Planning, Learning and Intelligent Decision Making

Markov chains

Markov property: the state at instant t is enough to predict the state at instant $t+1$

$$\Pr(X_{t+1} = y \mid X_{0..t} = x_{0..t}) = \Pr(X_{t+1} = y \mid X_t = x_t)$$

Markov chain: sequence of random variables $X_i \in \mathcal{X}$

- Follows the Markov property
- $\mathcal{X}$ is the **state space**
- $P \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ is the **transition probability matrix**

$$[P]_{xy} = \Pr(X_{t+1} = y \mid X_t = x)$$

The transition probability after k steps is given by $[P^k]_{xy} = \Pr(X_{t+k} = y \mid X_t = x)$

Irreducible chain: any state can be reached from any other state

$$\forall_{x,y \in \mathcal{X}}, \exists_{k \in \mathbb{N}} : [P^k]_{xy} > 0$$

- An irreducible chain has a single communicating class

Aperiodic chain: all states are aperiodic, otherwise periodic

- A state x is aperiodic if it's period d_x is 1

$$d_x = \gcd\{t \in \mathbb{N} \mid P^t(x|x) > 0, t > 0\} = 1$$

Distribution over a state space: μ is a distribution over $\mathcal{X}$ if

$$\mu = \begin{bmatrix} \Pr(X = x_1) & \Pr(X = x_2) & \dots & \Pr(X = x_{|\mathcal{X}|}) \end{bmatrix}$$

Stationary distribution: $\mu P = \mu$

Positive chain: possesses a stationary distribution, otherwise a null chain

Ergodic chain: irreducible and aperiodic

- An ergodic chain possesses a stationary distribution μ^* such that for any initial distribution μ_0, $\lim_{k \to \infty} \mu_0 P^k = \mu^*$

Hidden Markov models

Hidden Markov model: sequence of pairs of random variables (X_i, Z_i), where

- The state at instante t is enough to predict the state at instant $t+1$
- The state at instante t is enough to predict the observation at instant t
- $\mathcal{X}$ is the **state space**
- $\mathcal{Z}$ is the **observation space**
- $P \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ is the **transition probability matrix**

$$[P]_{xy} = \Pr(X_{t+1} = y \mid X_t = x)$$

- $O \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Z}|}$ is the **observation probability matrix**

$$[O]_{xz} = \Pr(Z_t = z \mid X_t = x)$$

Estimation: given a sequence of observations $z_{0..t}$

- **Filtering**: estimate the final state $P_{\mu_0}(X_t = x \mid Z_{0..t} = z_{0..t})$ for an initial distribution μ_0
  - Forward algorithm
- **Smoothing**: estimate the sequence of states
  - Forward-Backward algorithm for marginal smoothing (only one state)
  - Viterbi algorithm for smoothing (valid sequence of states)
- **Prediction**: predict future states
  - Forward algorithm followed by markov property

Forward mapping: given a sequence of observations, the forward mapping $\alpha_t : \mathcal{X} \to \mathbb{R}$ is defined as

$$\alpha_t(x) = P_{\mu_0}(X_t = x, Z_{0..t} = z_{0..t})$$

- We can compute $\mu_{t|0:t}(x) = P_{\mu_0}(X_t = x \mid Z_{0..t} = z_{0..t})$ from $\alpha_t(x)$

$$\mu_{t|0:t}(x) = \frac{\alpha_t(x)}{\sum_{y \in \mathcal{X}} \alpha_t(y)} \qquad \mu_{t|0:t} = \frac{\alpha_t}{||\alpha_t||_1}$$

- The forward mapping can be computed recursively

$$\alpha_t(x) = O(z_t \mid x) \sum_{y \in \mathcal{X}} P(x \mid y) \alpha_{t-1}(y)$$

$$\alpha_t^\top = \text{diag}(O(z_t \mid \cdot)) P^\top \alpha_{t-1}^\top$$

$$\alpha_0^\top = \text{diag}(O(z_t \mid \cdot)) \mu_0^\top$$

- The forward algorithm uses the forward mapping for filtering

Backward mapping: given a sequence of observations, the backward mapping $\beta_t : \mathcal{X} \to \mathbb{R}$ is defined as

$$\beta_t(x) = P(Z_{t+1..T} = z_{t+1..T} \mid X_t = x)$$

- We can compute $\mu_{t|0:T}(x) = P_{\mu_0}(X_t = x \mid Z_{0..T} = z_{0..T})$ from $\alpha_t(x)$ and $\beta_t(x)$

$$\mu_{t|0:T}(x) = \frac{\alpha_t(x)\beta_t(x)}{\sum_{y\in\mathcal{X}} \alpha_t(y)\beta_t(y)} \qquad \mu_{t|0:T} = \frac{\alpha_t \odot \beta_t}{||\alpha_t \odot \beta_t||_1}$$

