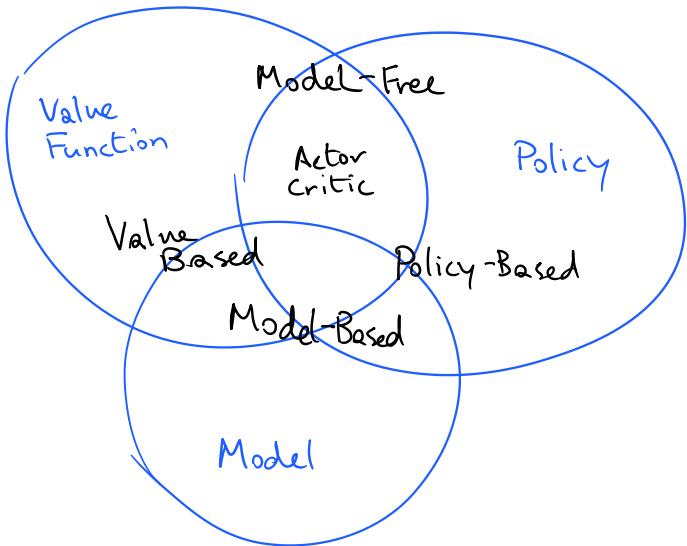
↳ use RNN

- Dynamics Model

$$M(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a)$$

$$R(s, a) = E(R_{t+1} | S_t = s, A_t = a)$$

- RL Agent Taxonomy



- Fundamental Problems
 - Learning → learn unknown dynamics model
→ improve policy based on experience
 - Planning → given known dynamics, find optimal policy
- Exploration vs Exploitation → exploration may seem suboptimal
but it might lead to finding even better states
 - Exploration → gain new info about environment
 - Exploitation → maximize reward w/ known info
- Prediction vs Control
 - Prediction → evaluate given policy
→ find true value of states
 - Control → find the best policy

► MDPs

- gain → total reward w/ discount
- $$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$
- Markov Reward Process (MRP) → no actions
 - value function
- $$V(s) = E(G_t | S_t = s)$$

$$= E(R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s)$$

- Bellman Equation

$$v(s) = R_s + \gamma \sum_{s' \in S} M(s, s') v(s')$$

prob going from s to s'

↳ matrix form

$$\bar{v} = \bar{R} + \gamma \hat{M} \bar{v}$$

$$\bar{v} = (I - \gamma \hat{M})^{-1} \bar{R} \rightarrow \text{solution}$$

- Markov Decision Process (MDP)

- policy

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

↳ MRP given a policy

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

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

- Q-function → action-value

$$Q_\pi(s, a) = E_\pi(C_t \mid S_t = s, A_t = a)$$

- Bellman Expectation Equation

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) Q_\pi(s, a)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} M(s, a, s') v_\pi(s')$$

- Bellman Optimality Equation

$$v^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = R(s, a) + \sum_{s' \in S} M(s, a, s') v^*(s')$$

- Additional Topics \rightarrow MDPs
 \rightarrow POMDPs
 \rightarrow Ergodic MDPs

► Dynamic Programming

- DP \rightarrow the problem must have a sequential component
 \rightarrow breakdown problem into subproblems & combine solutions
- this is all assuming agent knows $M(s, a, s')$ & $R(s, a)$
 \hookrightarrow how to solve for $v^*(s)$ & $Q^*(s, a)$ when S is large
- Policy Evaluation

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in S} M(s, a, s') v_k(s') \right)$$

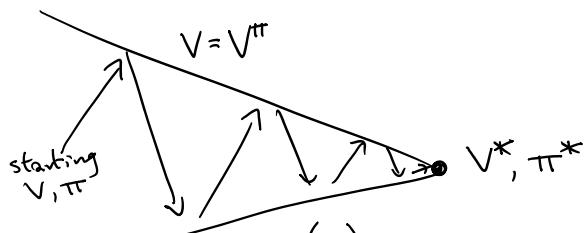
$$\bar{v}^{k+1} = \bar{R}^\pi + \gamma \bar{M}^\pi \bar{v}^k$$

