Lecture Notes for CS 726 - Spring 2022

Eeshaan Jain

March 27, 2022

These are my lecture notes taken during the Advanced Machine Learning (CS 726) course at IIT Bombay during the Spring 2022 session.

Contents

*List of Algorithms

Probabilistic Modeling

Probability Theory

We will briefly review probability in a rigorous sense.

We define events considering we have a space of possible outcomes denoted by Ω. $\mathcal{S}$ is a set of measurable events, to which we assign probabilities, and each event $\alpha \in \mathcal{S}$ is a subset of Ω.

In a single coin toss, we have

$$\Omega = \{H, T\}$$

The event space necessarily satisfies three properties -

1. It contains the empty event $\emptyset$ and the trivial event Ω.
2. It is closed under union, i.e if $\alpha, \beta \in \mathcal{S}$, so is $\alpha \cup \beta$.
3. It is closed under complementation, i.e if $\alpha \in \mathcal{S}$, so is $\Omega - \alpha$.

Definition 1 (Probability distribution)**.** A probability distribution P over $(\Omega, \mathcal{S})$ is a mapping of events in $\mathcal{S}$ to real values satisfying

- ⋄ $P(\alpha) \geq 0$ for all $\alpha \in \mathcal{S}$
- ⋄ $P(\Omega) = 1$
- ⋄ If $\alpha, \beta \in \mathcal{S}$ and $\alpha \cap \beta = \emptyset$, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$

A direct consequence of the properties is:

$$P(\emptyset) = 0$$
$$P(\alpha \cup \beta) = P(\alpha) + P(\beta) - P(\alpha \cap \beta)$$

Conditional probability answers the question - after learning that event α is true, how does our belief about β change? Formally, we define

$$P(\beta|\alpha) = \frac{P(\alpha \cap \beta)}{P(\alpha)} \tag{1}$$

It can be checked that this satisfies Definition 1 and is a probability distribution. Noting that $P(\alpha \cap \beta) = P(\alpha)P(\beta|\alpha)$, we define the chain rule of conditional probabilities

$$P(\alpha_1 \cap \cdots \cap \alpha_k) = P(\alpha_1)P(\alpha_2|\alpha_1) \cdots P(\alpha_k|\alpha_1 \cap \cdots \cap \alpha_{k-1}) \tag{2}$$

We further define the Bayes' rule

$$P(\alpha|\beta) = \frac{P(\beta|\alpha)P(\alpha)}{P(\beta)} \tag{3}$$

Here, $P(\alpha|\beta)$ is called the *posterior*, $P(\beta|\alpha)$ is the *likelihood*, $P(\alpha)$ is the *prior* and $P(\beta)$ is the *marginal probability* of the structure in context. We can generalize Equation 3 as

$$P(\alpha|\beta \cap \gamma) = \frac{P(\beta|\alpha \cap \gamma)P(\alpha|\gamma)}{P(\beta|\gamma)} \tag{4}$$

Now, we formally define the notion of random variables, which intuitively can be considered to be attribute reporters.

Definition 2 (Random Variable)**.** A random variable X is a *measurable* function $X : \Omega \to \mathcal{S}$. The probability that X takes values in a set $s \in \mathcal{S}$ is written as

$$\Pr(X \in s) = \Pr(\{\omega \in \Omega | X(\omega) \in s\}) \tag{5}$$

The marginal distribution over a random variable X is the distribution over events that can be described using X, and is denoted by $P(X)$. More generally, if we want to describe a distribution over a set of random variables $\mathcal{X} = \{x_1, \cdots, x_n\}$ called the *joint distribution* denoted as $P(x_1, \cdots, x_n)$. The full assignment to the variables is denoted as $\xi \in \text{Val}(\mathcal{X})$. The space corresponding to the joint assignment in $\mathcal{X}$ is called the *canonical outcome space*.

Now, we glance at independencies, a core component of Probabilistic Graphical Models.

Definition 3 (Independence)**.** An event α is independent of an event β denoted by $P \models (\alpha \perp\!\!\!\perp \beta)$, if $P(\alpha|\beta) = P(\alpha)$ or $P(\beta) = 0$.

Proposition 4. *A distribution satisfies* $(\alpha \perp\!\!\!\perp \beta)$ *if and only if*

$$P(\alpha \cap \beta) = P(\alpha)P(\beta) \tag{6}$$

Proof. Skipped (hint: Use the definition of conditional probability). □

Definition 5 (Conditional Indpendence)**.** An event α is conditionally independent of event β given γ in P, denoted by $P \models (\alpha \perp\!\!\!\perp \beta|\gamma)$ if $P(\alpha|\beta \cap \gamma) = P(\alpha|\gamma)$ or if $P(\beta \cap \gamma) = 0$.

Proposition 6. *P satisfies* $(\alpha \perp\!\!\!\perp \beta|\gamma)$ *if and only if*

$$P(\alpha \cap \beta|\gamma) = P(\alpha|\gamma)P(\beta|\gamma) \tag{7}$$

Definition 7. Let $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ be sets of random variables. $\mathbf{X}$ is conditionally independent of $\mathbf{Y}$ given $\mathbf{Z}$ in a distribution P if P satisfies $(\mathbf{X} = \mathbf{x} \perp\!\!\!\perp \mathbf{Y} = \mathbf{y}|\mathbf{Z} = \mathbf{z})$ for all values of $\mathbf{x} \in \text{Val}(\mathbf{X}), \mathbf{y} \in \text{Val}(\mathbf{Y})$ and $\mathbf{z} \in \text{Val}(\mathbf{Z})$. We say that the variables in $\mathbf{Z}$ are *observed*. If $\mathbf{Z}$ is empty, then we say that $\mathbf{X}$ and $\mathbf{Y}$ are marginally independent.

Proposition 8. *The distribution P satisfies* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z})$ *if and only if*

$$P(\mathbf{X}, \mathbf{Y}|\mathbf{Z}) = P(\mathbf{X}|\mathbf{Z})P(\mathbf{Y}|\mathbf{Z}) \tag{8}$$

The following properties hold for conditional independencies:

1. *Symmetry:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}) \implies (\mathbf{Y} \perp\!\!\!\perp \mathbf{X}|\mathbf{Z})$
2. *Decomposition:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W}|\mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z})$
3. *Weak Union:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W}|\mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}, \mathbf{W})$

Decomposition can also be stated as

$$\mathbf{X} \perp\!\!\!\perp \{\mathbf{Y}, \mathbf{Z}\} \implies \mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{X} \perp\!\!\!\perp \mathbf{Z}$$

Weak Union can also be stated as

$$\mathbf{X} \perp\!\!\!\perp \{\mathbf{Y}, \mathbf{Z}\} \implies \mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}$$

But note that, if $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}$ and $\mathbf{Z} \not\perp\!\!\!\perp \{\mathbf{X}, \mathbf{Y}\}$ then it is not necessary to have $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}$

4. *Contraction:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{W} | \mathbf{Z}, \mathbf{Y})$ & $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} | \mathbf{Z})$

If our distribution is positive (i.e, for all non-empty $\alpha \in \mathcal{S}, P(\alpha) > 0$), we have another property

Contraction can also be stated as

$$\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}, \mathbf{X} \perp\!\!\!\perp \mathbf{Z} \implies \mathbf{X} \perp\!\!\!\perp \{\mathbf{Y}, \mathbf{Z}\}$$

◇ *Intersection:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}, \mathbf{W})$ & $(\mathbf{X} \perp\!\!\!\perp \mathbf{W} | \mathbf{Z}, \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} | \mathbf{Z})$

Theorem 9. *Consider $\mathcal{X} = \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$. Then $P \models (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z})$ if and only if we can write*

$$P(\mathcal{X}) = \phi_1(\mathbf{X}, \mathbf{Z}) \phi_2(\mathbf{Y}, \mathbf{Z}) \tag{9}$$

Proof. Skipped. □

Probabilistic Graphical Models

Given a set of n random variables $\mathcal{X} = \{x_1, x_2, \cdots x_n\}$ where n is large, we want to build a joint probability distribution P over this set. Explicitly representing the joint distribution is computationally expensive, since just having binary values variables requires the joint distribution to specify $2^n - 1$ numbers, and for more practical variables, the count is too large.

We want to efficiently represent, estimate and answer inference queries on the distribution.

An example of a query can be - `Estimate the fraction of people with a bachelor's degree`.

Alternatives to explicit joint distributions

▷ Can we assume all columns are independent? **NO** - this is obviously a very bad assumption.
▷ Can we use data to detect highly correlated column pairs, and estimate their pairwise frequencies? **MAYBE** - but there might be too many correlated pairs, and the method is ad hoc.
To solve the above two not so good ways, we explore conditional independencies. It may be possible that `income` $\not\perp\!\!\!\perp$ `age` but `income` $\perp\!\!\!\perp$ `age`|`experience`.

Note that we write that a set X is conditionally independent of Y given Z, i.e $X \perp\!\!\!\perp Y | Z$ if

$$\Pr(X|Y, Z) = \Pr(X|Z)$$

Probabilistic graphical models use a graph-based representation as the basis for compactly encoding a complex distribution over a high-dimensional space.

It is convenient to represent the independence assumption using a graph. The so called graphical model has nodes as the variables (continuous or discrete), and the edges represent direct interaction. If we consider directed edges, we talk about Bayesian Networks, and if we consider undirected edges, we talk about Markov Random Fields.

Essentially the graphical model is a combination of the graph and potentials.

Definition 10 (Potentials)**.** Potentials $\psi_c(\mathbf{x}_c)$ are scores for assignment of values to subsets c of directly interacting variables. We factorize the probability as a product of these potentials, i.e

$$\Pr(\mathbf{x} = x_1, \cdots, x_n) \propto \prod \psi_s(\mathbf{x}_s) \tag{10}$$

Bayesian Networks

Bayesian Networks, also referred to as *directed graphical models* are a family of probability distributions that has a compact parameterization representable using a directed graph.

It is known that

$$\Pr(x_1, x_2, \cdots, x_n) = \Pr(x_1)\Pr(x_2|x_1)\Pr(x_3|x_2, x_1)\cdots\Pr(x_n|x_{n-1}, \cdots, x_1) \tag{11}$$

A compact Bayesian Network is a distribution in which each factor in the above equation depends on the *parent* variables represented by $\text{Pa}(x_i)$ for variable x_i. Thus, we have

$$\Pr(x_i|x_{i-1}, x_{i-2}, \cdots, x_1) = \Pr(x_i|\text{Pa}(x_i)) \tag{12}$$

and the corresponding potentials at each node in terms of its parents are

$$\psi_i(x_i, \text{Pa}(x_i)) = \Pr(x_i|\text{Pa}(x_i)) \tag{13}$$

Thus,

$$\Pr(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} \Pr(x_i|\text{Pa}(x_i)) \tag{14}$$

Consider the situation when each variable can take d values. The naive approach gives us $\mathcal{O}(d^n)$ parameters. If we think of the potentials as probability tables (with the rows corresponding to $\text{Pa}(x_i)$) and columns corresponding to the values of x_i, with entries as $\psi_i(x_i, \text{Pa}(x_i))$, we can notice that if $|\text{Pa}(x_i)| \leq k$, then the number of parameters are $\mathcal{O}(d^{k+1})$, and for n variables, we have $\mathcal{O}(nd^{k+1})$, which provides us the compact representation.

Definition

Now we formally define these -

Definition 12 (Bayesian Network)**.** A Bayesian Network is a directed graph $G = (V, E)$ together with

- ◇ a random variable x_i for each node $i \in V$
- ◇ a potential $\psi_i(x_i, \text{Pa}(x_i))$ for each node $i \in V$

For a variable x_i in our Bayesian Network $\mathcal{G}$, denote $\text{ND}(x_i)$ as the non-descendents of x_i. The following local conditional independencies hold in $\mathcal{G}$ -

$$x_i \perp\!\!\!\perp \text{ND}(x_i)|\text{Pa}(x_i) \tag{15}$$

Example 11 shows the independencies in a simple Bayesian Network.

Example 11.

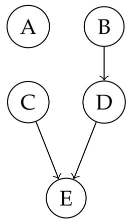

Figure 1: Sample BN

Consider the BN above. We will consider each variable at a time.

- ◇ A has no parent, and has no descendent. Thus,
$$A \perp\!\!\!\perp B, C, D, E$$
- ◇ B has no parent, but has D as a descendent. Thus,
$$B \perp\!\!\!\perp A, C$$
- ◇ C has no parent, but has E as a descendent. Thus,
$$C \perp\!\!\!\perp A, B, D$$
- ◇ D has B as a parent, and has E as the descendent. Thus,
$$D \perp\!\!\!\perp A, C|B$$
- ◇ E has C and D as parents, but has no descenent. Thus,
$$I \perp\!\!\!\perp A, B|C, D$$

Definition 13 (Factorization)**.** Let $\mathcal{G}$ be a Bayesian Network graph over the variables $\{X_i\}_{i=1}^{n}$. We say that a distribution P over the same space factorizes according to $\mathcal{G}$ if P can be expressed as a product described in Equation 14. Such factorization is also known as the chain rule for Bayesian Networks, and is denoted as Factorize$(P, \mathcal{G})$.

