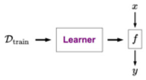


Feature Vector $\phi(x) = [\phi_1(x), \ldots, \phi_d(x)]$.

$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$

Linear Classifier

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x)) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(x) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(x) < 0 \\ ? & \text{if } \mathbf{w} \cdot \phi(x) = 0 \end{cases}$$

The score on an example (x, y) is $\mathbf{w} \cdot \phi(x)$, how **confident** we are in predicting +1.

The margin on an example (x, y) is $(\mathbf{w} \cdot \phi(x))y$, how **correct** we are.

Zero One Loss for binary classifier

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$
$$= \mathbf{1}[\underbrace{(\mathbf{w} \cdot \phi(x))y}_{\text{margin}} \leq 0]$$

Linear Regression

The **residual** is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots the target y.

Squared Loss and absdev

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\underbrace{f_{\mathbf{w}}(x) - y}_{\text{residual}})^2$$

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \phi(x) - y|$$

Objective : Minimize the Train Loss

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainLoss}(\mathbf{w})$$

Gradient Descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} (\mathbf{w} \cdot \phi(x) - y)^2$$

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} 2(\underbrace{\mathbf{w} \cdot \phi(x) - y}_{\text{prediction} - \text{target}})\phi(x)$$

Stochastic Gradient Descent

For each $(x, y) \in \mathcal{D}_{\text{train}}$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Hinge Loss

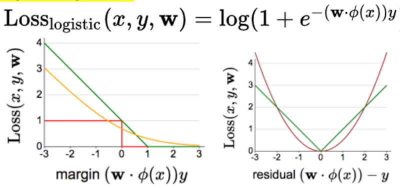
$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$

Gradient of Hinge Loss

$$\nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } \mathbf{w} \cdot \phi(x)y < 1 \\ 0 & \text{if } \mathbf{w} \cdot \phi(x)y > 1. \end{cases}$$

Logistic Regression

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w} \cdot \phi(x))y})$$

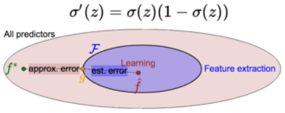


The logistic function maps $(-\infty, \infty)$ to $[0, 1]$:

$$\sigma(z) = (1 + e^{-z})^{-1}$$

Derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$



Approximation error: how good is the hypothesis class?

Estimation error: how good is the learned predictor **relative to** the hypothesis class?

$$\underbrace{\text{Err}(\hat{f}) - \text{Err}(g)}_{\text{estimation}} + \underbrace{\text{Err}(g) - \text{Err}(f^*)}_{\text{approximation}}$$

Regularization

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}}[\text{TrainLoss}(\mathbf{w})] + \lambda \mathbf{w})$$

K Means

$$\text{Loss}_{\text{kmeans}}(z, \mu) = \sum_{i=1}^{n} \|\phi(x_i) - \mu_{z_i}\|^2$$

For each point $i = 1, \ldots, n$:

Assign i to cluster with closest centroid:

$$z_i \leftarrow \arg\min_{k=1,\ldots,K} \|\phi(x_i) - \mu_k\|^2$$

For each cluster $k = 1, \ldots, K$:

Set μ_k to average of points assigned to cluster k:

$$\mu_k \leftarrow \frac{1}{|\{i : z_i = k\}|} \sum_{i : z_i = k} \phi(x_i)$$

Initialize $\mu_1, \ldots, \mu_K$ randomly.

