Lecture 6: Inference Pat 1

谢丹
清华大学数学系

November 9, 2025

Clustering Methods: Overview and Preview I

Previously Covered Methods

We have explored two fundamental clustering approaches:

K-Means Clustering

- Starts with K initial centroids
- **Iteratively updates**:
  - Cluster assignments
  - Centroid positions
- Based on **minimum distance** criterion

Gaussian Mixture Models (GMM)

- Probabilistic model with parameters:
  - Mixing coefficients: π_k
  - Means: μ_k
  - Covariances: Σ_k
- **Iteratively updates** parameters via EM algorithm
- Based on **probability density** estimation

Clustering Methods: Overview and Preview II

Common Assumption

Both methods assume clusters are centered around specific points (centroid-based).

Next: Alternative Clustering Paradigms

We will now explore two fundamentally different clustering approaches that overcome limitations of centroid-based methods.

Hierarchical Clustering: Overview

Core Idea

Build a tree-like hierarchy of clusters where clusters at higher levels contain clusters from lower levels.

Two Main Approaches

1. **Agglomerative** (Bottom-up)
   - Start with each point as a cluster
   - Iteratively merge closest clusters
2. **Divisive** (Top-down, less common)
   - Start with one cluster
   - Recursively split clusters

Agglomerative Hierarchical Clustering Algorithm

Step-by-Step Process

1. Start with n clusters (each point is its own cluster)
2. Compute $n \times n$ proximity matrix
3. Repeat until one cluster remains:
   - Find two closest clusters C_i and C_j
   - Merge C_i and C_j
   - Update proximity matrix

Proximity Matrix Update

After merging clusters C_i and C_j into C_{ij}, update distances to other clusters C_k using linkage method.

Linkage Methods

Single Linkage

Minimum distance between clusters:

$$d(C_i, C_j) = \min_{a \in C_i, b \in C_j} d(a, b)$$

Average Linkage

Average distance between clusters:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{a \in C_i} \sum_{b \in C_j} d(a, b)$$

Complete Linkage

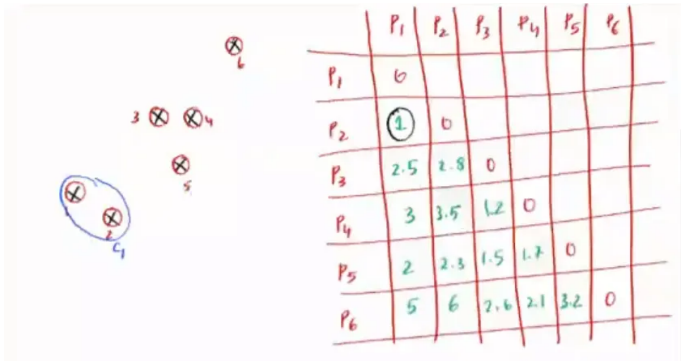
Maximum distance between clusters:

$$d(C_i, C_j) = \max_{a \in C_i, b \in C_j} d(a, b)$$

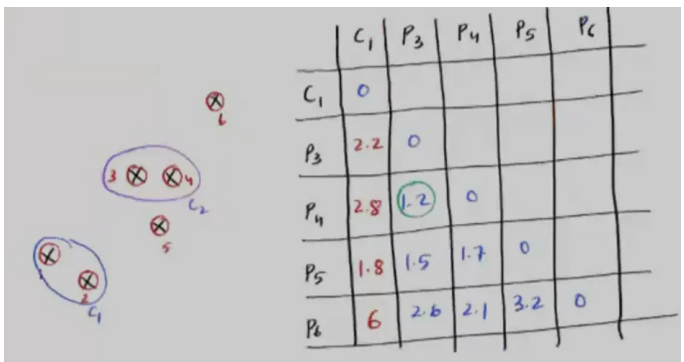
Ward's Method

Minimize increase in within-cluster variance:

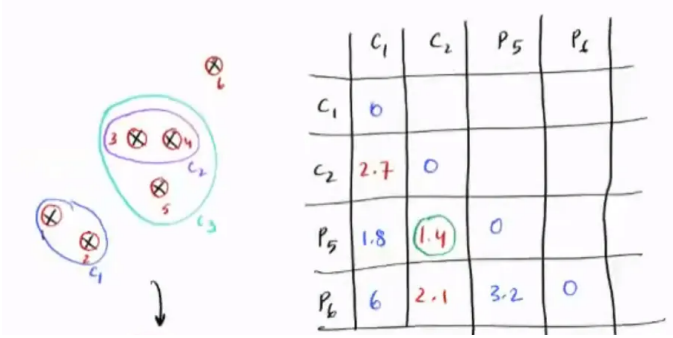
$$d(C_i, C_j) = \frac{|C_i||C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2$$