- The backward mapping can be computed recursively

$$\beta_t(x) = \sum_{y\in\mathcal{X}} O(z_{t+1} \mid y)\beta_{t+1}(y)P(y \mid x)$$

$$\beta_t^\top = P\,\mathrm{diag}(O(z_t \mid \cdot))\beta_{t+1}^\top$$

$$\beta_T^\top = \mathbf{1}$$

Maximizing forward mapping: given a sequence of observations, the maximizing forward mapping $m_t : \mathcal{X} \to \mathbb{R}$ is defined as

$$m_t(x) = \max_{x_{0..t-1}} P_{\mu_0}(X_t = x, X_{0..t-1} = x_{0..t-1}, Z_{0..t} = Z_{0..t})$$

Viterbi algorithm

1. Initialize $m_0 = diag(O(z_0 \mid \cdot))\mu_0^\top$
2. For $\tau = 1, \dots, T$ do
   i. Forward update
   $m_t = diag(O(z_t \mid \cdot)) \max(P^\top diag(m_{t-1}))$
   ii. $i_t = \arg\max(P^\top diag(m_{t-1}))$
3. Let $x_T^* = \arg\max_{x\in\mathcal{X}} m_T(x)$
4. For $\tau = T-1, \dots, 0$ do
   i. Backtrack $x_t^* = i_{t+1}(x_{t+1}^*)$
5. Return $x_{0..T}^*$

Preferences

A strict preference is a binary relation between outcomes:

- When outcome x is preferred to y, we write $x \succ y$
- If two outcomes are indifferent, we write $x \sim y$
- Properties
  - $\succ$ is anti-symetric, transitive and negative transitive
  - $\succeq$ is complete and transitive
  - $\sim$ is reflexive, symmetric and transitive
- A relation is called rational if it is complete and transitive

Rational preference: a relation that is complete and transitive, such as $\succeq$.

Let $\succeq$ be a rational preference on a set of possible outcomes $\mathcal{X}$. There is a utility function $u : \mathcal{X} \to \mathbb{R}$ such that $u(x) \geq u(y) \iff x \succeq y$.

An action with outcome $X(a)$ has value $Q(a) = u(X(a))$

When utilities are negative, they are called costs.

Markov decision processes

Markov decision processes: decision process X_i, A_i, C_i that satisfies the markov property

- $\mathcal{X}$ is the **state space**
- $\mathcal{A}$ is the **action space**
- For each action $a \in \mathcal{A}$, P_a is the **transition probability matrix**

$$[P_a]_{xy} = \Pr(X_{t+1} = y \mid X_t = x, a_t = a)$$

- $c(x, a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the **immediate cost function**

History: random variable containing all states and actions seen by an agent up to a time step t

$$H_t = \{X_0, A_0, X_1, A_1, \dots, X_{t-1}, A_{t-1}, X_t\} \in \mathcal{H}_t$$

Policy: mapping from histories to distributions over actions $\pi : \mathcal{H} \to \Delta(\mathcal{A})$

$$\pi(a \mid h) = \Pr(A_t = a \mid H_t = h)$$

- **Deterministic**: actions are selected with probability 1
- **Markov**: the distribution over actions depends only on the last state and t

$$\pi(a \mid h_t) = \pi_t(a \mid x)$$

- **Stationary**: the distribution over actions depends only on the last state

$$\pi(a \mid h_t) = \pi(a \mid x)$$

If an agent follows a stationary policy, the resulting process is a Markov cost process with transition probability matrix P_π and costs c_π

$$[P_\pi]_{xy} = \sum_{a\in\mathcal{A}} \pi(a \mid x)P_a(y \mid x) \qquad c_\pi(x) = \sum_{a\in\mathcal{A}} \pi(a \mid x)c(x, a)$$

Expected discounted cost-to-go: given a policy and an initial state, the cost-to-go function is

$$J^\pi(x) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t c_t \mid X_0 = x\right]$$

