9/18/2025

Probability

Axioms of probability

Axiom 1: Non-negativityP(A) ⩾ 0

For any event A ∈ SA \in \mathcal SA ∈ S:

P(A) ⩾ 0

Ƅ Probability can never be negative.

Axiom 2: Normalization

The probability of the entire sample space is **1**:

P(Ω)=1

Ƅ Something in the sample space must happen.

Axiom 3: Countable Additivity

For any countable sequence of **mutually disjoint** events A1,A2,A3,…A_1, A_2, A_3, ...

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i), \quad \text{if } A_i \cap A_j = \emptyset \text{ for } i \neq j.$$

Symbol	Meaning	Example
Ω (Omega)	The **sample space** — the set of *all possible outcomes*	Ω = {1, 2, 3, 4, 5, 6}
S	The **event space** — a set of subsets of Ω that we assign probabilities to	S = {∅, {1}, {2}, … , {6}, {1,3,5}, {2,4,6}, Ω}
α	A specific **event** (i.e., one element of S)	α = {1, 3, 5} → “the die shows an odd number”

Sample space:

Event Space: (S)

The event space S is the collection of all events to which you are willing to assign a probability.

If you want to assign a probability **only** to the event *"getting 3"*, you also need to be able to assign a probability to its **complement** (i.e. *"not getting 3"*).

The complement of {3} is {1,2,4,5,6}.

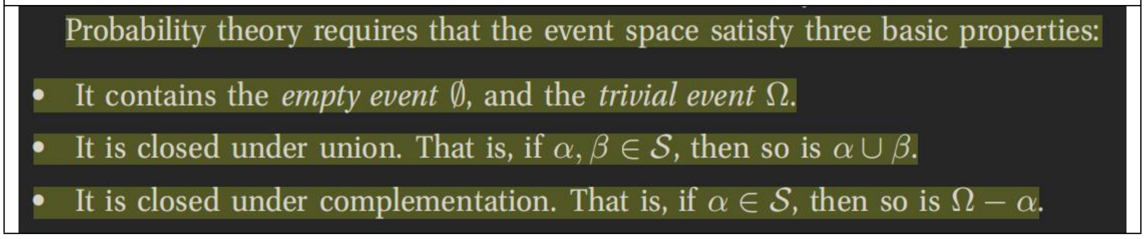
So the **smallest valid event space** in that case would be:

S = { ∅ , {3} , {1,2,4,5,6} , Ω }

TIP:

- We include both the event and its complement so that the event space is closed under complements (one of the required properties).
- every event in the event space must have its complement in the event space.

Konnor



Probability theory requires that the event space satisfy three basic properties:

- It contains the *empty event* $\emptyset$, and the *trivial event* Ω.
- It is closed under union. That is, if $\alpha, \beta \in \mathcal{S}$, then so is $\alpha \cup \beta$.
- It is closed under complementation. That is, if $\alpha \in \mathcal{S}$, then so is $\Omega - \alpha$.

General rule for building an event space (also called a σ –algebra)

An event space **S** over a sample space **Ω** must satisfy **all three** of the following properties:

Property	Meaning
1. $\emptyset \in S$	The **empty set** must be in the event space
2. Closed under **complements**	If $\alpha \in S$, then $\Omega \setminus \alpha$ must also be in S

Property	Meaning
3. Closed under **(countable) unions**	If $\alpha_1, \alpha_2, \alpha_3, \ldots \in S$ then $\alpha_1 \cup \alpha_2 \cup \alpha_3 \cup \ldots$ is also in S

In a **finite** sample space (like a die), "countable unions" simply means: If two events are in **S**, then their **union** is also in **S**.

So for a finite Ω you can simplify as:

$\varnothing \in S$

$\alpha \in S \rightarrow \Omega \setminus \alpha \in S$

$\alpha, \beta \in S \rightarrow (\alpha \cup \beta) \in S$

Probablity Distrubution (Probablity Density factor)

A probability distribution P over (Ω, S) is a mapping from events in S to real values that satisfies
probability Distribution

the following conditions:
- $P(\alpha) \geq 0$ for all $\alpha \in S$.
- $P(\Omega) = 1$.
- If $\alpha, \beta \in S$ and $\alpha \cap \beta = \varnothing$, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$

Probability Definition: there are two types of probablity definition

Aspect	Frequentist	Subjective (Bayesian)
Meaning	Long-run frequency	Degree of belief
Nature	Objective	Subjective
Experiment	Needs repeatable trials	Can be one-time events
Example	P(rolling 3 on a die) = 1/6	P(rain tomorrow) = 0.7

8/18/2025

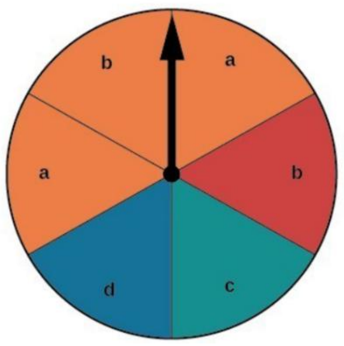
Addition rule of Probability

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If the two events are mutually exclusive

$P(A \cap B) = 0$ therefor $P(A \cup B) = P(A) + P(B)$

Why subtract $P(A \cap B)$?



There are a total of 6 sections, and 3 of them are orange. So the probability of spinning orange is 3/6=1/2. There are a total of 6 sections, and 2 of them have a b. So the probability of spinning a b is 2/6=1/3. If we added these two probabilities, we would be counting the sector that is both orange and b twice. To find the probability of spinning an orange or a b, we need to subtract the probability that the sector is both orange and has a b.

probability of disjoint union = sum of probabilities

(axiom 3 of probablity)

$$P(A) = \sum_{i=1}^{n} P(A \cap B_i)$$

$$P(A) = \sum_{i=1}^{n} P(A \mid B_i)\, P(B_i)$$

General Rule (Inclusion–Exclusion Principle)

When you have more than one event, you need to be careful not to double-count the overlaps.
For **three** events, the probability of their union is:

$P(A\cup B\cup C)=P(A)+P(B)+P(C) - P(A\cap B) - P(A\cap C) - P(B\cap C) + P(A\cap B\cap C)$

Solution:

P(A∪B∪C) = P (A u (B u C))

P (A u (B u C)) = P(A) + P(BuC) - P(A n (B u C))

But P (BuC) = P(B) + P(C) - P(B n C)

:- P (A u (B u C)) = P(A) + P(B) + P(C) - P(B n C) - P(A n (B u C))

But P(A n (B u C)) = P((A n B) u (A n C))

P((A n B) u (A n C)) = P(A n B) + P(A n C) - P((A n B) n (A n C))

But: P((A n B) n (A n C)) = P(A n B n C)

:- P (A u (B u C)) = P(A) + P(B) + P(C) - P(B n C) - (P(A n B) + P(A n C) - P((A n B) n (A n C)))

:- P (A u (B u C)) = P(A) + P(B) + P(C) - P(B n C) - P(A n B) - P(A n C) + P(A n B n C))

Conditional Probability:

1. $A \perp B \mid C$ = ?

(read: *"A is independent of B* **given** C"), it means that **once we know** C, information about B doesn't change the probability of A anymore.

Solution:

Normally, for any events (without independence):

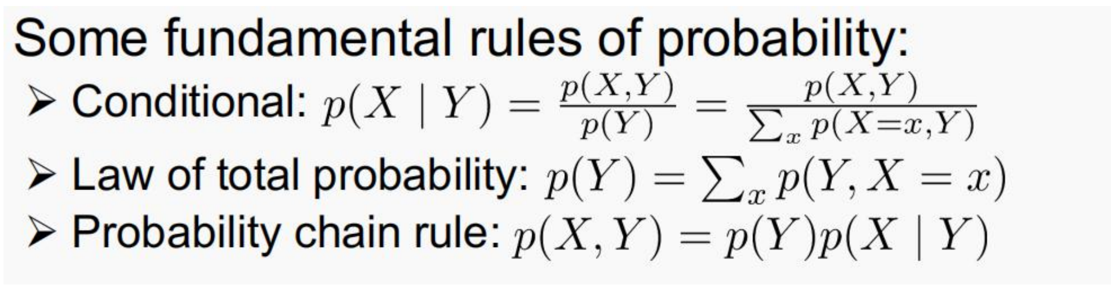
$P(A\cap B \mid C)=P(A \mid B,C)\, P(B \mid C)$ # $P(A \mid B,C)$ equivalent to $P(A \mid B \cap C)$

But **conditional independence** tells us that **A doesn't depend on B once C is known**, so: NB (if B does not depend on A the P(A | B) = P(A))

$P(A \mid B,C)=P(A \mid C)$ $P(A \mid B, C)$

Plugging this into the general rule gives:

$$P(A \cap B \mid C) = P(A \mid C)\, P(B \mid C)$$



Some fundamental rules of probability:

- Conditional: $p(X \mid Y) = \frac{p(X,Y)}{p(Y)} = \frac{p(X,Y)}{\sum_x p(X=x,Y)}$
- Law of total probability: $p(Y) = \sum_x p(Y, X = x)$
- Probability chain rule: $p(X,Y) = p(Y)p(X \mid Y)$

Question:

This is A more general conditional version of Bayes' rule, where all our probabilities are conditioned on
some background event γ, also holds

Proof the left side is true

$$P(\alpha \mid \beta \cap \gamma) = \frac{P(\beta \mid \alpha \cap \gamma)P(\alpha \mid \gamma)}{P(\beta \mid \gamma)}$$

$P(\alpha \mid \beta \cap \gamma) = P(\alpha \cap \beta \cap \gamma) = P(\beta \cap \alpha \cap \gamma)$

$$P(\alpha \mid \beta \cap \gamma) = \frac{P(\beta \cap \alpha \cap \gamma)}{P(\beta \cap \gamma)}$$

So lets take the $P(\beta \cap \alpha \cap \gamma)$

$P(\beta \cap \alpha \cap \gamma) = P(\beta \mid \alpha \cap \gamma)\, P(\alpha \cap \gamma)$
But $P(\alpha \cap \gamma) = P(\alpha \mid \gamma)\, P(\gamma)$
:- $P(\beta \cap \alpha \cap \gamma) = P(\beta \mid \alpha \cap \gamma)P(\alpha \mid \gamma)\, P(\gamma)$
Now lets take the $P(\beta \mid \gamma)$
$P(\beta \cap \gamma) = P(\beta \mid \gamma)P(\gamma)$

Now Plug both inside the original question

$$P(\alpha \mid \beta \cap \gamma) = \frac{P(\beta \mid \alpha \cap \gamma)P(\alpha \mid \gamma)\, P(\gamma)}{P(\beta \mid \gamma)P(\gamma)}$$

$$P(\alpha \mid \beta \cap \gamma) = \frac{P(\beta \mid \alpha \cap \gamma)P(\alpha \mid \gamma)}{P(\beta \mid \gamma)}$$

8/20/2025

Random Variables

$$\sum_{i=1}^{k} P(X = x_i) = 1$$

Random variables depend on **chance or outcomes of an experiment**, while non-random variables are **fixed, constant, or deterministic**.

Intuition

Suppose you want the probability that someone **likes coffee given they live**

in Ethiopia: (RV likeCoffe = ture or false)

P(like coffee | Ethiopia) = P(like coffe n Ethiopia) / P(Ethiopia) =>

P(Ethiopia) = $\sum_{i=1}^{k} P(likeCoffe = x_i, Ethiopia) = 1$

(Probability of Ethiopia for every possible values of liking coffee)
P(Ethiopia) = P(like coffe n Ethiopia) + P(doesn't like cofee n Ethiopia)

Two ways to classify random variables

1. By type of values they take:

1.1 Categorical (qualitative): Values are labels or categories

Example: Eye color = {Brown, Blue, Green}

1.2 Numerical / Real-valued (quantitative): Values are numbers

Example: Height = 1.65 m, 1.72 m

2. By how many values they can take:

2.1 **Discrete**: Can count the values (1, 2, 3, …)

Example: Number of students in a class

2.2 **Continuous**: Can take any value in an interval (infinite possibilities)

Example: Weight = 50.2 kg, 50.25 kg, 50.251 kg …

Multinomial distribution

$$\sum_{i=1}^{k} P(X = x_i) = 1$$

- Describes the number of times **each of several categories** occurs in *n* independent trials
- Example: Roll a 3-sided die 10 times → count how many times **1**, how many times **2**, and how many times **3** appear
- Random variable gives a **vector of counts**, one for each category

Think of it as **repeating a multi-category trial multiple times and counting how often each category happens**

Bernoulli distribution

Describes **a single trial** with **two possible outcomes**
Example: Toss one coin → Head (1) or Tail (0)
Random variable takes **only one value** (0 or 1)
Think of it as **one yes/no experiment**

Binomial distribution

Describes the **total number of successes in *n* independent Bernoulli trials**
Example: Toss a coin **10 times** → count how many Heads appear
Random variable takes values from **0 up to n**
Think of it as **repeating the Bernoulli experiment multiple times and counting how many "successes" you get**

Probability Density Function (PDF)

Because there are **infinitely many possible values**, we can't say "the probability that X = 25.2" — that would be **zero** (since it's one exact point).
So instead, we use the **Probability Density Function (PDF)**, written as **p(x)**.

Key things about the PDF:

It shows **how dense** the probabilities are around each value.
It's like saying: "where is the probability more concentrated?"
The **area under the curve** gives the probability, not the height of the curve.

✅ Example:
If you draw the PDF curve of height vs. width:

The *area* under the curve between 25 and 30 = probability that 25°C ≤ X ≤ 30°C.
The *area under the whole curve* = 1 (means 100% total probability).

Formula:

$$\int p(x)\,dx = 1$$

(“The total area under the curve equals 1.”)

Cumulative Distribution Function (CDF)

The **CDF** (written F(a)) tells you:
e "What's the probability that X is less than or equal to a certain number (a)?"

So:

$$F(a) = P(X \leq a) = \int_{-\infty}^{a} p(x)dx$$

To find the probability between two numbers (say 3 and 6):

$$P(3 \leq X \leq 6) = F(6) - F(3)$$

Probability Mass Function (PMF)

For **discrete variables** (like dice or coin tosses):

You can count the outcomes.
You use a **PMF**, not a PDF.

Example:
If X is the result of a dice roll,

$$p(3) = P(X = 3)$$

And the total probability of all outcomes adds up to 1:

$$p(1) + p(2) + ... + p(6) = 1$$

This not Explains Koller page 21, clearly

X is a set containing random variables

x = {w1, w2}

x = {w1 = rainy, w2 = sunny}

Val(X) = all possible values for the random variables (rainy, sunny, stormy, foggy)

Y ⊆ X, Y is a subset of X

Y = {w1}+

x⟨Y⟩ takes the assignment x (value of x) and keeps only the values of Y

X = the random variable (or set of variables)

Val(X) = all *possible* assignments that X can take **x** ∈ **Val(X)** → **x** is *one* *specific assignment* (one possible value) of X

For two assignments **x** (over variables X) and **y** (over variables Y), we write **x ~ y** if they **agree on the variables they have in common**.

That is: $x\langle X \cap Y\rangle = y\ X \cap Y\rangle$

→ **Both assignments give the same values to the variables that are in the intersection** of X and Y.

1. X and Y are sets of random variables

X and Y are **not necessarily independent** — they are just sets of variables.

For example:

X = {Weather forecast from Station 1}

Y = {Weather forecast from Station 2}

2. x and y are assignments

x = one particular forecast from X (e.g., "rainy")

y = one particular forecast from Y (e.g., "rainy")

3. x ~ y

The notation x ~ y just means:
"x and y agree on the variables they share in common (X ∩ Y)"
In your TV station example:
Suppose both X and Y include "today's weather"
Then X ∩ Y = {today's weather}
x⟨X ∩ Y⟩ = "rainy" (Station 1)
y⟨X ∩ Y⟩ = "rainy" (Station 2)
Since they match → x ~ y ✅

8/21/2025

⊨ is called the **semantic entailment symbol** (sometimes read as "models" or "entails").

In probability and logic, it means:

> **$P \models (\alpha \perp \beta \mid \gamma)$**
> → "In the probability distribution P, it is true that α is conditionally independent of β given γ."

So, ⊨ is like saying **"according to the model/distribution P, this statement holds."**

In logic:
$M \models \varphi$ means *the model M satisfies (or makes true) the formula φ.*

Expectation

Expectation

Defnition 1: is a weighted average of all possible outcomes. If some utcomes are more likely, they “pull” the expectation toward them.
Defnition 2: if you randomly sample from a distribution what would you think “average of the samples will likely be”.
===

october 20, 2025

Continuous Variables and Continuous Domain (Simple Explanation)

Think of **continuous variables** as numbers that can take **any value in a range**, not just whole numbers.

✅ Example:

> The **temperature** outside could be 25.2°C, 25.25°C, or 25.256°C — it can take *any* value, not just 25 or 26.
> That’s a **continuous variable**.

Probability Density Function (PDF)

Because there are **infinitely many possible values**, we can’t say “the probability that X = 25.2” — that would be **zero** (since it’s one exact point).
So instead, we use the **Probability Density Function (PDF)**, written as **p(x)**.

Key things about the PDF:

> It shows **how dense** the probabilities are around each value.
> It’s like saying: “where is the probability more concentrated?”
> The **area under the curve** gives the probability, not the height of the curve.

✅ Example:
If you draw the PDF curve of height vs. width:

> The *area* under the curve between 25 and 30 = probability that 25°C ≤ X ≤ 30°C.
> The *area under the whole curve* = 1 (means 100% total probability).

Formula:

$$\int p(x)dx = 1$$

(“The total area under the curve equals 1.”)

Cumulative Distribution Function (CDF)

The **CDF** (written F(a)) tells you:
e “What’s the probability that X is less than or equal to a certain number (a)?”

So:

$$F(a) = P(X \leq a) = \int_{-\infty}^{a} p(x)dx$$

To find the probability between two numbers (say 3 and 6):

$$P(3 \le X \le 6) = F(6) - F(3)$$

Probability Mass Function (PMF)

For **discrete variables** (like dice or coin tosses):

You can count the outcomes.

You use a **PMF**, not a PDF.