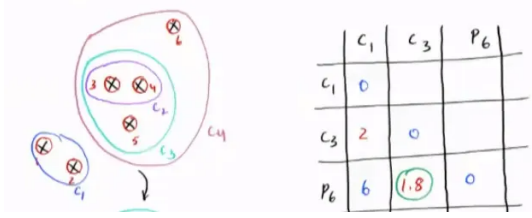
	P_1	P_2	P_3	P_4	P_5	P_6
P_1	0					
P_2	1	0				
P_3	2.5	2.8	0			
P_4	3	3.5	1.2	0		
P_5	2	2.3	1.5	1.7	0	
P_6	5	6	2.6	2.1	3.2	0



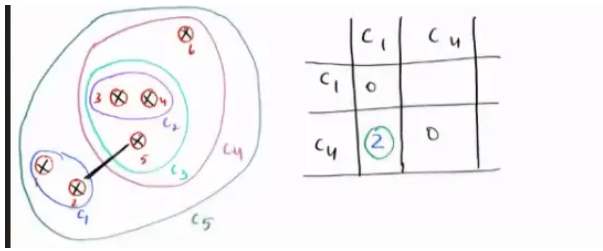
	C_1	P_3	P_4	P_5	P_6
C_1	0				
P_3	2.2	0			
P_4	2.8	1.2	0		
P_5	1.8	1.5	1.7	0	
P_6	6	2.6	2.1	3.2	0

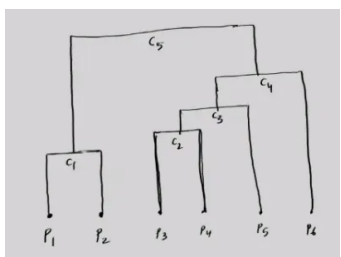


	c_1	c_2	p_5	p_4	
c_1	6				
c_2	2.7	0			
p_5	1.8	1.4	0		
p_6	6	2.1	3.2	0	



	c_1	c_3	p_6
c_1	0		
c_3	2	0	
p_6	6	1.8	0


1
2
3
4
5
C1
C2
C3
C4
C5
C1
C4
C1
0
C4
2
0


C5
C4
C3
C2
C1
P1
P2
P3
P4
P5
P6

The optimal number of clusters can be found using a dendrogram (the last diagram in last slide): Find the longest vertical line and cut it.

DBSCAN: Core Concepts

Philosophy

Cluster based on density connectivity rather than distance from centroids. $MinPts$ is the number of minimal points.

Key Definitions

- **ε -neighborhood**: $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$
- **Core point**: $|N_\epsilon(p)| \geq MinPts$
- **Border point**: In ε of core point but $|N_\epsilon(p)| < MinPts$
- **Noise point**: Neither core nor border

DBSCAN: Reachability Concepts

Directly Density-Reachable

Point q is directly density-reachable from p if:

- p is a core point
- $q \in N_\epsilon(p)$

Density-Reachable

Point q is density-reachable from p if there exists a chain $p_1, \ldots, p_n$ with $p_1 = p$, $p_n = q$ where each p_{i+1} is directly density-reachable from p_i.

Density-Connected

Points p and q are density-connected if there exists point o such that both p and q are density-reachable from o.

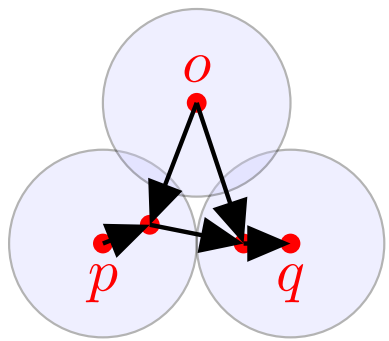


Figure: Density connectivity

Simplified DBSCAN Algorithm I

Step 1 — Identify Point Types

Identify all points as either:

- **Core point**: Has at least `MinPts` within ε distance
- **Border point**: Within ε of a core point but has $<$ `MinPts` neighbors
- **Noise point**: Neither core nor border point

Step 2 — Process Core Points

For each unclustered core point:

2a Create a new cluster

2b Expand cluster by adding all points that are:

- Unclustered
- Density-connected to the current core point

Step 3 — Assign Border Points

Simplified DBSCAN Algorithm II

For each unclustered border point, assign it to the cluster of the nearest core point.