- a policy π^* is optimal if

$$\forall_{\pi, x\in\mathcal{X}} : J^{\pi^*}(x) \leq J^\pi(x)$$

Calculating J^π:

$$J^\pi(x) = c_\pi(x_0) + \gamma \sum_{x\in\mathcal{X}} J^\pi(x)P_\pi(x \mid x_0)$$

$$J^\pi = c_\pi + \gamma P_\pi J^\pi \implies J^\pi = (I - \gamma P_\pi)^{-1} c_\pi$$

Computing J^π with value iteration:

1. Initialize $k = 0, J_0 = 0$
2. do
    i. $J_{k+1} = T_\pi J_k = c_\pi + \gamma P_\pi J_k$
    ii. $k = k + 1$
3. while $||J_k - J_{k-1}|| > \epsilon$

Computing J^* with value iteration:

1. Initialize $k = 0, J_0 = 0$
2. do
    i. $J_{k+1} = T J_k = \min_{a \in \mathcal{A}}(c_a + \gamma P_a J_k)$
    ii. $k = k + 1$
3. while $||J_k - J_{k-1}|| > \epsilon$

Q-functions: mapping from state-action pairs to values $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

$$Q^\pi(x, a) = c(x, a) + \gamma \sum_{x' \in \mathcal{X}} P_a(x' \mid x) J^\pi(x')$$
$$Q^\pi(\cdot, a) = c(\cdot, a) + \gamma P_a J^\pi$$

- We can compute the cost-to-go functions from Q-functions

$$J^\pi(x) = \mathbb{E}_\pi[Q^\pi(x, a)] \qquad J^*(x) = \min_{a \in \mathcal{A}}[Q^*(x, a)]$$

- We can compute the optimal policy from Q^*

$$\pi^*(x) = \arg\min_{a \in \mathcal{A}}[Q^*(x, a)]$$

Computing Q^π with value iteration:

1. Initialize $k = 0, Q_0 = 0$
2. Do
    i. $Q_{k+1} = H_\pi Q_k = C + \gamma P \mathbb{E}_\pi[Q]$
    ii. $k = k + 1$
3. While $||Q_k - Q_{k-1}|| > \epsilon$

Computing Q^* with value iteration:

1. Initialize $k = 0, J_0 = 0$
2. Do
    i. $Q_{k+1} = H\, Q_k = C + \gamma P \min[Q_k]$
    ii. $k = k + 1$
3. While $||Q_k - J_{k-1}|| > \epsilon$

Greedy policy with respect to (w.r.t.) J:

$$\pi_g(x) = \arg\min_{a \in \mathcal{A}}[c(x, a) + \gamma \sum_{y \in \mathcal{X}} P_a(y \mid x) J(y)]$$
$$= \arg\min_{a \in \mathcal{A}} Q$$
$$J^{\pi_g}(x) \leq J^\pi(x)$$

Computing π^* with policy iteration (policy optimization)

1. Initialize $k = 0$
2. Initialize π_0 randomly
3. Do
    i. $J^{\pi_k} = (I - \gamma P_{\pi_k})^{-1} c_{\pi_k}$
    ii. $\pi_{k+1} = \pi_g^{J^{\pi_k}}$
    iii. $k = k + 1$
4. While $\pi_k \neq \pi_{k-1}$

- Takes less iterations than VI, but takes more time per iteration

Large problems suffer the curse of dimensionality and can't be solved exactly. Possible approximations include:

- State aggregation
    - Limited representaton power
    - Retains convergence guarantees
- Linear approximation: states described as a vector of features
    - Good representation power, difficult to choose good features
    - Does not retain convergence guarantees
- Averagers
    - Do not extrapolate
    - Retains convergence guarantees

Partially observable Markov decision processes

Partially observable MDPs

- The state and action at instante t is enough to predict the state at instant $t + 1$
- The state at instante t and action at instant $t - 1$ is enough to predict the observation at instant t
- $\mathcal{X}$ is the **state space**
- $\mathcal{A}$ is the **action space**
- $\mathcal{Z}$ is the **observation space**
- $P_a \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ is the **transition probability matrix** for each action a

$$[P_a]_{xy} = \Pr(X_{t+1} = y \mid X_t = x, A_t = a)$$

- $O_a \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Z}|}$ is the **observation probability matrix** for each action a

$$[O_a]_{xz} = \Pr(Z_t = z \mid X_t = x, A_{t-1} = a)$$

- $c(x, a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the **immediate cost function**

Belief: $b_t = \mu_{t|0..t}$

$$b_{t+1} = \frac{b_t P_{a_t} \operatorname{diag}(O_{a_t}(z_{t+1} \mid \cdot))}{||b_t P_{a_t} \operatorname{diag}(O_{a_t}(z_{t+1} \mid \cdot))||_1}$$

