Reinforcement Learning

- MDP $\langle S, A, P, R, \gamma \rangle$ (POMDP)
- Bellman equations (V^π, Q^π, V^*, Q^*)
  - π^* given V^*/Q^*
- Dynamic Programming:
  - Policy evaluation, Policy iteration, Value iteration
    - update rule
    - 2 steps
    - update rule
  - async DP, function approx, divergence
  - contraction proof (Bellman expectation & optimality backups)
- Model-free prediction:
  - MC & TD policy evaluation
  - TD(n) & TD(λ)
- Model-free control:
  - MC policy iteration, Sarsa, Sarsa(λ)
  - Q-learning, Deep Q-learning (exp replay & stationary targets)
- Policy gradient:
  - $\nabla_\theta J(\theta) = \dots$
  - Reinforce (baseline), Actor-Critic
- Model-based:
  - Sample-based planning; forward plan, simulation based plan, MCTS
  - Dyna

- Advantage actor-critic
- Exploration vs exploitation:
  - multi-armed bandit
  - minimize regret
  - UCB, confidence in face of uncertainty

RL

$$v^\pi(s) = \sum_a \pi(s,a)\, q^\pi(s,a)$$

$$= \sum_a \pi(s,a)\left[R_s^a + \gamma \sum_{s'} P_{ss'}^a\, v^\pi(s')\right]$$

$$q^\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a\, v^\pi(s')$$

$$= R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(s',a')\, q^\pi(s',a')$$

Bellman (expectation) eqns

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

$$= \max_a R_s^a + \gamma \sum_{s'} P_{ss'}^a\, v^*(s')$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a\, v^*(s')$$

$$= R_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q^*(s',a')$$

Bellman Optimality eqns

Policy evaluation:
- iterate Bellman expation eqn

Policy iteration:
- Policy evaluation + Greedy Policy improvement

Value iteration:
- iterate Bellman optimality eqn

Can do async, or with priority (by Bellman error)

$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$

- MC Policy evaluation: (from full episode)
$$v^\pi(S_t) = v^\pi(S_t) + \alpha\left(G_t - v^\pi(S_t)\right)$$
- TD Policy evaluation: (TD error, $G_t^{(1)} - V(S_t)$)
$$V(S_t) = V(S_t) + \alpha\left(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{G_t^{(1)}} - V(S_t)\right)$$

can get into policy iteration too

equivalent can be done w/ $Q(s,a)$

TD(n) $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

TD(λ) $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} = R_{t+1} + \gamma\left[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\right]$

- Sarsa: policy iteration w/ TD policy evaluation
  $Q(S,A) = Q(S,A) + \alpha [R + \gamma Q(S',A') - Q(S,A)]$
  (Sarsa (λ) uses TD(λ) error, to do online need eligibility traces)
- Q-Learning: off-line Sarsa

Expected Sarsa →
$Q(S,A) = Q(S,A) + \alpha [R + \gamma \sum_a \pi(a|S') Q(S',a') - Q(S,A)]$

for greedy π,
$Q(S,A) = Q(S,A) + \alpha [R + \gamma \max_{a'} Q(S',a') - Q(S,A)]$

- DQL:
  - experience replay stabilize gradients
  - target network improve stability (stationary targets ⇒ SL problem)
- Policy Gradient:
  $\nabla_\theta J(\theta) = E[Q^{\pi_\theta}(S,A) \nabla_\theta \log \pi_\theta(A|S)]$
- REINFORCE:
  - Sample G_t from full episode (Monte-Carlo Policy Gradient)
    $\theta_{k+1} = \theta_k + \alpha \sum_t G_t \nabla_\theta \log \pi_{\theta_k}(A_t|S_t)$
- Actor-Critic:

policy evaluation →
  - Critic: $V(S)$ or $Q(S,A)$, update w/ TD (or TD(n)/TD(λ))