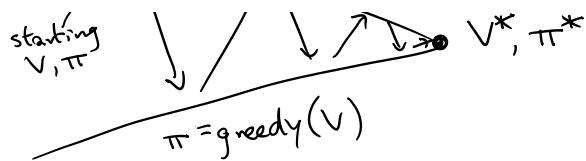
- Improving Policy \rightarrow greedily

$$\pi'(s|a) = \underset{a \in A}{\operatorname{argmax}} \left[R(s, a) + \gamma \sum_{s' \in S} M(s, a, s') v_\pi(s') \right]$$

- Policy Iteration

- algorithm:
 - initialize π (arbitrarily)
 - repeat until convergence ($\pi = \pi'$)
 - estimate v^π by evaluating policy
 - improve policy $\pi' = \text{greedy } (\pi^\pi)$
 - $\pi' \leftarrow \pi$





- Value Iteration \rightarrow skip policy eval

$$v_{k+1}(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} M(s, a, s') v_k(s') \right]$$

- DP Algorithms

<u>Problem</u>	<u>Bellman Eqn.</u>	<u>Algorithm</u>
Prediction	Expectation	Policy Eval
Control	Expectation + Greedy Improvement	Policy Iteration
Control	Optimality	Value Iteration

- Async DP

- in-place DP \rightarrow analogue to SGD
- prioritized sweeping \rightarrow update based on Bellman error
- real-time DP \rightarrow update only those states agent visits
- Problem Size \rightarrow only works up to $|S| \sim \leq 6$
 - For larger problems use sample backups instead full-backups

► Prediction

- Monte Carlo \rightarrow update $V(s)$ after episode

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

$$G_t = \sum_{\text{episode}} R_t$$

- Temporal Difference \rightarrow adjust $V(s)$ after each action \rightarrow bootstrapping

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- MC vs TD

<u>TD</u>	<u>MC</u>
bootstraps	learns at the end of an episode
online or episodic	episodic only
learn from incomplete sequences	high variance
works in non-terminating systems	no bias
low variance	Converges to min mean squared error of
some bias	$\sum \sum (C_t^k - V(S_t^k))^2$
converges to Markov model solution	change depends on many random actions
uses Markov property	starting state doesn't matter

- $TD(\lambda) \rightarrow$ generalizing MC & TD

- use eligibility function to rank how much you can learn about other states based on your current experience

↳ using both α frequency & recency heuristic

$$E_o(s) \leftarrow 0$$

$$E_{t+1}(s) \leftarrow \gamma \lambda E_t(s) + \mathbb{1}(S_t = s)$$

- now adjust all states each step according to eligibility

$$S_t = R_t + \gamma V(S_{t+1}) - V(S_t) \quad \text{TD error}$$

$$V(s) \leftarrow V(s) + \alpha S_t E_t(s)$$

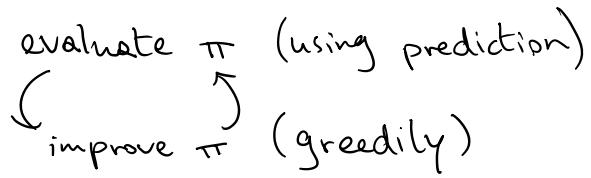
- MC is equivalent to $TD(1)$

- TD is equivalent to $TD(0)$

► Control

- now learn best policy

↳ naive approach



- greedy policy improvement \rightarrow given dynamics & value function

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}} \quad R(s, a) + M(s, a, s') V(s')$$

↳ model-free

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}} \quad Q(s, a)$$

- Exploration vs Exploitation $\rightarrow \varepsilon$ -greedy action selection

- w/ prob. $1-\varepsilon$ choose greedy action

- w/ prob. ε choose random action

- Monte-Carlo Control \rightarrow policy iteration

- policy eval \rightarrow MC $Q \leftarrow Q^\pi$

- policy improvement $\rightarrow \varepsilon$ -greedy

- gradually $\downarrow \varepsilon$ in ∞ limit \rightarrow GLIE

↳ in k^{th} episode w/ π

↳ for each S_t & A_t in episode

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(R_t - Q(S_t, A_t))$$

$$\varepsilon \leftarrow \frac{1}{k}$$

$$\pi(s) \leftarrow \underset{a \in A}{\operatorname{argmax}} \quad Q(s, a)$$

- SARSA \rightarrow TD(0) control
 \rightarrow on policy \rightarrow choose future actions using π