Definition 14. Let P be a distribution over $\mathcal{X}$. We define $\mathcal{I}(P)$ to be the set of independent assertions of the form $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}$ that hold in P.

We can now write "P satisfies the local independencies associated with $\mathcal{G}$" as $\mathcal{I}_\ell(\mathcal{G}) \subseteq \mathcal{I}(P)$.

Definition 15 (Independency-Map)**.** Let $\mathcal{K}$ be any graph object associated with a set of independencies $\mathcal{I}(\mathcal{K})$. We call $\mathcal{K}$ an I-map for a set of independencies $\mathcal{I}$ if $\mathcal{I}(\mathcal{K}) \subseteq \mathcal{I}$.

Thus for $\mathcal{G}$ to be an I-map for P, any independence that asserts in $\mathcal{G}$ must also assert in P, but P can have additional independencies not reflected in $\mathcal{G}$.

Remark 16 (Notation Alert). Note that we will use the following interchangeably - P satisfies the local conditional independencies satisfied by $\mathcal{G}$ and $\mathcal{G}$ is an I-map for P, i.e

$$\text{Local-CI}(P, \mathcal{G}) \equiv \mathcal{I}_\ell(\mathcal{G}) \subseteq \mathcal{I}(P) \tag{16}$$

Definition 17 (I-equivalence)**.** Two Bayesian Networks $\mathcal{G}_1$ and $\mathcal{G}_2$ are $\mathcal{I}$-equivalent, if the encode the same dependencies, i.e

$$\mathcal{I}(\mathcal{G}_1) = \mathcal{I}(\mathcal{G}_2) \tag{17}$$

Theorem 18. *If $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same skeleton, and the same v-structures (see D-separation), then they are $\mathcal{I}$-equivalent.*

Proof. Skipped. □

Theorem 19. *Given a distribution $P(x_1, x_2, \cdots, x_n)$ and a directed acyclic graph (DAG) $\mathcal{G}$,*

$$\textit{Local-CI}(P, \mathcal{G}) \iff \textit{Factorize}(P, \mathcal{G}) \tag{18}$$

Proof. ($\implies$) We essentially need to show that if $\mathcal{G}$ is an I-map for P, then P factorizes according to $\mathcal{G}$. Consider a topologically sorted order $x_1, x_2, \cdots, x_n$ in $\mathcal{G}$. Local-CI$(P, \mathcal{G})$ tells us that

$$\Pr(x_i | x_1, \cdots, x_{i-1}) = \Pr(x_i | \text{Pa}(x_i))$$

We can write

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, \cdots, x_{i-1})$$

Each term in the product can be simplified due to the notion of Local-CI stated above, and we reach Equation 14, proving factorization.

($\impliedby$) Proof has been skipped. □

Minimal Construction

Our goal is to construct a minimal and correct BN $\mathcal{G}$ to represent P. A DAG $\mathcal{G}$ is correct if all Local-CIs that are implied in $\mathcal{G}$ hold in P, and a DAG $\mathcal{G}$ is minimal if we cannot remove any edge(s) from $\mathcal{G}$ and still get a correct BN for P.

In the setting, we define our oracle $\mathscr{O}$ to whom we can ask any query of the type "`Is X` $\perp\!\!\!\perp$ `Y|Z?`" pertaining to P and get a boolean answer. We will query the oracle several times to build up our BN. The following algorithm constructs such a BN -

```
Variables: $x_1, x_2, \cdots, x_n \longleftarrow$ ordered variables in $\mathcal{X}$
Independencies: $\mathcal{I} \longleftarrow$ set of independencies
$\mathcal{G} \longleftarrow$ Empty graph over $\mathcal{X}$
for $i = 1$ to $n$ do
    $\mathbf{U} \longleftarrow \{x_1, \cdots, x_{i-1}\}$ // Set of candidate parents of $x_i$
    for $U' \subseteq \{x_1, \cdots, x_{i-1}\}$ do
        if $U' \subset U$ and $(x_i \perp\!\!\!\perp \{x_1, \cdots, x_{i-1}\} - U' | U') \in \mathcal{I}$ then
            $\mathbf{U} \longleftarrow U'$
        end
    end
    // Now we have the minimal set $\mathbf{U}$ satisfying
       $(x_i \perp\!\!\!\perp \{x_1, \cdots, x_{i-1}\} - \mathbf{U} | \mathbf{U})$
    // Now we set $\mathbf{U}$ to be the parents of $x_i$
    for $x_j \in \mathbf{U}$ do
        Add $x_j \to x_i$ in $\mathcal{G}$
    end
end
return $\mathcal{G}$
```

Algorithm 1: Minimal Bayesian Network Construction (I-Map)

We know sketch rough proofs for the claims of the algorithm.

Theorem 20. *The BN $\mathcal{G}$ constructed by algorithm 1 is minimal, i.e we cannot remove any edge from the BN while maintaining the correctness of the BN for P.*

Proof. By construction. A subset of $\text{ND}(x_i)$ were available when we chose parents of $\mathbf{U}$ minimally. □

Theorem 21. *$\mathcal{G}$ constructed by the above algorithm is correct, i.e, the local-CIs induced by $\mathcal{G}$ hold in P.*

Proof. The construction is such that Factorize$(P, \mathcal{G})$ holds everytime. Since Factorize$(P, \mathcal{G}) \implies$ Local-CI$(P, \mathcal{G})$, the constructed BN satisfies the local-CIs of P. □

Question 22 (Construction of BN)**.** *Draw a Bayesian network over five variables $x_1, \cdots, x_5$ assuming the variable order x_1, x_2, x_3, x_4, x_5. For this* *ordering, assume that the following set of local CIs hold in the distribution:*

$$x_1 \perp\!\!\!\perp x_2 \quad x_3 \perp\!\!\!\perp x_2 | x_1 \quad x_4 \perp\!\!\!\perp x_1, x_3 | x_2 \quad x_5 \perp\!\!\!\perp x_1, x_2 | x_3, x_4$$

Answer 23. Due to the ordering, we begin by inserting x_1 into the BN. Then we follow Algorithm 1 as follows:

1. x_2: Predecessor - x_1
   - *Query 1* - Is $x_2 \perp\!\!\!\perp x_1 | \emptyset$? : *Result 1* - True

   Thus, x_2 has no parents.
2. x_3: Predecessors - x_1, x_2
   - *Query 1* - Is $x_3 \perp\!\!\!\perp \{x_1, x_2\} | \emptyset$? : *Result 1* - False
   - *Query 2* - Is $x_3 \perp\!\!\!\perp x_2 | x_1$? : *Result 2* - True

   Thus, x_3 has x_1 as a parent, and x_2 as a non-descendent.
3. x_4: Predecessors - x_1, x_2, x_3
   - *Query 1* - Is $x_4 \perp\!\!\!\perp \{x_1, x_2, x_3\} | \emptyset$? : *Result 1* - False
   - *Query 2* - Is $x_4 \perp\!\!\!\perp \{x_2, x_3\} | x_1$? : *Result 2* - False
   - *Query 3* - Is $x_4 \perp\!\!\!\perp \{x_1, x_3\} | x_2$? : *Result 3* - True

   Thus, x_4 has x_2 as a parent, and x_1, x_3 as non-descendents.
4. x_5: Predecessors - x_1, x_2, x_3, x_4
   Check that it has x_3, x_4 as parents and x_1, x_2 as non-descendents.

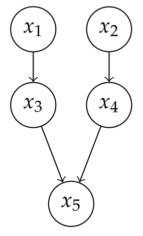

Figure 2: BN Example

Thus, finally we get the BN as in Figure 2

Remark 24 (Importance of ordering). It is possible that a different ordering in $\mathcal{X}$ gives rise to a different BN, which although may be minimal, but may not be *optimal*. A minimal BN is defined for a given ordering, while an optimal BN is defined over all orderings. Example 25 shows such a case.

Example 25. Consider the BN shown in Figure 3, with the ordering x_1, x_2, x_3. We have $x_1 \perp\!\!\!\perp x_2$ as the only CI holding. Now consider our order changes to x_3, x_2, x_1. We follow the procedure as before, first inserting x_3 into the BN. Now

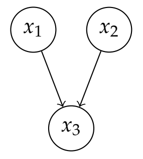

Figure 3: BN Ordering (optimal)

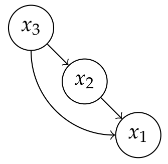

Figure 4: BN Ordering (non-optimal)

1. x_2: Predecessor - x_3
   - *Query 1* - Is $x_2 \perp\!\!\!\perp x_3 | \emptyset$? : *Result 1* - False

   Thus, we have x_3 as a parent of x_2.
2. x_1: Predecessor - x_3, x_2

- *Query 1* - Is $x_1 \perp\!\!\!\perp \{x_2, x_3\} | \emptyset$? : *Result 1* - False
- *Query 2* - Is $x_1 \perp\!\!\!\perp x_2 | x_3$? : *Result 2* - False
- *Query 3* - Is $x_1 \perp\!\!\!\perp x_3 | x_2$? : *Result 3* - False

Thus, we have both x_2 and x_3 as parents of x_1.

Thus we get the BN as in Figure 4, and see that the BN is minimal, but not optimal.

D-Separation

Our goal is to know when we can guarantee $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}$ holds given a BN $\mathcal{G}$. The further discussion provides some cases where we can guarantee $\mathbf{X} \not\perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}$.

1. **Direct Connection:** If there is an edge $X \rightarrow Y$, then regardless of any $\mathbf{Z}$, we can find examples where they influence each other.
2. **Indirect Connection:** This means that there is a trail between the nodes in the graph. We consider the simple case when we a 3-node graph and Z is between X and Y. Consider the 4 diagrams to the left for reference.

   (a) *Indirect causal effect:* X cannot influence Y via Z if Z is observed.

   (b) *Indirect evidential effect:* This is similar to the previous case as dependence is a symmetric notion. Thus, X can influence Y via Z, only if Z is not observed.

   (c) *Common cause:* The conclusion is similar to (a) and (b).

   (d) *Common effect:* (v-structure) This case is a bit tricky to understand, but the crux is that X can influence Y when either Z or one of Z's descendents is observed.

   If we have flow of influence from X to Y via Z, we say that the trail $X \rightleftharpoons Y \rightleftharpoons Z$ is active.

$$\left.\begin{array}{r} \text{Causal trail: } X \rightarrow Z \rightarrow Y \\ \text{Evidential trail: } Y \rightarrow Z \rightarrow X \\ \text{Common cause: } X \leftarrow Z \rightarrow Y \end{array}\right\} \text{Active if and only if } Z \text{ is observed}$$
$$\star \;\; \text{Common effect: } X \rightarrow Z \leftarrow Y \Big\}$$
$$\hookrightarrow \text{Active if and only if } Z \text{ or one of } Z\text{'s descendent is observed} \tag{19}$$

Now, we can create a general notion of trails -

Definition 26. Let $\mathcal{G}$ be a BN, and $x_1 \rightleftharpoons \cdots \rightleftharpoons x_n$ be a trail in $\mathcal{G}$. Let $\mathbf{Z} \subset \{\text{observed variables}\}$. The trail is active given $\mathbf{Z}$ if

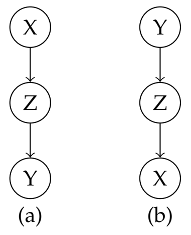
Figure 5: Causal and evidential effect

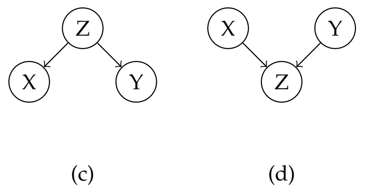
Figure 6: Common cause and common effect

- ⋄ Whenever we have a v-structure $x_{i-1} \rightarrow x_i \leftarrow x_{i+1}$, then x_i or one of its descendents are in $\mathbf{Z}$
- ⋄ No other node along the trail is in $\mathbf{Z}$.

We can see that if $x_1 \in \mathbf{Z}$ or $x_n \in \mathbf{Z}$, then the trail is inactive.

Essentially in a DAG, $\mathbf{Z}$ d-separates $\mathbf{X}$ from $\mathbf{Y}$ if all paths $\mathcal{P}$ from any $\mathbf{X}$ to $\mathbf{Y}$ is blocked by $\mathbf{Z}$.

A path $\mathfrak{P}$ is *blocked* if it is inactive, i.e there is no flow of influence.

Definition 27 (d-separation)**.** Let $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ be three sets of nodes in $\mathcal{G}$. We say that $\mathbf{X}$ and $\mathbf{Y}$ are d-separated given $\mathbf{Z}$, i.e d-sep$_{\mathcal{G}}(\mathbf{X}; \mathbf{Y}|\mathbf{Z})$ if there is no active trail between any node $x \in \mathbf{X}$ and $y \in \mathbf{Y}$ given $\mathbf{Z}$.