Step 4 — Handle Noise Points

Leave all noise points unclustered (marked as outliers).

Final Result

- Dense regions form clusters of arbitrary shapes
- Sparse regions are identified as noise
- No need to pre-specify number of clusters

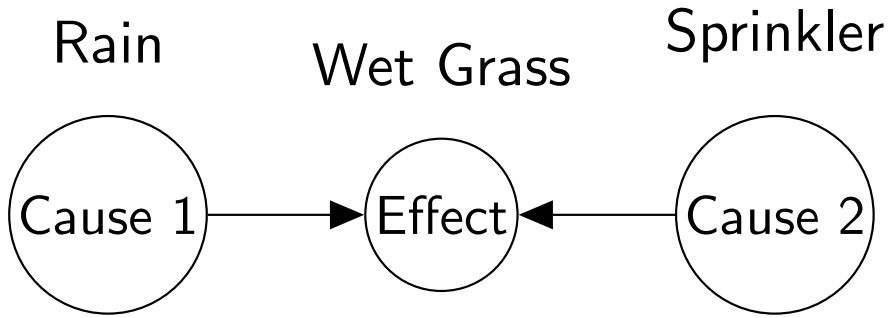
DBSCAN vs Hierarchical Clustering: Comparison

Hierarchical Clustering	DBSCAN
Cluster Shape Tends to find spherical clusters Shape determined by linkage	**Cluster Shape** Finds arbitrarily shaped clusters Shape follows density contours
Noise Handling Poor - sensitive to outliers Outliers can distort entire structure	**Noise Handling** Excellent - explicit noise category Noise points don't affect clusters
Parameters Linkage method, distance metric Number of clusters (for cutting)	**Parameters** ε (eps), MinPts No need for cluster count
Complexity $O(n^3)$ naive, $O(n^2 \log n)$ optimized $O(n^2)$ space	**Complexity** $O(n^2)$ naive, $O(n \log n)$ with indexing $O(n)$ space
Output Dendrogram (hierarchical structure)	**Output** Flat partition + noise points

CHAPTER 7:
Inference

Explain Away Phenomenon: Three-Node Example

Let's first explain an interesting fact about the inference.



Bayesian Network Structure

- Two independent causes that can produce the same effect
- This is a **v-structure** or **collider**:
  $Rain \rightarrow WetGrass \leftarrow Sprinkler$
- The causes are independent *a priori* but become dependent when we observe the effect

Probability Model Specification

Prior Probabilities

- $P(Rain = T) = 0.2$
- $P(Sprinkler = T) = 0.1$
- $Rain \perp Sprinkler$

Conditional Probability Table

$P(WetGrass|Rain, Sprinkler)$:

Rain	Sprinkler	WetGrass	Probability
F	F	T	0.0
F	F	F	1.0
F	T	T	0.9
F	T	F	0.1
T	F	T	0.8
T	F	F	0.2
T	T	T	0.98
T	T	F	0.02

Key Insight

- Initially: $Rain \perp Sprinkler$
- After observing $WetGrass = T$: $Rain \not\perp Sprinkler|WetGrass$
- The causes become **negatively correlated**

The "Explain Away" Effect Step by Step

Step 1: Initial Beliefs (No Evidence)

- $P(Rain = T) = 0.2$
- $P(Sprinkler = T) = 0.1$
- $P(Rain = T \cap Sprinkler = T) = 0.2 \times 0.1 = 0.02$

Step 2: Observe Wet Grass

We observe: $WetGrass = True$

What happens to our beliefs about the causes?