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
    Initialize  $S$ 
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Repeat (for each step of episode):
        Take action  $A$ , observe  $R, S'$ 
        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
         $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
         $S \leftarrow S'; A \leftarrow A'$ ;
    until  $S$  is terminal
  
```

- SARSA(λ) \rightarrow analogue of TD(λ)
 \rightarrow eligibility list to learn in advance

```

Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
Repeat (for each episode):
     $E(s, a) = 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
    Initialize  $S, A$ 
    Repeat (for each step of episode):
        Take action  $A$ , observe  $R, S'$ 
        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
         $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$ 
         $E(S, A) \leftarrow E(S, A) + 1$ 
        For all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :
             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$ 
             $E(s, a) \leftarrow \gamma \lambda E(s, a)$ 
         $S \leftarrow S'; A \leftarrow A'$ 
    until  $S$  is terminal
  
```

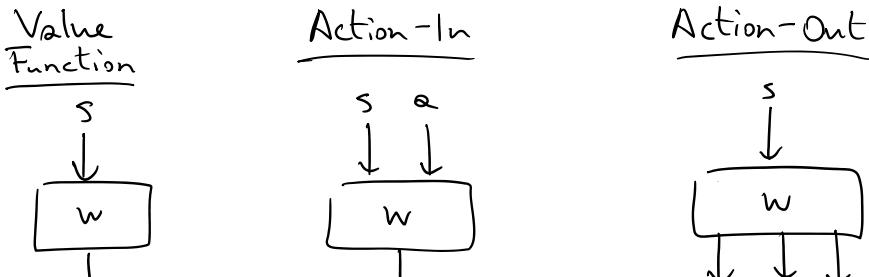
- Q-Learning \rightarrow on or off policy

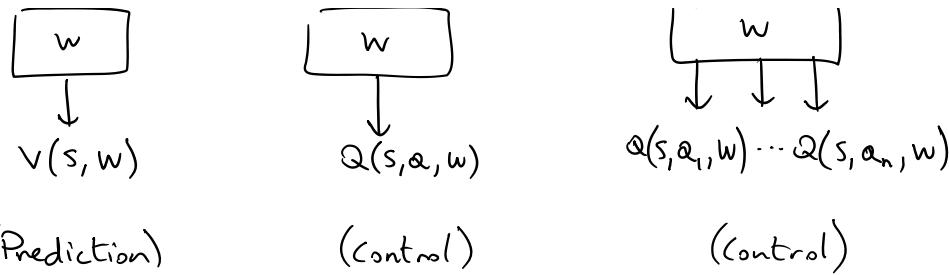
$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \max_{A'} \gamma Q(S', A') - Q(S, A))$$

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
    Initialize  $S$ 
    Repeat (for each step of episode):
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
        Take action  $A$ , observe  $R, S'$ 
         $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
         $S \leftarrow S'$ ;
    until  $S$  is terminal
  
```

- Value Function Approximation
 - for problems w/ too many states for table look-up \rightarrow e.g. continuous
 - Types





- Types of Approximators
 - linear combinations of features
 - neural network
 - decision trees
 - nearest neighbor
 - Fourier/wavelet bases
- Linear Function Approximation

$$v(s, \bar{w}) = \bar{x}(s) \cdot \bar{w}$$

- Loss

$$L(\bar{w}) = E_{\pi} \left[(v_{\pi}(s) - v(s, \bar{w}))^2 \right]$$

true value

- update rule

$$\nabla_w v(s, w) = \bar{x}(s)$$

$$\Delta \bar{w} = \alpha (v_{\pi}(s) - v(s, \bar{w})) \bar{x}(s)$$

- incremental method \rightarrow approximate v_{π} by experience

$$\Delta \bar{w} = \alpha (R_{t+1} + \gamma v(S_{t+1}, \bar{w}) - v(S_t, \bar{w})) \bar{\nabla}_w v(s, \bar{w})$$

- same idea w/ Q function

- Convergence Issues

- prediction

gradient TD follows true ∇ of Bellman eqn.