For $t = 1, \ldots, T$:

Step 1: set assignments z given μ

Step 2: set centroids μ given z

Legend: b actions/state, solution depth d, maximum depth D

Algorithm	Action costs	Space	Time
DFS	zero	$O(D)$	$O(b^D)$
BFS	constant ≥ 0	$O(b^d)$	$O(b^d)$
DFS-ID	constant ≥ 0	$O(d)$	$O(b^d)$
Backtracking	any	$O(D)$	$O(b^D)$

UCS Algorithm

Add s_{start} to **frontier** (priority queue)

Repeat until frontier is empty:

Remove s with smallest priority p from frontier

If $\text{IsEnd}(s)$: return solution

Add s to **explored**

For each action $a \in \text{Actions}(s)$:

Get successor $s' \leftarrow \text{Succ}(s, a)$

If s' already in explored: continue

Update **frontier** with s' and priority $p + \text{Cost}(s, a)$

DP vs UCS

N total states, n of which are closer than end state

Algorithm	Cycles?	Action costs	Time/space
DP	no	any	$O(N)$
UCS	yes	≥ 0	$O(n \log n)$

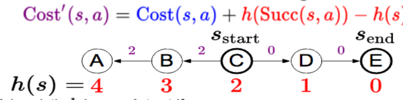
Structured Perceptron Alogrithm

- For each action: $\mathbf{w}[a] \leftarrow 0$
- For each iteration $t = 1, \ldots T$:
  - For each training example $(x, y) \in \mathcal{D}_{\text{train}}$:
    - Compute the minimum cost path y' given $\mathbf{w}$
    - For each action $a \in y$: $\mathbf{w}[a] \leftarrow \mathbf{w}[a] - 1$
    - For each action $a \in y'$: $\mathbf{w}[a] \leftarrow \mathbf{w}[a] + 1$

A* Algorithm

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s, a) = \text{Cost}(s, a) + h(\text{Succ}(s, a)) - h(s)$$



A heuristic h is **consistent** if

- $\text{Cost}'(s, a) = \text{Cost}(s, a) + h(\text{Succ}(s, a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

Heuristic is admissible if $h(s) \leq \text{FutureCost}(s)$

Relaxation

A **relaxation** P_{rel} of a search problem P has costs that satisfy:

$$\text{Cost}_{\text{rel}}(s, a) \leq \text{Cost}(s, a).$$

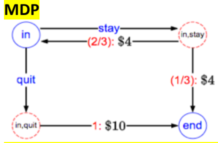
Given a relaxed search problem P_{rel}, define the **relaxed heuristic** $h(s) = \text{FutureCost}_{\text{rel}}(s)$, the minimum cost from s to an end state using $\text{Cost}_{\text{rel}}(s, a)$.

And also that means h(s) is consistent.

Note : Heuristic Tradeoff: efficiency(relaxed) vs tightness(not too relaxed)

Note:if h1 and h2 are consistent then max (h1,h2) is too.

MDP



$$\sum_{s' \in \text{States}} T(s, a, s') = 1$$

Search State plus Reward and Transition Prob, Disc factor

A **policy** π is a mapping from each state $s \in \text{States}$ to an action $a \in \text{Actions}(s)$.

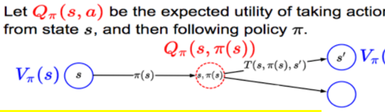
Path: $s_0, a_1 r_1 s_1, a_2 r_2 s_2, \ldots$ (action, reward, new state).

The **utility** with discount γ is

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$$

Let $V_\pi(s)$ be the expected utility received by following policy π from state s.

Let $Q_\pi(s, a)$ be the expected utility of taking action a from state s, and then following policy π.



Policy Evaluation: recurrences and Algorithm

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

Initialize $V_\pi^{(0)}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \ldots, t_{\text{PE}}$:

For each state s:

$$V_\pi^{(t)}(s) \leftarrow \underbrace{\sum_{s'} T(s, \pi(s), s')[\text{Reward}(s, \pi(s), s') + \gamma V_\pi^{(t-1)}(s')]}_{Q^{(t-1)}(s, \pi(s))}$$

Convergence: $\max_{s \in \text{States}} |V_\pi^{(t)}(s) - V_\pi^{(t-1)}(s)| \leq \epsilon$

Optimal value and Policy

The **optimal value** $V_{\text{opt}}(s)$ is the maximum value attained by any policy.

$$Q_{\text{opt}}(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}(s')].$$

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a) & \text{otherwise.} \end{cases}$$

$$\pi_{\text{opt}}(s) = \arg\max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a)$$

Value Iteration Algorithm:

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \ldots, t_{\text{VI}}$:

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s, a)}$$

Note: converges if mdp is acyclic or disc factor < 1

Reinforcement Learning: no T or R defined in MDP.