Example:
If X is the result of a dice roll,

p(3)=P(X=3)p(3) = P(X = 3)p(3)=P(X=3)

And the total probability of all outcomes adds up to 1:

p(1)+p(2)+...+p(6)=1p(1) + p(2) + ... + p(6) = 1p(1)+p(2)+...+p(6)=1

Probability Mass Function (PMF)
For discrete variables (like dice or coin tosses):

You can count the outcomes.

You use a PMF, not a PDF.

Example:
If X is the result of a dice roll,
p(3)=P(X=3)p(3) = P(X = 3)p(3)=P(X=3)
And the total probability of all outcomes adds up to 1:
p(1)+p(2)+...+p(6)=1p(1) + p(2) + ... + p(6) = 1p(1)+p(2)+...+p(6)=1

8-5-2025

1. Representation (How we describe probability distributions with graphs)

Bayesian networks (BNs):

Definitions – A BN is a probabilistic model that represents variables and their conditional dependencies using a **directed acyclic graph (DAG)**.

Directed graphs – Nodes = random variables, Edges = direct causal/conditional relationships.

Independencies – BNs encode conditional independencies: a variable is independent of its non-descendants given its parents.

Markov Random Fields (MRFs):

Undirected vs directed models – Unlike BNs, MRFs use **undirected graphs** to represent mutual dependencies (good for things like image segmentation where relationships are symmetric).

Independencies – MRFs use the **Markov property**: a node is independent of others given its neighbors.

Conditional Random Fields (CRFs) – A type of MRF used for prediction tasks; models the conditional distribution P(Y ∣ X)P(Y | X)P(Y ∣ X), common in NLP and computer vision.

Type of Variable	Example	Function Used	Key Operation	Probability at Exact Value
Discrete	Rolling a dice	PMF (p(x))	Sum (∑)	Non-zero (e.g. P(X=3)=1/6)
Continuous	Measuring temperature	PDF (p(x))	Integral (∫)	Always 0

---

2. Inference (How we answer probability questions with these models)

Variable elimination

The **inference problem**: Given a graphical model, compute probabilities like marginals or conditionals.

Variable elimination: Systematically eliminate variables by summing them out.

Complexity: Depends on graph structure (can be exponential in the worst case).

Belief propagation

Junction tree algorithm – Convert graph into a tree-like structure to do exact inference efficiently.

Exact inference – Works for graphs without cycles.

Loopy belief propagation – Apply the same idea even with cycles (approximate, but works well in practice).

MAP inference (Maximum a Posteriori)

Find the **most probable assignment** of variables, not just probabilities.

Max-sum message passing – A variant of belief propagation for MAP.

Graph cuts – Efficient algorithm for MAP in vision problems.

Linear programming relaxations & dual decomposition – Advanced optimization techniques.

Sampling-based inference

Instead of exact math, **simulate samples** to estimate probabilities.

Monte Carlo sampling – General random sampling.

Forward sampling – Generate from the model step by step.

Rejection sampling – Generate then reject inconsistent samples.

Importance sampling – Weight samples for efficiency.

Markov Chain Monte Carlo (MCMC) – Build dependent samples that converge to the true distribution.

Applications – Useful when exact inference is impossible.

Variational inference

Approximate a complicated distribution with a simpler one.

Variational lower bounds – Replace hard integrals with optimization problems.

Mean Field – Assume independence between variables for tractability.

Marginal polytope relaxations – Approximation techniques for complex dependencies.

---

3. Learning (How we estimate model parameters/structure from data)

Directed models (Bayesian networks)

Maximum likelihood estimation (MLE) – Fit parameters to maximize likelihood of data.

Learning theory basics – Generalization, bias-variance tradeoff.

MLE for BNs – Straightforward if graph structure is known.

Undirected models (MRFs/CRFs)

Exponential families – Common parametric forms (e.g., logistic regression, CRFs).

MLE with gradient descent – Need iterative optimization (partition function is tricky).

Learning in CRFs – Estimate parameters for conditional models.

Latent variable models

Latent variables – Hidden/unobserved factors (e.g., clusters).

Gaussian Mixture Models (GMMs) – Example of a latent variable model.

Expectation Maximization (EM) – Standard algorithm for learning with hidden variables.

Bayesian learning

Bayesian paradigm – Treat parameters as random variables with priors.

Conjugate priors – Priors that make posterior math easy.

Examples – Normal-Normal, Beta-Binomial, etc.

Structure learning

Goal: Learn the **graph structure** (not just parameters) from data.

Chow-Liu algorithm – Efficient method for tree-structured networks.

AIC / BIC – Model selection criteria that trade off fit vs complexity.

Bayesian structure learning – Uses Bayesian methods to score possible structures.

Representation			
Graphical Structural Representation		Quantitative Representation	
baysian nw	marcove nw		

8-30-2025

Bayesian Network

The Scenario (Example)

Imagine we have a much simpler medical world with just **3 binary variables**:

1. `HasFlu` (F)
2. `HasFever` ®
3. `Coughs` (C)

And our simple Bayesian Network has this structure:

`HasFlu` → `HasFever`
`HasFlu` → `Coughs`

This means `HasFlu` is the parent of both `HasFever` and `Coughs`.

Our Tiny "dataset.dat"

Let's say our training data has only **10 patient records**. Each number represents a patient's full

state (like in your assignment).

`[3, 1, 3, 0, 3, 1, 0, 1, 3, 0]`

`NB: Here the variables are represented using a 3 bit binary digit.`

We need to convert these to binary to see the symptoms. Remember, the integer is encoded as a binary number where the rightmost bit is `HasFlu`.

Integer	Binary (F, R, C)	Meaning
0	`000`	**No Flu**, No Fever, No Cough
1	`001`	**Has Flu**, No Fever, No Cough
3	`011`	**Has Flu**, No Fever, **Has Cough**
0	`000`	**No Flu**, No Fever, No Cough
3	`011`	**Has Flu**, No Fever, **Has Cough**
1	`001`	**Has Flu**, No Fever, No Cough
0	`000`	**No Flu**, No Fever, No Cough
1	`001`	**Has Flu**, No Fever, No Cough
3	`011`	**Has Flu**, No Fever, **Has Cough**
0	`000`	**No Flu**, No Fever, No Cough

(Note: In this tiny example, `HasFever` never occurs, which is unrealistic but helps show how smoothing works.)

The "Empty CPTs" We Need to Fill

Our model has three variables. Their CPTs will look like this before learning:

1. CPT for `HasFlu` (it has no parents, so it's simple)

P(`HasFlu` = true)	
?	

2. CPT for `HasFever` (parent: `HasFlu`)

HasFlu	P(HasFever = true | HasFlu)
false	?
true	?

3. CPT for Coughs (parent: HasFlu)

HasFlu	P(Coughs = true | HasFlu)
false	?
true	?

Parameter Learning: Calculating the Numbers

Now, we use the data to calculate the ? values.

1. Learn P(`HasFlu` = true)

Count total patients: 10

Count patients with Flu (HasFlu=1): Let's find all integers that have the first bit set: 1, 3, 1, 3, 1, 3. That's **6 patients**.

Apply Laplace Smoothing:

```
P(F = true) = (Count(F=true) + 1) / (Total Patients + 2) = (6 + 1) / (10 + 2)
= 7/12 ≈ 0.583
```

Calculated Number for CPT: 0.583

2. Learn P(`HasFever` = true | `HasFlu`)

This has two rows. We calculate each separately.

Row 1: Given `HasFlu = false` **(0)**

Count patients with `HasFlu = false`**:** The patients with integer 0. This happened **4 times**.

From those 4, count how many have `HasFever = true`**:** Look at the binary for 0: `000`. The fever bit is 0. This happened **0 times**.

Apply Smoothing:

`P(R=true | F=false) = (0 + 1) / (4 + 2) = 1/6 ≈ 0.167`

Row 2: Given `HasFlu = true` **(1)**

Count patients with `HasFlu = true`**:** We already know this is **6**.

From those 6, count how many have `HasFever = true`**:** Look at the data. Patients with flu are integers 1 and 3. Their binary is `001` and `011`. The fever bit is always 0. This happened **0 times**.

Apply Smoothing:

`P(R=true | F=true) = (0 + 1) / (6 + 2) = 1/8 = 0.125`

Calculated Numbers for CPT: `0.167` and `0.125`

3. Learn P(`Coughs` = true | `HasFlu`)

Again, two rows.

Row 1: Given `HasFlu = false` **(0)**

Count patients with `HasFlu = false`**: 4 times** (integer 0).

From those 4, count how many have `Coughs = true`**:** Look at the binary for 0: `000`. The cough bit is 0. This happened **0 times**.

Apply Smoothing:

`P(C=true | F=false) = (0 + 1) / (4 + 2) = 1/6 ≈ 0.167`

Row 2: Given `HasFlu = true` (1)

Count patients with `HasFlu = true`: 6 times.

From those 6, count how many have `Coughs = true`: Patients with flu are integers 1 and 3. Integer 1 (`001`) has no cough. Integer 3 (`011`) has a cough. So cough happened **3 times** (for each integer 3 in the list).

