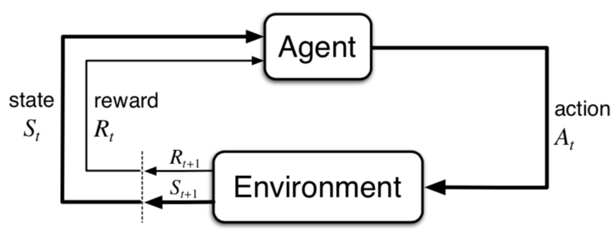
Reinforcement Learning (RL)

Symbols

a action
A advantage
α learning rate, step size
b bandit, baseline estimate
c cost, context (= state)
d difference error
Δ difference
D replay buffer
E eligibility trace
ϵ exploration rate ($0 \sim 1$)
Φ state transition function
g gain
γ discount factor
∇ gradient
H horizon
J expected return
L loss / regret
λ trace decay ($0 \sim 1$)
Pr transition distribution
π policy (*state* $\rightarrow$ *action*)
q quality
r reward
R return
s state
t time
τ trajectory
θ parameter weight
v value
w weight
y target / prediction
$_*$ optimal
$'$ next / derivative
$\hat{}$ estimation
$\|\|$ vector norm

Agent-Environment Interface



General model

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

General challenges

- choosing the right *Representation* of the problem / state
- *Generalization* beyond the cases trained on
- *Temporal Credit Assignment*: what caused this outcome?
- balance *Exploitation* vs *Exploration* (short vs long term)

Concepts (no state)

$$\underbrace{a}_{\text{state}} \in \underbrace{A}_{\text{finite set of actions}}$$

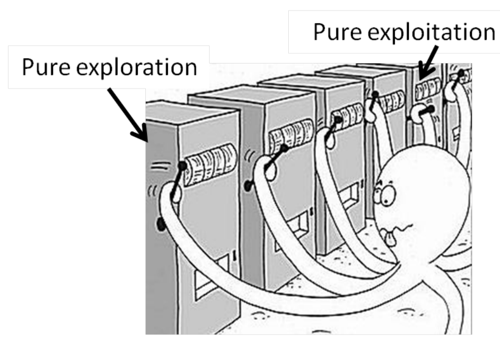
$$\underbrace{R}_{\text{return}} = \underbrace{\sum_{t=0}^{\overbrace{H}^{\text{horizon } (\infty?)}} r_t}_{\text{total } reward \text{ over time}}$$

$$\underbrace{G_t}_{\text{gain}} = \underbrace{\sum_{k=0}^{H} \overbrace{\gamma^k}^{\text{discount factor}} r_{t+k+1}}_{\text{total discounted reward over time}} \tag{1}$$

$$\underbrace{L}_{\text{regret}} = \underbrace{\overbrace{G^*}^{\text{optimal gain}} - \overbrace{G_\pi}^{\text{gain under policy}}}_{\text{gain missed under policy}}$$

Bandit algorithms

- *multi-armed bandit problem*: which slot arm gives most?
- no notion of state!
- this leaves just *exploration vs exploitation*



Bandits = heuristics. Image from research.microsoft.com

$$\underbrace{n_a}_{\text{action's use count}} = \underbrace{\sum_{t=0}^{T} \begin{cases} 1 & a_t = a \\ 0 & \text{otherwise} \end{cases}}_{\text{tries for action so far}}$$

$$\underbrace{\hat{r}_a}_{\text{action's expected reward}} = \underbrace{\sum_{t}^{a} \frac{r_t}{n_a}}_{\text{average action reward so far}} \tag{2}$$

Greedy

- 100% exploitation
- wants to pick best arm but won't know enough
- – if 1st arm ok won't try others (stuck in local optimum)

$$\underbrace{a_t}_{\text{action at time } t} = \underbrace{\operatorname*{argmax}_{a \in A} \hat{r}_a}_{\text{action with max expected reward}}$$

$$\underbrace{L_T}_{\text{total regret}} \propto \underbrace{O(n)}_{\text{order of magnitude } linear \text{ to the number of actions}} \tag{3}$$

Optimistic-Greedy

- like greedy but large initial $\hat{r}_a$
- \+ tries all
- – no 2nd chance for arms unlucky on first try

ϵ-Greedy

- at probability ϵ pick randomly
- \+ not fooled by unlucky first try
- – linear regret from constant exploration

