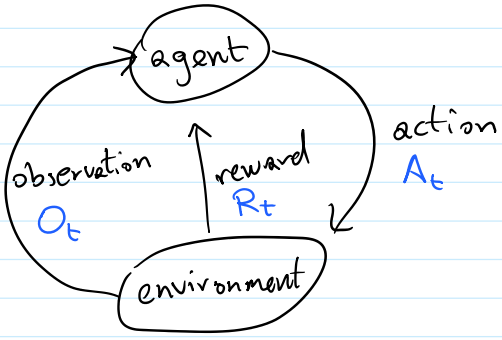
RL Basics

Monday, July 10, 2017 9:02 AM

Intro



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

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, \ldots, S_t)$$

- fully observable: $O_t = S_t$

- POMDPs → partially observable states

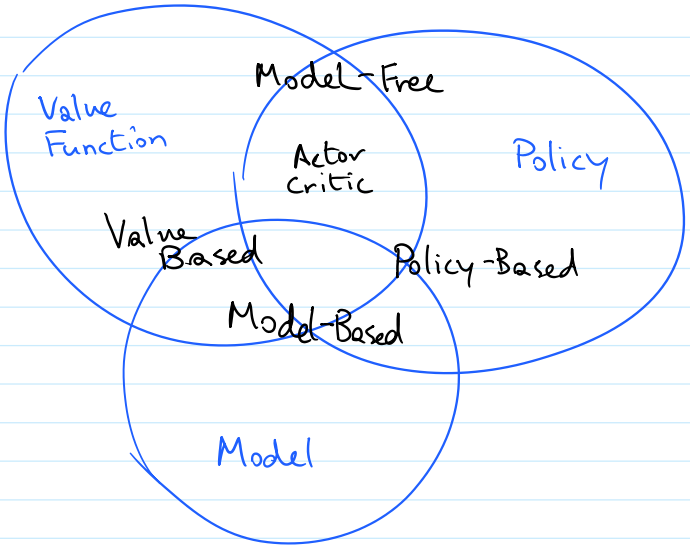
$$S_t = \sigma(S_{t-1} W_s + O_t W_s)$$

↳ use RNN

- Dynamics Model

$$M(s, a, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$$
$$R(s, a) = E(R_{t+1} \mid S_t = s, A_t = a)$$

- 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

$$V(s) = \mathbb{E}(G_t \mid S_t = s)$$

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

- Bellman Equation

prob going from s to s'

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

↳ matrix form

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

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

- 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

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}(G_t \mid S_t = s, A_t = a)$$

- 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 → ∞ MDPs
  → POMDPs
  → Ergodic MDPs

Dynamic Programming

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

- Policy Evaluation

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

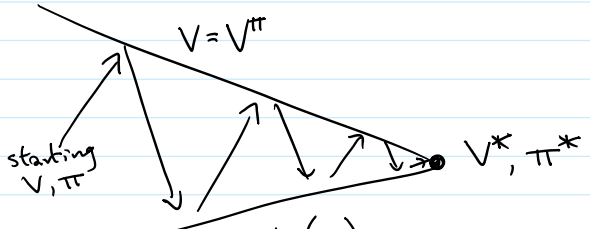
$$\bar{V}^{k+1} = \bar{R}^{\pi} + \gamma \hat{M}^{\pi} \bar{V}^{k}$$

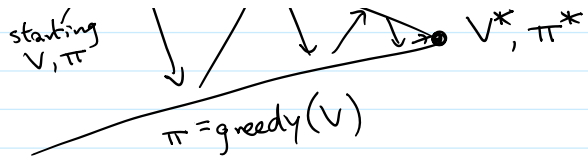
- Improving Policy → greedily

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

- Policy Iteration

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





- Value Iteration → 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

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

- Async DP
  - in-place DP → analogue to SGD
  - prioritized sweeping → update based on Bellman error
  - real-time DP → update only those states agent visits
- Problem Size → only works up to $|S| \sim 10^6$
  - For larger problems use sample backups instead full-backups

Prediction

- Monte Carlo → 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 → adjust V(s) after each action → bootstrapping

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

- MC vs TD

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

- TD(λ) → generalizing MC & TD

  - use eligibility function to rank how much you can learn about other states based on your current experience
    - ↳ using both a frequency & recency heuristic

$$E_0(s) \leftarrow 0$$

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

  - now adjust all states each step according to eligibility

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

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

  - MC is equivalent to TD(1)
  - TD is equivalent to TD(0)

- Control

- now learn best policy
  ↳ naive approach

evaluate π (using prediction)
↑ ↓
improve π (greedily)

- greedy policy improvement → given dynamics & value function

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

↳ model-free

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

- Exploration vs Exploitation → ε-greedy action selection
  - w/ prob. $1-\varepsilon$ choose greedy action
  - w/ prob. ε choose random action

- Monte-Carlo Control → policy iteration
  - policy eval → MC $\quad Q \leftarrow Q^{\pi}$
  - policy improvement → ε-Greedy
  - gradually ↓ ε in ∞ limit → GLIE