Step 3: Updated Beliefs

Using Bayes' theorem:

$$P(R|W) = \frac{P(W|R)P(R)}{P(W)}$$

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)}$$

Calculating the Evidence

Probability of Wet Grass

$$
\begin{aligned}
P(W = T) &= \sum_{r,s} P(W = T|r, s)P(r)P(s) \\
&= P(W = T|R = F, S = F)P(R = F)P(S = F) \\
&+ P(W = T|R = F, S = T)P(R = F)P(S = T) \\
&+ P(W = T|R = T, S = F)P(R = T)P(S = F) \\
&+ P(W = T|R = T, S = T)P(R = T)P(S = T) \\
&= 0.0 \times 0.8 \times 0.9 + 0.9 \times 0.8 \times 0.1 \\
&+ 0.8 \times 0.2 \times 0.9 + 0.98 \times 0.2 \times 0.1 \\
&= 0 + 0.072 + 0.144 + 0.0196 = 0.2356
\end{aligned}
$$

Posterior Probabilities

Individual Posteriors

$$P(R|W) = \frac{P(W|R)P(R)}{P(W)}$$
$$= \frac{0.8 \times 0.2}{0.2356} \approx 0.679$$
$$P(S|W) = \frac{P(W|S)P(S)}{P(W)}$$
$$= \frac{0.9 \times 0.1}{0.2356} \approx 0.382$$

Key Observation

- Both probabilities increased:
  $P(R)$: 0.2 $\rightarrow$ 0.679
  $P(S)$: 0.1 $\rightarrow$ 0.382
- But wait... there's more!

The "Explain Away" Revealed

Joint Posterior Probability

What is $P(R = T, S = T|W = T)$?

$$P(R = T, S = T|W = T) = \frac{P(W = T|R = T, S = T)P(R = T)P(S = T)}{P(W = T)}$$
$$= \frac{0.98 \times 0.2 \times 0.1}{0.2356}$$
$$= \frac{0.0196}{0.2356} \approx 0.083$$

The Explain Away Effect!

- If independent:
  $P(R, S|W) = P(R|W)P(S|W) \approx 0.679 \times 0.382 \approx 0.259$
- Actual: $P(R, S|W) \approx 0.083$
- **They are much less likely to co-occur than expected!**

Mathematical Generalization

General Common Effect Structure

For any v-structure: $A \rightarrow C \leftarrow B$

- Initially: $A \perp B$
- After observing C: $A \not\perp B|C$
- The correlation is generally negative

Bayes' Theorem Explanation

$$P(A, B|C) = \frac{P(C|A, B)P(A)P(B)}{P(C)}$$
$$\propto P(C|A, B)P(A)P(B)$$

The conditional probability $P(C|A, B)$ creates the dependence.

Key Insight for Inference

- This is why inference algorithms must carefully handle v-structures

Summary

Key Points

- **Explain away** occurs in common effect structures (v-structures)
- Independent causes become dependent when their common effect is observed
- The dependence is typically **negative correlation**
- One cause can "explain away" the need for the other cause

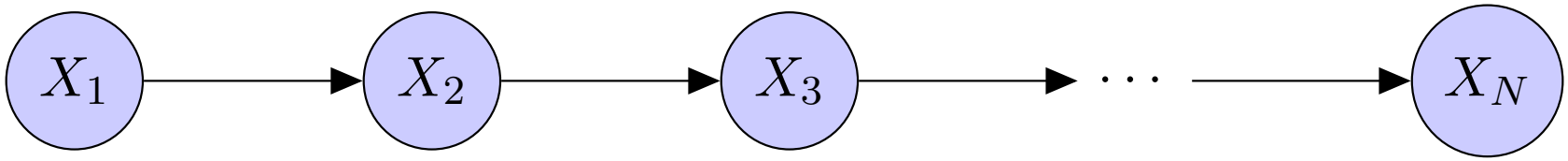
Importance in Probabilistic Inference

- Crucial for correct belief updating in Bayesian networks
- Explains counter-intuitive reasoning patterns
- Fundamental to understanding conditional independence
- Essential for efficient inference algorithms

Section 1: Exact inference

The Chain Graph Structure

Bayesian Network



Joint Probability Factorization

$$P(\mathbf{X}) = P(X_1) \prod_{i=2}^{N} P(X_i|X_{i-1})$$

Inference Goal

Compute marginals:

$$P(X_i) = \sum_{\mathbf{X} \setminus X_i} P(\mathbf{X})$$

Key Insight: Exploiting Factorization I

Factorization Enables Efficient Computation

We observe that the joint probability distribution admits a factorized form, allowing us to exchange the order of summation when computing marginals. This crucial insight leads to a dramatic reduction in computational complexity.