Belief MDP

Belief MDPs: can adapt MDP results to POMDPs

- $\mathcal{B}$ is the state space containing all beliefs
- $\mathcal{A}$ is the action space
- $P_B \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ is the **transition probability matrix**
- $c_B(b,a) : \mathcal{B} \times \mathcal{A} \to \mathbb{R}$ is the **immediate cost function**

$$c_B(b,a) = \sum_{x\in\mathcal{X}} b(x)c(x,a)$$

We can adapt MDP results to POMDPs through belief-MDPs

Heuristic solutions using the MDP associated with the POMDP:

- MLS heuristic:

$$\pi_{\text{MLS}}(b) = \pi_{\text{MDP}}(\arg\max_{x\in\mathcal{X}} b(x))$$

- AV heuristic

$$\pi_{\text{AV}}(b) = \arg\max_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} b(x)\mathbb{I}(a = \pi_{\text{MDP}}(x))$$

- Q-MDP heuristic

$$\pi_{\text{QMDP}}(b) = \arg\min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} b(x)Q^*_{\text{MDP}}(x,a)$$

- FIB heuristic:

$$Q_{\text{FIB}}(x,a) = c(x,a) + \gamma \sum_{z\in\mathcal{Z}} \min_{a'\in\mathcal{A}} \sum_{x'\in\mathcal{X}} P_a(x' \mid x)O_a(z \mid x')Q_{\text{FIB}}(x',a')$$
$$\pi_{\text{FIB}}(b) = \arg\min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} b(x)Q_{\text{FIB}}(x,a)$$

Exact solutions:

- Value iteration: the cost-to-go is PWLC (picewise linear and concave)
  - Point-based methods: select a finite set of beliefs to perform updates
- Policy iteration: representing policies as graphs

Complexity of MDPs:

- MDPs are solvable in polynomial-time
- Infinite horizon POMDPs are undecidable
- Finite-horizon POMDPs are PSPACE-complete
- POMDPs are non-approximable

Learning

Learning problem:

- T is the task
- P is the performance measure
- E is the training experience

Types of learning:

- Supervised: P is the correctness of actions in all possible situations
- Reinforcement: P is the accumulated cost of actions in all possible situations
- Unsupervised: P is the quality of structure found

No free lunches: some hypothesis are more likely than others

Inductive bias: we use assumptions to reduce the search space

Assumptions must compromise between:

- Bias: error introduced by assumptions about the hypothesis
- Variance: error introduced by the variability of the training set

Nomenclature

- Task: lean $\pi : \mathcal{X} \to \Delta(\mathcal{A})$
- Experience: dataset of examples $\mathcal{D} = \{(x_0,a_0),(x_1,a_1),\dots,(x_N,a_N)\}$, where pairs are generated from an unknown distribution μ_D
- Performance: expected cost or risk of policy π
  - $L(\pi) = \mathbb{E}_{\mu_D}[\ell(x,a;\pi)] = \sum_{x,a} \ell(x,a;\pi)\mu_D(x,a)$
  - Ideally, $\pi^* = \arg\min_\pi L(\pi)$
  - Empirical risk: $L(\pi) \approx \hat{L}_N(\pi) = \frac{1}{N}\sum_{n=1}^N \ell(x_n,a_n;\pi)$
  - ℓ is a similarity measure

Inductive learning assumption: if we learn form a sufficiently large set of examples, we will do well in the actual task

Learining from examples:

- **MDP induced metric**: uses bissimulation metric to choose an action based on the closest example from $\mathcal{D}$
- **Inverse reinforcement learning (IRL)**: recover the task from the optimal policy

- If the cost function to be computed can be arbitrary, the teacher must demonstrate the optimal action in all states
- We can get c from an optimal policy, however, if $c = 0$, all policies are optimal, and the problem is ill-defined

$$(P_\pi - P_a)(I - \gamma P_\pi)^{-1} c \leq 0$$

Stochastic approximation

Reinforcement learning:

- At each step, the agent collects a data point (x_t, a_t, c_t, x_{t+1})
- The agent must compute the optimal policy by collecting many data points
- Agent learns from reward and punishment

Methods:

- Model-based methods
- Value-based methods: Monte Carlo, TD(λ), Q-learning, SARSA
- Policy-based methods: policy gradient

Model-based methods:

- Given a sample (x_t, a_t, c_t, x_{t+1})
- Update model

$$\hat{c}(x_t, a_t) = \hat{c}(x_t, a_t) + \alpha_t(c_t - \hat{c}(x_t, a_t))$$
$$\hat{P}(x' \mid x_t, a_t) = \hat{P}(x' \mid x_t, a_t) + \alpha_t(\mathbb{I}(x_{t+1} = x') - \hat{P}(x' \mid x_t, a_t))$$

- Use value iteration or policy iteration to get Q^* or π^*

$$Q_{t+1}(x_t, a_t) = \hat{c}(x_t, a_t) + \gamma \sum_{x' \in \mathcal{X}} \hat{P}(x' \mid x_t, a_t) \min_{a' \in \mathcal{A}} Q_t(x', a')$$

Exploration in Monte Carlo RL:

- Given a trajectory obtained with policy π

$$\tau = \{x_0, c_0, x_1, c_1, \ldots, c_{T-1}, x_T\}$$

- Compute loss $L(\tau) = \sum_{t=0}^{T} \gamma^t c_t$
- Update

$$J_{t+1}(x_0) = J_t(x_0) + \alpha_t[L(\tau) - J_t(x_0)]$$

Policy optimization in **Monte Carlo RL**:

- Given a trajectory obtained with policy π

$$\tau = \{x_0, a_0, c_0, x_1, a_1, c_1, \ldots, c_{T-1}, x_T\}$$

- Compute loss $L(\tau) = \sum_{t=0}^{T} \gamma^t c_t$
- Update

$$Q_{t+1}(x_0, a_0) = Q_t(x_0, a_0) + \alpha_t[L(\tau) - Q_t(x_0, a_0)]$$

- Compute an improved policy and repeat
- We can update the cost-to-go for every state-action pair visited along a trajectory
- Sufficient exploration requires exploring starts for all state-action pairs

Temporal difference - TD(0):

- Given a sample (x_t, c_t, x_{t+1}), where action was selected from π
- Update

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t[c_t + \gamma J_t(x_{t+1}) + J_t(x_t)]$$

TD(λ):

- Given a sample (x_t, c_t, x_{t+1}), where action was selected from π
- Update

$$z_{t+1}(x) = \lambda\gamma z_t(x) + \mathbb{I}(x = x_t)$$
$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

- Converges for any $\lambda \in [0, 1]$
- Each update uses informations from multiple steps

Q-learning:

- Given a sample (x_t, a_t, c_t, x_{t+1})
- Update using temporal difference δ_t

$$Q_{t+1}(x_t, a_t) = \\ = Q_t(x_t, a_t) + \alpha_t \underbrace{[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t)]}_{\delta_t}$$

- Most known form of RL

E vs E:

- **Exploration**: trying new actions
- **Exploitation**: using the knowledge already acquired to select the better actions
- RL algorithms require visiting every state-action pair infinitely often (in practise, a large number of times), which would mean making sub-optimal actions for a long time

E vs E heuristics:

- ϵ-greedy
  - Select random action with probability ϵ
  - Select greedy action (action with smallest Q-value) with probability $1 - \epsilon$
- Boltzmann policy: η controls exploration, may grow with time

$$\pi(a \mid x) = \frac{e^{-\eta Q_t(x,a)}}{\sum_{a \in \mathcal{A}} e^{-\eta Q_t(x,a')}}$$

- Optimistic initialization
  - All Q-values are initialized to 0 or negative
  - Agent always selects greedy action

SARSA:

- Given a sample $(x_t, a_t, c_t, x_{t+1}, a_{t+1})$
- Update using temporal difference δ_t

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t[\underbrace{c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)}_{\delta_t}]$$

- Must be combined with policy improvement to learn Q^*

On vs off policy:

- **Off-policy**: learns a value of one policy while following another (Q-learning)
- **On-policy**: learns the value of the followed policy (SARSA, TD)

Gradient descent: solves curse of dimensionality by limiting the possible functions

- Start somewhere
- Compute gradient
- Take a step in the opposite direction
- Repeat until convergence

Monte carlo methods are stochastic gradient descent methods and converge, possibly to a local minimum.

TD(λ) with linear function approximation converges as long as policy π induces an ergodic Markov chain and $\mathbb{E}_{x\sim\mu}(\phi(x)\phi^\top(x))$ is non singular.