↳ in k^{th} episode w/ π
↳ for each S_t & A_t in episode

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

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

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

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

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

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-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) → analogue of TD(λ)
  → 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., ε-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 → on or off policy

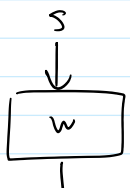
$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \max_{a'} \gamma Q(S',a') - Q(S,A)\right)$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-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

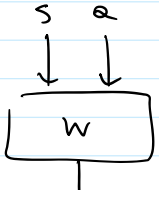
- Value Function Approximation
  - for problems w/ too many states for table look-up → eg. continuous

- Types

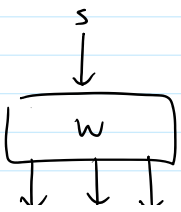
Value Function

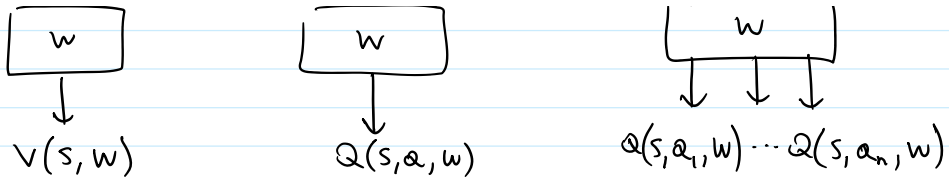


Action-In



Action-Out





(Prediction) (Control) (Control)

- Types of Approximators
  - linear combinations of features
  - neural network
  - decision trees
  - nearest neighbor
  - fourier/wavelet bases

} differentiable, non-stationary, non-iid

- Linear Function Approximation

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

- loss (true value)

$$L(\bar{w}) = E_\pi\left[\left(v_\pi(S) - \bar{x}(S)\cdot\bar{w}\right)^2\right]$$

- update rule

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

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

- incremental method → approximate v_π by experience

$$\Delta\bar{w} = \alpha\left(R_{t+1} + \gamma\, v(S_{t+1}, \bar{w}) - v(S_t, \bar{w})\right)\nabla_w v(S, \bar{w})$$

- 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

$$L_i(w_i) = \mathbb{E}_{(s,a,r,s')\sim D_i}\left[\left(r + \gamma \max_{a'} Q(s', a', \underbrace{w_i^-}_{\text{fixed}}) - Q(s, a, w_i)\right)^2\right]$$

- Linear Analytical Solution

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

↳ don't know v_t^π → MC → $v_t^\pi \approx G_i$
→ TD(0) → $v_t^\pi \approx R_{t+1} + \gamma V(S_{t+1}, \bar{w})$
→ TD(λ) → G_t^λ

- convergence

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

- LSTDQ → 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
    $\pi \leftarrow \pi'$
    $Q \leftarrow$ **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 π
end function

▶ Policy Gradients

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

- Policy Objective Function
  - episodic env. → use start value

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

  - continuing env → use average value

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

for Markov chain for π_θ

↳ or average reward per step

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

- Optimization
  - w/out gradient → Hill climbing
    → Nelder-Mead
    → Genetic Algorithms
  - gradient based → grad descent
    → conjugate grad
    → quasi-Newton

- Finite Differences → 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 → requires lots of samples to estimate grad
  → 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 → weigh actions by linear combos

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

↳ score function

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

- Gaussian Policy → 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 → single action & reward, no sequence

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

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

sample reward

- Policy Gradient Theorem → for sequences → use value function

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

long-term reward

- Monte-Carlo Policy Grad → 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 θ arbitrarily
  for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **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 θ
end function

↳ slow → return is unbiased but high variance

- Actor-Critic → use parameterized value function

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

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

EX) critic is linear TD(0)

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    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

↳ now policy is improved by following policy gradient instead of greedily

- using advantage function → ↓ variance
  → how much better is this action compared to others

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

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

↳ estimating advantage function → learn Q & V separately
  ↳ inefficient → lots of params

↳ use TD error

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

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

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

↳ you only have to learn value function not Q function

- λ Critics → using TD(λ)

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

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

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

- 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 → for continuous action spaces

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

↳ using Fisher information matrix

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

- vanilla policy gradient → noise gets worse as training ↑
  ↳ not w/ natural policy gradients

- Natural Actor-Critic → use compatible FA
  → update actor param (θ) toward critic params (w)