We use the same notation as $\mathcal{I}(P)$ as we can show that the independencies in $\mathcal{I}(\mathcal{G})$ are those guaranteed to hold for every distribution over $\mathcal{G}$ (Theorem 29).

Definition 28 (Global Markov independencies)**.** The set

$$\mathcal{I}(\mathcal{G}) \overset{\text{def}}{=} \{(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}) : \text{d-sep}_{\mathcal{G}}(\mathbf{X}; \mathbf{Y}|\mathbf{Z})\} \tag{20}$$

denoting the set of independencies corresponding to d-separation is the set of global Markov independencies.

Theorem 29. *The d-separation test identifies the complete set of conditional independencies that hold in all distributions that conform to a given Bayesian Network.*

Proof. Skipped. □

Now, we look at another way to check d-separation over BNs, but first we define some terms.

Definition 30 (Ancestral Graph)**.** Given a graph $G = (V, E)$ and a set of nodes to focus on, say $V^* \subseteq V$, the ancestral graph G^A is a subgraph induced by $V^A = V^* \cup \mathcal{A}(V^*)$ where $\mathcal{A}(V^*)$ denotes the ancestors of V^*. Thus,

$$G^A = G\langle V^A \rangle = (V^A, \{(u,v)|(u,v) \in E \text{ and } u, v \in V^A\}) \tag{21}$$

Definition 31 (Markov Blanket)**.** Given a random variable Y in a random variable set $\mathcal{X} = X_1, X_2, \cdots, X_n$, it's Markov Blanket is any subset $\mathcal{S}$ of $\mathcal{X}$, conditioned on which other variables are independent with Y, i.e

$$Y \perp\!\!\!\perp \mathcal{X} \setminus \mathcal{S}|\mathcal{S} \tag{22}$$

Thus, we can infer Y from $\mathcal{S}$ itself, and the rest of the elements are redundant in observation.

Remark 32. Essentially we are finding an equivalent undirected graph for a DAG. We find all pairs of non-adjacent nodes having a common child, and add an undirected edge between them. Then we transform all directed edges in the resulting graph to undirected edges.

Definition 33 (Moral graph)**.** A moral graph of a directed acyclic graph G is an undirected graph in which each node of the original G is now connected to its *Markov Blanket*.

```
Given: Bayesian Network $\mathcal{G}$, Condition to check $\mathcal{C}$: $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}$
$\mathcal{C} \leftarrow$ False
$G = (V, E) \leftarrow$ Underlying DAG in $\mathcal{G}$.
$G^A = (V^A, E^A) \leftarrow$ Ancestral graph of $G$
$G^A_M = (V^A_M, E^A_M) \leftarrow$ Moral graph of $G^A$ using Note 32
// Delete the nodes in Z and all its connections
for $z \in \mathbf{Z}$ do
    $\Xi \leftarrow \{\}$
    for $u \in V$ such that $\xi = (u, z) \in E^A_M$ do
        $\Xi \leftarrow \Xi \cup \xi$
    end
    $E^A_M \leftarrow E^A_M \setminus \Xi$
    $V^A_M \leftarrow V^A_M \setminus \{z\}$
end
if X and Y are disconnected in the resulting graph then
    $\mathcal{C} \leftarrow$ True
end
```

Algorithm 2: Checking for independence in a BN

Definition 34 (Perfect map)**.** A graph $\mathcal{G}$ is a perfect map (P-map) for a set of independencies $\mathcal{I}$ if $\mathcal{I}(\mathcal{G}) = \mathcal{I}$. Also, $\mathcal{G}$ is P-map for a distribution P if $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{P})$.

Limitations

Consider the following set of CIs

$$x \perp\!\!\!\perp y \quad y \perp\!\!\!\perp z \quad z \perp\!\!\!\perp x \quad x \not\perp\!\!\!\perp \{y, z\} \tag{23}$$

If you try to draw the BN for any ordering of the variables $\{x, y, z\}$, you can check that you get extraneous edges, and we fail to capture at least one of the conditions above. Thus, a symmetric dependency of this sort is not possible to be represented by Bayesian Networks. This, gives rise to a different field of graphical modeling using *Markov Networks*, which can represent some of these situations.

Markov Random Fields

Intuition

We saw previously that we cannot draw a *perfect* I-map such that $\mathcal{I}(\mathcal{G}) = \mathcal{I}(P)$ for any distribution P using directed graphical models. Such too is the case with undirected graphical models, but they help us to represent some of these independencies which directed graphs couldn't.

To be added.

Cliques

Definition 35 (Complete Graph)**.** A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

Definition 36 (Clique)**.** A clique C in an undirected graph $G = (V, E)$ is a subset of vertices, $C \subseteq V$ such that every two distinct vertices are adjacent. Thus, the subgraph induced by C, i.e $G\langle C \rangle$, is a complete graph.

Definition 37 (Maximal Clique)**.** A clique that cannot be extended by including one more adjacent vertex (i.e it does not exist exclusively within the vertex set of a larger clique) is a maximal clique.

Definition 38 (Maximum Clique)**.** A maximum clique of a graph G, is a clique such that there is no other clique with more vertices.

With each clique C, we associate a potential function ψ, which is a provisional function of its arguments that assigns a pre-probabilistic score of their joint distribution. It is to note that ψ must be non-negative, but it shouldn't be interpreted as probability.

Gibbs Fields

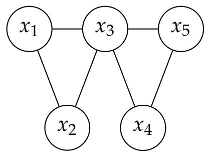

Figure 7: A Gibbs Field

A Gibbs Field is a representation of a set of random variables and their relationships. An example is in Figure 7. In this, the edges are undirected and imply some correlation between the connected nodes.

Consider clique potentials as $\psi_i(c_i)$. Then the joint probability for any set of random variables $\mathcal{X} = \{x_1, \cdots, x_n\}$ represented by a Gibbs Field can be written as a product of clique potentials

$$P(\mathcal{X}) = \frac{1}{Z} \prod_{c_i \in C} \psi_i(c_i) \tag{24}$$

Z is a normalizing constant required to create a valid probability distribution, i.e

$$Z = \sum_{x} \prod_{c_i \in C} \psi_i(C_i) \tag{25}$$

Remark 39. For any Gibbs Field, there is a subset $\hat{C}$ of C consisting of only maximal cliques, which are not proper subsets of any other cliques. We write the potentials for these maximal cliques as products of all potentials of their sub-cliques, and thus state the joint probability as

$$P(\mathcal{X}) = \frac{1}{Z} \prod_{c_i \in \hat{C}} \hat{\psi}_i(c_i) \tag{26}$$

Formal Definition

Definition 40 (Markov Random Field)**.** A Markov Random Field (MRF) is a probability distribution P over variables $x_1, \cdots, x_n$ defined by an undirected graph G in which nodes correspond to variables x_i and has the form

$$P(x_1, x_2, \cdots, x_n) = \frac{1}{Z} \prod_{c \in C} \psi_c(x_c) \tag{27}$$

where

$$Z = \sum_{x_1, \cdots, x_n} \prod_{c \in C} \psi_c(x_c) \tag{28}$$

is the ***partition function*** which is the normalizing constant ensuring the distribution sums to 1.

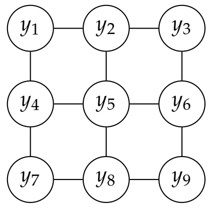

Figure 8: Relations in image pixels

As we saw earlier, if we have symmetric interactions, then UGMs become useful (such as labeling pixels in an image - see Figure 8). Define $y_i = 1$ if the pixel is a part of the foreground, and 0 else. Taking cliques of size 1, we have the potential functions $\psi_1(0)$ to $\psi_9(0)$ and $\psi_1(1)$ to $\psi_9(1)$. Now considering cliques of size 2, we have $\psi(0,0), \psi(0,1), \psi(1,0)$ and $\psi(1,1)$. Thus we write

$$\Pr(y_1, \cdots, y_9) \propto \prod_{k=1}^{9} \psi_k(y_k) \prod_{(i,j) \in E(G)} \psi(y_i, y_j) \tag{29}$$

Conditional Independencies

From now on, we will work on the UGM in Figure 8. Let

$$V = \{y_1, \cdots, y_9\}$$

. We define three types of CIs in UGMs as follows

In a graph G with vertices $V = \{x_1, \cdots, x_n\}$, $\mathcal{N}(x_i)$ denotes the neighbors of x_i in the graph

1. **Local CI:** $y_i \perp\!\!\!\perp V - \mathcal{N}(y_i) - \{y_i\} | \mathcal{N}(y_i)$

$$y_1 \perp\!\!\!\perp y_3, y_5, y_6, y_7, y_8, y_9 | y_2, y_4$$

2. **Pairwise CI:** $y_i \perp\!\!\!\perp y_j | V - \{y_i, y_j\}$ if $(y_i, y_j) \notin E(G)$

$$y_1 \perp\!\!\!\perp y_3 | y_2, y_4, y_5, y_6, y_7, y_8, y_9$$

3. **Global CI: $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} | \mathbf{Z}$** if **Z** separates **X** and **Y** in the graph

$$y_1, y_2, y_3 \perp\!\!\!\perp y_7, y_8, y_9 | y_4, y_5, y_6$$

Checking for CI in MRFs is much more easier than BNs. The way to check is through graph separability. Consider the example given in **Global CI**. If we remove y_4, y_5 and y_6 from the graph along with their edges, we see that the components y_1, y_2, y_3 is disconnected from y_7, y_8, y_9, and hence the CI holds.

Theorem 41. *Let G be an undirected graph of $V = \{x_1, \cdots, x_n\}$ nodes, and let $P(x_1, \cdots, x_9)$ be a distribution. If P is represented by G, that is, if it can be factorized as per the cliques of G, then P will also satisfy the global-CIs of G. Thus*

$$\textit{Factorize}(P, G) \implies \textit{Global-CI}(P, G) \tag{30}$$

Note that for any arbitrary distribution, the converse doesn't hold, i.e in general for a distribution P

$$\text{Factorize}(P, G) \not\Longleftarrow \text{Global-CI}(P, G) \tag{31}$$

We see this through a counter example. Consider the UGM in Figure 9, for the probability distribution $P(x_1, x_2, x_3, x_4)$ such that $P(x_1, x_2, x_3, x_4) = \frac{1}{8}$ when x_1, x_2, x_3, x_4 can take values from $\{0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001\}$ else 0. It can be manually checked that all 4 Global-CIs hold in the graph, for example $x_1 \perp\!\!\!\perp x_3 | x_2, x_4$. Now consider the factors in the edges as $\psi(x_i, x_j)$. These will be positive, but that cannot represent the probability for $x_1, x_2, x_3, x_4 = 0101$.

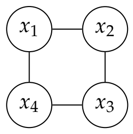

Figure 9: Sample UGM

Also, it is trivial to see that

$$\text{Global-CI} \implies \text{Local-CI} \tag{32}$$

But again through a counter example, we will show that the converse doesn't hold, i.e

$$\text{Local-CI} \not\Longleftarrow \text{Global-CI} \tag{33}$$

Consider a distribution over 5 binary variables $P(x_1, \cdots, x_5)$ where $x_1 = x_2, x_4 = x_5$ and $x_3 = x_2 \wedge x_4$. Consider G as in Figure 10. Notice that all 5 Local-CIs hold in the graph, for example $x_1 \perp\!\!\!\perp \{x_3, x_4, x_5\} | x_2$. But notice that the graph also tells us that $x_2 \perp\!\!\!\perp x_4 | x_3$, but this is not present in the distribution P.

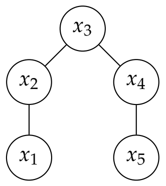

Figure 10: Sample UGM

We also notice that

$$\text{Local-CI} \implies \text{Pairwise-CI} \tag{34}$$

But again through a counter example, we show that the converse doesn't hold, i.e

$$\text{Pairwise-CI} \not\Longleftarrow \text{Local-CI} \tag{35}$$

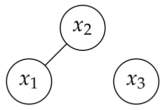

Figure 11: Sample UGM

Consider $P(x_1, x_2, x_3)$ defined over 3 binary variables such that $P(x_1, x_2, x_3) = \frac{1}{2}$ if $x_1 = x_2 = x_3$ and 0 else. Let G be as in Figure 11. See that both the Pairwise-CIs, i.e $x_1 \perp\!\!\!\perp x_3 | x_2$ and $x_2 \perp\!\!\!\perp x_3 | x_1$ hold in the graph, but the local CI $x_1 \perp\!\!\!\perp x_3$ doesn't hold.