Apply Smoothing:

```
P(C=true | F=true) = (3 + 1) / (6 + 2) = 4/8 = 0.5
```

Calculated Numbers for CPT: `0.167` and `0.5`

The Final, Learned CPTs

After parameter learning, our previously empty CPTs are now filled with these **calculated numbers**:

1. CPT for `HasFlu`

P(`HasFlu` = true)
0.583

2. CPT for `HasFever`

`HasFlu`	P(`HasFever` = true | `HasFlu`)
false	0.167
true	0.125

3. CPT for `Coughs`

`HasFlu`	P(`Coughs` = true | `HasFlu`)
false	0.167

`HasFlu`	P(`Coughs` = **true** | `HasFlu`)
true	0.5

These numbers are the **parameters** of your model. They are all calculated directly from the data in `dataset.dat` using counting and Laplace smoothing. This entire process is **parameter learning**.

Representation

- the first task when drawing graphs or Bayesian NW

BN (Bayesian network):-

. indicating the direction of influence or causality (from cause to effect)

. edges are directed from cause --> effect

. edges are acyclic (no loops allowed)

. BN assums most relationships are conditionally independent with each other (that is why it reduces the complextity of having multiple dimensions since it leaves out independent variables and represents RVs with meaningfull relationship)

Example: Flu --> Caugh --> fatigue

What is Image Segmentation?

Image segmentation = dividing an image into meaningful parts (regions, objects, or boundaries).
Instead of treating every pixel separately, segmentation groups pixels with similar properties (color, texture, intensity, etc.) so we can analyze objects in the image.

The segmentation is then found by minimizing an **energy function** combining:

Unary costs → how well a pixel fits a label.

Pairwise costs → how smooth/consistent neighboring labels are.

When to Avoid MRF and Use CRF Instead?

Use CRF if you have labeled data for training, complex data (e.g., noisy Sentinel-2 images), or need to incorporate multiple features (e.g., color, texture, elevation) to improve accuracy.

Important terms and concepts

Marginal Independence

Two variables **X** and **Y** are **marginally independent** if knowing one does **not** change the probability of the other.

Outfit and **Happiness** are both influenced by Weather.

Without knowing the Weather:

If you see someone wearing a raincoat (Outfit = raincoat), you can guess it's rainy, which also means lower Happiness.

So Outfit and Happiness are **dependent**.

$P(\text{Outfit},\text{Happiness}) \neq P(\text{Outfit}) \cdot P(\text{Happiness})$

Conditional Independence

Two variables **X** and **Y** are **conditionally independent given Z** if:

$P(X,Y \mid Z)=P(X \mid Z) \cdot P(Y \mid Z)$

This means: once we know **Z**, knowing **X** gives us no extra info about **Y**.

- A **latent variable** is a variable that **we don't directly observe** in the data.Instead, it's **inferred indirectly** from the relationships between the observed variables.
- They often represent **underlying causes** or **unmeasured factors**.

Psychology (IQ tests)

Observed: test scores in math, reading, logic.

Latent: "intelligence" (not directly measured, only inferred).

Markov property is a principle in probability and statistics (especially in stochastic processes) that says: - **The future depends only on the present, not on the past.**

Each node is conditionally independent of all other nodes given its neighbors

Markov Random Field

Represents how variables mutually influence each other, using undirected links and factorized weights instead of cause-and-effect arrows.

Potential functions (ψ sigh)

are flexible "preference scores" showing how strongly variables in an MRF agree or align. They aren't probabilities themselves — they *become* probabilities after dividing by the partition function Z.

– **potential function**, also called a **factor**, is **not** a probability.
It's a **positive number** that tells us *how compatible* or *how likely to go together* a certain combination of variable values is.

Think of it like a **"preference score"** or **"energy level"** between variables.

joint probability distribution (JPD) is the probability of all variables taking specific values at the same time.

Marginal probability = overall probability of observing today's satellite images (regardless of the true weather).
Probability of a variable regardless of others (can be computed from joint probabilities).
probability of a variable, possibly estimated from data or computed from the model.)

Prior = your initial guess about tomorrow's weather (say, 70% sunny, 30% rainy).
Likelihood = how well today's satellite images match each possibility.
Posterior = updated belief after combining prior + likelihood.

PDF as a **curve of likelihood** — the area under the curve between two points gives the probability.

- **Factors / Potentials** → score combinations of variables (like a table).
- **A factor (or potential function)** is a table or function that assigns a "score" to each combination of variable values.
- **Purpose**: Measures how compatible a combination of variables is. High score → more likely, low score → less likely.
- **Cliques** → groups of variables that are fully connected; each clique gets a factor.

: A clique is a subset of nodes in a graph where every node is connected to every other node in that subset.
joint probability distribution (JPD): is the probability of all variables taking specific values at the same time.

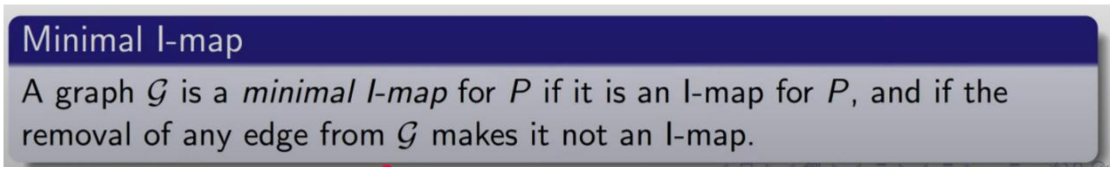
Cross check and Read

I-map: it is drawing the graph given the probablity distributio
I-map: only needs to encode some independencies that hold in the distribution without asserting any false independencies. It does **not** have to capture all of the distribution's conditional independencies.

Minial i-map: is also drawing the graph given the probabltity distribution but. the graph should contain all the random variables given in the probablity distribution.

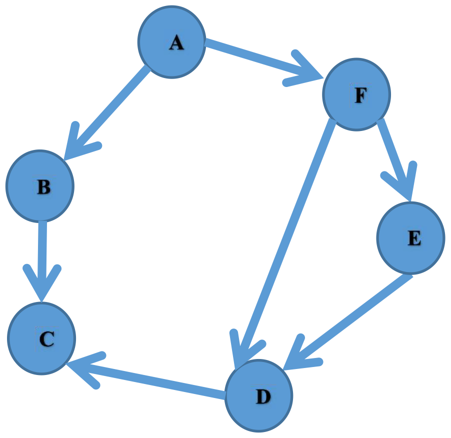
P-map (Probability Map)
Definition: A P-map is a graph that represents a specific probability distribution. It is a graph where the structure corresponds to the actual probability values assigned to the variables, typically reflecting how the joint distribution can be factored into conditional distributions.

P-map (Probability map) is often considered synonymous with a **minimal I-map** (Independence map) in the context of graphical models.



Minimal I-map

A graph $\mathcal{G}$ is a *minimal I-map* for P if it is an I-map for P, and if the removal of any edge from $\mathcal{G}$ makes it not an I-map.



Chordal Graphs:

A graph is chordal (or "perfect") if every cycle of four or more vertices has a chord, which is an edge that is not part of the cycle but connects two vertices in the cycle. Chordal graphs have nice properties that make them useful in probabilistic reasoning.

The cycle is: A - B - C - D - E - F
The Chord is: F - D

The core idea of PGMs is: instead of storing the giant joint probability table directly (which grows exponentially), we factorize it using the graph structure

==

9-10-2025

INFRENCE

Variable Elimination: A method for exact inference in probabilistic graphicalmodels (e.g., Bayesian networks). It computes marginal probabilities by systematically eliminating variables one by one, summing out their contributions using factor operations (e.g., multiplication and summation). Efficient for sparse networks but can be computationally expensive with many variables.

Sum-Product / Max-Product Algorithm: A message-passing algorithm used in factor graphs for inference. The Sum-Product variant computes marginal probabilities by passing messages between nodes to sum out variables, while Max-Product finds the most probable configuration (MAP estimation) by maximizing over variables. Both are exact on trees but approximate on loopy graphs.

Junction Tree Algorithm: An exact inference algorithm for Bayesian networks, transforming the model into a junction tree (a tree of cliques) to handle loops. It uses message passing between cliques to compute marginals efficiently, making it suitable for complex, loopy networks like yours (A → C ← B → D).

Loopy Belief Propagation: An approximate inference method extending belief propagation to graphs with loops. It iteratively passes messages between nodes, even in cyclic structures, to estimate marginal probabilities. While not guaranteed to converge or be exact, it's computationally efficient and often works well in practice (e.g., in error-correcting codes or neural networks).

