Topics

- Temporal-difference Learning
- n-step TD
- Planning
Temporal-difference learning

Bootstrap method (update value estimates from other value estimates)

On-policy prediction:
\[ V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(s)] \]

Policy control:
Sarsa:
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma - Q(S_t, A_t)] \]

Q-learning:
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma - Q(S_t, A_t)] \]

Expected Sarsa:
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma - Q(S_t, A_t)] \]
Maximization bias

\[ \max_a Q(S', a) \] overestimates the max Q value

Solution:
Keep two Qs whose estimates are independent: \( Q_1, Q_2 \)

Unbiased estimate of max Q value:
\[ Q_1(S', \text{argmax}_a Q_2(S', a)) \]
or:
\[ Q_2(S', \text{argmax}_a Q_1(S', a)) \]
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in S^+$, $a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probability:
      $$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left( R + \gamma Q_2(S', \text{argmax}_a Q_1(S', a)) - Q_1(S, A) \right)$$
    else:
      $$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left( R + \gamma Q_1(S', \text{argmax}_a Q_2(S', a)) - Q_2(S, A) \right)$$
  $S \leftarrow S'$
  until $S$ is terminal
n-step TD

n-step TD prediction:

Total reward from $t$ to $t + n$:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

Update function:

$$V(S_t) \leftarrow V(S_t) + \alpha [G_{t:t+n} - V(S_t)]$$
**n-step TD for estimating** $V \approx v_\pi$

**Input:** a policy $\pi$
**Algorithm parameters:** step size $\alpha \in (0, 1]$, a positive integer $n$
**Initialize** $V(s)$ arbitrarily, for all $s \in S$
**All store and access operations (for $S_t$ and $R_t$) can take their index mod $n + 1$

**Loop for each episode:**
- Initialize and store $S_0 \neq$ terminal
- $T \leftarrow \infty$
- **Loop for** $t = 0, 1, 2, \ldots$
  - If $t < T$, then:
    - Take an action according to $\pi(\cdot | S_t)$
    - Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    - If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
  - $\tau \leftarrow t - n + 1$ (\(\tau\) is the time whose state’s estimate is being updated)
  - If $\tau \geq 0$:
    - $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    - If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ \((G_{\tau:\tau+n})\)
    - $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  - Until $\tau = T - 1$
n-step TD control

\[ G_{t:t+n} : \text{Total reward from time } t \text{ to } t + n \]

Sarsa:
\[
R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n
\]

Expected Sarsa:
\[
R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n
\]

Function for both: \( Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_{t:t+n} - Q(S_t, A_t)] \)
**n-step Sarsa for estimating** $Q \approx q_*$ or $q_\pi$

Initialize $Q(s,a)$ arbitrarily, for all $s \in S, a \in A$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t, A_t$, and $R_t$) can take their index mod $n + 1$

Loop for each episode:
- Initialize and store $S_0 \neq$ terminal
- Select and store an action $A_0 \sim \pi(\cdot|S_0)$
- $T \leftarrow \infty$
- Loop for $t = 0, 1, 2, \ldots :$
  - If $t < T$, then:
    - Take action $A_t$
    - Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
  - If $S_{t+1}$ is terminal, then:
    - $T \leftarrow t + 1$
  - else:
    - Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    - $\tau \leftarrow t - n + 1$ \hspace{1em} (\tau is the time whose estimate is being updated)
    - If $\tau \geq 0$:
      - $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      - If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
      - $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
  - If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$

Until $\tau = T - 1$
n-step Off-policy prediction

\[ \rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \]

\[ V(S_t) \leftarrow V(S_t) + \alpha \rho_{t:t+n-1} [G_{t:t+n} - V(S_t)] \]
n-step Off-Policy control

Sarsa:
\[ G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) \]
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho_{t+1:t+n}[G_{t:t+n} - Q(S_t, A_t)] \]

Expected Sarsa:
\[ G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n})Q(S_{t+n}, a) \]
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho_{t+1:t+n-1}[G_{t:t+n} - Q(S_t, A_t)] \]
n-step TD: Off-policy Per-decision Methods: State value

Without control variate:

\[ G_{t:h} = \rho_t (R_{t+1} + \gamma G_{t+1:h}) \]

With control variate:

\[ G_{t:h} = \rho_t (R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t) V(S_t) \]