Example

Sequential Marginalization Strategy Instead of summing over all variables simultaneously, we can proceed sequentially:

- First marginalize the last node, effectively removing it from consideration
- Then proceed to the next node, leveraging the simplified structure
- Continue this process until reaching the target variable

General Framework: Factor Graphs

Key Insight: Exploiting Factorization II

For arbitrary graphical models, we formalize this approach using factor graphs:

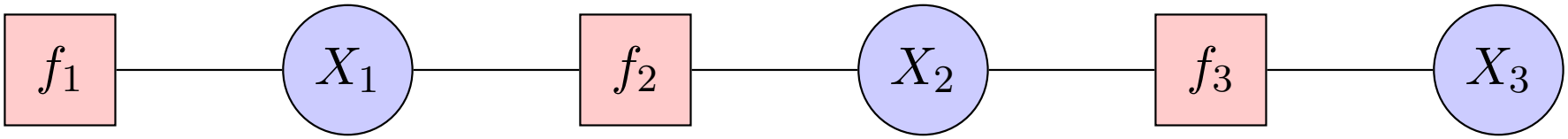
- Explicitly represent the factorization structure
- Enable systematic message-passing protocols
- Provide a unified framework for efficient inference

Complexity Reduction

This strategy transforms the computation from exponential to polynomial complexity, making exact inference feasible for large-scale models!

Factor Graph Representation I

Factor Graph consists of variables and the factors in the joint probability



Factors

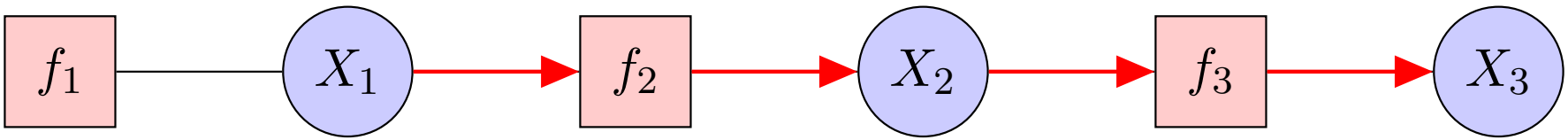
$$f_1(X_1) = P(X_1)$$
$$f_2(X_1, X_2) = P(X_2|X_1)$$
$$f_3(X_2, X_3) = P(X_3|X_2)$$
$$\vdots$$
$$f_N(X_{N-1}, X_N) = P(X_N|X_{N-1})$$

Factor Graph Representation II

Key Insight

Factorization enables efficient local computations via message passing

Message Passing: Forward Pass (α) I



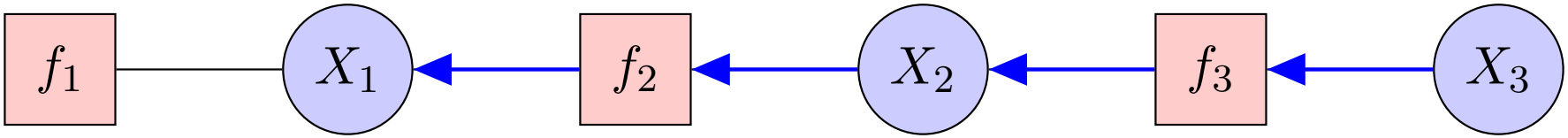
Forward Messages

$$\alpha_2(X_2) = \sum_{x_1} f_1(x_1) f_2(x_1, X_2)$$

$$\alpha_3(X_3) = \sum_{x_2} \alpha_2(x_2) f_3(x_2, X_3)$$

$$\alpha_i(X_i) = \sum_{x_{i-1}} \alpha_{i-1}(x_{i-1}) f_i(x_{i-1}, X_i)$$

Message Passing: Backward Pass (β)



Backward Messages

$$\beta_{N-1}(X_{N-1}) = \sum_{x_N} f_N(X_{N-1}, x_N)$$

$$\beta_{N-2}(X_{N-2}) = \sum_{x_{N-1}} f_{N-1}(X_{N-2}, x_{N-1})\beta_{N-1}(x_{N-1})$$

$$\beta_i(X_i) = \sum_{x_{i+1}} f_{i+1}(X_i, x_{i+1})\beta_{i+1}(x_{i+1})$$

Marginal Computation I

Combining Messages

For any variable X_i, combine forward and backward information:

$$P(X_i) \propto \alpha_i(X_i) \cdot \beta_i(X_i)$$

Normalization

$$P(X_i) = \frac{\alpha_i(X_i)\beta_i(X_i)}{\sum_{x_i} \alpha_i(x_i)\beta_i(x_i)}$$

