Reinforcement Learning

Symbols

a action, arm (multi-armed bandit problem)

A advantage

α learning rate, step size

b bandit, baseline estimate, bias

c cost, context (= state)

d difference error

Δ difference

D replay buffer

E eligibility trace

ϵ exploration rate (0 1)

ϕ state transition function

g gain (reward over time)

γ discount factor

∇ gradient (spacial derivative)

H horizon

J expected return (see loss)

L expected (squared) loss (see return)

λ trace decay (0 1)

m momentum

Pr transition distribution

π policy ($state \rightarrow action$)

q quality

r reward

R return

ρ regret

s state

t time

τ trajectory

θ parameter weight (learning target)

v value

vm variance momentum

w weight, winning probability

y target / prediction

$_*$ optimal

$'$ next / derivative

$\hat{}$ estimation

$|||$ vector norm

Reward

$$G_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{1}$$

for *discount factor* γ and (infinite?) *horizon* H.

Bandits

- model picking actions without state
- e.g. multi-armed bandit
- reduces RL to exploration vs exploitation

They estimate $\hat{r}_a \approx \mathbb{E}[r|a]$

Greedy

$$\begin{aligned} n_a &= \textstyle\sum_t^a 1 \\ \hat{r}_a &= \textstyle\sum_t^a \frac{r_t}{n_a} \\ a_t &= \operatorname*{argmax}_{a \in A} \hat{r}_a \end{aligned} \tag{2}$$

Optimistic-Greedy

large initial $\hat{r}_a$, R

ϵ-Greedy

for probability ϵ, pick randomly

Upper Confidence Bound

try all for k rounds, then

$$a_t = \operatorname*{argmax}_{a \in A} \hat{r}_a + \sqrt{\frac{2logt}{n_a}} \tag{3}$$

Contextual Bandit

LinUCB:

$$\begin{aligned} x_{t,a} &= \Phi(s_{t,a}) \\ \mathbb{E}[r_{t,a}|x_{t,a}] &= \theta_a \cdot x_{t,a} \\ \hat{\theta}_a &= A^{-1} b \end{aligned} \tag{4}$$

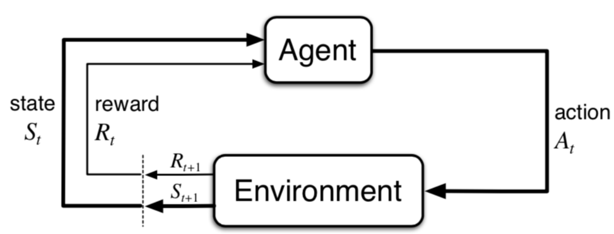
Posterior / Thompson sampling

Pick actions by probability they maximize expected reward

Greedy in the Limit with Infinite Exploration (GLIE)

- infinitely explores
- converges on greedy

Agent-Environment Interface



The Agent at each step t receives a representation of the environment's *state*, $S_t \in S$ and it selects an action $A_t \in A(s)$. From its action the agent receives a *reward*, $R_{t+1} \in R \in \mathbb{R}$.

Policy

A *policy* maps a state to an action

$$\pi_t(a|s) \tag{5}$$

probability to pick an action $A_t = a$ if $S_t = s$.

Markov Decision Process (MDP)

a 5-tuple (S, A, P, R, γ)

$$\begin{aligned} &\text{state transition probabilities:} \\ &p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\} \\ &\text{expected reward for state-action-nexstate:} \\ &r(s',s,a) = \mathbb{E}[R_{t+1}|S_{t+1} = s', S_t = s, A_t = a] \end{aligned} \tag{6}$$

Value Function

Value (expected return, total discounted reward) of a specific state s under policy π for a MDP:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)[r + \gamma v_\pi(s')] \\ v_*(s) &= \max_\pi v_\pi(s) \end{aligned} \tag{7}$$

Action-Value (Q) Function

expected reward for state-action pairs:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')] \\ q_*(s,a) &= \max_\pi q_\pi(s,a) \end{aligned} \tag{8}$$

rewriting v_* for $q_*(s,a)$:

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a) \tag{9}$$

i.e. state value under optimal policy = expected return from its best action.

Bellman Equation

Recursive property $Value$ 7 / Q 8 functions

Model-based Methods (known MDP)

Exhaustive Search

usually computationally unviable.