Model based Monte Carlo(optimal policy)

Data: $s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$

$$\hat{T}(s, a, s') = \frac{\#\text{ times } (s,a,s') \text{ occurs}}{\#\text{ times } (s,a) \text{ occurs}}$$

$$\widehat{\text{Reward}}(s, a, s') = r \text{ in } (s, a, r, s')$$

$$\hat{Q}_{\text{opt}}(s, a) = \sum_{s'} \hat{T}(s, a, s')[\widehat{\text{Reward}}(s, a, s') + \gamma \hat{V}_{\text{opt}}(s')]$$

Model Free Monte Carlo

$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$

$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

$\hat{Q}_\pi(s, a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$

(and s, a doesn't occur in $s_0, \cdots, s_{t-2}$)

On each (s, a, u):

$$\hat{Q}_\pi(s, a) \leftarrow \hat{Q}_\pi(s, a) - \eta[\underbrace{\hat{Q}_\pi(s, a)}_{\text{prediction}} - \underbrace{u}_{\text{target}}]$$

Bootstrapping Methods: SARSA

The main advantage that SARSA offers over model-free Monte Carlo is that we don't have to wait until the end of the episode to update the Q-value.

On each (s, a, r, s', a'):

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta[\underbrace{r}_{\text{data}} + \underbrace{\gamma \hat{Q}_\pi(s', a')}_{\text{estimate}}]$$

Q-Learning Algorithm(Off Policy and no Succ States reqd)

On each (s, a, r, s'):

$$\hat{Q}_{\text{opt}}(s, a) \leftarrow (1 - \eta)\underbrace{\hat{Q}_{\text{opt}}(s, a)}_{\text{prediction}} + \eta\underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a')$

Exploration Policy: Exploration vs Exploitation(RL tmplat)

For $t = 1, 2, 3, \ldots$

Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

Epsilon Greedy Policy for choosing actions

$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_{a \in \text{Actions}} \hat{Q}_{\text{opt}}(s, a) & \text{probability } 1 - \epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

Function Approx. :Large State Spaces, hard to explore

Define **features** $\phi(s, a)$ and **weights** $\mathbf{w}$:

$$\hat{Q}_{\text{opt}}(s, a; \mathbf{w}) = \mathbf{w} \cdot \phi(s, a)$$

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s, a)$$

Adversarial Games: search state plus these:

$\text{Utility}(s)$: agent's utility for end state s

$\text{Player}(s) \in \text{Players}$: player who controls state s

Game Evaluation Recurrence(Generic)

$$V_{\text{eval}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{agent}}(s, a) V_{\text{eval}}(\text{Succ}(s, a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s, a) V_{\text{eval}}(\text{Succ}(s, a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Expectimax Recurrence(opponent takes average)

$$V_{\text{exptmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s, a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s, a) V_{\text{exptmax}}(\text{Succ}(s, a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Minimax Recurrence

$$V_{\text{minmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Relationship between game values

V(Ex,pi7) >= V(max,pi7) >= V (max,pimin) >= V(Ex,pimin)

Expectiminimax Algorithm

$$V_{\text{exptminmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{coin}}(s,a) V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{coin} \end{cases}$$