Upper Confidence Bound (UCB)

try all for k rounds, then

$$a_t = \operatorname*{argmax}_{a \in A}[\underbrace{\hat{r}_a}_{\text{reward}} + \underbrace{\sqrt{\frac{2 \log t}{n_a}}}_{\text{under-explored bonus}}] \tag{4}$$

- explore then gradually exploit
- \+ logarithmic regret

Contextual Bandit

Linear UCB (LinUCB)

$$\begin{aligned} x_{t,a} &= \Phi(s_{t,a}) \\ \mathbb{E}[r_{t,a}|x_{t,a}] &= \theta_a \cdot x_{t,a} \\ \hat{\theta}_a &= A^{-1} b \end{aligned} \tag{5}$$

Posterior / Thompson sampling

Pick actions by probability they maximize expected reward

Greedy in the Limit with Infinite Exploration (GLIE)

- infinitely explores
- converges on greedy

Markov Decision Process (MDP)

a 5-tuple (S, A, P, R, γ)

$$
\begin{aligned}
\underbrace{s}_{\text{state}} &\in \underbrace{S}_{\text{finite set of states}} \\
\underbrace{\pi_t(a|s)}_{\text{policy}} &: \text{a mapping from a state to an action} \\
p(s'|s,a) &= \underbrace{Pr\{S_{t+1}=s'|S_t=s,A_t=a\}}_{\text{state transition probabilities}} \\
r(s',s,a) &= \underbrace{\mathbb{E}[R_{t+1}|S_{t+1}=s',S_t=s,A_t=a]}_{\text{expected return for state-action-nexstate}} \\
\underbrace{v_\pi(s)}_{\text{value function}} &= \underbrace{\mathbb{E}_\pi[G_t|S_t=s]}_{\text{expected gain under policy } \pi \text{ in state } s} \\
&= \underbrace{\sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)[r+\gamma v_\pi(s')]}_{\text{recursive definition (Bellman equation)}} \\
\underbrace{q_\pi(s,a)}_{\text{action-value (Q) function}} &= \underbrace{\mathbb{E}_\pi[G_t|S_t=s,A_t=a]}_{\text{expected reward under policy } \pi \text{ for a state-action pair}} \\
&= \underbrace{\sum_{s',r} p(s',r|s,a)[r+\gamma V_\pi(s')]}_{\text{recursive definition (Bellman equation)}} \\
\underbrace{q_*(s,a)}_{\text{optimal Q function}} &= \max_\pi q_\pi(s,a) \\
\underbrace{v_*(s)}_{\text{state value under optimal policy}} &= \max_\pi v_\pi(s) \\
&= \underbrace{\max_{a\in A(s)} q_{\pi_*}(s,a)}_{\text{expected return from its best action}}
\end{aligned} \tag{6}
$$

Contraction Mapping

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

$$d(f(x), f(y)) \leq kd(x, y) \tag{7}$$

Contraction Mapping theorem

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

Model-based Methods (known MDP)

Exhaustive Search

brute force, usually computationally unviable.

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:

$$V(S_t) += \alpha[G_t - V(S_t)] \tag{8}$$

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:

$$V(S_t) += \alpha[R_{t+1} + \gamma V(S_{t+1} - V(S_t)] \tag{9}$$

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)
    $y \leftarrow r + \gamma Q(s', a')$
    $Q(s,a) += \alpha[y - Q(s,a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

n-step Sarsa (n-step TD) for n-step Q-Return:

$$
\begin{aligned}
q_t^{(n)} &= \gamma^n Q(S_{t+n}) + \textstyle\sum_{i=1}^n \gamma^{i-1} R_{t+i} \\
Q(s_t, a_t) &+= \alpha \left[q_t^{(n)} - Q(s_t, a_t)\right]
\end{aligned} \tag{10}
$$

Forward View Sarsa(λ) (/ TD(λ))

$$
\begin{aligned}
q_t^\lambda &= (1-\lambda) \textstyle\sum_{n=1}^\infty \lambda^{n-1} q_t^{(n)} \\
Q(s_t, a_t) &+= \alpha \left[q_t^\lambda - Q(s_t, a_t)\right]
\end{aligned} \tag{11}
$$

Backward View Sarsa(λ) (/ TD(λ))