Contraction Mapping

For metric space (X, d) and $f : X \rightarrow X$, f is a *contraction* given a *Lipschitz coefficient* $k \in [0, 1)$ where for all x / y in X:

$$d(f(x), f(y)) \leq kd(x,y) \tag{10}$$

Contraction Mapping theorem

For complete metric space (X, d) and contraction $f : X \rightarrow X$, there is only 1 fixed point x^* where $f(x^*) = x^*$.
For point x in X, and $f^n(x)$ inductively defined by $f^n(x) = f(f^{n1}(x))$, $f^n(x) \rightarrow x^*$ as $n \rightarrow \infty$, yielding a unique optimal solution for DP.

Dynamic Programming (DP)

+ bootstrap (learn mid-episode)

Find π_* for V / Q:

Policy Iteration

Initialize $V(s) \in \mathbb{R}$e.g. 0, $\Delta \leftarrow 0$, $\pi(s) \in A$ for all $s \in S$
1. Policy Evaluation
while $\Delta < \theta$ *(e.g. 0.001)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
2. Policy Improvement
policy-stable $\leftarrow$ *true*
while *not policy-stable* **do**
  foreach $s \in S$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    policy-stable $\leftarrow$ *old-action* $\neq \pi(s)$
  end
end

Policy iteration methods:

- gradient-based (policy gradient methods): gradient ascent
- gradient-free: simulated annealing, cross-entropy search or methods of evolutionary computation
- value search/iteration: stop after 1 state sweep
- async DP: update iteratively, no full sweeps
- generated policy iteration (GPI)

Value Iteration

ditch $V(s)$ convergence for policy improvement and truncated policy eval step in 1 operation:

Initialize $V(s) \in \mathbb{R}$e.g. 0, $\Delta \leftarrow 0$
while $\Delta < \theta$ *(e.g. 0.001)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
output: deterministic policy $\pi \approx \pi_*$ where
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

Model-free methods

Monte Carlo (MC) Methods

- uses **averaging sample returns** per state-action pair
- episodic

\+ sampling

Initialize for all $s \in S, a \in A(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
while *forever* **do**
  Pick $S_0 \in S$ and $A_0 \in A(S_0)$, all $p(s,a) > 0$
  Generate an episode starting at S_0, A_0 following π
  foreach *pair s,a in the episode* **do**
    $G \leftarrow$ return for first occurrence of s,a
    Append G to $Returns(s,a)$)
    $Q(s,a) \leftarrow average(Returns(s,a))$
  end
  foreach *s in the episode* **do**
    $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$
  end
end

estimate for non-stationary problems:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \quad (11)$$

for learning rate α, how much we want to forget about past experiences.

Temporal Difference (TD)

\+ DP's bootstrap

\+ MC's sampling

- substitutes expected discounted reward G_t from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1} - V(S_t)] \quad (12)$$

State-action-reward-state-action (SARSA)

- on-policy TD control
- can use priors

Initialize $Q(s,a)$ arbitrarily and
  $Q(terminal - state,) = 0$
foreach *episode* $\in$ *episodes* **do**
  Pick a from s by policy from Q (e.g. ϵ-greedy)
  while *s is not terminal* **do**
    Take action a, observe r, s'
    Pick a' from s' by policy from Q (e.g., ϵ-greedy)
    $y \leftarrow r + \gamma Q(s',a')$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[y - Q(s,a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

n-step Sarsa (n-step TD)

for n-step Q-Return:

$$\begin{aligned} q_t^{(n)} &= \gamma^n Q(S_{t+n}) + \textstyle\sum_{i=1}^{n} \gamma^{i-1} R_{t+i} \\ Q(s_t,a_t) &\leftarrow Q(s_t,a_t) + \alpha\left[q_t^{(n)} - Q(s_t,a_t)\right] \end{aligned} \quad (13)$$

Forward View Sarsa(λ) (/ TD(λ))

$$\begin{aligned} q_t^{\lambda} &= (1-\lambda)\textstyle\sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \\ Q(s_t,a_t) &\leftarrow Q(s_t,a_t) + \alpha\left[q_t^{\lambda} - Q(s_t,a_t)\right] \end{aligned} \quad (14)$$

Backward View Sarsa(λ) (/ TD(λ))

\+ more efficient

\+ can update at every time-step

- eligibility traces : find cause in frequency vs recency

$$\begin{aligned} E_0(s,a) &= 0 \\ E_t(s,a) &= \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = t, A_t = a) \\ \delta_t &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\ Q(s,a) &\leftarrow Q(s,a) + \alpha \delta_t E_t(s,a) \end{aligned} \quad (15)$$