We have made a lot of statements about converses not holding in arbitrary distributions, but the natural question to arise is, can we find distributions where all the relations hold? The answer is yes, and is shown by the following theorem, also called the *fundamental theorem of random fields* -

Theorem 42 (Hammerseley Clifford Theorem)**.** *If a positive distribution $P(x_1, \cdots x_n)$ confirms to the Pairwise-CIs of a UDGM G, then it can be factorized as per the cliques C of G as*

A distribution $P(\mathbf{x})$ is positive, if $P(\mathbf{x}) > 0 \; \forall \; \mathbf{x}$.

$$P(x_1, \cdots, x_n) \propto \prod_{C \in G} \psi_C(\mathbf{y}_C) \tag{36}$$

Proof. Skipped. □

Thus, in summary, for any arbitrary distribution P and UGM H,

$$\begin{aligned} &\text{Factorize}(P, H) \implies \text{Global-CI}(P, H) \implies \\ &\text{Local-CI}(P, H) \implies \text{Pairwise-CI}(P, H) \end{aligned} \tag{37}$$

and if P is positive, then

$$\text{Pairwise-CI}(P, H) \implies \text{Factorize}(P, H) \tag{38}$$

Hence, for a positive distribution, all three types of CIs are *equivalent*.

Minimal Construction

The question to answer is, given a positive distribution $P(x_1, \cdots, x_n)$ as an oracle $\mathcal{O}$ to which we can ask the query - is $X \perp\!\!\!\perp Y | Z$ and get a boolean answer, we need to draw a minimal and correct UGM G to represent P.

Denote $V = \{x_1, x_2, \cdots, x_n\}$ as the set of all variables. We see that there are two methods to draw the UGM -

1. *Using Pairwise-CIs:* For each pair of vertices (x_i, x_j), if $x_i \perp\!\!\!\perp x_j | V - \{x_i, x_j\}$ in P, add an edge between x_i and x_j in G.

2. *Using Local-CIs:* For each vector x_i, find the smallest subset U such that $x_i \perp\!\!\!\perp V - U - \{x_i\} | U$ in P. Then, add U to $\mathcal{N}(x_i)$ in P.

Example 43. To be added.

We had seen Markov Blankets before, but we re-define them in terms of a UGM.

Definition 44 (Markov Blanket)**.** The Markov Blanket (MB) of a variable x_i is the smallest subset of variables V that makes x_i conditionally independent of others given the MB, i.e

$$x_i \perp\!\!\!\perp V - MB(x_i) - \{x_i\} | MB(x_i) \tag{39}$$

Theorem 45. *The MB of a variable is always unique for a positive distribution.*

Proof. We will prove the following by contradiction. Let $x_i \in V$ and M_1, M_2 be two MBs. Let $\alpha = M_1 - M_2$ and $\beta = M_2 - M_1$, $M = M_1 \cap M_2, W = V - (M_1 \cup M_2)$. Note that, by definition, $x_i \perp\!\!\!\perp V - M_2 | M_2$ and $x_i \perp\!\!\!\perp V - M_1 | M_1$. Using this, we can write

$$x_i \perp\!\!\!\perp W, \alpha | M, \beta \quad x_i \perp\!\!\!\perp W, \beta | M, \alpha$$

For positive distributions, using intersection property, we can write

$$x_i \perp\!\!\!\perp W, \alpha, \beta | M$$

This implies that M is also a MB, but that is a contradiction since M_1 and M_2 were supposed to be minimal. Hence, the MB is unique. □

Definition 46 (Immorality)**.** In a directed acyclic graph, the structure of the form $x \rightarrow y \leftarrow z$ is an immorality provided there is no edge between x and z.

Interesetingly, UGMs were initially used to model interactions of atoms in gases and solids in 1800. A few other places where they are used are in

1. Markov Random Fields - Image Segmentation
2. Conditional Random Fields - Information Extraction
3. Social Networks
4. Bio-informatics - Annotating active sites in proteins

With this, we can restate the equivalence of BNs -
*Two BNs $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent **iff** they have the same skeleton structure and the same set of immoralities.*

Conversion to and from Bayesian Networks

Theorem 47. *In a Bayesian Network $\mathcal{G}$, the Markov Blanket of a variable x_i is given as*

$$MB(x_i) = Pa(x_i) \cup Ch(x_i) \cup Sp(x_i) \tag{40}$$

where $Pa(x_i)$, $Ch(x_i)$ and $Sp(x_i)$ denote the parents, children and spouses (unmarried shared parent) of the children of x_i (if exists).

Proof. Only a flavor of the proof is provided. We have seen moralization of the Bayesian Network $\mathcal{G}$, and when we get $\mathcal{G}^M$ (i.e the moralized graph), notice that removing the parents, children and spouses disconnects the node from the graph. □

For example, in Figure 12, the MB of x_2 is given as

$$MB(x_2) = \{x_4\} \cup \{x_1\} \cup \{x_3\}$$

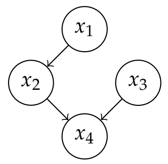

Figure 12: Sample BN

Theorem 48. *A Bayesian Network will have a perfect MRF if it has no immoralities.*

Proof. Skipped. □

We can ask the reverse question too. What condition should be posed on the MRF to have a perfect BN?

Definition 49 (Chordal Graph)**.** A cordal graph is a simple graph in which every graph cycle of length four or greater has a cycle chord.

With this, we can state that

Theorem 50. *An MRF can be perfectly converted to a BN if and only if it is chordal.*

Proof. Skipped. □

Inference Queries

We have seen two major types of compact representations of joint probability distributions in terms of graphs. Summarizing the expressions of the joint distribution, we can write

$$\text{For UGM: } \Pr(x_1, \cdots, x_n) = \frac{1}{Z} \prod_C \psi_C(x_C) \tag{41}$$

$$\text{For DGM: } \Pr(x_1, \cdots, x_n) = \prod_i \Pr(x_i | \text{Pa}(x_i)) \tag{42}$$

We get a very compact representation if $\text{Pa}(x_i)$ is small.

Given a probability distribution P, we can ask two major types of queries -

1. *Marginal probability queries over a sm'all subset of variables:* Given P, what is the marginal probability of x_1?.

$$\begin{aligned} \Pr(x_1) &= \sum_{x_2, \cdots, x_n} \Pr(x_1, \cdots, x_n) \\ &= \sum_{x_2=1}^{m} \cdots \sum_{x_n=1}^{m} \Pr(x_1, \cdots, x_n) \end{aligned} \tag{43}$$

   We can see that if each variable takes m values, then the brute-force computation of the marginal probability will take $\mathcal{O}(m^{n-1})$ time.

2. *Most likely labels of remaining variables (MAP queries):* Here, we ask questions of the form,

$$\mathbf{x}^* = \arg\max_{x_1, \cdots, x_n} \Pr(x_1, \cdots, x_n) \tag{44}$$

   An example of such a query could be - find the most likely entity labels of all words in a sentence.

Example 51 (Exact Inference)**.** Say we have a probability distribution over three binary variables as

$$P(x_1, x_2, x_3) = \frac{1}{Z} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3)$$

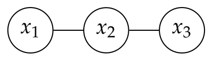

Figure 13: UGM for $P(x_1, x_2, x_3)$

The UGM for this is shown in Figure 13. Say we have the potential tables (each entry being $\psi_{ij}(a, b)$ representing the potential) as

		x_2	
		0	1
x_1	0	5	2
	1	1	4

		x_3	
		0	1
x_2	0	2	10
	1	5	3

For example, we see that $\psi_{12}(0, 0) = 5$. Let us find $P(x_1)$.

$$P(x_1) = \frac{1}{Z} \sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3)$$

We multiply the above two tables to get an intermediate potential distribution $\psi_{123}(x_1, x_2, x_3)$ and get a three dimensional table as follows (note that the columns denote x_2 and the rows denote x_1)

$x_3 = 0$

	0	1
0	10	10
1	2	20

$x_3 = 1$

	0	1
0	50	6
1	10	12

For example, $\psi_{12}(0,0)\psi_{23}(0,0) = 2 \times 5 = 10$. The next computation is to sum over x_3.

$$P(x_1) = \frac{1}{Z} \sum_{x_2 \in \{0,1\}} \psi^*_{12}(x_1, x_2)$$

The table after sum denoting $\psi^*_{12}(x_1, x_2)$ is

		x_2	
		0	1
x_1	0	60	16
	1	12	32

Now we eliminate x_2 by summing over the row values, thus finally

$$\psi^*_1(x_1) = \frac{1}{Z} \begin{bmatrix} 76 \\ 44 \end{bmatrix}$$

Since this $P(x_1) = \psi^*_1(x_1)$, we immediately get to know that $Z = 76 + 44 = 120$.

Clearly, we see through the example that the calculation, even for three variables is cumbersome. Image doing this for thousands!

From the table in the above example, we can also calculate the assignment which gives the maximum probability. Note the ψ_{123} table made, and see that $x_1 = 0, x_2 = 0, x_3 = 1$ has the score of 50 giving the highest probability. But let us write this in a more algorithmic way

$$\mathbf{x}^* = \arg\max_{x_2} \arg\max_{x_2} \arg\max_{x_3} \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$$

Let us construct the table $\psi^{\max}_{12}(x_1, x_2)$ from the ψ_{123} table

		x_2	
		0	1
x_1	0	50 for $x_3 = 1$	10 for $x_3 = 0$
	1	10 for $x_3 = 0$	20 for $x_3 = 0$

Similarly, $\psi^{\max}_1(x_1)$ will be

$$\begin{bmatrix} 50 \text{ for } x_2 = 0, x_3 = 1 \\ 20 \text{ for } x_2 = 1, x_3 = 0 \end{bmatrix}$$

At last, we can do an argmax over x_1 to get the assignment $x_1 = 0, x_2 = 0, x_3 = 1$ for the score of 50.

Clearly, after the example, it is clear that we want to avoid the exponential overhead that brute-force approach applies.

Exact Inference on Chains

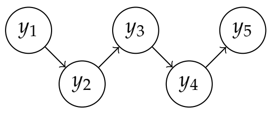

Figure 14: Chain graph

Consider the chain show in Figure 14.

We see that in the graph we would have potentials of the form $\psi_i(y_i, y_{i+1})$, and

$$\Pr(y_1, \cdots, y_n) = \prod_i \psi_i(y_i, y_{i+1}) \tag{45}$$

Note: Since we don't have immoralities, the MRF is equivalent to the undirected version of the graph. Say we want to calculate

$$\Pr(y_5 = 1) = \sum_{y_1, \cdots, y_4} \Pr(y_1, y_2, y_3, y_4, 1) \tag{46}$$

The key idea to reducing computations is to push summations past the multiplications, i.e

$$\begin{aligned}
\Pr(y_5 = 1) &= \sum_{y_1, \cdots, y_4} \Pr(y_1, y_2, y_3, y_4, 1) \\
&= \sum_{y_1}\sum_{y_2}\sum_{y_3}\sum_{y_4} \psi_1(y_1, y_2)\psi_2(y_2, y_3)\psi_3(y_3, y_4)\psi_4(y_4, 1) \\
&= \sum_{y_1}\sum_{y_2} \psi_1(y_1, y_2) \sum_{y_3} \psi_2(y_2, y_3) \sum_{y_4} \psi_3(y_3, y_4)\psi_4(y_4, 1) \\
&= \sum_{y_1}\sum_{y_2} \psi_1(y_1, y_2) \sum_{y_3} \psi_2(y_2, y_3)\mathcal{B}_3(y_3) \\
&= \sum_{y_1}\sum_{y_2} \psi_1(y_1, y_2)\mathcal{B}_2(y_2) \\
&= \sum_{y_1} \mathcal{B}_1(y_1)
\end{aligned} \tag{47}$$

We denote $\mathcal{B}_i(y_i)$ as the *belief* which flows from node $i+1$ to i. This is an efficient computation. In general, if we have a chain with n variables and each can take m values, the above algorithm (breaking into beliefs) takes time in order of $\mathcal{O}(nm^2)$.

Notice that we did the efficient computation for chains, the natural question is, for what other graphs can this be done?

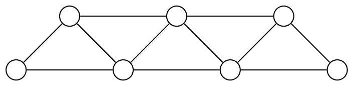
Figure 15: Triangular graph

Another one is shown in Figure 15. We define potential over each triangle (say ψ_{123}). If we follow a similar idea as the algorithm above, the time required for this computation will be $\mathcal{O}(nm^3)$.

Hardness of Inference and 3-SAT

The above discussion might lead to the thought that any graph G which can be factorized into small clique sizes might have an efficient computation method (i.e polynomial time) of calculating the marginal probability.

The answer sadly is no, and a counter example is the grid graph shown in Figure 16.

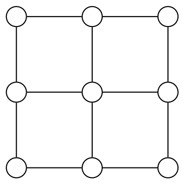
Figure 16: Grid graph

We will now reduce the 3-SAT to inference in Bayesian Networks.