Example

Marginal Computation II

3-Node Chain

$$P(X_2) \propto \alpha_2(X_2) \cdot \beta_2(X_2)$$

$$\alpha_2(X_2) = \sum_{x_1} P(x_1)P(X_2|x_1)$$

$$\beta_2(X_2) = \sum_{x_3} P(x_3|X_2)$$

Complete Two-Pass Algorithm

Step 1: Forward Pass

1. Initialize: $\alpha_1(X_1) = P(X_1)$
2. For $i = 2$ to N: $\alpha_i(X_i) = \sum_{x_{i-1}} \alpha_{i-1}(x_{i-1}) P(X_i | x_{i-1})$

Step 2: Backward Pass

1. Initialize: $\beta_N(X_N) = 1$
2. For $i = N - 1$ to 1: $\beta_i(X_i) = \sum_{x_{i+1}} P(x_{i+1} | X_i) \beta_{i+1}(x_{i+1})$

Step 3: Compute Marginals

For $i = 1$ to N:

$$P(X_i) = \frac{\alpha_i(X_i)\beta_i(X_i)}{\sum_{x_i} \alpha_i(x_i)\beta_i(x_i)}$$

Computational Complexity

Complexity Analysis

- **Naive approach**: $O(K^N)$
- **Sum-Product**: $O(NK^2)$
- **Speedup**: Exponential to polynomial!

Key Operations per Step

- Message computation: $O(K^2)$
- $N-1$ messages in each direction
- Total: $O(NK^2)$ operations

Memory Requirements

- Store all α_i: $O(NK)$
- Store all β_i: $O(NK)$
- Total: $O(NK)$ storage

Optimal for Chains

Sum-Product is provably optimal for exact inference in chain graphs

The General Inference Problem

- Joint probability distribution over variables $\mathbf{X} = \{X_1, \ldots, X_N\}$:

$$P(\mathbf{X}) = \frac{1}{Z} \prod_c \psi_c(\mathbf{X}_c)$$

- ψ_c: Factor/potential function (both undirected and directed graph model)
- Z: Normalization constant

Goal

Compute marginal distribution for variable X_i:

$$P(X_i) = \sum_{\mathbf{X} \setminus X_i} P(\mathbf{X})$$

Challenge

Naive summation has exponential complexity! Sum-Product algorithm provides efficient solution for **tree-structured graphs**.

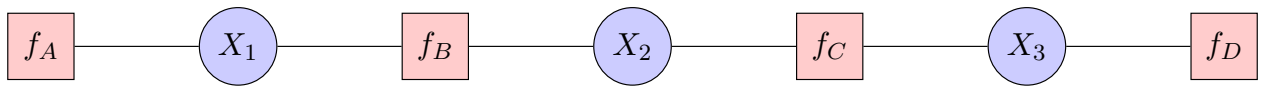
Factor Graph Representation

Two Types of Nodes

- **Variable Nodes** (circles)
- **Factor Nodes** (squares)

Example

$$P = f_A(X_1) f_B(X_1, X_2) f_C(X_2, X_3) f_D(X_3)$$



Why Factor Graphs?

- Clear representation of factorization
- Explicit message passing rules
- Generalizes Bayesian Networks and MRFs

Message Passing Rules I

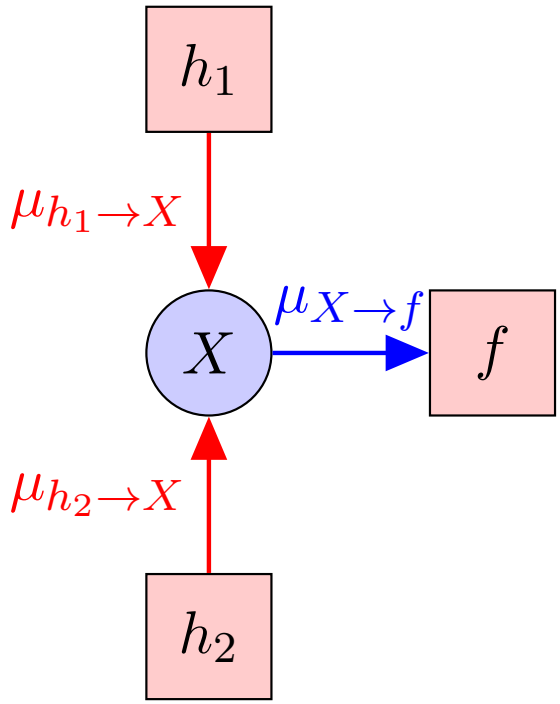
Two Message Types

1. $\mu_{X \to f}(X)$: Variable $\to$ Factor
2. $\mu_{f \to X}(X)$: Factor $\to$ Variable