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

- control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

(✓) = chatters around near-optimal value function

- Experience Replay DQN → batch method → reuse training data

- take action → ϵ -greedy
- store transition in replay memory $(s_t, a_t, r_{t+1}, s_{t+1})$
- sample random mini batches from memory
- use 2 networks → fix one to ↑ stability
→ train the other w/ Q values from fixed network
- optimize MSL loss w/ SGD

$$L_i(\bar{w}_i) = E_{(s, a, r, s') \sim D_i} \left[(r + \gamma \max_{a'} Q(s', a', \bar{w}_i^-) - Q(s; a, \bar{w}_i))^2 \right]$$

- Linear Analytical Solution

$$\bar{w} = (\bar{x}(s_i) \bar{x}(s_j))^{-1} \bar{x}(s_i) v_i^\pi$$

$$\begin{aligned} \hookrightarrow \text{don't know } v_t^\pi &\rightarrow \text{MC} \rightarrow v_t^\pi \approx G_t \\ &\rightarrow \text{TD}(0) \rightarrow v_t^\pi \approx R_{t+1} + \gamma V(s_{t+1}, \bar{w}) \\ &\rightarrow \text{TD}(\lambda) \rightarrow G_t^\lambda \end{aligned}$$

- convergence

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

- LSTDQ \rightarrow least squares q-learning

LSTDQ algorithm: solve for total update = zero

$$0 = \sum_{t=1}^T \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \mathbf{x}(S_t, A_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^T \mathbf{x}(S_t, A_t) (\mathbf{x}(S_t, A_t) - \gamma \mathbf{x}(S_{t+1}, \pi(S_{t+1})))^\top \right)^{-1} \sum_{t=1}^T \mathbf{x}(S_t, A_t) R_{t+1}$$

```

function LSPI-TD( $\mathcal{D}, \pi_0$ )
     $\pi' \leftarrow \pi_0$ 
    repeat
         $\pi \leftarrow \pi'$ 
         $Q \leftarrow \text{LSTDQ}(\pi, \mathcal{D})$ 
        for all  $s \in \mathcal{S}$  do
             $\pi'(s) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}} Q(s, a)$ 
        end for
    until ( $\pi \approx \pi'$ )
    return  $\pi$ 
end function

```

► Policy Gradients

- Policy-based methods \rightarrow parameterize policy directly
 \rightarrow instead of just ϵ -greedy w/ value function
- advantages \rightarrow better convergence
 \rightarrow works for high dim/continuous action spaces
 \rightarrow can learn stochastic policies
- disadvantages \rightarrow converges to local optimum
 \rightarrow high variance

- Policy Objective Function

- episodic env. \rightarrow use start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}[v_1]$$

- continuing env. \rightarrow use average value

$$J_{\text{avg}}(\theta) = \sum d^{\pi_\theta}(s) V^{\pi_\theta}(s) \quad \text{stationary distr.}$$

↳ or average reward per step

for Markov Chain
for π_θ

$$J_{\text{avgR}}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R(s, a)$$

- Optimization

- w/out gradient \rightarrow Hill climbing
 - \rightarrow Nelder-Mead
 - \rightarrow Genetic Algorithms
- gradient based \rightarrow grad descent
 - \rightarrow conjugate grad
 - \rightarrow quasi-Newton
- Finite Differences \rightarrow perturb parameters slightly to approx grad

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon}$$

↳ u_k is 1 in k^{th} dim & 0 else

- works for non-differentiable policies
- very inefficient \rightarrow requires lots of samples to estimate grad
 - \rightarrow only for small parameter space
- Analytical Gradient

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \underbrace{\nabla_\theta \log \pi_\theta(s, a)}_{\text{score function}}$$

- Softmax Policy \rightarrow weigh actions by linear coeffs

$$\pi_\theta(s, a) \propto e^{\theta \cdot \phi(s, a)}$$

↳ score function

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - E_{\pi_\theta}[\phi(s, \cdot)]$$