Definition 52 (3-SAT Problem)**.** Given n boolean variables $x_1, \cdots x_n$ such that $x_i \in \{T, F\}$. We define a literal ℓ to be the variable x_i or its negation $\neg x_i$ or $\bar{x}_i$. Given a set of K clauses $C_1, C_2, \cdots, C_K$ with each clause being

$$C_j = \ell_{j_1} \vee \ell_{j_2} \vee \ell_{j_3} \tag{48}$$

The 3-SAT problem is to decide if there exists an assignment of values to the n variables such that

$$C_1 \wedge C_2 \wedge \cdots \wedge C_K = T \tag{49}$$

Example 53. Consider $n = 4, K = 3$ and

$$\begin{aligned} C_1 &= x_1 \vee \bar{x}_2 \vee \bar{x}_3 \\ C_2 &= x_2 \vee x_3 \vee \bar{x}_4 \\ C_3 &= x_4 \vee \bar{x}_1 \vee \bar{x}_2 \end{aligned}$$

In this case, having all $x_i = T$ for $i = \{1, 2, 3, 4\}$ solves the problem.

In the above example, we by chance got lucky and solved the problem, but in general for a large number of variables, it is not possible to go over all possible combinations of values, since it requires an exponential amount of time.

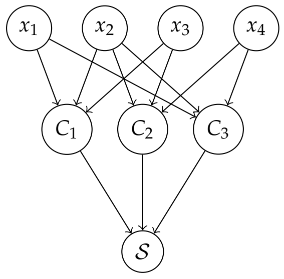

Figure 17: 3-SAT as BN

Now we represent 3-SAT as a Bayesian Network.

Let us do that in a *layer* sense. Let the first layer have all the variables as nodes and the next layer have all the clauses. Each clause will have 3 parents due to Equation 48. Finally, the third layer would have $\mathcal{S}$, which is the satisfiability (Equation 49), and it's parents would be all the clauses. Figure 17 shows the BN of Example 53.

Coming back to the general setting, for each variable x_i, we denote

$$\Pr(x_i) = \begin{cases} \frac{1}{2} & x_i = F \\ \frac{1}{2} & x_i = T \end{cases} \tag{50}$$

We also need to define $\Pr(C_j | \ell_{j_1}, \ell_{j_2}, \ell_{j_3})$. To do this, we assign a non-zero probability to only those which make $C_j = T$. This can be done uniformly (say out of the 8 assignments, 5 give a non-zero value, then 1 for each of those assignments, and 0 to rest - this is done because each C_j is a deterministic function of the literals). Finally, we write the last probability $\Pr(\mathcal{S} | C_1, \cdots, C_K)$ as 1 if $C_1, \cdots, C_K = T$, i.e all are true, and in the rest of the cases, we assign it as zero (note the difference here - the table for each C_i had 8 rows, and the table for $\mathcal{S}$ has 2^K rows). The 2^K shows that it is not polynomial. This is again, not efficient.

One small change we can do is that instead of having a single $\mathcal{S}$ in the last layer, have $K-1$, such that each $\mathcal{S}_i$ is connected to C_{i-1} and C_i as parents, and each $\mathcal{S}_i$ is a parent of $\mathcal{S}_{i+1}$. This allows us to create the probability table as $\Pr(\mathcal{S}_j|\mathcal{S}_{j-1}, C_{j-1}, C_j)$ which represents the logic

$$\mathcal{S}_j = \mathcal{S}_{j-1} \wedge C_{j-1} \wedge C_j \tag{51}$$

This allows each $\mathcal{S}_j$ with 8 variables, bringing in the needed efficiency. More specifically, the space required now is polynomial, since each S_j requires only 2^4 space, each C_j requires 2^5 space and each x_j requires just constant (2) space. Thus overall the space required is $\mathcal{O}((K-1) \cdot 2^4 + K \cdot 2^5 + 2)$

Finally, if we can answer $\Pr(\mathcal{S}_j = 1) > 0$ positively, then we know that a 3-SAT assignment exists, else it does not.

Variable Elimination on General Graphs

We saw that using brute-force (i.e an exponential number of operations), we could calculate the normalizer Z. This is impractical, and hence we need a more efficient way to do so. Let's define the problem again -

Given an arbitrary set of potentials $\psi_C(x_C)$ in a graph G where C are the cliques in G, we need to find

$$Z = \sum_{x_1, \cdots, x_n} \prod_C \psi_C(x_C)$$

The algorithm to do so is as follows:

```
1 Input: Graph $G$
2 Variables: $x_1, x_2, \cdots, x_n$ present in a good ordering
3 $\mathcal{F} \longleftarrow \{\psi_C(x_C)$ where $C$ = cliques in $G\}$
4 for $i = 1$ to $n$ do
5     $\mathcal{F}_i \longleftarrow$ factors in $\mathcal{F}$ containing $x_i$
6     $\mathcal{M}_i \longleftarrow$ product of factors in $\mathcal{F}_i$
7     $m_i \longleftarrow \sum_{x_i} \mathcal{M}_i$
8     $\mathcal{F} \longleftarrow (\mathcal{F} - \mathcal{F}_i) \cup \{m_i\}$
9 end
```

Algorithm 3: Variable Elimination

At the end, $\mathcal{F}$ consists of only a constant. Note that the product of factors isn't trivial, i.e we would need to multiply probability tables. To understand Algorithm 3, let's see an example.

Example 54. Say we have been given 5 variables, and the cliques are

$$\psi_{12}(x_1, x_2), \psi_{24}(x_2, x_4), \psi_{23}(x_2, x_3), \psi_{45}(x_4, x_5), \psi_{35}(x_3, x_5)$$

The corresponding graph is in Figure 18. We can see that

$$Z = \sum_{x_1 \cdots x_5} \psi_{12}(x_1 x_2)\psi_{24}(x_2 x_4)\psi_{23}(x_2 x_3)\psi_{45}(x_4 x_5)\psi_{35}(x_3 x_5)$$

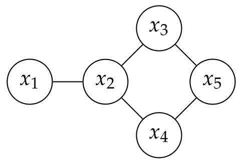

Figure 18: UGM for Example

Say our good ordering is x_1, x_2, x_3, x_4, x_5. So we start

- ◇ First variable x_1 -

$$\begin{aligned}\mathcal{F}_1 &= \{\psi_{12}(x_1, x_2)\}\\ \mathcal{M}_1(x_1, x_2) &= \psi_{12}(x_1, x_2)\\ m_1(x_2) &= \sum_{x_1} \mathcal{M}_1\\ \mathcal{F} &= \{\psi_{24}(x_2, x_4), \psi_{23}(x_2, x_3), \psi_{45}(x_4, x_5), \psi_{35}(x_3, x_5), m_1(x_2)\}\end{aligned}$$

- ◇ Second variable x_2 -

$$\begin{aligned}\mathcal{F}_2 &= \{\psi_{24}(x_2, x_4), \psi_{23}(x_2, x_3), m_1(x_2)\}\\ \mathcal{M}_2(x_2, x_3, x_4) &= \psi_{12}(x_2, x_4)\psi_{23}(x_2, x_3)m_1(x_2)\\ m_2(x_3, x_4) &= \sum_{x_2} \mathcal{M}_2\\ \mathcal{F} &= \{\psi_{45}(x_4, x_5), \psi_{35}(x_3, x_5), m_2(x_3, x_4)\}\end{aligned}$$

- ◇ Third variable x_3 -

$$\begin{aligned}\mathcal{F}_3 &= \{\psi_{35}(x_3, x_5), m_2(x_3, x_4)\}\\ \mathcal{M}_3(x_3, x_4, x_5) &= \psi_{35}(x_3, x_5)m_2(x_3, x_4)\\ m_3(x_4, x_5) &= \sum_{x_3} \mathcal{M}_3\\ \mathcal{F} &= \{\psi_{45}(x_4, x_5), m_3(x_4, x_5)\}\end{aligned}$$

- ◇ Fourth variable x_4 -

$$\begin{aligned}\mathcal{F}_4 &= \{\psi_{45}(x_4, x_5), m_3(x_4, x_5)\}\\ \mathcal{M}_4(x_4, x_5) &= \psi_{45}(x_4, x_5)m_3(x_4, x_5)\\ m_4(x_5) &= \sum_{x_4} \mathcal{M}_4\\ \mathcal{F} &= \{m_4(x_5)\}\end{aligned}$$

The above example showed how $\mathcal{F}$ is a singleton set at the end. We can also modify Algorithm 3 to get $\Pr(x_i)$ as follows -

- ◇ In line 1 of Algorithm 3, we choose a good ordering such that x_i is last
- ◇ The for loop in line 3 runs only for $n - 1$ iterations

- ⋄ After this, at the end, $\mathcal{F}$ will consist of unnormalized values, sum of which will give Z, and each term divided by Z will give the required probability.

What if we want to compute the MAP query? For that, we do the follwing -

- ⋄ In line 6, we have $\hat{m}_i = \max_{x_i} \mathcal{M}_i$ and we have to keep around the maximizing assignment
- ⋄ In the end $\mathcal{F}$ consists of the required argmax.

Theorem 55. *The complexity of the Variable Elimination algorithm is $\mathcal{O}(nm^w)$ where w is the maximum number of variables in any factor.*

Sketch of proof. The bottleneck step in the algorithm's for loop is computing the product of factors, and in general if the factor has κ variables, then the time to do the product will be $\mathcal{O}(m^\kappa)$. □

In Example 54, we see that the time complexity is $\mathcal{O}(nm^3)$. If we started with x_2, our time complexity would've been $\mathcal{O}(nm^4)$.

More interestingly, if we have a star graph (Figure 19), and if we start with the centre node first, we encounter a very severe penalty in terms of time complexity. This elimination order will give you $\mathcal{O}(m^n)$ running time, while removing the non-central nodes first gives you just $\mathcal{O}(nm^2)$ running time.

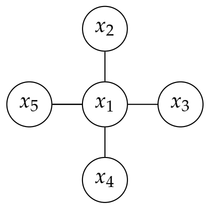
Figure 19: Star Graph

Unfortunately, choosing the optimal elimination order is NP hard in general. But for chordal (triangulated) graphs, the algorithm is polynomial time. But another problem we stumble upon is that if our graph is not triangular, optimal triangulation is NP hard (but there exist many heuristics to do this in polynomial time).

Definition 56 (Simplicial)**.** A vertex in a graph G is simplicial if its neighbors form a complete set.

Theorem 57. *Every triangulated graph is either complete or has at least two non-adjacent simplicial vertices.*

Proof. To be added. □

The goal is to find an optimal ordering for inferring $\Pr(x_1)$, which means x_1 should be last.

```
1 Input: Graph $G$, $n$ = number of vertices in $G$
2 for $i = 1$ to $n$ do
3     $\pi_i \longleftarrow$ any simplicial vertiex in $G$ except 1
4     Remove $\pi_i$ from $G$
5 end
6 return ordering $\pi_1, \cdots, \pi_{n-1}$
```

Algorithm 4: Optimal ordering for triangulated graph

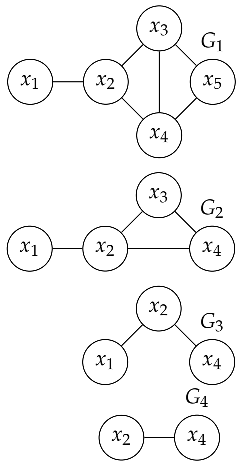

Figure 20: Sequence of graphs

Example 58. Consider the triangulated graph G_1 given on the right for which we have to find the optimal ordering. We go over the iterations as follows

1. In G_1, we have x_1 and x_5 as simplicial vertices. Say we remove x_5 first, to get G_2.
2. In G_2, we have x_1, x_3 and x_4 as the simplicial vertices. Say we remove x_3 to get G_3.
3. In G_3, we have x_1 and x_4 as simplicial vertices. Say we remove x_1 to get G_4.
4. In G_4 we have x_2 and x_4 as simplicial vertices. Say we remove x_2.

The sequence of graphs is shown in Figure 20.

Thus, finally we get the ordering x_5, x_3, x_1, x_4, x_2 as an optimal ordering.

Multiple Inference Queries

The above subsection showed how we can calculate the optimal ordering and a single inference query. But say, we have been given a chain graph with potentials as $\psi_{i,i+1}(x_i, x_{i+1})$, say we need all $\Pr(x_1), \cdots, \Pr(x_n)$, can we do that faster? A no-brain method would be to use variable elimination n times to get $\mathcal{O}(n^2m^2)$.

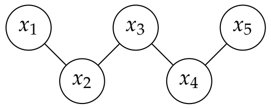

Figure 21: Chain graph

Say I have the chain graph in Figure 21. If we need to calculate $\Pr(x_1)$, we first remove x_5. This is followed by removing x_4, x_3 and x_2.

Now if we want to calculate $\Pr(x_2)$. We can reuse the computation done in removing x_5, x_4 and x_3.

We will see that if we skillfully reuse such computation, if each variable elimination run takes time $\mathfrak{T}$, the time for n inference queries will take just $2\mathfrak{T}$.