SARSA with linear function approximation converges as long as policy π induces an ergodic Markov chain and $\mathbb{E}_{x,a\sim\mu}(\phi(x,a)\phi^\top(x,a))$ is non singular. More stable than Q-learning, as long as the learning policy changes soothly with Q_t

Q-learining with linear function approximation may diverge due to:

- Function approximation
- Bootstrapping (using the previous value to calculate the new)
- Off-policy updates

Turning RL into SL (supervised learning):

- Task: learn Q^*
- Experience: $\mathcal{D}_{\text{supervised}} = \{(x_n, a_n, target_n), n = 1, \dots, N\}$
- Performance: $L(\theta) = \frac{1}{N}\sum_{n=1}^{N}(target_n - Q_\theta(x_n, a_n))^2$

Policy gradient

- Consider a family of parameterized policies $\{\pi_w, w \in \mathbb{R}^P\}$

$$\pi_w(a \mid x) = \frac{e^{w^\top \phi(x,a)}}{\sum_{a' \in \mathcal{A}} e^{w^\top \phi(x,a')}}$$

- w is the parameter of the policy
- V is a function of w

Proto-PG algorith

- Given a trajectory obtained with policy π_{w_t}

$$\tau = \{x_0, a_0, c_0, x_1, a_1, c_1, \dots, c_{T-1}, x_N\}$$

- Compute gradient

$$\hat{\nabla}_w V(w_t) \approx \sum_{n=0}^{N} \gamma^n \nabla_w \log \pi_{w_t}(x_{t+n}, a_{t+n}) Q^{\pi_{w_t}}(x_{t+n}, a_{t+n})$$

- Update

$$w_{t+1} = w_t - \alpha_t \hat{\nabla}_w V(w_t)$$

REINFORCE algorithm:

- Given a trajectory obtained with policy π_{w_t}

$$\tau = \{x_0, a_0, c_0, x_1, a_1, c_1, \dots, c_{T-1}, x_N\}$$

- Compute gradient

$$\hat{\nabla}_w V(w_t) \approx \sum_{n=0}^{N} \nabla_w \log \pi_{w_t}(x_{t+n}, a_{t+n}) \sum_{m=0}^{N} \gamma^m c_{t+m}$$

- Update

$$w_{t+1} = w_t - \alpha_t \hat{\nabla}_w V(w_t)$$

- On-policy algorithm
- A baseline can be used to decrease the variance of the gradient estimate

Sequential prediction

Weighted majority algorithm:

- Initialize weights $w_0(a) = 1, \forall_{a \in \mathcal{A}}$
- Make prediction based on weighted majority vote
- Update weights of wrong predictors as $w_{t+1}(a) = w_t(a)(1 - \eta)$
- After a mistake, at least half of the sources decreased

- Maximum number of mistakes
  $M \leq 2(1+\eta)m + \frac{2\log(N)}{\eta}$

Exponentially weighted averager:

- Initialize weights $w_0(a) = 1, \forall_{a \in \mathcal{A}}$
- Select an action according to the distribution
  $p_t(a) = \frac{w_t(a)}{||w_t||_1}$
- Update weights of all actions $w_{t+1} = w_t(a)e^{-\eta c_t(a)}$
- Makes no assumptions on how costs are selected
- Depends logarithmically on $|\mathcal{A}|$
- It's regret is sublinear in T

EXP3 (Exponentially weighted algorithm for Exporation and Exploitation):

- Initialize weights $w_0(a) = 1, \forall_{a \in \mathcal{A}}$
- Select an action according to the distribution
  $p_t(a) = \frac{w_t(a)}{||w_t||_1}$
- Update weights of all actions $w_{t+1} = w_t(a)e^{-\eta \frac{c_t(a)}{p_t(a)}\mathbb{I}(a=a_t)}$
- After a mistake, at least half of the sources decreased
- Makes no assumptions on how costs are selected
- Depends sublinearly on $|\mathcal{A}|$
- It's regret is sublinear in T
- Can be used for the multi-armed bandit problem

UCB algorithm (Upper Confidence Bound):

- Execute each action once
- From then on, at each step t select action
  $a^* = \arg\min \hat{c}(a) - \sqrt{\frac{2\log(t)}{N_t(a)}}$
- Assumes that costs follow an unknown but fixed distribution
- It's regret is sublinear in T
- Makes use of the priciple of pptimism in the face of uncertainty (unknown is probably better)

Monte Carlo Tree Search: application of UCB

1. Selection: select a node to expand, using UCB
2. Expansion: expand it
3. Simulation: simulate the process starting from the node
4. Back-propagation: back-propagate the result