- Gaussian Policy \rightarrow for continuous action spaces

$$\mu(s) = \theta \cdot \phi(s)$$

$$a \sim N(\mu(s), \sigma^2)$$

↳ score function

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{(a - \mu(s)) \phi(s)}{\sigma^2}$$

- One-Step MDP \rightarrow single action & reward, no sequence

$$J(\theta) = \sum_s d(s) \sum_a \pi_{\theta}(s, a) R(s, a) = E_{\pi_{\theta}}[r]$$

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(s, a) r \right]$$

sample reward

- Policy Gradient Theorem \rightarrow for sequences \rightarrow use value function

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(s, a) Q^{\pi_{\theta}}(s, a) \right]$$

long-term reward

- Monte-Carlo Policy Grad \rightarrow use return v_t as a sample of $Q^{\pi_{\theta}}(s, a)$

$$v_t = \sum_{t'=0}^t r_{t'}$$

$$\Delta \theta = \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$$

↳ algorithm

```

function REINFORCE
    Initialise  $\theta$  arbitrarily
    for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_{\theta}$  do
        for  $t = 1$  to  $T-1$  do
             $\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$ 
        end for
    end for
    return  $\theta$ 
end function

```

↳ slow \rightarrow return is unbiased but high variance

- Actor-Critic \rightarrow use parameterized value function

$$Q_w(s, a) \approx Q^{\pi_{\theta}}(s, a)$$

- critic \rightarrow estimates value function
 \rightarrow policy evaluation
- actor \rightarrow improve policy using critic's value

Ex) critic is linear TD(0)

```

function QAC
  Initialise  $s, \theta$ 
  Sample  $a \sim \pi_\theta$ 
  for each step do
    Sample reward  $r = \mathcal{R}_{s,a}^a$ ; sample transition  $s' \sim \mathcal{P}_{s,a}$ .
    Sample action  $a' \sim \pi_\theta(s', a')$ 
     $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$ 
     $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$ 
     $w \leftarrow w + \beta \delta \phi(s, a)$ 
     $a \leftarrow a', s \leftarrow s'$ 
  end for
end function

```

\hookrightarrow now policy is improved by following policy gradient instead of greedily

- using advantage function \rightarrow \downarrow variance
 \rightarrow how much better is this action compared to others

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

\hookrightarrow estimating advantage function \rightarrow learn Q & V separately
 \hookrightarrow inefficient \rightarrow lots of params

\hookrightarrow use TD error

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \delta^{\pi_\theta}]$$

$$V^{\pi_\theta}(s) \approx V_w(s)$$

\hookrightarrow you only have to learn value function not Q function

- λ Critics \rightarrow using TD(λ)

$$\delta_t = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$$

$$e_t = \gamma e_{t-1} + \phi(s_t)$$

$$\Delta w = \alpha \delta_t e_t$$

- λ Actors

$$\delta = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$$

$$e_{t+1} = \lambda e_t + \nabla_\theta \log \pi_\theta(s, a)$$

$$\Delta \theta = \alpha \delta e_t$$

- Natural Policy Gradient \rightarrow for continuous action spaces

$$\nabla_\theta^{\text{nat}} \pi_\theta(s, a) = C_\theta^{-1} \nabla_\theta \pi_\theta(s, a)$$

\hookrightarrow using Fisher information matrix

$$C_\theta = \nabla_\theta \log \pi_\theta(s, a) \otimes \nabla_\theta \log \pi_\theta(s, a)$$