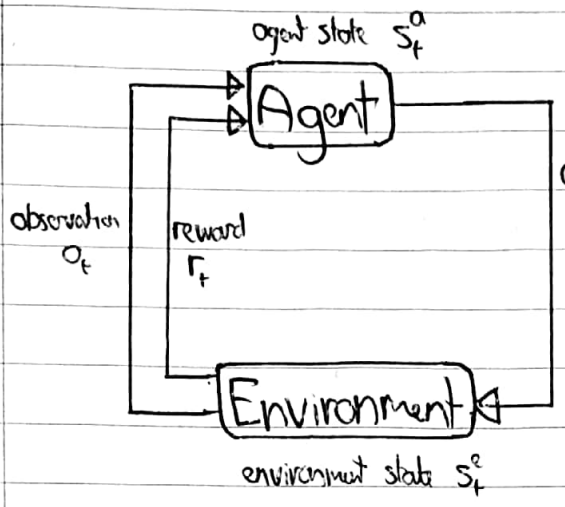
policy gradient →
  - Actor: π_θ, update by policy gradient
    (REINFORCE update, but G_t replaced with TD-error)
- Dyna:
  - every step learning from reality, learn N steps from model
    (experience buffer is a simple type of model)
- MCTS:
  - selection + expansion w/ tree policy, simulation w/ default policy, update $Q(S,A)$ on path to root
- Regret:
  $L_t = \sum_{t=1}^{t} v^* - q(a_t) = \sum_a N_t(a) \Delta_a \quad (\Delta_a = v^* - q(a_a))$

for ε-greedy →
$E[v^* - Q_t(A_t)] \geq \frac{\varepsilon}{|A|} \sum_a \Delta_a$ ⇒ linear in t w/ constant ε

- UCB:
  $P(E(X) > \bar{X}_t + u) \leq e^{-2tu^2}$

$P(q(a) > Q_t(a) + U_t(a)) = p \leq e^{-2N_t(a)U_t(a)^2} \Rightarrow U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$

Introduction to Reinforcement Learning



- Reward hypothesis:
  any goal can be formalized as outcome of maximising a cumulative reward

o_t function of s_t^e
s_t^a function of history; sequence of (o_t, a_t, r_t)
a_t function of s_t^a
r_t function of s_{t-1}^e and a_{t-1}

- Markov state:
  s_t Markov $\Leftrightarrow P(s_{t+1} | s_t) = P(s_{t+1} | s_{1:t})$
  - future independent of past given present
  - even if environment state is Markov, observation may not be (partially observable environment)

- Agent components:
  - $\pi(a|s)$ Policy: map agent state to action
  - $q_\pi(s,a), v_\pi(s)$, Value function: expected future reward (discounted by γ)
    - depends on policy
  - $s', r = m(s,a)$ Model: predict next state or reward
    - can be used to help derive good policy

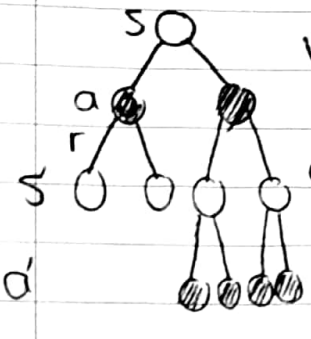
- Some combination of these 3 gives an RL agent

Markov Decision Processes

- Markov process:
  - tuple, $\langle S, P \rangle$
    - S = finite set of states
    - $P_{ss'} = P(S_{t+1} = s' \mid S_t = s)$
- Markov reward process:
  - tuple $\langle S, P, R, \gamma \rangle$
    - $R_s = E(R_{t+1} \mid S_t = s)$
    - γ = discount factor $(0 \leq \gamma \leq 1)$ ← γ can be 1 only if all sequences terminate
  - Return is discounted future reward
    $$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$
  - Value is expected return
    $$v(s) = E(G_t \mid S_t = s)$$
    $$v(s) = E(R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s)$$ ← Bellman eqn for MRP
    $$\underline{v} = \underline{R} + \gamma \underline{\underline{P}}\, \underline{v}$$
    $$\Rightarrow \underline{v} = (\underline{\underline{I}} - \gamma \underline{\underline{P}})^{-1} \underline{R}$$ ($O(n^3)$ inversion; unfeasible for large state space)
- Markov Decision Process:
  - tuple $\langle S, \cancel{R}, A, P, R, \gamma \rangle$
    - A is finite set of actions
    - $P^a_{ss'} = P(S_{t+1} = s' \mid S_t = s, A_t = a)$
    - $R^a_s = E(R_{t+1} \mid S_t = s, A_t = a)$
  - Policy is distribution over actions given states ← fully defining agent behaviour; stationary
    $$\pi(s, a) = P(A_t = a \mid S_t = s)$$
- Markov Reward Process:
  - State and reward sequence from following policy π in an MDP
- Value function
  $$v^\pi(s) = E_\pi(G_t \mid S_t = s)$$ ← expected G_t if follow policy π
  $$q^\pi(s, a) = E_\pi(G_t \mid S_t = s, A_t = a)$$ ← take action a, then follow policy π
- ~~Bellman eqns:~~