+ more efficient
+ can update at every time-step
- eligibility traces : find cause in frequency vs recency

$$\begin{aligned} E_0(s,a) &= 0 \\ E_t(s,a) &= \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = t, A_t = a) \\ \delta_t &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\ Q(s,a) &\mathrel{+}= \alpha\delta_t E_t(s,a) \end{aligned} \tag{12}$$

Linear Function Approximation

+ efficient
+ generalize

update temporal difference error, minimize squared loss:

$$\begin{aligned} \delta &\leftarrow r_t + \gamma\theta^T\Phi(s_{t+1}) - \theta^T\Phi(s_t) \\ \theta &\mathrel{+}= \alpha\delta\Phi(s_t) \\ J(\theta) &= ||\delta||^2 \end{aligned} \tag{13}$$

Q Learning

$$\begin{aligned} \delta &\leftarrow r_t + \gamma \operatorname*{argmax}_{a\in A} Q(\Phi(s_{t+1},a);\theta_i^-) - Q(\Phi(s_t,a);\theta_i) \\ Q(s,a) &\mathrel{+}= \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \end{aligned} \tag{14}$$

Replay Memory

update θ by SGD

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\,\delta\nabla_{\theta_i} Q(\Phi(s_t,a_t);\theta) \tag{15}$$

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.

$$\begin{aligned} y &= r_t + \gamma \max_{a\in A} Q(\Phi(s_{t+1},a);\theta_i^-) \\ \underbrace{\nabla_i J(\theta_i)}_{\text{loss function}} &= \mathbb{E}_{(s,a,r,s')\sim \underbrace{U(D)}_{\text{memory}}}[(\underbrace{y}_{\text{target}} - \underbrace{Q(s,a;\theta_i)}_{\text{prediction}})^2] \end{aligned} \tag{16}$$

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)

$$p(s_t,a_t,r_t,s_{t+1}) \propto r_t + \gamma \max_{a\in A} Q(\Phi(s_{t+1},a);\theta_i^-) \tag{17}$$

Double DQN

solve bias from reused θ, faster

$$y = r_t + \gamma Q(\Phi(s_{t+1}, \operatorname*{argmax}_{a\in A} Q(\Phi(s_{t+1},a);\theta_i));\theta_i^-) \tag{18}$$

Direct Policy Search

learn π.

Policy Gradient

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

$$L = \mu(logp(a|s)\cdot R(s)) \tag{19}$$

Likelihood Ratio

$$\begin{aligned} & \text{return state-action trajectory:} \\ R(\tau) &= \textstyle\sum_{t=0}^T R(s_t,a_t) \\ & \text{expected return:} \\ J(\theta) &= \mathbb{E}[\textstyle\sum_{t=0}^T R(s_t,a_t);\pi_\theta] \\ &= \textstyle\sum_\tau^T R(s_t,a_t);\pi_\theta \\ & \text{find } \theta \text{ to max:} \\ &= \textstyle\sum_\tau P(\tau;\theta)R(\tau) \\ & \text{yielding:} \\ \max_\theta J(\theta) &= \max_\theta \textstyle\sum_\tau P(\tau;\theta)R(\tau) \\ & \text{gradient chasing reward:} \\ \nabla_\theta J(\theta) &= \textstyle\sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)R(\tau) \\ & \text{sampled over m trajectories:} \\ &= \tfrac{1}{m}\textstyle\sum_{i=1}^m P(\tau;\theta)\nabla_\theta \log P(\tau_i;\theta)R(\tau_i) \\ & \text{combined:} \\ &= \tfrac{1}{m}\textstyle\sum_{i=1}^m \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau_i) \end{aligned} \tag{20}$$

REINFORCE

- check policy/episode's states/actions/rewards
- calc episode return for collected rewards
- update model params toward policy gradient

$$\nabla_\theta J(\theta) = \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t)R \tag{21}$$

desired loss function:

$$\frac{1}{m}\sum_{t=1}^m \nabla_\theta \log \pi_\theta(a_t|s_t)R \tag{22}$$

Baselined REINFORCE

$$\nabla_\theta J(\theta) = \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t)\ \underbrace{(R - V_{\Phi(s_t)})}_{\text{baselined reward}} \tag{23}$$

Actor-critic

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

$$\underbrace{A_\pi(s,a)}_{\text{advantage}} = Q(s,a) - V(s) \tag{24}$$

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