Remark 59. Refer to Example 54. In this notice that the arguments of $\mathcal{M}_i(\cdot)$ are cliques with *induced* edges. For example, we have $\mathcal{M}_1(x_1, x_2)$ and clearly $\{x_1, x_2\}$ forms a clique. Second, we have $\mathcal{M}_2(x_2, x_3, x_4)$ and notice that when we add or *induce* an edge between x_2 and x_3, $\{x_2, x_3, x_4\}$ forms a clique. This can be extended to all $\mathcal{M}_i(\cdot)$. For our notion of reusing, we are interested in the maximal cliques formed, and you can check that they refer to the cliques formed by the arguments of $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$.

Junction Trees

The junction tree algorithm is an optimal general-purpose algorithm for **exact** marginal or MAP queries, and can simultaneously compute many such queries. It utilizes efficient data structures and overall has a complexity of $\mathcal{O}(m^w N)$ where w is the size of the largest clique in the triangulated graph and each variable can take m values. It is to note that

- ◇ Viterbi algorithm of Hidden Markov Models
- ◇ Forward-backward algorithm of Kalman Filters

are special cases of junction trees.

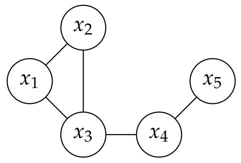

Figure 22: Sample graph for JT

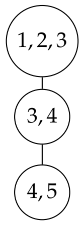

Figure 23: Junction Tree example

Definition 60 (Junction Tree)**.** Junction tree JT of a triangulated graph G with nodes $x_1, \cdots, x_n$ is a tree where the nodes are the maximal cliques of G and the edges obey the *running intersection property*. This property states that if any two nodes contain variable x_i, then x_i is present in every node in the unique path between them.

Example 61. For the graph given in Figure 22, we have the maximal cliques as $\{x_1, x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}$. Thus, the junction tree for the graph is given in Figure 23

Theorem 62. *A graph will have a junction tree if and only if it is chordal.*

Proof. Skipped. □

Construction of a junction tree

If our graph is chordal, we have efficient polynomial time algorithms to create a JT. We first enumerate a set of maximal cliques covering our graph G, then we connect the cliques to get a tree satisfying the *running intersection property*. Note that if our graph is non-triangulated, we need to triangulate it first using heuristics, since optimal triangulation is NP-hard.

Definition 63 (Optimal triangulation)**.** A triangulation which gives rise to a JT where the size of the largest clique is smallest is called the optimal triangulation.

A general method for finding heuristics is -

```
for i = 1 to n
choose the vertex for which some score is minimum
connect all neighbors of chosen vertex
remove the chosen vertex from the graph
```

Some heuristics of triangulation are

1. Choose the vertex with smallest degree and connect all its neighbors
2. Chose the vertex which will require the smallest number of edges to connect neighbors

Example 64. Creation of a JT from a UGM:

1. Consider the graph in Figure 24 as our starting graph.

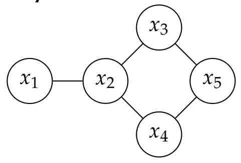

Figure 24: Non-chordal graph

2. We triangulate the graph to get the graph in Figure 25. Notice that the maximal cliques are $C_1 = \{x_1, x_2\}, C_2 = \{x_2, x_3, x_4\}$ and $C_3 = \{x_3, x_4, x_5\}$.

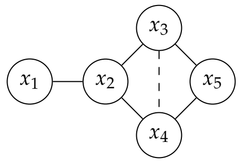

Figure 25: Chordal graph

3. These cliques act as nodes in our junction tree, and we connect the three cliques such that they satisfy the running intersection property. I have dropped the x and just written the indices to avoid clutter. Note in Figure 26 that now we have two types of nodes - in circle we have the cliques, and in rectangles we have the *separators*. Separators are variables present in the intersection of the nodes, thus
$$\text{Separator}(C_i, C_j) = C_i \cap C_j \tag{52}$$

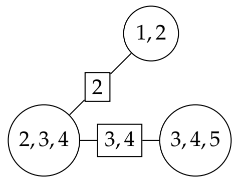

Figure 26: Clique-node graph

4. Now we assign potentials to all cliques. Note that from Example 54, we had potentials $\psi_{12}, \psi_{23}, \psi_{24}, \psi_{35}, \psi_{45}$. Thus here we can assign ψ_{12} to C_1, ψ_{23}, ψ_{24} to C_2 and ψ_{35}, ψ_{45} to C_3.

   Remark 65. If we encounter any ambiguity in assigning potentials, i.e we can assign potentials to more than once clique-node, then we can arbitrarily assign it to any one.

Theorem 66. *Every triangulated graph has a simplicial vertex.*

Proof. Skipped. □

We now state the algorithm to find the maximal cliques in a triangulated graph

```
1 Input: Triangulated graph $G$, $n =$ number of vertices in $G$
2 for $i = 1$ to $n$ do
3     $\pi_i \longleftarrow$ any simplicial vertiex in $G$
4     $C_i \longleftarrow \{\pi_i\} \cup \mathcal{N}(\pi_i)$
5     Remove $\pi_i$ from $G$
6 end
7 return the maximal cliques from $C_1, \cdots, C_n$
```
Algorithm 5: Finding maximal cliques in a chordal graph

Theorem 67. *A clique tree that satisfies the running intersection property maximized the number of separator variables.*

Proof. To be added. □

```
1 Input: Cliques $C_1, \cdots, C_k$
2 Form a complete weighted graph $H$ with cliques as nodes and
    edge weights = size of the intersection of the two cliques it
    connects
3 $T \longleftarrow$ maximum weight spanning tree of $H$
4 return $T$ as the junction tree
```

Algorithm 6: Forming a junction tree from a clique-node graph

Note that once we have the clique graph, we can make a weighted clique graph from that by adding $|\mathcal{S}|$ as the weight for each each between C_i and C_j where $\mathcal{S} = C_i \cap C_j$, and the problem of finding the junction tree boils down to finding the maximum weight spanning tree of the weighted graph.

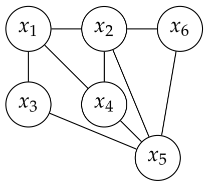

Figure 27: Undirected graph

Example 68. Consider the undirected graph H in Figure 27. Say the potentials are defined over cliques of size 2.

To triangulate, say we pick a heuristic - smallest degree first. Start with x_6 and notice that its neighbors x_2 and x_5 are connected. Next we choose x_3 and connect x_1 and x_5. You can go on further, but notice that the graph is already triangulated (since on removing x_1 the graph becomes complete). Thus an ordering we can have is

$$x_6, x_3, x_1, x_2, x_5, x_4$$

Next we choose the maximal cliques. This can be done using Algorithm 5. This gives the set of maximal cliques as

$$C_1 = \{x_3, x_1, x_5\}$$
$$C_2 = \{x_6, x_2, x_5\}$$
$$C_3 = \{x_1, x_2, x_4, x_5\}$$

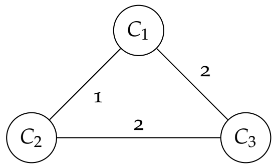

Figure 28: Complete Weighted Graph

We then make the complete weighted graph as in Figure 28. Clearly by removing the edge $C_1 - C_2$, we get the maximum spanning tree, and that gives the junction tree as shown in Figure 29.

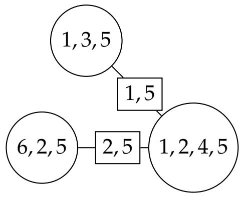

Figure 29: Junction Tree

To assign the potentials, we can assign ψ_{13}, ψ_{35} to C_1, $\psi_{14}, \psi_{12}, \psi_{45}, \psi_{24}$ to C_3 and finally ψ_{25}, ψ_{26} to C_2. The potential of a clique is the product of the potentials assigned to it.

Message Passing

Say each node c, which is a clique in the JT, sends a message $m_{c\to c'}(\cdot)$ to its neighbors c' once it has messages from every other neighbor $\mathcal{N}(c) - \{c'\}$.

$$m_{c\to c'}(\mathbf{x}_s) = \sum_{\mathbf{x}_{c-s}} \psi_c(\mathbf{x}_c) \prod_{d\in\mathcal{N}(c)-\{c'\}} m_{d\to c}(\mathbf{x}_{d\cap c}) \tag{53}$$

For a MAP query, we can replace the $\sum$ with max. Note that the $\sum$ sums over all the variables present in clique but not in separator.

We can write

$$\Pr(\mathbf{x}_c) \propto \psi_c(\mathbf{x}_c) \prod_{d \in \mathcal{N}(c)} m_{d \to c}(\mathbf{x}_{d \cap c}) \tag{54}$$

And to get the marginal probability of any x_i, we can just sum over the rest, i.e

$$\Pr(x_i) = \sum_{\mathbf{x}_c - x_i} \Pr(\mathbf{x}_c) \tag{55}$$

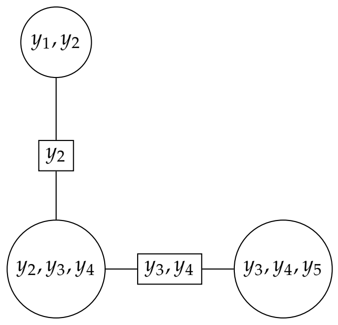
Figure 30: Junction Tree

Example 69. Consider the JT shown in Figure 30. Note that we have edge potentials, and each clique has the product of such potentials present. Each node can send a message once it has messages from neighbors.

Initially, $C_1 = \{y_1, y_2\}$ or $C_3 = \{y_3, y_4, y_5\}$ can initiate the message passing since they have single neighbors. Thus, we have

1. C_1 initiates message $m_{12 \to 234}(y_2) = \sum_{y_1} \psi_{12}(\mathbf{y}_{12})$ to $C_2 = \{y_2, y_3, y_4\}$.
2. C_3 sends message $m_{345 \to 234}(\mathbf{y}_{34}) = \sum_{y_5} \psi_{345}(\mathbf{y}_{345})$ to C_2
3. C_2 sends message $m_{234 \to 345} = \sum_{y_2} \psi_{234}(\mathbf{y}_{234}) m_{12 \to 234}(y_2)$ to C_3
4. C_2 sends message $m_{234 \to 12}(y_2) = \sum_{\mathbf{y_{34}}} \psi_{234}(\mathbf{y}_{234}) m_{345 \to 234}(\mathbf{y}_{34})$ to C_1

We also write that $\Pr(y_1) \propto \sum_{y_2} m_{234 \to 12}(y_2)$.

Remark 70 (Intuition behind message passing). Message from c to c' denotes the result of VE of potentials on the side of the tree that contains the clique c but not c' leaving only the separator variables $s = c \cap c'$.

Addition of Evidence

In such queries, we have an evidence or conditioning set $\mathbf{x}_e$, and we need to find similar types of sub-queries shown before, i.e

1. $\Pr(x_1 | \mathbf{x}_e) = \sum_{x_2, \cdots, x_m} \Pr(x_1, \cdots, x_n | \mathbf{x}_e)$
2. $\mathbf{x}^* | \mathbf{x}_e = \arg\max_{x_1, \cdots, x_m} \Pr(x_1, \cdots, x_n | \mathbf{x}_e)$

where $\{x_2, \cdots, x_m\} = V - \mathbf{x}_e - \{x_1\}$. The trick to add evidence is to change the potentials.

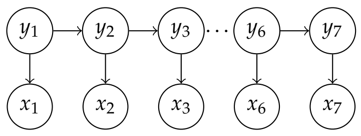
Figure 31: Sample HMM

Example 71 (Viterbi Algorithm)**.** Consider the Hidden Markov Model shown in Figure 31. Define edge potentials as $\Pr(y_i | y_{i-1})$ and $\Pr(x_i | y_i)$. Also, let the evidence variables be $\mathbf{x} = x_1, \cdots, x_n =$

$o_1, \cdots, o_n$. We need to find the most likely values of the hidden state variables $\mathbf{y} = y_1, \cdots, y_n$, i.e

$$\arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x} = \mathbf{o})$$

We redefine the potentials as

$$\psi_i(y_{i-1}, y_i) = \Pr(y_i|y_{i-1})\Pr(x_i = o_i|y_i)$$

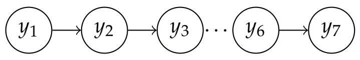
Figure 32: Sample HMM

Since we have fixed the observations of $x_i = o_i$, we have a 1D table instead of a 2D potential table. This gives the reduced chain graph as in Figure 32. Now we use the message passing algorithm shown earlier, just replacing sum with max. For ease of calculation, only consider a three node chain, with the following probabilities:

$$\Pr(y_i = 0|y_{i-1}) = \begin{cases} 0.9 & \text{if } y_{i-1} = 0 \\ 0.2 & \text{if } y_{i-1} = 1 \end{cases}$$

$$\Pr(x_i = 0|y_i) = \begin{cases} 0.7 & \text{if } y_i = 0 \\ 0.6 & \text{if } y_i = 1 \end{cases}$$

Also, $\Pr(y_1 = 1) = 0.5$. Say our observations are

$$[x_1, x_2, x_3] = [0, 0, 0]$$

We want to calculate

$$\arg\max \Pr(y_1, y_2, y_3|x_1, x_2, x_3 = [0, 0, 0])$$

Now, we calculate the potential

$$\psi_{12}(y_1, y_2) = \Pr(y_2|y_1)\Pr(x_1 = 0|y_1)\Pr(y_1)$$

$\psi_{12}(y_1, y_2)$

	0	1
0	$0.9 \times 0.7 \times 0.5$	$0.1 \times 0.7 \times 0.5$
1	$0.2 \times 0.6 \times 0.5$	$0.8 \times 0.6 \times 0.5$

Note that when we calculate

$$\psi_{23}(y_2, y_3) = \Pr(y_3|y_2)\Pr(x_2 = 0|y_2)\Pr(x_3 = 0|y_3)$$

Remark 72 (Approximate Inference). Note the followings points:

- ◇ Exact inference is NP hard. First define the tree width w of a triangulated graph as one less than the size of the maximal clique. The complexity of exact inference is $\mathcal{O}(m^w)$.
- ◇ It is seen that real-life graphs produce large cliques on triangulation. For example, an $n \times n$ grid has a tree width of n. A Kalman filter on K parallel state variables influencing a common observation variable has a tree width of size $K + 1$.