Leaf Initialization

- Variable leaf: $\mu_{X \to f}(X) = 1$
- Factor leaf: $\mu_{f \to X}(X) = f(X)$

Message Rules



Message Passing Rules II

- **Variable $\rightarrow$ Factor:** $\mu_{X \to f}(X) = \prod_{h \in \mathsf{ne}(X) \setminus \{f\}} \mu_{h \to X}(X)$
- **Factor $\rightarrow$ Variable:**

$$\mu_{f \to X}(X) = \sum_{\mathbf{Y}} \left(f(\mathbf{Y}, X) \cdot \prod_{Y \in \mathsf{ne}(f) \setminus \{X\}} \mu_{Y \to f}(Y) \right)$$

General Algorithm Steps

Step 1: Choose Root

Pick an arbitrary node as the root of the tree.

Step 2: Leafward Pass (Collection)

- Start from leaves, pass messages inward toward root
- Node sends message after receiving from all children

Step 3: Rootward Pass (Distribution)

- Root sends messages back toward leaves
- Complete message propagation in both directions

Step 4: Compute Marginals

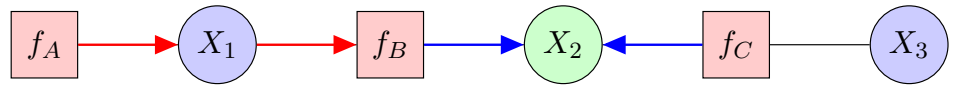
For each variable X_i:

$$P(X_i) = \frac{1}{Z} \prod_{f \in \mathsf{ne}(X_i)} \mu_{f \to X_i}(X_i)$$

Worked Example

Factorization

$$P(X_1, X_2, X_3) = f_A(X_1) f_B(X_1, X_2) f_C(X_2, X_3)$$



Compute $P(X_2)$

$$\mu_{X_1 \to f_B} = f_A(X_1)$$

$$\mu_{X_3 \to f_C} = 1$$

$$\mu_{f_C \to X_2} = \sum_{x_3} f_C(X_2, x_3)$$

$$\mu_{f_B \to X_2} = \sum_{x_1} f_B(x_1, X_2) f_A(x_1)$$

Final Marginal

$$P(X_2) \propto \mu_{f_B \to X_2} \cdot \mu_{f_C \to X_2}$$

$$P(X_2) \propto \left[\sum_{x_1} f_A f_B \right] \cdot \left[\sum_{x_3} f_C \right]$$

Key Properties and Applications

Computational Efficiency

- **Naive:** $O(K^N)$
- **Sum-Product:** Linear in nodes
- Cost per message: $O(K^m)$ for factor with m variables

Theoretical Guarantees

- **Exact** for tree-structured graphs
- Computes **all marginals** efficiently
- Foundation for many algorithms

Applications and Extensions

- **Forward-Backward Algorithm** (HMMs)
- **Kalman Filter**
- **Loopy Belief Propagation** for graphs with cycles
- **Expectation Propagation**

Limitation

Exact inference only for **trees**. For graphs with cycles, becomes approximate but often works well in practice.

Summary

- **General framework** for inference on tree-structured graphs
- Uses **factor graph** representation for clarity
- Based on **message passing** between variable and factor nodes
- Provides **efficient exact marginals** via two-pass algorithm
- **Foundation** for many important algorithms in machine learning
- Extends to **approximate inference** for loopy graphs

The Big Picture

The Sum-Product algorithm transforms the exponentially hard problem of marginalization into a linear-time procedure by exploiting graph structure and local computations, demonstrating the power of graphical models for efficient probabilistic reasoning.

The Problem: MAP Inference

Sum-Product vs Max-Product

- **Sum-Product**: Computes marginal distributions $P(X_i)$
- **Max-Product**: Finds most probable configuration

MAP Inference Problem

Find the assignment that maximizes the joint probability:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} P(\mathbf{x}) = \arg\max_{\mathbf{x}} \prod_c \psi_c(\mathbf{x}_c)$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a complete assignment.