Speeding up minimax : Eval func and Alpha Beta pruning

Depth Limited Search

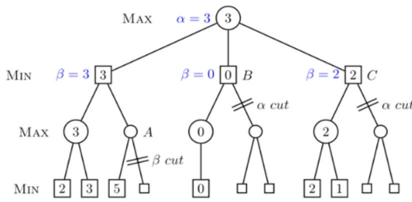
$$V_{\text{minmax}}(s, d) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \text{Eval}(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a), d) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a), d-1) & \text{Player}(s) = \text{opp} \end{cases}$$

AlphaBeta Pruning

alpha: lwr bound on val of max node

beta : uppr bound on val of min node

(store $\alpha_s = \max_{s' \preceq s} a_s$ and $\beta_s = \min_{s' \preceq s} b_s$)



TD Learning(On policy and need Succ states)

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{V}_\pi(s;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_\pi(s';\mathbf{w}))}_{\text{target}}]\nabla_{\mathbf{w}} \hat{V}_\pi(s;\mathbf{w})$$

$$V(s;\mathbf{w}) = \mathbf{w} \cdot \phi(s)$$

$$\nabla_{\mathbf{w}} V(s;\mathbf{w}) = \phi(s)$$

Simultaneous Games(Rock paper scissors)

For every simultaneous two-player zero-sum game with a finite number of actions:

$$\max_{\pi_A} \min_{\pi_B} V(\pi_A, \pi_B) = \min_{\pi_B} \max_{\pi_A} V(\pi_A, \pi_B),$$

where π_A, π_B range over **mixed strategies**.

Non Zero Sum Games(Prisoner's dilemma)

In any finite-player game with finite number of actions, there exists at least one Nash equilibrium.

A **Nash equilibrium** is (π_A^*, π_B^*) such that no player has an incentive to change his/her strategy:

$V_A(\pi_A^*, \pi_B^*) \geq V_A(\pi_A, \pi_B^*)$ for all π_A

$V_B(\pi_A^*, \pi_B^*) \geq V_B(\pi_A^*, \pi_B)$ for all π_B

Factor graph

Variables:

$X = (X_1, \dots, X_n)$, where $X_i \in \text{Domain}_i$

Factors:

$f_1, \dots, f_m$, with each $f_j(X) \geq 0$

Scope of a factor f_j: set of variables it depends on.

Arity of f_j is the number of variables in the scope.

Unary factors (arity 1); **Binary** factors (arity 2).

Each **assignment** $x = (x_1, \dots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

Objective $\quad \arg\max_x \text{Weight}(x)$

A CSP is a factor graph where all factors are **constraints**:

$$f_j(x) \in \{0, 1\} \text{ for all } j = 1, \dots, m$$

The constraint is satisfied iff $f_j(x) = 1$.

An assignment x is **consistent** iff $\text{Weight}(x) = 1$ (i.e., **all** constraints are satisfied).

$\text{Backtrack}(x, w, \text{Domains})$:

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i
- Order **VALUES** Domain_i of chosen X_i
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - $\text{Domains}' \leftarrow \text{Domains}$ via **LOOKAHEAD**
  - $\text{Backtrack}(x \cup \{X_i : v\}, w\delta, \text{Domains}')$

Let $D(x, X_i)$ be set of factors depending on X_i and x but not on unassigned variables.

Forward Checking(Look Ahead)

- After assigning a variable X_i, eliminate inconsistent values from the domains of X_i's neighbors.
- If any domain becomes empty, don't recurse.
- When unassign X_i, restore neighbors' domains.

Least constrained value: useful when **all** factors are constraints (all assignment weights are 1 or 0)

Most constrained variable: useful when **some** factors are constraints (can prune assignments with weight 0)

Forward checking: need to actually prune domains to make heuristics useful!

Arc Consistency: eliminate values from domains-> reduce branching

A variable X_i is **arc consistent** with respect to X_j if for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that $f(\{X_i : x_i, X_j : x_j\}) \neq 0$ for all factors f whose scope contains X_i and X_j.

$\text{EnforceArcConsistency}(X_i, X_j)$: Remove values from Domain_i to make X_i arc consistent with respect to X_j.