Generalized Belief Propogation

Here, we tr to run some kind of message passing algorithms on graphs which look like junction trees. Instead of creating an exact JT which satisfies the running intersection property, we try to create a cluster graph with two relaxations -

1. The nodes are arbitrary clusters instead of cliques in the chordal graph. We only ensure that all potentials are subsumed.

2. Instead of adding separator nodes, we add a subset of intersecting variables so as to satisfy the running intersection property.

Example 73. Consider the JT creation in Example 64. For that graph, we see that we get maximal clique size of 3. Suppose we want to maintain a clique size of 2.

We create a *factor graph* such that the nodes of the factor graph correspond to the edge potentials given. The cluster/factor graph will be as shown in Figure 33. Note that factor graphs are special kinds of cluster graphs.

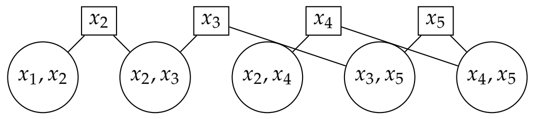

Figure 33: Factor Graph

Belief propogation algorithms are approximate message passing algorithms, and differ in the order of sending of messages. Note that in general graph can have loops and thus we can't apply tree-based two phase methods.
Variants of scheduling order of propogating beliefs are -

1. Simple loopy belief propogation

2. Tree-reweighted message passing

3. Residual belief propogation

There are other classes too, which are

1. Sampling

2. Combinatorial Algorithms

3. Greedy algorithms: relaxation labeling

4. Variatinal methods - mean-field & structured mean-field

5. Linear and Quadratic Programming based approaches

Learning from Data

Each graphical model consists of two major components - the graph structure, and the potentials given the graph structure.

Graph Structure

There are two major methods to learn the graph:

1. **Manual:** A domain expert manually designs the graphs. This is popular in applications where we are well-versed with the underlying dependency structure. For example - Quick Medical Reference (QMR) systems for disease-symptoms matching, Kalman Filters, Grid graphs in Computer Vision, Hidden Markov Models (HMM) in speech recognition/information extraction.

2. **Learning from Examples:** It can be shown that recovering the graph structure is NP hard. Usually learning methods are branch and bound search problems, which are particularly useful in dynamic situations.

Parameters in Potentials

As before, there are the two same methods for learning:

1. **Manual:** Done by a domain expert usually for infrequently constructured graphs (QMR systems), or where the potentials are a trivial function of the attributes of the connected graphs (grid graphs). This is a relevant method for Bayesian Networks, as potentials correspond to conditional probabilities. Thus in data-starved regions, with priors we can use Bayesian Networks.

2. **Learning from Examples:** A popular method where humans cannot make objective assignments. Two major subdomains are table potentials, where each entry is a parameter (HMMs), and potentials with shared parameters and data attributed (CRFs).

Given a sample of data $\mathcal{D}$ generated from a distribution P, being represented by a known graphical model G, our aim is to learn the potentials. There can be many scenarios to consider -

1. **Variables:**

   (a) In each training instance, we have observed all the variables $\mathcal{X}$. Such a setting is called fully supervised setting.

   (b) In each training instance, we have observed a subset of variables $\mathfrak{X} \subset \mathcal{X}$. Such a setting is called partially observed setting.

2. **Potentials:**

3. In BNs, we don't have a log-partition function $\log Z$, and thus in most cases, we'd be able to find closed form solutions for potentials.

4. In UGMs, $\log Z$ causes a lot of trouble. Since potentials are attached to arbitrary overlapping subset of variables, we require gradient descent kind of iterative algorithms.

Representation of potentials as parameters includes -

1. **Generative:** $P(\mathbf{x}) = P(x_1, \cdots, x_n)$ is represented as our G. The training samples are $\mathcal{D} = \{\mathbf{x}^1, \cdots, \mathbf{x}^N\}$ where $\mathbf{x}^i = \{x_1^i, x_2^i, \cdots, x_n^i\}$, and we finally learn the potentials $\psi_C(\mathbf{x}_C)$.

2. **Conditional:** We work with $P(\mathbf{y}|\mathbf{x}) = P(y_1, \cdots, y_n|\mathbf{x})$ represented by our G over $\mathbf{y}$ variables. Our training instances in this case would be 2-tuples $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \cdots, (\mathbf{x}^N), \mathbf{y}^N\}$ and we learn the potentials $\psi_C(\mathbf{y}_C, \mathbf{x})$.

Learning under Conditional Representation

We now focus on the conditional representation of potentials, and provide a general framework for parameter learning.

Consider the conditional distribution $\Pr(\mathbf{y}|\mathbf{x}, \theta)$, potentials are function of $\mathbf{x}$, and we want to learn θ. Say $\mathbf{y} = y_1, \cdots, y_n$ forms a graphical model G.

Say G is undirected, then

$$\begin{aligned}\Pr(y_1, \cdots, y_n|\mathbf{x}, \theta) &= \frac{\prod_C \psi_C(\mathbf{y}_C, \mathbf{x}, \theta)}{Z_\theta(\mathbf{x})} \\ &= \frac{1}{Z_\theta(\mathbf{x})} \exp\left(\sum_C F_\theta(\mathbf{y}_C, C, \mathbf{x})\right)\end{aligned} \tag{56}$$

where $F_\theta(\mathbf{y}_C, C, \mathbf{x}) = \log \psi_C(\mathbf{y}_C, \mathbf{x}, \theta)$, $Z_\theta(\mathbf{x}) = \sum_{\mathbf{y}'} \exp\left(\sum_C F_\theta(\mathbf{y}'_C, C, \mathbf{x})\right)$.

Now, we can think of $F_\theta(\mathbf{y}_C, C, \mathbf{x})$ in many ways

1. **Log-linear model** over features defined by the user (eg. in CRFs, Maxent models). Say we gave K features, and each feature is represented as $f_k(\mathbf{y}_C, C, \mathbf{x})$. Thus,

$$F_\theta(\mathbf{y}_C, C, \mathbf{x}) = \sum_{k=1}^{K} \theta_k f_k(\mathbf{y}_C, C, \mathbf{x}) \tag{57}$$

2. **Neural Network** which takes in $\mathbf{y}_C, C, \mathbf{x}$ and transforms them non-linearly into $\mathcal{Y} \in \mathbb{R}$, and θ are the parameters of the neural network.

Example 74 (Named Entity Recognition)**.** Consider the task of NER, where y_i can take 3 values, and the structure is represented as a chain graph. Thus essentially, the user enforces that only adjacent words in the sentence affect the labels taken. Since we have a chain graph, we take the cliques as edges, and we take our templatized functions as $\mathbf{f}(y_i, y_{i-1}, i, \mathbf{x})$, where $C = i$ is a short hand for $C = (i-1, i)$. Defining the features $f_i(y_i, y_{i-1}, i, \mathbf{x})$. We can manually write heuristics, or in something like BERT, we can have $\mathbf{f}(y_i, y_{i-1}, i, \mathbf{x}) = \mathbf{e}_i \in \mathbb{R}^K$ where e_i is the embedding generated by BERT.

Now, for training, we have been given N input pairs represented by $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \cdots, (\mathbf{x}^N, \mathbf{y}^N)\}$, and the form of F_θ. We learn θ through maximum likelihood, i.e

$$\max_\theta LL(\theta, \mathcal{D}) = \max_\theta \sum_{i=1}^{N} \log \Pr(\mathbf{y}^i | \mathbf{x}^i, \theta) \tag{58}$$

We can write

$$\begin{aligned} LL(\theta, \mathcal{D}) &= \sum_{i=1}^{N} \log \Pr(\mathbf{y}^i | \mathbf{x}^i, \theta) \\ &= \sum_{i=1}^{N} \log \left(\frac{1}{Z_\theta(\mathbf{x}^i)} \exp \left(\sum_C F_\theta(\mathbf{y}^i_C, C, \mathbf{x}^i) \right) \right) \\ &= \sum_{i=1}^{N} \left[\sum_C F_\theta(\mathbf{y}^i_C, C, \mathbf{x}^i) - \log Z_\theta(\mathbf{x}^i) \right] \end{aligned} \tag{59}$$

Note that computing F_θ is not difficult, but to calculate Z_θ for each i requires invoking an inference algorithm.

For training, we can use gradient descent. For now, let us assume a log-linear model as $F_\theta(\mathbf{y}^i_C, C, \mathbf{x}) = \theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i_C, C)$, and denote $\mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) = \sum_C \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i_C, C)$. Thus,

$$LL(\theta) = \sum_i \left(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \log Z_\theta(\mathbf{x}^i) \right) \tag{60}$$

We can add a regularizer to prevent over-fitting. Thus, our objective becomes

$$\max_\theta \sum_i \left(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \log Z_\theta(\mathbf{x}^i) \right) - \frac{\|\theta\|^2}{C} \tag{61}$$

The objective function is concave in θ, and thus we can reach the globally optimal value of θ using gradient descent. The gradient of the objective $L(\theta)$ is

$$\begin{aligned} \nabla L(\theta) &= \sum_i \left(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \frac{\sum_{\mathbf{y}'} \mathbf{f}(\mathbf{y}', \mathbf{x}^i) \exp(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i))}{Z_\theta(\mathbf{x}^i)} \right) - \frac{2\theta}{C} \\ &= \sum_i \left(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \sum_{\mathbf{y}'} \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) \Pr(\mathbf{y}' | \theta, \mathbf{x}^i) \right) - \frac{2\theta}{C} \\ &= \sum_i \left(\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \mathbb{E}_{\Pr(\mathbf{y}' | \theta, \mathbf{x}')} \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) \right) - \frac{2\theta}{C} \end{aligned} \tag{62}$$

where

$$\begin{aligned}\mathbb{E}_{\Pr(\mathbf{y}'|\theta,\mathbf{x}')} f_k(\mathbf{x}^i,\mathbf{y}') &= \sum_{\mathbf{y}'} f_k(\mathbf{x}^i,\mathbf{y}')\Pr(\mathbf{y}'|\theta,\mathbf{x}^i) \\ &= \sum_{\mathbf{y}'}\sum_{C} f_k(\mathbf{x}^i,\mathbf{y}'_C,C)\Pr(\mathbf{y}'|\theta,\mathbf{x}^i) \\ &= \sum_{C}\sum_{\mathbf{y}'_C} f_k(\mathbf{x}^i,\mathbf{y}'_C,C)\Pr(\mathbf{y}'_C|\theta,\mathbf{x}^i)\end{aligned} \tag{63}$$

Example 75. Consider an undirected graphical model on 3 binary variables $\{y_1, y_2, y_3\}$ represented as $\mathbf{y}$, and are forming a chain. We define two features for the model as

$$f_1(\mathbf{x}, y_j, j) = x_j y_j \qquad \text{where } x_j \text{ is the intensity of pixel } j$$
$$f_2(\mathbf{x}, (y_k, y_j), (k, j)) = [\![y_k \neq y_j]\!] \qquad \text{where } [\![\alpha]\!] = 1 \text{ if } \alpha = \text{true}$$

Consider the initial parameters as $\theta = [\theta_1, \theta_2] = [3, -2]$, and consider $\mathbf{x}^1 = [0.1, 0.7, 0.3]$ and $\mathbf{y}^1 = [1, 1, 0]$.

- ◇ The log-node potentials $F_\theta(y_j, C = j, \mathbf{x})$ are given as

  $$y_j = \theta \cdot \mathbf{f}(\mathbf{x}, y_j, j) = \theta_1 x_j y_j$$

  For y_1, we have the value as $[0, 0.3]$, y_2 will have the value $[0, 2.1]$ and y_3 will have $[0, 0.9]$.