Markov Decision Processes

- Bellman equations: (expectation)



$$v^\pi(s) = \sum_{a \in A} \pi(s,a)\left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^\pi(s')\right]$$

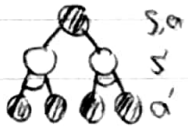
$$q^\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(s',a') q^\pi(s',a')$$

$q^\pi(s,a)$ spans the whole equation; $v^\pi(s')$ spans the inner sum.

- v^* & q^* are max v^π & q^π over all policies
  - solved MDP when found v^* / q^*
  - optimal policy $\pi^* \geq \forall \pi$ achieving $v^{\pi^*} = v^*$, $q^{\pi^*} = q^*$
    ($\pi \geq \pi' \Leftrightarrow v^\pi(s) \geq v^{\pi'}(s)$ for all s)
  - $\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_a q^*(s,a) \\ 0 & \text{o/w} \end{cases}$
- Bellman Optimality eqns:

$$v^*(s) = \max_a q^*(s,a) \qquad q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')$$

$$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s') \qquad q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q^*(s',a')$$



- non-linear; no closed form solution in general, need to iterate
- Partially Observable MDP:

$\langle S, A, O, P, R, Z, \gamma \rangle$

- O: finite set of observations
- Z: observation function, $Z_s^a = P(O_{t+1} = o \mid S_t = s, A_t = a)$

- Hidden Markov model with actions

Planning by Dynamic Programming

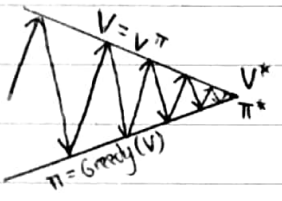
- Dynamic Programming:
  - break complex problem into subproblems, solve these & combine solutions to solve complex problems
  - good when:
    - (see Bellman eqns) → complex solution combination of simpler solutions
    - (value fn stores & reuses solutions) → subproblems overlap / repeat
    - ⇒ good for MDPs!

(Need full knowledge of MDP; then use DP to plan)

- Iterative policy evaluation:
  - iterate $V_{k+1}(s) = \sum_{a \in A} \pi(s,a)\left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s')\right]$

- Policy iteration:
  - estimate v^π
    - using iterative policy evaluation
  - generate $\pi' \geq \pi$
    - greedy policy on v^π
    - improves as $v^\pi(s) \leq q^\pi(s, \pi'(s)) = v^{\pi'}(s)$
    - if no change, $v^\pi(s) = \max_{a \in A} q^\pi(s,a)$ ⇒ Bellman optimality satisfied
    - ⇒ optimal!

(Can also be done by other algorithms (Generalised Policy Iteration))



- Value iteration:
  - iterate $V_{k+1}(s) = \max_{a \in A}\left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s')\right]$
  - ~~Can~~ once V^* found, can derive $\pi^* = \text{Greedy}(V^*)$

- all $O(mn^2)$ per iteration (m actions, n states)
  ($O(m^2n^2)$ if based on q-function)

- Asynchronous DP:
  - above have been done on all states per iteration; prohibitively expensive for large state spaces

- In-place DP:
  - updat as we go through states
- Prioritised sweeping:
  - update largest Bellman error $= \left| \max_{a \in A} \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s') \right] - V_k(s) \right|$
- Real time DP:
  - follow an agent in the MDP, updating states we visit
- Approximate DP:
  - use a function approximator for value function, train to minimise square loss over a sampled mini-batch of (or every) states
- Divergence:
  - Bootstrapping + General Function Approximation + Using state distribution not matching transition dynamics of MDP

improve value estimate by using estimate in future states

⇒ possibility to diverge (rare in practise)

- Contraction: (& optimality backup)
  - Bellman expectation backup is a contraction; brings value function closer by at least γ

⇒ by contraction mapping theorem all 3 algorithms converge to optimal policy or value function

Model-Free Prediction

- Estimate value function of unknown MDP

- Monte-Carlo Policy Evaluation:
  - $V^{\pi}(s) = E_{\pi}[G_t | S_t = s]$
  - compute expectation by full rollouts following policy π