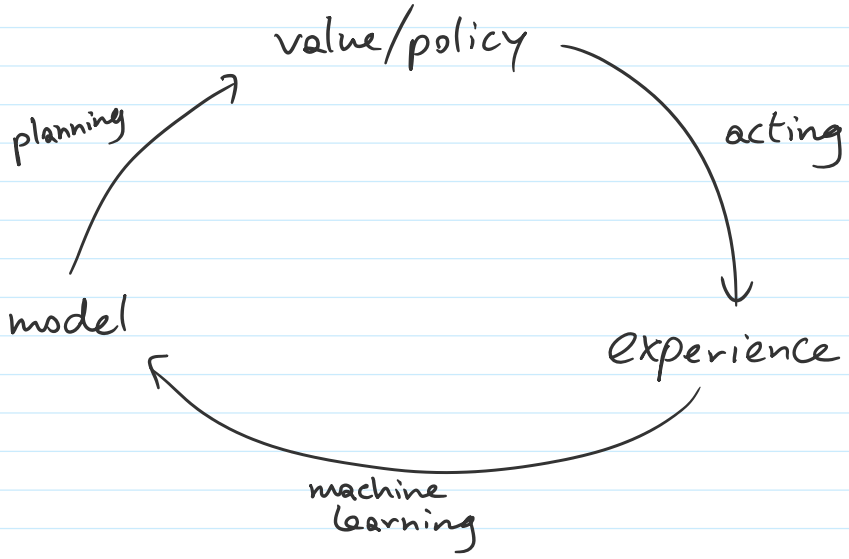
$$\nabla_w A_w(s, a) = \nabla_\theta \log \pi_\theta(s, a)$$

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

Integrating Planning & Learning

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

value/policy



- advantages → efficiently learn model from supervised training
  → reason about model uncertainty
- disadvantages → model adds source of approximation error

- model → dynamics & rewards, parameterized by η

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

↳ 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_η → density estimation problem

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

↳ Learn R_η → regression problem

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

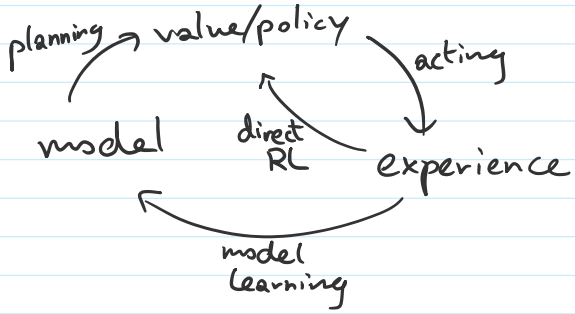
- learning & planning

1) learn model from training set
2) given model → solve MDP

↳ solve directly → using tree search
→ using DP → 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) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(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
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

↳ 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 → updated from simulated exp
  → specific knowledge to current situation

▶ Exploration vs Exploitation

- Approaches
  - Random → ε-greedy, softmax
  - optimism → estimate uncertainty
    → prefer high uncertainty
  - information state space → consider agents information as part of state
    → then the value of exploration is considered explicitly

- Types
  - state-action → explore state/action space
    → visit new states, take new actions
  - parameter → pick new parameters for a while
    → to avoid local optima & find global optimum

- Multi-Armed Bandit (MAB) → single step MDP → no states only actions
  - action-value $q(a) = \mathbb{E}[R \mid A = a]$
  - optimal value $v_* = q(a^*) = \max_a q(a)$
  - regret $I_t = \mathbb{E}[v_* - q(A_t)]$
  - total regret $L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} v_* - q(A_\tau)\right]$
    ↳ minimize total regret

$$L_t = \sum_a \mathbb{E}[N_t(a)]\,\Delta_a$$

$$\Delta_a = v_* - q(a)$$

  - lower bound for total regret over time → logarithmic
    ↳ decaying ε-greedy gets close if decay function is chosen carefully

- Estimating Confidence Bounds for action-values → optimism

- distribution free → w/out making assumptions about distr.

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

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

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

↳ solution to Multi armed bandit → achieves lower bound
→ UCB algorithm

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

- bayesian approach → using prior distribution

↳ Bayesian UCB :

1) use Bayes law to calculate posterior $p(w \mid R_1, \ldots, R_{t-1})$
2) compute posterior distribution
$$p(Q(a) \mid R_1, \ldots, R_{t-1}) = p(Q(a) \mid w)\, p(w \mid R_1, \ldots, R_{t-1})$$
3) estimate upper confidence bound $U_t(a) = c\sigma(a)$
4) pick $A_t = \underset{a}{\operatorname{argmax}}\ Q_t(a) + c\sigma(a)$

↳ Probability Matching → select action by sampling calculated prob. of how likely each action is best

$$\pi(a) = P\left[Q(a) = \max_{a'} Q(a') \mid R_1, \ldots, R_{t-1} \right]$$

↳ automatically is optimistic

↳ Thompson sampling → sample prior distr. & pick action w/ best sample

$$\pi(a) = E\left[1(Q(a) = \max_{a'} Q(a')) \mid R_1, \ldots, R_{t-1} \right]$$

- Information State Space

- exploration gains info → now quantify the value of this info

- transform MAB to full MDP → states are how often you've chosen each action

  ↳ now solve MDP using RL → model-free → Q-learning
  → model-based → Gittins indices → DP
  ↳ Bayes adaptive RL

- Contextual Bandits → now you have some info about bandits → states

- Model-Based → Rmax algorithm
  → assume new states have max reward
  ↳ the update model