- vanilla policy gradient \rightarrow noise gets worse as training \uparrow

\hookrightarrow not w/ natural policy gradients

- Natural Actor-Critic \rightarrow use compatible FA
 \rightarrow update actor param toward critic params w

$$\nabla_w A_w(s, a) = \nabla_\theta \log \pi_\theta(s, a)$$

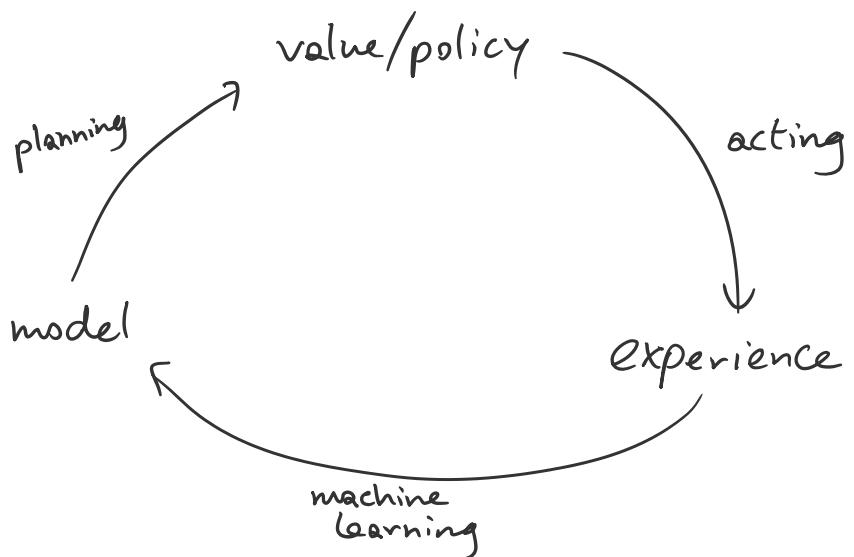
$$\nabla_\theta J(\theta) = C_\theta w$$

$$\nabla_\theta^{\text{nat}} J(\theta) = w$$

► Integrating Planning & Learning

- Model Based RL \rightarrow learn model directly from experience
 \rightarrow plan value function/policy w/ model

\nearrow value/policy \searrow



- advantages → efficiently learn model from supervised training
→ reason about model uncertainty
- disadvantages → model adds source of approximation error
- model → dynamics & rewards, parameterized by η

$$S_{t+1} \sim M_\eta(S_{t+1} | S_t, A_t)$$

$$R_{t+1} = R_\eta(R_{t+1} | S_t, A_t)$$

↳ assume transition prob is independent of reward

- given $\{S_1, A_1, R_2, S_2, A_2, \dots, S_T, R_T\}$

↳ learn $M_\eta \rightarrow$ density estimation problem

$$M_\eta : s, a \rightarrow s'$$

↳ learn $R_\eta \rightarrow$ regression problem

$$R_\eta : s, a \rightarrow r$$

- learning & planning

1) learn model from training set

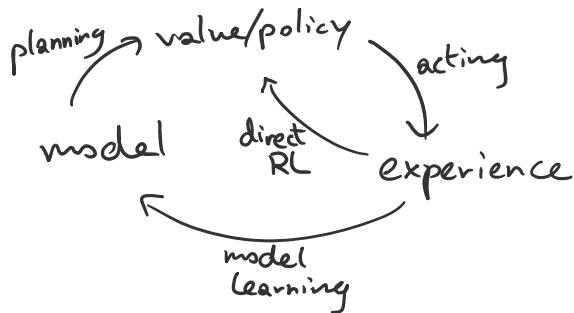
2) given model \rightarrow solve MDP

↳ solve directly \rightarrow using tree search

\rightarrow using DP \rightarrow value/policy iteration

↳ sample-based planning → use model-free RL
→ more efficient

- Dyna Learning → use both real & simulated
 - 1) Learn model from training set
 - 2) given model generate simulated experiences
 - 3) solve MDP using both real & simulated data
- you get more training data by learning model



- DYNA-Q algorithm

```

Initialize  $Q(s, a)$  and  $Model(s, a)$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}(s)$ 
Do forever:
  (a)  $S \leftarrow$  current (nonterminal) state
  (b)  $A \leftarrow \varepsilon\text{-greedy}(S, Q)$ 
  (c) Execute action  $A$ ; observe resultant reward,  $R$ , and state,  $S'$ 
  (d)  $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
  (e)  $Model(S, A) \leftarrow R, S'$  (assuming deterministic environment)
  (f) Repeat  $n$  times:
     $S \leftarrow$  random previously observed state
     $A \leftarrow$  random action previously taken in  $S$ 
     $R, S' \leftarrow Model(S, A)$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
  