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

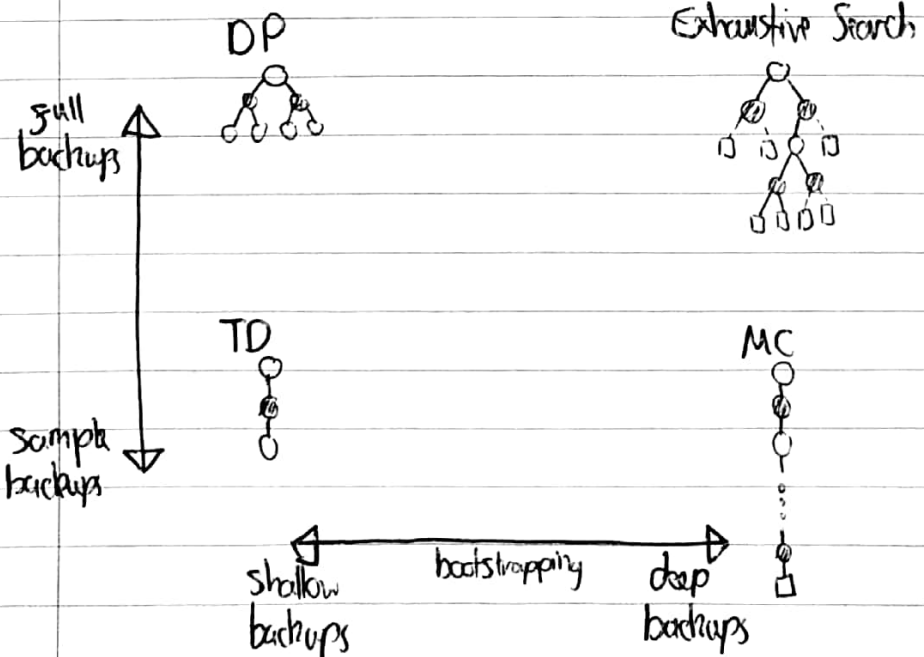
(α: learning rate; G_t: from 1 full episode following π)

- Temporal Difference:
  - Bootstrap:

TD error $\rightarrow \delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

$$V(S_t) \leftarrow \cancel{R_{t+1} + \gamma V_{\theta}(S_{t+1})} V(S_t) + \alpha\delta$$

  - only perform 1 step, then update estimate using estimate
  - \+ don't need to rollout to end
  - \+ works in never ending environments
  - \+ lower variance
  - − biased (a little)
  - − sensitive to initial value function
  - \+ exploits Markov property, converges to max likelihood Markov model (Monte-Carlo method doesn't)



- TD(λ):
  - use λ to decide 'degree' of bootstrapping

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

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \quad G_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i R_{t+i+1} + \gamma^n \cdot V(S_{t+n})$$

$$G_t^\lambda = R_{t+1} + \gamma\left[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\right]$$

- observe reward
- continue w/ weight γ
- bootstrap w/ weight $(1-\lambda)$
- continue w/ weight λ

- need full rollouts: can use eligibility traces to allow TD(λ) online
- $\lambda = 1$ roughly equivalent to MC
- converges with tabular or linear value function approximation

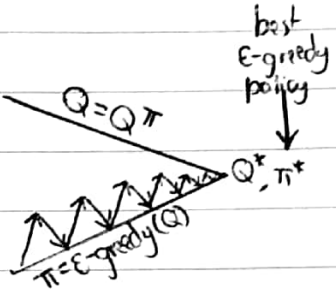
(TD(n) $\Rightarrow$ $V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$)

- n-step then bootstrap

Model-Free Control

- Optimize value function for unknown MDP
- Multi-armed bandit:
  - A = actions space
  - R_t is based on action $a_t \in A$
  - how to maximize $\sum_t R_t$?
  - need to trade-off exploration vs exploitation
  - eg ε-greedy: prob $(1-\varepsilon)$ take best-seeming action, prob ε take random action
- Policy iteration:
  - ~~estimate~~ Policy evaluation:
    - Monte-Carlo policy evaluation, $Q \approx Q^\pi$
  - Policy improvement:
    - ε-greedy policy using new Q