3.3s

what are these or what do they mean 1. factor graphs 2. MAP estimation 3. exact inference method 4. approximate inference method 5. message passing

1. **Factor Graphs**: A bipartite graphical representation of a probabilistic model, consisting of variable nodes and factor nodes. Factor nodes represent functions (e.g., probability distributions) that depend on connected variable nodes. They're used to visualize and perform inference in models like Bayesian networks or Markov random fields, making dependencies and computations clearer.
2. **MAP Estimation**: Stands for Maximum A Posteriori estimation. It's a method to find the most probable configuration of variables in a probabilistic model given evidence, by maximizing the posterior probability P(variables ∣ evidence) P(\text{variables} | \text{evidence}) P(variables ∣ evidence). It's widely used in decision-making or prediction tasks, differing from marginal inference by focusing on a single best outcome.
3. **Exact Inference Method**: A technique that computes the exact probability distributions (e.g., marginals or conditionals) in a probabilistic model, such as variable elimination or the junction tree algorithm. It guarantees accurate results but can be computationally intensive, especially for large or loopy networks, often requiring exponential time in the worst case.
4. **Approximate Inference Method**: A technique that estimates probability distributions when exact methods are infeasible, using approximations like sampling (e.g., Monte Carlo) or iterative methods (e.g., loopy belief propagation). It's faster and scalable but may introduce errors, making it suitable for complex models with many variables.
5. **Message Passing**: A general inference strategy where nodes in a graphical model (e.g., factor graphs or junction trees) exchange information (messages) to update beliefs or probabilities. Used in algorithms like sum-product or belief propagation, it enables distributed computation and is key to both exact and approximate inference.

---

1. Chain (Head-to-Tail) (Causal or Evidence Trail)

Note: parent = cause , child = Évidence

$$A \rightarrow B \rightarrow C$$

Information flows along the chain.

$A \perp C \mid B$

$A \perp C$

Rule:

Without conditioning: A and C are dependent.

With conditioning on B: $A \perp C \mid B$

Example:

A=Cloud → B=Rain→ C=Wet ground.

Rain and Wet ground are dependent, but if we know about Clouds, then Rain doesn't add extra info about Wet ground.

2- Fork (Tail-to-Tail) (Common Cause) (common Parent)

Note: parent = cause , child = Evidence

$$A \leftarrow B \rightarrow C$$

$A \perp C \mid$

One parent causes two children.

Rule:

Without conditioning: A and C are dependent. (therefor $A \perp C$ is wrong)

With conditioning on B: $A \perp C \mid B$

Example:

B=Genetic mutation

A=Heart disease

C=Diabetes.

bLODOD PRESS ←Genetic mutation →Diabetes.

Heart disease and Diabetes appear related, but once we know the mutation status, they're independent.

4. Collider (Head-to-Head) (Common Effect) (Decendants) (V SHAPE)

Note: parent = cause , child = Evidence $A \rightarrow C \leftarrow B$

Two causes meet at a common effect.

Rule:

Without conditioning: $A \perp B$

With conditioning on C: A not $\perp B \mid C$

Example (Explaining Away):

A=Burglary

B=Earthquake

C=Alarm.

Burglary --> Alarm. <-- Earthquake

Burglary and Earthquake are independent.

But if the Alarm goes off, and we know there was a Burglary, the probability of an Earthquake goes down (and vice versa).

Structure	Graph	Independence rule	Example
Head-to-Tail (Chain)	a→c→b	a⊥b | c	Rain → Clouds → Wet ground
Tail-to-Tail (Fork)	a←c→b	a⊥b | c	Mutation → Heart disease → Diabetes
Head-to-Head (Collider)	a→c←b	a⊥b, a not⊥ b | c	Burglary → Alarm ← Earthquake

Latent Variables

(also called Hidden Variables)

- A **latent variable** is a variable that **we don't directly observe** in the data.Instead, it's **inferred indirectly** from the relationships between the observed variables.
- They often represent **underlying causes** or **unmeasured factors**.

Psychology (IQ tests)

Observed: test scores in math, reading, logic.

Latent: "intelligence" (not directly measured, only inferred).

Explaining away" is one of the most **interesting Phenomena in Bayesian**

Networks, and it happens in the **Head-to-Head (collider) structure:**

A: Burglary

B: Earthquake

C: Alarm goes off

Without evidence (alarm not observed):
Burglary and Earthquake are independent — one doesn't affect the other.

If alarm = ON:

If you learn that a **burglary happened**, that **explains the alarm** → the probability of an earthquake being the cause **goes down**.

If no burglary, then the earthquake becomes a more likely explanation.

Both causes compete to "explain" the effect → **explaining away**.

We move away from burglary to earthquake that is why it is called explaining away

Active Path and Blocked Path

· If we **don't observe WetGround** → Rain and Sprinkler are independent (path is blocked).

· If we **observe WetGround** → The path becomes active → Rain and Sprinkler become dependent (explaining away: "If WetGround is wet, was it Rain or Sprinkler?").

Season ← Weather → FluCases

Active path =

. information flows, variables can affect each other,

. children (Season, FluCases) are dependent.

. If you don't know the Weather: "If it's winter (Season), flu cases are probably high." (Season gives info about FluCases).

Blocked path =

. information doesn't flow, variables are independent given the evidence,

. children become independent given the cause.

.If you do know the Weather: "If Weather = cold, then FluCases are high, but Season adds no new info." (Season is no longer useful once Weather is known).

D-Separation = "Directional Separation"

Purpose: Test for conditional independence between nodes in a Bayesian Network (Directed Acyclic Graph - DAG)

D-separation (Directional Separation) is a criterion used to decide whether two sets of variables are **independent** given some observed variables (evidence).

Two variables (say X and Y) are **d-separated** by a set of nodes Z if **all paths** between X and Y are **blocked** once we consider Z.

If there is at least one **active path**, then X and Y are **dependent** (not d-separated).

f PDF as a **curve of likelihood** — the area under the curve between two points gives the probability.

9/18/2025

Markov property is a principle in probability and statistics (especially in stochastic processes) that says:

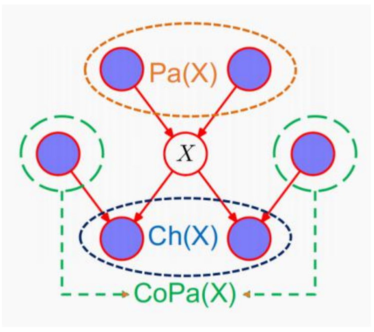
- The future depends only on the present, not on the past.

Bayes Ball Algorithm

If we throw a ball from x to z, if it passes to z so we say it is not blocked

The ball follows rules defined by the, canonical 3-node subgraphs we've discussed.

Marcov Blanket



$$\boxed{\begin{aligned} &Mb(X) \\ &= Pa(X)\ U\ Ch(X)\ U\ CoPa(X) \end{aligned}}$$

· **Markov Blanket**: The Markov blanket of a node X includes its parents (Pa(X)), children (Ch(X)), and co-parents (CoPa(X)). The statement "X conditionally independent of all other nodes, given its Markov blanket" means that once the values of Pa(X) Pa(X) Pa(X), Ch(X) Ch(X) Ch(X), and CoPa(X) are known, X is independent of all othernodes in the network.

Analogy: suppose there is a Father: Marcove blankate of the father are
1. his parents (mother and father)
2. his childrens
3. His childrens mother (if has children from diffent wives then all his childerens mothers)

Diagram Breakdown:

Parents (Pa(X)**):** Nodes that directly influence X (shown in orange dashed ovals).
Children (Ch(X) **):** Nodes that X directly influences (shown in blue dashed ovals).
Co-parents (CoPa(X)**):** Nodes that are parents of X's children but not parents of X itself (shown in green dashed ovals). These are the nodes that share children with X.

The Venn diagram on the left visually represents the overlap between Parents, Children, and Co-parents.

- **Why Co-parents?:**

  The question "Why co-parents?" is answered with "Explaining away."
  Explaining away occurs in a network where multiple causes (e.g., co-parents) can lead to the same effect (a child node). When one cause is observed, it reduces the likelihood of the other causes, making them dependent. For X to be conditionally independent of all other nodes given its Markov blanket, the co-parents must be included because they can influence X indirectly through shared children. Conditioning on the co-parents blocks this path, ensuring independence.

Clique is a fully connected group of variables (everyone in the group affects everyone else)

The term $\psi c(x_c)$ in the context of the image about factorized probability distributions (specifically for Markov Random Fields, or MRFs) represents a **factor** or **potential function** associated with a clique c c c in the graph. Let's break it down simply:

What It Means

ψc : This is a function that quantifies the relationship or compatibility between the variables in a specific group (called a clique c). It's a nonnegative value (never negative), which reflects the strength or likelihood of those variables taking certain values together.

xc) : This refers to the subset of random variables $x=(x1 ,…,xd)$ that belong to the clique c c c. For example, if c c c includes the indices {1,2}

Together, $\psi c(xc)$: This is the potential function evaluated for the specific values of the variables in clique c. It acts like a "score" that indicates how well those values fit together based on the relationships in the MRF.

What is Factorization (in probability)?

Factorization means breaking down a **big joint probability distribution** into **smaller, simpler pieces (factors)** that show **local relationships**.

Ѣ Instead of writing one huge probability table for all variables together, we multiply smaller tables that depend only on a few variables.

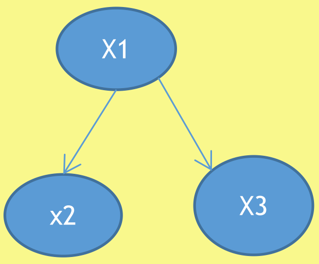
Understanding Factorization in Probability

What is Factorization?

Factorization means **breaking down a big, complicated probability distribution into smaller, easier parts**.
Instead of handling one huge function for all variables, we split it into smaller pieces (factors) that describe local relationships.

- This makes calculations and understanding much easier.