Linear Function Approximation

\+ efficient

\+ generalize

update temporal difference error, minimize squared loss:

$$\begin{aligned} \delta &\leftarrow r_t + \gamma\theta^T\phi(s_{t+1}) - \theta^T\phi(s_t) \\ \theta &\leftarrow \theta + \alpha\delta\phi(s_t) \\ J(\theta) &= ||\delta||^2 \end{aligned} \quad (16)$$

Q Learning

$$\begin{aligned} \delta &\leftarrow r_t + \gamma \operatorname{argmax}_{a \in A} Q(\phi(s_{t+1},a);\theta_i^-) - Q(\phi(s_t,a);\theta_i) \\ Q(s,a) &\leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \end{aligned} \quad (17)$$

Replay Memory

update θ by SGD

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\, \delta \nabla_{\theta_i} Q(\phi(s_t,a_t);\theta) \quad (18)$$

Deep Q Learning (DQL)

Made by *DeepMind*, uses a deep neural net (*Q-network*) for the Q function. Keeps N observations in a *memory* to train on.

$$\begin{aligned} y &= r_t + \gamma \max_{a \in A} Q(\phi(s_{t+1},a);\theta_i^-) \\ \nabla_i J(\theta_i) &= \mathbb{E}_{(s,a,r,s')\sim U(D)}[(y - \underbrace{Q(s,a;\theta_i)}_{\text{prediction}})^2] \end{aligned} \quad (19)$$

for network weights θ and experience replay history $U(D)$.

Initialize replay memory D with capacity N
Initialize $Q(s,a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  Pick a from s by policy from Q (e.g. ϵ-greedy)
  while *s is not terminal* **do**
    Take action a, observer r, s'
    Store transition (s,a,r,s') in D
    Sample random transitions from D
    $y_i \leftarrow \begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s',a';\theta) & \text{otherwise} \end{cases}$
    Perform gradient descent step on
      $(y_j - Q(s_j,a_j;\Theta))^2$
    $s \leftarrow s'$
  end
end

Prioritized Replay Memory

learn esp. from high loss (traumas)

$$p(s_t, a_t, r_t, s_{t+1}) \propto r_t + \gamma \max_{a \in A} Q(\Phi(s_{t+1}, a); \theta_i^-) \tag{20}$$

Double DQN

solve bias from reused θ, faster

$$y = r_t + \gamma Q(\Phi(s_{t+1}, \underset{a \in A}{\text{argmax}} Q(\Phi(s_{t+1}, a); \theta_i)); \theta_i^-) \tag{21}$$

Policy Gradient

learn π.

+ simpler than Q or V
+ allows stochastic policy (rock-paper-scissors)
− local optimum
− less sample efficient

$$L = \mu(logp(a|s) \cdot R(s)) \tag{22}$$

Likelihood Ratio

return state-action trajectory:

$$R(\tau) = \sum_{t=0}^{T} R(s_t, a_t) \tag{23}$$

expected return:

$$\begin{aligned} J(\theta) &= \mathbb{E}[\textstyle\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta] \\ &= \textstyle\sum_{\tau}^{T} R(s_t, a_t); \pi_\theta \end{aligned} \tag{24}$$

find θ to max:

$$J(\theta) = \sum_{\tau} P(\tau; \theta) R(\tau) \tag{25}$$

yielding:

$$\max_{\theta} J(\theta) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau) \tag{26}$$

$$\nabla_\theta J(\theta) = \sum_{\tau} P(\tau; \theta) \nabla_\theta logP(\tau; \theta) R(\tau) \tag{27}$$

sampled over m trajectories:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} P(\tau; \theta) \nabla_\theta logP(\tau^i; \theta) R(\tau^i) \tag{28}$$

gradient chases reward.

REINFORCE

- Get info on states/actions/rewards for policy during episode
- Calc episode return for collected rewards
- Update model params toward policy gradient

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R \tag{29}$$

desired loss function:

$$\frac{1}{m} \sum_{t=1}^{m} \nabla_\theta \log \pi_\theta(a_t|s_t) R \tag{30}$$

Baselined REINFORCE

use baselined rewards

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R - V_{\phi(s_t)}) \tag{31}$$

Actor-critic

+ less variance than Baselined REINFORCE
- actor (makes policy): policy gradient
- critic: policy iteration

advantage:

$$A_\pi(s, a) = Q(s, a) - V(s) \tag{32}$$

Asynchronous Advantage Actor Critic (A3C)

- 5-step Q-Value estimation
- shares params between actor/critic
+ run parallel, one policy
+ no more need for DQN's replay policy

github.com/tycho01/Reinforcement-Learning-Cheat-Sheet