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



- Greedy In the Limit w/ Infinite Exploration: (GLIE)
  - if all state-action pairs visited infinite times, policy converges on a greedy policy
  - ε-greedy is GLIE if $\varepsilon_k = 1/k$ (ε decays)
  - $\Rightarrow$ decay ε this way w/ policy iteration
- Sarsa:
  - use TD learning instead of MC for policy evaluation

On-policy: learn about π by sampling from π

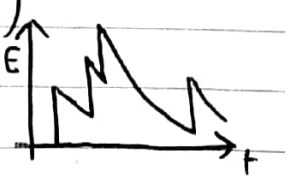
$$Q(S,a) \leftarrow Q(S,a) + \alpha\,\delta$$
$$\delta = \underbrace{R + \gamma Q(S', a')}_{\text{change to } G_t^{(n)} \text{ for n-step backup}} - Q(S,a)$$

- Sarsa(λ):

updated for every S, a: $Q(S,a) \leftarrow Q(S,a) + \alpha\,\delta_t\,E_t(S,a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, a_t)$$

eligibility traces:
$$E_0(S,a) = 0$$
$$E_t(S,a) = \gamma\lambda E_{t-1}(S,a) + I(S_t = S, A_t = a)$$



Off-policy
Learn about policy π
by sampling policy μ

- Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$$

  - if $\mu = \pi$: Expected Sarsa
  - if $\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$ & μ ε-greedy wrt $Q(s,a)$
    target simplifies to

$$R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

- converges to opt Q^* if take every action in every state infinite times
- Deep Q-Learning:
  - To improve performance:
  - Experience Replay:
    - Store many previous (S, a, r, s') transitions, & update using minibatch to stabilise gradients
    - don't just do recent part, or we'll overfit to it
  - Stationary targets:
    - target depends on Q-network; changes every iteration
    - instead, use target network, fixed for a while; problem closer to SL, improves stability

Policy Gradient

- Learn policy only; don't bother with value function or model
  - parametrise policy, $\pi_\theta(a|s) = P(a|s,\theta)$
  - + better convergence
  - + can learn stochastic policies
  - + policy could be simple whilst value/model complex
  - − local optima & does not generalize

- Policy gradient:
  - update parameters by gradient ascent (eg av. reward per time step, average state value, or initial state value)
    $\Delta\theta = \alpha \nabla_\theta J(\theta)$, $J(\theta)$ is policy objective function
  - need to compute
    $\nabla_\theta J(\theta) = \nabla_\theta E_d(V^{\pi_\theta}(S))$
  - aka contextual bandit
  - eg, 1 step case
    $\nabla_\theta J(\theta) = \nabla_\theta E(R(S,A))$
    $= E(\nabla_\theta \log \pi_\theta(A|S) \cdot R(S,A))$ (score function trick)
    (can compute this using MC samples)
    $\Rightarrow \theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t)$
  - also applies to long term reward, using $q^\pi(S,a)$
    $\nabla_\theta J(\theta) = E[q^{\pi_\theta}(S,A) \nabla_\theta \log \pi_\theta(A|S)]$
    (expectation over S & A)
  - don't need to know dynamics
    - eg, trajectory T w/ return $G(T)$
      $\nabla_\theta J(\theta) = E[G(T) \nabla_\theta \sum_{t=0} \log \pi(A_t|S_t)]$

- REINFORCE:
  - Monte-Carlo Policy Gradient
  - for each episode:
    $\theta \leftarrow \theta + \alpha \sum_t G_t \nabla_\theta \log \pi_\theta(A_t|S_t)$ (learn from policy evaluation)
  - problem: high variance
    - use a baseline $b(S) = V_\pi(S) \approx V_\pi(S)$ (similar idea to target function)
      $\theta \leftarrow \theta + \alpha \sum_t (G_t - b(S_t)) \nabla_\theta \log \pi_\theta(A_t|S_t)$

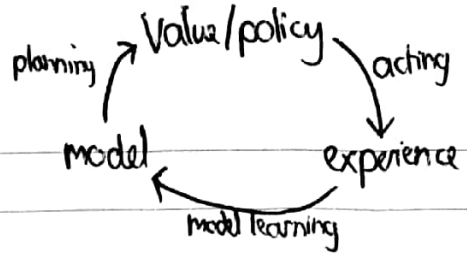
- Actor-Critic:
  - Critic:
    - update η of V_η by TD ← bootstrapping to reduce variance
  - Actor:
    - update Θ by policy gradient