In both cases:

\[ G_{t:t+n} = V(S_h) \]

\[ V(S_t) \leftarrow V(S_t) + \alpha [G_{t:t+n} - V(S_t)] \]
n-step TD: Off-policy Per-decision Methods: Action value

Without control variate:

\[ G_{t:h} = R_{t+1} + \gamma \rho_{t+1} G_{t+1:h} \]

With control variate:

\[ G_{t:h} = R_{t+1} + \gamma \rho_{t+1} (G_{t+1:h} - Q(S_{t+1}, A_{t+1})) + \gamma \overline{V}(S_{t+1}) \]

where \( \overline{V}(s) = \sum_a \pi(a|s)Q(s, a) \)
Off-Policy learning **without** importance sampling: n-step Tree Backup

\[ G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q(S_{t+1}, a) + \gamma \pi(A_{t+1}, S_{t+1})G_{t+1:t+n} \]
n-step $Q(\sigma)$

A blend between tree-backup and per-decision importance sampling
Types of Models

- Distributional model
- Sample model
What is planning?

model -> (planning) -> policy
Random-sample one-step tabular Q-planning

Loop forever:
1. Select a state, $S$, and an action $A$ at random
2. Send $S, A$ to a sample model and obtain a sample next reward $R$, and a sample next state, $S'$
3. Apply one-step tabular Q-learning to $S, A, R, S'$
   
   $$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$$
Online planning
Tabular Dyna-Q Overview

- Direct RL method: one-step tabular Q-learning
- Model-learning method:
  - Assumes environment is deterministic
  - Table-based
  - Given $A_t, S_t \rightarrow R_{t+1}, S_{t+1}$, stores model[$(S_t, A_t)$] = $(R_{t+1}, S_{t+1})$
Tabular Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $a, s$

Loop forever:
1. $S \leftarrow$ current (nonterminal) state
2. $A \leftarrow \epsilon - \text{greedy}(S, Q)$
3. Take action $A$; observe resultant reward $R$ and state $S'$
4. $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
5. $Model(S, A) \leftarrow (R, S')$ (assumes deterministic environment)
6. Loop 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+: Dyna-Q + heuristics for encouraging

• Provide an implicit reward to exploring stale transitions

\[ Q(S, A) \leftarrow Q(S, A) + \alpha[R + k\sqrt{\tau(S, A)} + \gamma \max_a Q(S', a) - Q(S, A)] \]

• Allow actions that had never been tried from a state to be considered in planning (initial model was that such an action led back to the same state with a reward of 0)
Sampling

- Uniform
- Prioritized Sweeping
- Trajectory
Monte Carlo Tree Search (MCTS)

Improves the decision that would be made by the rollout policy (or at least no worse)

Repeat while time remaining starting with current state:
1. **Selection**: Select a leaf node in the expanded tree
2. **Expansion**: Expand a child of the leaf node
3. **Simulation**: Follow rollout-policy from expanded node to simulate complete episode
4. **Backup**: Backup action values to nodes in the tree
Selection: choose a leaf node in the MCTS tree

Traverse MCTS tree until reach a node with unexpanded children.

Use Upper Confidence Bound for Trees (UCT) to decide most promising child.

\[ UCT(v) = \frac{q(v)}{n(v)} + c \sqrt{\frac{\log n(v, \text{parent})}{n(v)}} \]

- \( q(v) \): Total simulation reward for node \( v \)
- \( n(v) \): Total number of visits (simulation backups) for node \( v \)
Expansion: expand selected node

If selected node is not terminal, choose an untried action and create a new MCTS node for the state that generates
Simulation: rollout starting at expanded node

Monte Carlo simulation using rollout policy until terminal state is reached.

Record total reward.
Backup: backup action values to nodes in the MCTS tree

Update $n(v)$ and $q(v)$ for each node $v$ in the MCTS tree from simulation node up to root.

Note: If two-player competitive game, adjust reward to reflect who made the move.

For example, if reward is +1 (player 1 won):

- For $v$ reached from player 1 move, increment $q(v)$
- For $v$ reached from player 2 move, decrement $q(v)$
Selecting a final action

Probably don't want exploration term in UCT

- Child with highest $\frac{q(v)}{n(v)}$, or
- Child of root with highest $N(v)$—it's the one that was explored the most so must have been most promising overall.