AC 3 Algorithm

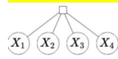
Add X_j to set.

While set is non-empty:

- Remove any X_k from set.
- For all neighbors X_l of X_k:
  - Enforce arc consistency on X_l w.r.t. X_k.
  - If Domain_l changed, add X_l to set.

Runtime O(ED^3). D -> largest domain E-> number of edges

N ARY constraints



Variables: $X_1, X_2, X_3, X_4 \in \{0, 1\}$

Factor: $[X_1 \vee X_2 \vee X_3 \vee X_4]$

Auxiliary Variables hold intermediate computation.

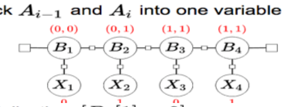
Initialization: $[A_0 = 0]$

Processing: $[A_i = A_{i-1} \vee X_i]$

Final output: $[A_4 = 1]$

i	0	1	2	3	4
X_i:		0	1	0	1
A_i:	0	0	1	1	1

pack A_{i-1} and A_i into one variable B_i



Initialization: $[B_1[1] = 0]$

Processing: $[B_i[2] = B_i[1] \vee X_i]$

Final output: $[B_4[2] = 1]$

Consistency: $[B_{i-1}[2] = B_i[1]]$

Variables: B_i is (pre, post) pair from processing X_i

Pruning Techniques only useful for constraints, does not help reduce actual runtime.

Finding Maximum Weight Assignments: Greedy Algorithm

Partial assignment $x \leftarrow \{\}$

For each $i = 1, \dots, n$:

  Extend:

    Compute weight of each $x_v = x \cup \{X_i : v\}$

  Prune:

    $x \leftarrow x_v$ with highest weight

Beam Search Algorithm:

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \dots, n$:

  Extend:

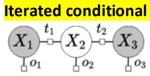
    $C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$

  Prune:

    $C \leftarrow K$ elements of C' with highest weights

Running time: $O(n(Kb)\log(Kb))$ with branching factor $b = |\text{Domain}|$, beam size K

Iterated conditional modes



$o_1(x_1)t_1(x_1, v)o_2(v)t_2(v, x_3)o_3(x_3)$ x1,v,x3

Initialize x to a random complete assignment

Loop through $i = 1, \dots, n$ until convergence:

  Compute weight of $x_v = x \cup \{X_i : v\}$ for each v

  $x \leftarrow x_v$ with highest weight

Gibbs Sampling: Sample an assign. Prop. to its weight

Note: The Gibbs update depends on the Markov blanket on a variable

Initialize x to a random complete assignment

Loop through $i = 1, \dots, n$ until convergence:

  Compute weight of $x_v = x \cup \{X_i : v\}$ for each v

  Choose $x \leftarrow x_v$ with probability prop. to its weight

Algorithms for max-weight assignments in factor graphs:

Backtracking search: exact, exponential time

Beam search: approximate, linear time

Iterated conditional modes: approximate, deterministic

Gibbs sampling: approximate, randomized

Conditioning Algorithm

To **condition** on a variable $X_i = v$, consider all factors $f_1, \dots, f_k$ that depend on X_i.

Remove X_i and $f_1, \dots, f_k$.

Add $g_j(x) = f_j(x \cup \{X_i : v\})$ for $j = 1, \dots, k$.

Conditionally Independent: conditioning on C, makes A and B independent

Let $A \subseteq X$ be a subset of variables.

Define $\text{MarkovBlanket}(A)$ be the neighbors of A that are not in A.

Elimination Algorithm

To **eliminate** a variable X_i, consider all factors $f_1, \dots, f_k$ that depend on X_i.

Remove X_i and $f_1, \dots, f_k$.