Example with 3 Variables



We'll use a **rain and umbrellas** story.

X_1: Rain (Yes / No)

X_2: You bring an umbrella (Yes / No)

X_3: Your friend brings an umbrella (Yes / No)

Step 1: Define Dependencies

Rain (X_1) affects **your umbrella** (X_2).

Rain (X_1) affects **your friend's umbrella** (X_3).

Your umbrella and your friend's umbrella are **not directly connected** (only through rain).

Step 2: Factorize the Joint Probability

Instead of writing the full joint probability:
$p(X1,X2,X3)$ which requires 2^3
We factorize as: $p(X1,X2,X3)= p(X1)\cdot p(X2 \mid X1)\cdot p(X3 \mid X1)$
Step 3: Assign Simple Probabilities

Rain (X_1):

X_1	$P(X_1)$
Yes	0.3
No	0.7

Your Umbrella (X_2 | X_1):

If X_1 = Yes (Rain):

X_2	$P(X_2 \mid X_1=Y)$
Yes	0.9
No	0.1

If X_1 = No (No Rain):

X_2	$P(X_2 \mid X_1=N)$
Yes	0.1
No	0.9

Friend's Umbrella (X_3 | X_1):

If X_1 = Yes (Rain):

X_3	$P(X_3 \mid X_1=Y)$
Yes	0.8
No	0.2

If X_1 = No (No Rain):

X_3	$P(X_3 \mid X_1=N)$
Yes	0.2
No	0.8

Full joint probability table

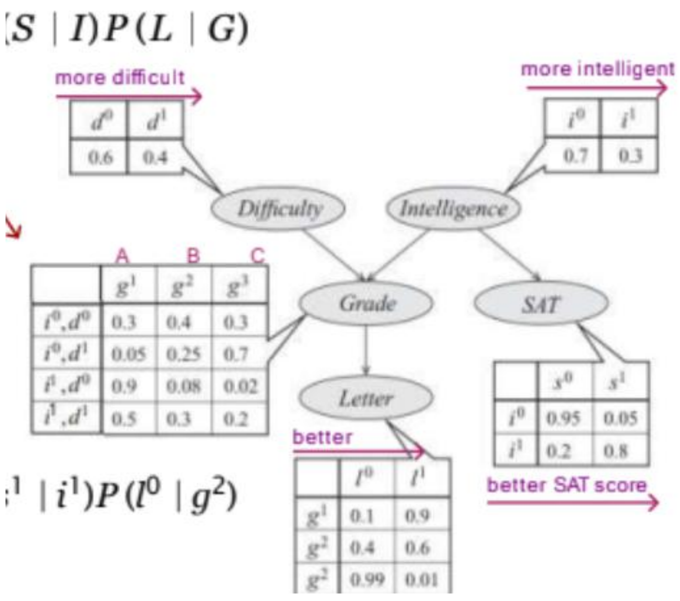
X1 Rain)	X2 (You)	X3 (Friend)	Calculation	Joint Prob
Yes	Yes	Yes	0.3 × 0.9 × 0.8	0.216
Yes	Yes	No	0.3 × 0.9 × 0.2	0.054
Yes	No	Yes	0.3 × 0.1 × 0.8	0.024
Yes	No	No	0.3 × 0.1 × 0.2	0.006
No	Yes	Yes	0.7 × 0.1 × 0.2	0.014
No	Yes	No	0.7 × 0.1 × 0.8	0.056
No	No	Yes	0.7 × 0.9 × 0.2	0.126
No	No	No	0.7 × 0.9 × 0.8	0.504

Step 4: Build Joint Probability by Multiplying Factors

Now we can compute any joint probability by multiplying.
e Example:
P(X1=Yes,X2=Yes,X3=Yes)= 0.3×0.9×0.8=0.216

The **big joint probability** is **factorized** into **smaller conditional pieces**.

This reduces complexity and makes computations manageable.



	d^0	d^1
	0.6	0.4

	i^0	i^1
	0.7	0.3

	g^1	g^2	g^3
i^0,d^0	0.3	0.4	0.3
i^0,d^1	0.05	0.25	0.7
i^1,d^0	0.9	0.08	0.02
i^1,d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^2	0.99	0.01

---

oct 8, 2025

Marcov Random Fields

Think of an MRF like a recipe:

The **graph** tells you *which ingredients must mix* (dependencies).

The **factors** ψc\psi_cψc are the *mixing bowls* — you can combine them in different ways.

As long as you end up with the same *overall mixture* (joint probability), your recipe is valid.

Formula

A probability distribution $p(x)$ over random variables $x = (x_1, \ldots, x_d)$ can be expressed as:

$$p(x) = \frac{1}{Z}\prod_{c\in\mathcal{C}}\psi_c(x_c)$$

where:

$\mathcal{C}$: a set of subsets of indices $\{1, \ldots, d\}$, representing groups (or cliques) of variables.
$\psi_c(x_c)$: nonnegative *factors* (also called *potential functions*) that describe how variables in each group c interact.
Z: the *normalizing constant* (or *partition function*), ensuring the total probability sums to 1:

$$Z = \int \prod_{c \in \mathcal{C}} \psi_c (x_c)\, dx$$

A **minimal factorization** occurs when

Show me Z using simple number example

Perfect — let's make Z (the **normalizing constant**) super clear with a **simple numeric example** ℓ

Step 1: Define a simple setup

Suppose we have **two binary random variables** $x_1, x_2 \in \{0, 1\}$.
We define one potential function (factor):

$$\psi(x_1, x_2) = \begin{cases} 2 & \text{if} x_1 = x_2 \\ 1 & \text{if} x_1 \neq x_2 \end{cases}$$

This means:

> The system "prefers" x_1 and x_2 to be the same (weight 2),
> and gives less weight (1) when they differ.

Step 2: Compute all possible combinations

x_1	x_2	$\psi(x_1, x_2$
0	0	2
0	1	1
1	0	1
1	1	2

Step 3: Compute Z

$Z = \sum_{x_1, x_2} \psi\,(x_1, x_2)\ Z = 2 + 1 + 1 + 2 = 6$

So, $Z = 6$.

Step 4: Get normalized probabilities

Each joint probability is:

$$p(x_1, x_2) = \frac{1}{Z} \psi(x_1, x_2)$$

x_1	x_2	$\psi(x_1, x_2$	$p(x_1, x_2)$
0	0	2	$2/6 = 0.333$

x_1	x_2	$\psi(x_1, x_2$	$p(x_1, x_2)$
0	1	1	$1/6$ $= 0.167$
1	0	1	$1/6$ $= 0.167$
1	1	2	$2/6$ $= 0.333$

✅ All probabilities sum to 1:

$$0.333 + 0.167 + 0.167 + 0.333 = 1$$

Step 5: Intuitive meaning

> $Z = 6$ acts as a **scaling factor** so that all weighted combinations (the unnormalized ψ values) turn into **valid probabilities**.
>
> Without Z, we'd only have *relative preferences* (not actual probabilities).

Would you like me to extend this with an example that uses **three variables (x_1, x_2, x_3)** to show how Z scales in larger systems?

Variable Elimination

- Variable Elimination is a *general inference method* for *any* factor graph

it applies to both **Bayesian Networks** and **Markov Random Fields (MRFs)**.

VE works by elimination all variables in turn until there is a factor with only query variable

Global Marcov:

: A distribution is Markov if and only if, for any B separating A and C , the conditional independence holds:

> p(xA,xC ∣ xB) = p(xA ∣ xB)p(xC ∣ xB) p(x_A, x_C | x_B)

Local Marcov:

Each node XS is independent of all other variables xV\s (all variables except s s s) given its neighbors xT(s), where T(s) is the set of neighbors of S. Mathematically:

Global property is a broader, graph-wide rule, while the local property is a specific case that applies to each node based on its local structure.

p(xs ∣ xV\s)=p(xs ∣ xT(s))

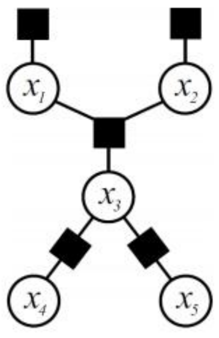
Hammersley Cliford Theorem:

$$p(x) \propto \prod_{c \in C} \psi_c\,(x_c)\,dx$$

probablity distribution defined by <u>**Product of non negative potential function over a clique**</u> is always marcove with respect to the graph G.

HyperGraphs:

hypergraph is a generalization of a standard graph in mathematics and computer science, where edges (called hyperedges) can connect more than two vertices, unlike traditional graphs where edges connect exactly two vertices.



Step-by-step core concepts (Probabilistic Graphical Models)

Below is a compact, ordered summary that pulls together the ideas in your text and answers the small questions you asked.