  - for each step:
    - Sample reward R_s^A, S'
    - Sample $A' \sim \pi_\Theta(S')$
    - $\delta \leftarrow R_s^A + \gamma V_\eta(S') - V_\eta(S)$ — TD error
    - $\eta \leftarrow \eta + \beta \delta \nabla_\eta V_\eta(S)$ — TD(0)
    - $\Theta \leftarrow \Theta + \alpha \delta \nabla_\Theta \log \pi_\Theta(S, A)$ — Policy Gradient
    - $A \leftarrow A'$, $S \leftarrow S'$

  - can reduce bias by using TD(n) instead of TD(0)
    $\delta_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n V_\eta(S_{t+n}) - V_\eta(S_t)$

- Continuous actions:
  - as only dealing with parameters on policy, easy generalise to continuous action space
  - Gaussian policy common; $a \sim N(\mu(S), \sigma^2)$
    $\nabla_\Theta \log \pi_\Theta(S, a) = \frac{1}{\sigma^2}(a - \mu(S)) \nabla \mu(S)$

Planning & Models



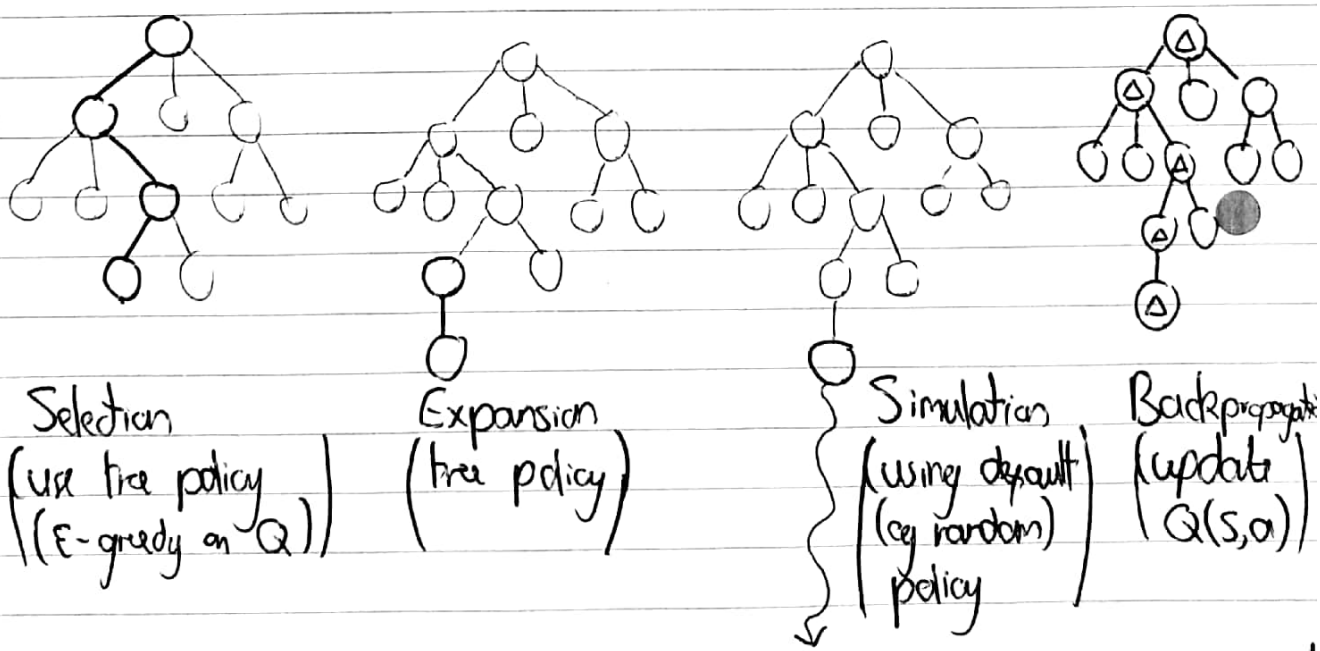
- Model:
  - $M_\eta = \langle P_\eta, R_\eta \rangle$, approximate state transitions & rewards of an MDP $\langle S, A, P, R \rangle$
  - Model learning:
    - estimate model from experienced transitions $\langle S_t, A_t \rangle \rightarrow \langle R_{t+1}, S_{t+1} \rangle$
    - SL problem; learn P & R functions
    - table based, linear, neural network...
  - Planning with model:
    - solve MDP $\langle S, A, P_\eta, R_\eta \rangle$ using planning algorithm
      - Value iteration, policy iteration...
    - Sample based planning:
      - sample experience from model & apply model-free RL to this (MC, Sarsa, Q-learning)