Add $f_{\text{new}}(x) = \max_{x_i} \prod_{j=1}^{k} f_j(x)$

Running time: $O(n \cdot |\text{Domain}|^{\text{max-arity}+1})$

The **treewidth** of a factor graph is the maximum arity of any factor created by variable elimination with the best variable ordering.

Bayesian Ntwks: Joint ,Marginal(sum) ,Conditional(select rows) Distribution: three types for random variables

Explaining away:Suppose two causes positively influence an effect. Conditioned on the effect, conditioning on one cause reduces the probability of the other cause

Let $X = (X_1, \dots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) \stackrel{\text{def}}{=} \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

All factors (local conditional distributions) satisfy:

$$\sum_{x_i} p(x_i \mid x_{\text{Parents}(i)}) = 1 \text{ for each } x_{\text{Parents}(i)}$$

Probabilistic Inference Strategy: $\mathbb{P}(Q \mid E = e)$

- Remove (marginalize) variables that are not ancestors of Q or E.
- Convert Bayesian network to factor graph.
- Condition on $E = e$ (shade nodes + disconnect).
- Remove (marginalize) nodes disconnected from Q.
- Run probabilistic inference algorithm (manual, variable elimination, Gibbs sampling, particle filtering).

HMM:

$$\mathbb{P}(H = h, E = e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$

Smoothing Query: Lattice Representation:

Forward: $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1}) w(h_{i-1}, h_i)$

Backward: $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1}) w(h_i, h_{i+1})$

Define $S_i(h_i) = F_i(h_i) B_i(h_i)$

Gibbs Sampling

Loop through $i = 1, \dots, n$ until convergence:

  Set $X_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$

  (notation: $X_{-i} = X \backslash \{X_i\}$)

Particle Filtering

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \dots, n$:

  Propose (extend):

    $C' \leftarrow \{x \cup \{X_i : x_i\} : x \in C, x_i \sim p(x_i \mid x_{i-1})\}$

  Reweight:

    Compute weights $w(x) = p(e_i \mid x_i)$ for $x \in C'$

  Resample (prune):

    $C \leftarrow K$ elements drawn independently from $\propto w(x)$

Max Likelihood Learning Algorithm

Input: training examples $\mathcal{D}_{\text{train}}$ of full assignments

Output: parameters $\theta = \{p_d : d \in D\}$

Count:

  For each $x \in \mathcal{D}_{\text{train}}$:

    For each variable x_i:

      Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

Normalize:

  For each d and local assignment $x_{\text{Parents}(i)}$:

    Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$

g	r	$p_R(r \mid g)$
d	1	0
d	2	0
d	3	0
d	4	2 / 3
d	5	1 / 3
c	1	1 / 2
c	2	0

data = {(d, 4), (d, 4), (d, 5), (c, 1), (c, 5)}

Regularization:Laplace Smoothing(add 6 for Dice)

just add to the count for each possible value, regardless of whether it was observed or not.Eg: X = 6 for dice

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

Expectation maximization (EM)algorithm

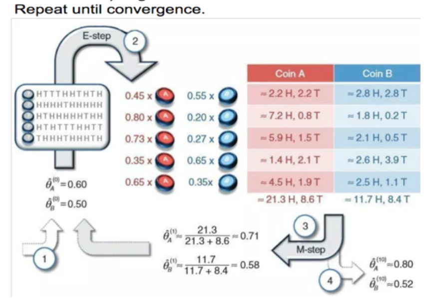
E-step:

- Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (use any probabilistic inference algorithm)
- Create weighted points: (h, e) with weight $q(h)$

M-step:

- Compute maximum likelihood (just count and normalize) to get θ

Repeat until convergence.



- Estimate Likely No of Heads and Tails for First Toss
- For A -: H=.45*5=2.2 , T=.45*5=2.2
- For B -: H==.55*5=2.8 , T = .55*5=2.8

Max Likelihood of p^n(1-p)^m = n/(n+m)

E step: for each missing assume all vals,create virtual bags and find probs. Mstep: find Max likelihood for all params.