---

1. **Graph basics & separation**
   - A graph = nodes (variables) + edges (dependencies).
   - **Separation:** a set B *separates* A and C if **every** path from any node in A to any node in C passes through B.
   - Intuition: if B separates A and C, then B blocks all information flow between them.
2. **Global Markov Property (GMP)** — graph-level independence
   - If B separates A and C in the graph, then the distribution satisfies $p(x_A, x_C|x_B) = p(x_A|x_B)\, p(x_C|x_B)$.
   - Equivalent conditional forms: $p(x_A|x_B, x_C) = p(x_A|x_B)$, etc.
   - Intuition: any two groups of variables separated by B are conditionally independent given B.
3. **Local Markov Property (LMP)** — node-level independence
   - For each node s, let $T(s)$ be its neighbors. Then $p(x_s|x_{V\setminus s}) = p(x_s|x_{T(s)})$.
   - The **Markov blanket** for a node in an *undirected* graph = its immediate neighbors. (In Bayesian networks the Markov blanket = parents ∪ children ∪ co-parents.)
   - Relationship to GMP: LMP is the per-node instance of separation. For strictly positive distributions the Local, Pairwise and Global Markov properties are equivalent.
4. **Hammersley-Clifford Theorem (HC)** — factorization ↔ Markov property
   - If a strictly positive distribution is Markov w.r.t. an undirected graph G, **then** it can be written as a product of non-negative potentials over the cliques:

     $p(x) = \frac{1}{Z}\prod_{C\in\mathcal{C}} \psi_C(x_C)$.

     Here $\mathcal{C}$ = set of cliques, $\psi_C \geq 0$ are clique potentials, and Z is the partition function (normalizing constant).
   - Converse: any strictly positive distribution of that form is Markov w.r.t. G.
   - **Positivity requirement** is important — HC needs strictly positive densities for the equivalence.

5. **Normalization and the partition function Z**
   - $Z = \sum_x \prod_C \psi_C(x_C)$ (or integral for continuous variables).
   - Why often omitted: texts write $p(x) \propto \prod_C \psi_C(x_C)$ because the *factorization form* is the key structural statement; the actual normalization Z is implied. But to make p a valid probability you **must** divide by Z.
   - Practical note: computing Z is usually hard (combinatorial), which is why many algorithms avoid computing it directly.

6. **Potential functions (scores) — what you can assign**
   - A potential $\psi_f(x_f)$ is any non-negative function that scores configurations of variables in factor f. Values are arbitrary but **non-negative**.
   - Yes — you *can* assign rule-based scores like:

     $\psi(A, B) = 2$ if $A = B$, and $\psi(A, B) = 1$ if $A \neq B$.

     Or $\psi(A, B) = 3$ when $A > B$ (if A, B have an order), else 1.

- These choices are valid; the final probabilities come from normalizing the product of all potentials.
- **Example (tiny):** binary A, B with ψ = 2 when equal, 1 when different. Config weights: (00)=2, (11)=2, (01)=1, (10)=1 so $Z = 6$. Then $p(00) = 2/6 = 1/3$, $p(01) = 1/6$, etc. This shows how normalization changes scores into probabilities.

7. **Transformations: marginalization vs conditioning**
   - **Marginalizing out C:** compute $p(A,B) = \sum_C p(A,B,C)$. Removing (summing over) C can *create* a direct dependence between A and B even if they were only connected through C. Intuition: different values of C couple A and B when you ignore C.
   - **Conditioning on C:** compute $p(A,B|C)$. If C separates A and B, conditioning on C can *break* the dependency: $p(A,B|C) = p(A|C)p(B|C)$.
   - Short rule: *marginalize → can create edges (dependencies); condition → can remove edges (independence) when the conditioned set blocks paths.*
8. **Factor graphs & hypergraphs**
   - A **factor graph** is a bipartite graph: variable nodes (circles) and factor nodes (squares). Each factor node connects to all variables it involves. It explicitly shows the factorization

     $p(x) \propto \prod_{f\in\mathcal{F}} \psi_f(x_f)$.
   - A **hypergraph** generalizes edges: a hyperedge can join any number of vertices (not just pairs). Factor graphs can be seen as hypergraphs where each factor = hyperedge connecting its variables.
   - Factor graphs make complex interactions (e.g. $\psi(x_1, x_2, x_3)$) explicit and are convenient for message-passing algorithms.
9. **LDPC codes as an application of PGMs**
   - LDPC codes ↔ factor graphs: variable nodes = code bits, factor nodes = parity checks. The sparse parity check matrix H defines which bits each check touches.
   - Valid codewords satisfy hard constraints: indicator potentials $I(H_i t = 0 \bmod 2)$. With channel noise, soft likelihoods $p(r|t) = \prod_n p(r_n|t_n)$ produce soft potentials.
   - Inference (decoding) uses **belief propagation** (message passing) exploiting local independence: each variable updates beliefs from neighboring checks. This is exactly the local Markov idea in action.
   - Sampling/inference note: undirected models need iterative methods (Gibbs, BP, MCMC) because there are no locally normalized conditional distributions as in Bayesian Networks.

10. **Inference & simulation differences (BN vs MRF)**
    Bayesian Networks (directed): give locally normalized conditionals $p(x_i|\text{parents})$ → can do **ancestral sampling** (sample parents → children).
    Markov Random Fields / factor graphs (undirected): only a global normalization Z → ancestral sampling not available; use Gibbs sampling, MCMC, or belief propagation instead.
    Computationally important: computing marginals or Z is often intractable for large graphs; approximate methods are common.

Quick cheat-summary (one-liners)

Global: separation in graph ⇒ conditional independence.
Local: node independent of non-neighbors given neighbors (Markov blanket).

Hammersley-Clifford: positive Markov distributions factor over cliques (with partition function Z).
Potentials: arbitrary non-negative scores (relative scale matters; normalize with Z).
Marginalize: sum out a variable $\rightarrow$ may create dependencies.
Condition: fix a variable $\rightarrow$ may break dependencies.
Factor graph: explicit factors (hyperedges) $\rightarrow$ good for message passing.
LDPC: parity checks = factors; decoding = belief propagation on a factor graph.

Variable Elimination:

For a chain graph (e.g., A → B → C → D), marginals can be computed efficiently by reusing computations instead of naive summation.

Two major limitations of variable elimination:
1. Computation **exponential** in size of the largest intermediate factor (equivalently, largest clique in clique tree)
2. Computation is not reused for computing a series of marginals

✅ Belief Propagation = Message-passing framework
✅ Sum-Product Algorithm = Belief Propagation applied to compute marginals
✅ Max-Product Algorithm = Belief Propagation applied to find the most likely assignment (MAP)

- **Factor node** represents a function (or factor) that defines the relationship between a set of variables in the model. This function, often denoted as ψ\psiψ or fff, encodes the joint probability distribution or a local potential over the variables it connects.

Sum-Product / Max-Product Algorithm

Forward-Backward Algorithm (for Chains/HMMs):
- o Forward messages: Filtered posteriors incorporating past observations.
- o Backward messages: Incorporate future observations for smoothed posteriors.
- o Marginal: Product of forward/backward messages.
- o Time: O(T K^2) for T nodes, K states.

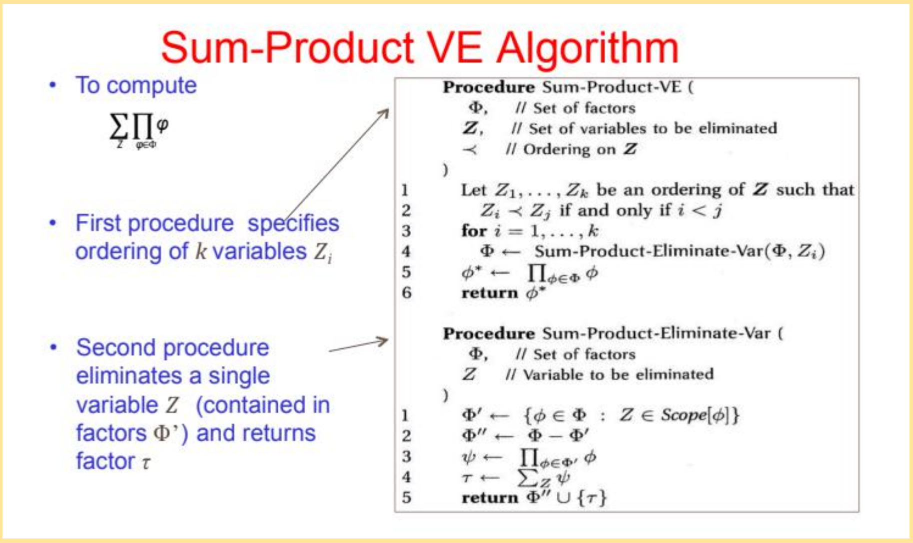
Sum-Product Belief Propagation (BP) for Trees:

- o Generalizes to tree-structured pairwise MRFs or factor graphs.
- o Messages passed from leaves to root, then root to leaves.
- o Variable-to-factor message: Product of incoming messages.
- o Factor-to-variable message: Sum over product of factor and incoming messages.
- o Marginal: Product of incoming messages (normalized).
- o Key: Messages depend on Markov blanket; recursive decomposition of subtrees.
- o Complexity: O(|E| K^2) for edges E (linear in graph size).

Non-Pairwise MRFs:

- o Convert to factor graphs; messages handle higher-order factors (complexity

O(K^{M+1}) for M-variable factor).