Problem: performance limited to optimal policy for MDP $\langle S, A, P_\eta, R_\eta \rangle$; not optimal for true MDP

- Model-based vs Model-free:
  - can be fuzzy distinction; eg experience replay buffer; effectively stores a model & samples from this to train
    - can store experience in more complex forms too
  - can do, eg, prioritised experience replay; train on transitions with largest Bellman error

- Dyna:
  - learn using real & simulated experience; don't distinguish the two
  - initialise $Q(s,a)$ & $Model(s,a)$
  - do:
    - sample a using $Q(s,a)$ (eg ε-greedy)
    - observe R_s^a & s'
    - update $Q(s,a)$ using Q-learning
    - update $Model(s,a) \leftarrow (s', R_s^a)$
    - repeat N times:
      - randomly choose s, pick a chosen in s last time encountered, find s', R_s^a using Model, update $Q(s,a)$ using Q-learning

- Planning:
  - given fixed model, how to use optimally to select next action
  - forward search:
    - search from now to termination to determine best action
    - use model
  - simulation-based search:
    - simulate episodes from now on
    - use model-free RL on the episodes
      - MC control → MC search
      - Sarsa → TD search
  - Monte-Carlo Tree Search:
    - simulate K episodes, using current simulation policy π
    - build search tree from this
    - evaluate $Q(S,a)$ (for all S,a leaves) by mean return from episodes from S,a
    - Select action by $\arg\max_a Q(S_t, a)$



Building Agents

$A_t = agent(R_t, O_t)$

(policy is a simple agent, only looking at O_t)
(learning done through looking at R_t too)

- eg neural Q-learning:
  - network $q_\theta : O_t \to (q[a_1], \dots, q[a_m])$
  - ε-greedy policy: $\pi(q_t) \to a_t$
  - Q-learning squared loss
  - optimizer to minimize loss (SGD, RMSProp, Adam)
- Advantage Actor critic:
  - Representation: $(S_{t-1}, O_t) \to S_t$
  - network $V_\eta : S \to V$ — n-step TD loss on V_η
  - network $\pi_\theta : S \to \pi$ — n-step REINFORCE loss on π_θ
  - copies π^m of π_θ as policy — optimizers for each
    $S_t^m \to A_t^m$

- Exploration vs Exploitation:
  - aim to minimize total regret

$$L_t = \sum_{i=1}^{t} \underbrace{V^* - q(a_i)}_{\text{regret = opportunity cost for 1 step}}, \qquad V^* = \max_{a \in A} q(a)$$

$$q(a) = E[R_t \mid A_t = a]$$

  - ε-greedy: prob $(1-\varepsilon)$: greedy action, prob ε: random action

$$E(V^* - Q_t(A_t)) \geq \frac{\varepsilon}{|A|} \sum_{a \in A} \Delta_a \qquad (\Delta_a = V^* - q(a))$$

hard to find decay rate
- $\Rightarrow$ constant ε = linear total regret
- $\to$ decaying ε = logarithmic asymptotic total regret

- Optimism in the face of uncertainty:
  - when very uncertain about reward given action, better to explore that one
- Upper confidence bounds:
  - estimate upper confidence bounds $U_t(a)$ for each action (eg 95% confidence bound) $(P(q(a) > Q_t(a) + U_t(a)) \approx$
  - select action with highest UCB

large $U_t(a)$ → = uncertain about a

$\Rightarrow$ Optimistic in face of uncertainty

- UCB for Multi-Armed Bandit:

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

for iid $x_1, \ldots, x_t$, $\bar{X}_t = \frac{1}{t}\sum x_i$

$$\Rightarrow P[q(a) > Q_t(a) + U_t(a)] \leq \exp(-2N_t(a)U_t(a)^2) = p$$

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

decay p over time, eg $p = \frac{1}{t}$
($\Rightarrow$ select optimal action as $t \to \infty$)

$$\Rightarrow a_t = \underset{a \in A}{\operatorname{argmax}}\, Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \qquad (c > 0)$$

(UCB algorithm achieves logarithmic total regret)