Key Idea: Sum $\rightarrow$ Max

The algorithm is identical to Sum-Product except for one crucial change:

Sum-Product Message (Factor $\rightarrow$ Variable)

$$\mu_{f \to X}(X) = \sum_{\mathbf{Y}} \left(f(\mathbf{Y}, X) \cdot \prod_{Y \in \mathsf{ne}(f) \setminus \{X\}} \mu_{Y \to f}(Y) \right)$$

Max-Product Message (Factor $\rightarrow$ Variable)

$$\mu_{f \to X}(X) = \max_{\mathbf{Y}} \left(f(\mathbf{Y}, X) \cdot \prod_{Y \in \mathsf{ne}(f) \setminus \{X\}} \mu_{Y \to f}(Y) \right)$$

$$\sum \rightarrow \max$$

Complete Algorithm Steps

Step 1: Message Passing

- Identical to Sum-Product structure
- Choose root node
- Perform upward pass (leaves to root)
- Perform downward pass (root to leaves)
- **But use MAX instead of SUM**

Step 2: Compute Max-Marginals at Root

For root variable X_r:

$$P^*(X_r) = \prod_{f \in \mathsf{ne}(X_r)} \mu_{f \to X_r}(X_r)$$

- $P^*(X_r)$ is the **max-marginal**
- Proportional to maximum probability achievable for each value of X_r

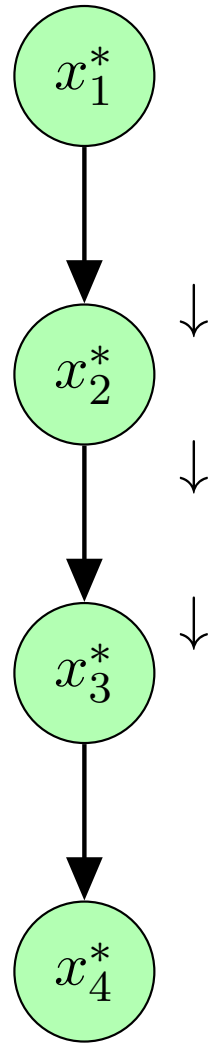
Step 3: Backtrack for MAP Assignment I

Backtracking Process

1. **Start at root**:

$$x_r^* = \arg\max_{x_r} P^*(x_r)$$

2. **Recurse downward**: For each child, choose value consistent with parent's choice that maximizes incoming messages.
3. **Continue** until all variables assigned.



Example

Step 3: Backtrack for MAP Assignment II

For chain $X_1 \to X_2 \to X_3 \to X_4$:

- Choose $x_4^* = \arg\max \mu_{3\to 4}(x_4)$
- Choose $x_3^* = \arg\max_{x_3} [P(x_4^*|x_3) \cdot \mu_{2\to 3}(x_3)]$
- Continue to x_2^*, x_1^*

Max-Sum Algorithm: Log-Domain Version I

Why Use Log-Domain?

- Avoids numerical underflow
- Converts multiplication to addition
- More numerically stable

Transformations

Original	**Log-Domain**
Multiplication	Addition
Maximization	Maximization
$\prod$	$\sum$
$\max$	$\max$

Max-Sum Messages

Max-Sum Algorithm: Log-Domain Version II

- **Variable to Factor**:
  $$\log \mu_{X \to f}(X) = \sum_{h \in \mathsf{ne}(X) \setminus \{f\}} \log \mu_{h \to X}(X)$$
- **Factor to Variable**: $\log \mu_{f \to X}(X) =$
  $$\max_{\mathbf{Y}} \left[\log f(\mathbf{Y}, X) + \sum_{Y \in \mathsf{ne}(f) \setminus \{X\}} \log \mu_{Y \to f}(Y) \right]$$

Comparison: Sum-Product vs Max-Product

Aspect	**Sum-Product**	**Max-Product**
Goal	Compute marginals $P(X_i)$	Find MAP assignment $\mathbf{x}^*$
Operation	Sum over other variables	Max over other variables
Output	Probability distributions	Single best configuration
Complexity	$O(K^m)$ per message	$O(K^m)$ per message
Backtrack	Not needed	Required for MAP assignment
Normalization	Required during/after	Optional (only for probabilities)
Applications	Bayesian inference, estimation	Decoding, segmentation, optimal control