Sum-Product VE Algorithm

- To compute $\sum_{Z}\prod_{\varphi\in\Phi}\varphi$
- First procedure specifies ordering of k variables Z_i
- Second procedure eliminates a single variable Z (contained in factors Φ') and returns factor τ

```
Procedure Sum-Product-VE (
    $\Phi$,     // Set of factors
    $Z$,        // Set of variables to be eliminated
    $\prec$     // Ordering on $Z$
)
    Let $Z_1, \ldots, Z_k$ be an ordering of $Z$ such that
        $Z_i \prec Z_j$ if and only if $i < j$
    for $i = 1, \ldots, k$
        $\Phi \leftarrow$ Sum-Product-Eliminate-Var($\Phi$, $Z_i$)
    $\phi^* \leftarrow \prod_{\phi\in\Phi}\phi$
    return $\phi^*$

Procedure Sum-Product-Eliminate-Var (
    $\Phi$,     // Set of factors
    $Z$         // Variable to be eliminated
)
    $\Phi' \leftarrow \{\phi \in \Phi : Z \in Scope[\phi]\}$
    $\Phi'' \leftarrow \Phi - \Phi'$
    $\psi \leftarrow \prod_{\phi\in\Phi'}\phi$
    $\tau \leftarrow \sum_{Z}\psi$
    return $\Phi'' \cup \{\tau\}$
```

```
function SumProductVE(Φ, Z):
    // Φ = list of all factors (like tables of numbers)
    // Z = list of variables we want to eliminate (sum over)

    while Z is not empty:                          // Keep going until all variables are gone
        pick next variable Z_i from Z              // Choose one variable to eliminate now
        factors_with_Zi = all φ in Φ that include Z_i  // Find all factors that depend on Z_i
        ψ = product of factors_with_Zi             // Multiply those factors together → new big factor
        τ = sum out Z_i from ψ                     // Sum over all values of Z_i → remove Z_i
        remove factors_with_Zi from Φ              // Remove the old factors we used
        add τ to Φ                                 // Add the new factor (without Z_i) back
        remove Z_i from Z                          // Z_i is gone forever from the list
    end                                            // Loop repeats with remaining variables

    return product of all values in the final factor  // At the end, only one factor left → multiply all
its values
end
```

Maximum A Posteriori (MAP) Inference:

- **Max-Product/Max-Sum**: Replace sums with max (log-domain for stability).
- **Viterbi Algorithm:** Special case for HMMs; forward max messages, backward traceback for joint maximizer.
- **Max-marginal:** Gives per-node maximizers (Viterbi-style backtrack resolves ties).

Example:

- Detailed walkthrough of sum-product on a small factor graph, computing marginals by passing messages.

Homework Mention:

- Q1: Variable Elimination (hand calculation).
- Q2: Implement sum-product BP for tree factor graph.
- Q3: LDPC using loopy sum-product BP.

Summary:

- Exact marginal/MAP inference in trees.
- Generalizes Forward-Backward/Viterbi.

Junction Tree Algorithm

Limitations of Variable Elimination:

- Exponential in largest clique; no reuse for multiple marginals (e.g., all HMM marginals: O(T K^2) vs. naive O(T^2 K^2)).

Clique Tree:

- Built from elimination order; nodes are maximal cliques.

Junction Tree:

- Clique tree with running intersection property (shared variables on paths).
- From triangulated (chordal) graphs (no chordless cycles ≥4).
- Theorem: Clique trees from elimination are junction trees.
- Encodes conditional independencies via separators.

Algorithm (Shafer-Shenoy BP):

- Triangulate graph (any elimination order works; optimal NP-hard).
- Build junction tree (quadratic time worst-case).
- Run sum-product BP on cliques: Messages over separators, marginals by summing residuals.
- Computes all clique marginals efficiently (reuses messages).
- Complexity: Exponential in maximal clique size.

Steps:

- Convert to undirected if needed.
- Triangulate, build tree, execute BP.

Summary:

- Exact all-marginals in general graphs.
- BP on clique tree; caches computations.

Loopy Belief Propagation

Motivation:

- For graphs with cycles, exact methods (e.g., junction tree) may be intractable.
- Approximate by iterating BP updates on cyclic graphs.

Algorithm (Sum-Product):

- Initialize messages (constant or random).
- Parallel (synchronous): Update all messages using previous iteration.
- Asynchronous (sequential): Update using latest messages (faster convergence).
- Beliefs: Approximate marginals from product of incoming messages.
- For factor graphs: Similar, with factor/variable marginals.

Convergence:

- No guarantees; may oscillate or enter limit cycles.
- Measures: Max/total change in messages/beliefs < epsilon.
- Improves with: Message damping (partial updates), asynchronous schedules.
- Computation tree view: Iterations approximate tree unrolling; converges if edges weak.

Examples:

- Ising model: Convergence depends on small cycles; damping helps.
- LDPC Codes: Binary variables, parity check factors; loopy BP decodes efficiently.

Loopy Belife Propagation:

LBP works by iteratively passing "messages" between nodes along edges, updating beliefs until convergence. It's called "loopy" because it handles graphs with cycles (loops).

Junction Tree Algorithm

1. **Cluster nodes to break cycles**: This step transforms the original graph (which may have cycles) into a tree structure by grouping variables into clusters (or cliques). The goal is to eliminate loops, making exact inference possible.

2. Once the junction tree is built, Belief Propagation (BP) is applied exactly. Messages are passed between clusters, and marginal probabilities are computed by collecting and distributing evidence across the tree. This leverages the tree's property where information flows without ambiguity.

Rule for building the clique tree

Connect cliques by the **largest possible intersection**, ensuring all cliques remain connected and satisfy the running intersection property.

The typical construction algorithm is:

Create a complete graph where each clique is a node.

Weight each edge by intersection size.

Build a **maximum-weight spanning tree** (MWST).

“Maximum-Weight” Spanning Tree (MWST)?

Sometimes edges have **weights** — numbers that measure how “strong” or “important” that connection is.

A **Maximum-Weight Spanning Tree (MWST)** is:

> the spanning tree that connects all the nodes while **maximizing the total sum of edge weights**.

That means we keep the *strongest* possible connections (highest weights), while still avoiding cycles.

Conditioned Versions of Directed & Undirected Models

1. **Conditional Random Fields (CRFs):**

    An *undirected* graphical model used for **predicting structured outputs** (e.g., sequences or labels).
    Models **P(Y | X)** directly without modeling P(X).
    Useful in tasks like part-of-speech tagging and image segmentation.
2. **Conditional Bayesian Networks (CBNs):**

    A *directed* graphical model that represents **conditional dependencies** among variables given some observed inputs.
    Models **P(Y | X)** using directed edges and Conditional Probability Distributions (CPDs).

Representing Conditional Probabilities

1. **Deterministic CPD:**

    Defines outcomes as **deterministic functions** of parent variables (no uncertainty).
    Example: If A = true always causes B = true.
2. **Noisy-OR Model:**

    Simplifies CPDs when multiple independent causes can produce the same effect.
    Each cause has a certain probability of activating the effect.
3. **Logistic CPD:**

    Uses the **logistic (sigmoid) function** to model binary outcomes.
    Common in **classification tasks** and **logistic regression**.
4. **Linear Gaussian CPD:**

    Assumes the child variable is a **linear combination** of parent variables plus Gaussian noise.
    A special case of the **Generalized Linear Models (GLMs)** family.
5. **Neural Network CPDs:**

    Use **neural networks** to define complex, **nonlinear conditional probability distributions.**
    Highly expressive and can capture intricate dependencies.

Types of Graph

1. Undirected Graph

The **edges (connections)** between nodes have **no direction**.

Example: Friendship network — if A is friends with B, then B is also friends with A.

Represented as: A — B

2. Directed Graph (Digraph)

The edges **have direction** (arrows).

Eample: Following on Twitter — A → B (A follows B, but B may not follow A)

Represented as: A → B

3. Complete Graph

Every pair of distinct nodes is **directly connected** by an edge.

If there are nnn nodes, there are n(n−1)/2n(n−1)/2n(n−1)/2 edges (for undirected graphs).

Example: In a group of 4 friends, everyone knows everyone.

4. Acyclic Graph

A graph with **no cycles** — meaning, you can’t start at one node and come back to it by following the edges.

Example: A tree structure (like a family tree).

5. Partially Directed Acyclic Graph (PDAG)

A mix of **directed** and **undirected** edges, but still has **no cycles**.

Often used in **Bayesian networks** to represent uncertain directions of influence.

Example: A → B — C (A affects B, but the direction between B and C is not decided).

6. Induced Graph

A **subset** of nodes from a larger graph **and all edges connecting those nodes**

Example: If you pick nodes {A, B, C} from a bigger graph, the induced graph includes these nodes **and** all edges among them.

- ## 7. Subgraph

Any smaller graph made from part of a larger graph.

It may include **some nodes and some edges** (not necessarily all that connect them).

Induced graphs are a **special type of subgraph**.

Learning

Goals of Learning
1. Density Estimation
   - KL divergence, Log-loss
2. Prediction
   - Classification error
3. Knowledge Discovery

we start from:
start from graphical model (to acqure the model we can use our knowledge or ml) -->

Loss Functions:

· Use Hamming Loss for multi-label tasks where partial correctness is meaningful, as it better captures performance at the individual label level.
· Use Classification Error when exact label set matching is critical, though it’s less common in multi-label settings.