```

↳ DYNA-Q+ → gives bonus for exploration
→ good if env changes

- Simulation-Based Search
 - simulate experience from current state using model
 - use model-free algorithm to find best policy
- Dyna-2 → treating simulated experience specially
 - 2 sets of feature weights → long-term memory
→ short-term memory
 - long-term memory → updated from real experience
→ general knowledge

- short term memory \rightarrow updated from simulated exp
 \rightarrow specific knowledge to current situation

► Exploration vs Exploitation

- Approaches
 - Random $\rightarrow \epsilon$ -greedy, softmax
 - optimism \rightarrow estimate uncertainty
 \rightarrow prefer high uncertainty
 - information state space \rightarrow consider agents information as part of state
 \rightarrow then the value of exploration is considered explicitly
- Types
 - state-action \rightarrow explore state/action space
 \rightarrow visit new states, take new actions
 - parameter \rightarrow pick new parameters for α while
 \rightarrow to avoid local optima & find global optimum
- Multi-Armed Bandit \rightarrow single step MDP \rightarrow no states only actions (MAB)
 - action-value $q(a) = E[R | A=a]$
 - optimal value $v_* = q(a^*) = \max_a q(a)$
 - regret $I_t = E[v_* - q(A_t)]$
 - total regret $L_t = E\left[\sum_{\tau=1}^t v_* - q(A_\tau)\right]$
 \hookrightarrow minimize total regret
 - $$L_t = \sum_a E[N_t(a)] \Delta_a$$
 - $$\Delta_a = v_* - q(a)$$
 - lower bound for total regret over time \rightarrow logarithmic
 \hookrightarrow decaying ϵ -greedy gets close if decay function is chosen carefully
 - Estimating Confidence Bounds for action-values \rightarrow optimism

- distribution free \rightarrow w/out making assumptions about distr.

\hookrightarrow Hoeffding's Theorem \rightarrow for bound rewards
 \rightarrow gives an upper bound for the error of estimating the true mean w/ a sample mean

$$P[E[X] > \bar{X}_t + u] \leq e^{-2tu^2}$$

$$\bar{X}_t = \frac{1}{t} \sum_{t'=1}^t X_{t'}$$

\hookrightarrow solution to Multi armed bandit \rightarrow achieves lower bound
 \rightarrow UCB algorithm

$$A_t = \operatorname{argmax}_a \left(Q_t(a) + \sqrt{\frac{2 \log t}{N_t(a)}} \right)$$

- bayesian approach \rightarrow using prior distribution

\hookrightarrow Bayesian UCB :

1) use Bayes law to calculate posterior $p(w | R_1, \dots, R_{t-1})$

2) compute posterior distribution

$$p(Q(a) | R_1, \dots, R_{t-1}) = p(Q(a) | w) p(w | R_1, \dots, R_{t-1})$$

3) estimate upper confidence bound $U_t(a) = c\sigma(a)$

4) pick $A_t = \operatorname{argmax}_a Q_t(a) + c\sigma(a)$

\hookrightarrow Probability Matching \rightarrow select action by sampling calculated prob. of how likely each action is best

$$\pi(a) = P[Q(a) = \max_a Q(a') | R_1, \dots, R_{t-1}]$$

\hookrightarrow automatically is optimistic

\hookrightarrow Thompson sampling \rightarrow sample prior distr. & pick action w/ best sample

$$\pi(a) = E[1(Q(a) = \max_{a'} Q(a')) | R_1, \dots, R_{t-1}]$$

- Information State Space

- exploration gains info \rightarrow now quantify the value of this info

- transform MAB to full MDP \rightarrow states are how often you've chosen each action
 - \hookrightarrow now solve MDP using RL
 - \rightarrow model-free \rightarrow Q-learning
 - \rightarrow model-based \rightarrow Gittins indices \rightarrow DP
 - \hookrightarrow Bayes adaptive RL
- Contextual Bandits \rightarrow now you have some info about bandits \rightarrow states
- Model-Based \rightarrow Rmax algorithm
 - \rightarrow assume new states have max reward
 - \hookrightarrow the update model