- ◇ The log-edge potentials $F_\theta((y_1, y_2), C = (1, 2), \mathbf{x})$ are given as

  $$\theta_2 f_2(\mathbf{x}, (y_1, y_2), (1, 2))$$

  For $(1, 2)$ we have the value as $[0, -2, -2, 0]$ corresponding to $y_1 y_2 = \{00, 01, 10, 11\}$ (can consider it to be a matrix for ease of understanding). Since our edge potentials don't depend on x, $F_\theta((y_2, y_3), C = (2, 3), \mathbf{x}) = [0, -2, -2, 0]$.

Example 76. Consider parameter learning for $\mathbf{y} = \{y_1, \cdots, y_6\}$, where $y_j = \pm 1$. Let us define 8 features for the variables as

$$\begin{aligned} f_1(y_j, y_{j+1}) &= [\![y_j + y_{j+1} > 1]\!], \quad 1 \leq j \leq 5 \\ f_2(y_1, y_3) &= -2y_1y_3 \\ f_3(y_2, y_3) &= y_2y_3 \\ f_4(y_3, y_4) &= y_3y_4 \\ f_5(y_2, y_4) &= [\![y_2y_4 < 0]\!] \\ f_6(y_4, y_5) &= 2y_4y_5 \\ f_7(y_3, y_5) &= -y_3y_5 \\ f_8(y_5, y_6) &= [\![y_5 + y_6 > 0]\!] \end{aligned}$$

Consider

$$\mathbf{f}(\mathbf{y}) = [f_1, \cdots, f_8] \text{ and } \theta = [1\ 1\ 1\ 2\ 2\ 1\ \ -1\ 1]^\top$$

The underlying graphical model is given in Figure 34 For the above graph, we can draw the junction tree to find

$$Z = \sum_{\mathbf{y}} \exp\left(\theta^\top \mathbf{f}(\mathbf{x}, \mathbf{y})\right)$$

For clique C, we have $\psi_C(\mathbf{y}_C) = \exp\left(\theta \cdot \mathbf{f}_C(\mathbf{x}, \mathbf{y}_C)\right)$.

The junction tree for the above graph is given in Figure 35. Say from left to right in the JT, we call the cliques $\mathscr{C}_1$ to $\mathscr{C}_4$. For $\mathscr{C}_2$, the log-potential will be $2 \cdot f_5(y_2, y_4) + 1 \cdot f_1(y_3, y_4) + 2 \cdot f_4(y_3, y_4)$. We can easily find for others too.

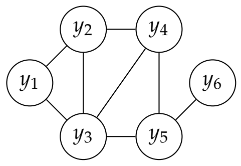

Figure 34: UGM for example

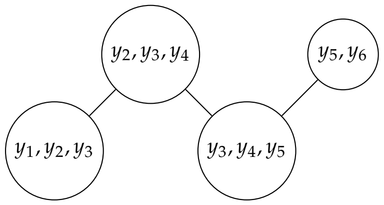

Figure 35: JT for above UGM

Now, coming back to the expectation, we need to compute $\mathbb{E}_{\Pr(\mathbf{y}|\theta^\top, \mathbf{x}^i)} f_k(\mathbf{x}^i, \mathbf{y})$. To do this, we can proceed as follows:

1. Represent the probability $\Pr(\mathbf{y}|\theta^t, \mathbf{x}^i)$ as UGM where nodes are $y_1, \cdots y_n$ and the potential $\psi_C(\mathbf{y}_C, \mathbf{x}, \theta)$ for clique C is $\exp\left(\theta^t \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i_C, C)\right)$.

2. Run a sum-product inference algorithm on the above UGM and compute for each C, $\mathbf{y}_C$ the marginal probability $\mu(\mathbf{y}_C, C, \mathbf{x}^i)$.

3. Using these nodes, compute

$$\mathbb{E}_{\Pr(\mathbf{y}|\theta^t, \mathbf{x}^i)} f_k(\mathbf{x}^i, \mathbf{y}) = \sum_C \sum_{\mathbf{y_C}} \mu(\mathbf{y}_C, C, \mathbf{x}^i) f_k(\mathbf{x}^i, C, \mathbf{y}_C)$$

Having done this, we continue Example 75. We can calculate marginals $\mu(y_j, j)$ and $\mu(y_k, y_j, (k, j))$, and write

$$\begin{aligned}
\mathbb{E}[f_1(\mathbf{x}^1, \mathbf{y})] &= \sum_j \sum_{y'_j} \mu_j(y'_j, j) x_j y'_j = \sum_j y_j x_j \mu_j(1, j) \\
&= 0.1\mu(1,1) + 0.7\mu(1,2) + 0.3\mu(1,3) \\
\mathbb{E}[f_2(\mathbf{x}^2, \mathbf{y})] &= \sum_C \sum_{y'_C} \mu_j(y'_C, C) f_2(y'_C, C, x) \\
&= \mu(1,0,(1,2)) + \mu(0,1,(1,2)) + \mu(1,0,(2,3)) + \mu(0,1,(2,3))
\end{aligned}$$

Now, value of

$$\begin{aligned}
f_1(\mathbf{x}^1, \mathbf{y}^1) &= \sum_C f_1(\mathbf{x}^1, \mathbf{y}^1_C, C) = \sum_{j=1}^{3} f_1(\mathbf{x}^1, y^1_j, j) = \sum_{j=1}^{3} x_j y_j \\
&= 0.1 \times 1 + 0.7 \times 1 + 0.3 \times 0 = 0.8 \\
f_2(\mathbf{x}^1, \mathbf{y}^1) &= [\![y^1_1 \neq y^1_2]\!] + [\![y^1_2 \neq y^1_3]\!] = 1
\end{aligned}$$

Thus we can write the gradients for each parameter as

$$\nabla L(\theta_1) = 0.8 - \mathbb{E}(f_1(\mathbf{x}^1, \mathbf{y})) - 2 \cdot \frac{3}{C}$$
$$\nabla L(\theta_2) = 1 - \mathbb{E}(f_2(\mathbf{x}^1, \mathbf{y})) + 2 \cdot \frac{2}{C}$$

Training Algorithm

```
1  Input: $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$, $\mathbf{f} : f_1, \cdots, f_K$
2  Output: $\theta = \arg\max \sum_{i=1}^N (\theta \cdot \mathbf{f}(\mathbf{x}^i, \mathbf{y}^i) - \log Z_\theta(\mathbf{x}^i)) - \frac{\|\theta\|^2}{C}$
3  Initialize: $\theta^0 = \mathbf{0}$
4  for $t = 1$ to $T$ do
5      for $i = 1$ to $N$ do
6          $g_{k,i} = f_k(\mathbf{x}^i, \mathbf{y}^i) - \mathbb{E}_{\Pr(\mathbf{y}'|\theta^\top, \mathbf{x}^i)} f_k(\mathbf{x}^i, \mathbf{y}')$ for $k = 1, \cdots, K$
7      end
8      $g_k = \sum_i g_{k,i}$ for $k = 1, \cdots, K$
9      $\theta_k^t = \theta_k^{t-1} + \gamma_t (g_k - 2\frac{\theta_k^{t-1}}{C})$
10     Exit: if $\|\mathbf{g}\| \approx 0$
11 end
```

Algorithm 7: Training Algorithm for Parameter Learning

The running time for Algorithm 7 in case of chain graph is $\mathcal{O}(INn(m^2 + K))$ where I is the total number of iterations. But in general, for a graph with tree-width of w, the running time has m^{w+1} instead of m^2.

The above algorithm is a generalized framework, and we can see its use in Bayesian Networks now.

Local Conditional Probability for Bayesian Networks

For a Bayesian Network

$$\begin{aligned} \Pr(y_1, \cdots, y_n | \mathbf{x}, \theta) &= \prod_j \Pr(y_j | \mathbf{y}_{\mathrm{Pa}(j)}, \mathbf{x}, \theta) \\ &= \prod_j \frac{\exp\left(F_\theta(\mathbf{y}_{\mathrm{Pa}(j)}, y_j, j, \mathbf{x})\right)}{\sum_{y_j'=1}^m \exp\left(F_\theta(\mathbf{y}_{\mathrm{Pa}(j)}, y_j', j, \mathbf{x}\right)} \end{aligned} \tag{64}$$

The potentials above are locally normalized. Now, we can write the likelihood as

$$
\begin{aligned}
LL(\theta, \mathcal{D}) &= \sum_{i=1}^{N} \log \Pr(\mathbf{y}^i | \mathbf{x}^i, \theta) \\
&= \sum_{i=1}^{N} \log \left(\prod_j \Pr(y^i_j | \mathbf{y}^i_{\mathrm{Pa}(j)}, \mathbf{x}^i, \theta) \right) \\
&= \sum_i \sum_j \log \Pr(y^i_j | \mathbf{y}^i_{\mathrm{Pa}(j)}, \mathbf{x}^i, \theta) \\
&= \sum_i \sum_j F_\theta(\mathbf{y}^i_{\mathrm{Pa}(j)}, y^i_j, j, \mathbf{x}^i) - \log \sum_{y'=1}^{m} \exp\left(F_\theta(\mathbf{y}^i_{\mathrm{Pa}(j)}, y'_j, j, \mathbf{x}^i\right)
\end{aligned} \tag{65}
$$

We can notice that we are essentially doing a softmax over $F_\theta(\cdot)$, and thus we are doing a normal classification task.

Table Potentials in Feature Framework

Consider a generative model (i.e $\mathbf{x}^i$ does not exist) - such as an Hidden Markov Model.

◇ $F_\theta(\mathbf{y}^i_{\mathrm{Pa}(j)}, y^i_j, j) = \log P(y^i_j, \mathbf{y}^i_{\mathrm{Pa}(j)})$, and the normalizer vanishes.

◇ $\Pr(y_j | \mathbf{y}_{\mathrm{Pa}(j)})$ is a table of real values denoting the probability of each value of x_j corresponding to each combination of values of the parents (θ^j).

- If each variable takes m values, and has k parents, then each $\Pr(y_j | \mathbf{y}_{\mathrm{Pa}(j)})$ will require m^{k+1} parameters in θ^j.

$$
\theta^j_{vu_1,\cdots,u_k} = \Pr(y_j = v | \mathbf{y}_{\mathrm{Pa}(j)} = [u_1, \cdots, u_k]) \tag{66}
$$

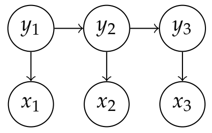

Figure 36: Sample HMM

Example 77 (HMM Parameter Learning)**.** Consider the HMM shown in Figure 36. Say $y_j \in \{1, 2, 3\}$ and $x_j \in \{A, B, C, D\}$. The above table

(y_1, x_1)	(y_2, x_2)	(y_3, x_3)	(y_4, x_4)
1, A	1, B	2, A	3, C
2, B	1, A	3, A	3, D
1, B	1, B	2, C	3, D

contains the values for $\mathcal{D} = (N = 3, n = 4)$. We need to calculate the probability of variable given its parent. Since y_1 has no parent, we can write

$$
P(y_1) = \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \frac{2}{3} & \frac{1}{3} & 0 \\ \hline \end{array}
$$

The values from left to right denote $\theta^1_1, \theta^1_2, \theta^1_3$. For y_2 we need to find $\theta^2_{v:u} = P(y_2 = v|y_1 = u)$ and this will have 9 values.

$P(y|y') =$

	$y = 1$	$y = 2$	$y = 3$
1	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
2	$\frac{1}{3}$	0	$\frac{2}{3}$
3	0	0	1

Now, to find $\theta^{x_1}_{u:v} = P(x_1 = u, y_1 = v)$, we will have 12 values.

For the above, we can notice that

$$\begin{aligned}
LL(\theta, \mathcal{D}) &= \max_\theta \sum_i \sum_j \log P(y^i_j | \mathbf{y}^j_{\text{Pa}(j)}) \\
&= \max_\theta \sum_i \sum_j \log \theta^j_{y_j \mathbf{y}_{\text{Pa}(j)}} \quad \text{s.t} \sum_v \theta^J_{vu_1,\cdots,u_k} = 1 \; \forall \; j, u_1, \cdots, u_k \\
&= \max_\theta \sum_i \sum_j \log \theta^j_{y_j \mathbf{y}_{\text{Pa}(j)}} - \sum_j \sum_{u_1,\cdots,u_k} \lambda^j_{u_1,\cdots,u_k} \left(\sum_v \theta^j_{vu_1,\cdots,u_k} - 1 \right)
\end{aligned}$$

Using gradient descent, we can get that

$$\theta^j_{vu_1,\cdots,u_k} = \frac{\sum_{i=1}^N [\![y^i_j = v, \mathbf{y}_{\text{Pa}(j)} = u_1, \cdots, u_k]\!]}{\sum_{i=1}^N [\![\mathbf{y}_{\text{Pa}(j)} = u_1, \cdots, u_k]\!]} \tag{67}$$

Going back to the previous example, we can write for $P(y_1)$, the variables

$$\theta^1_1 = \frac{\sum_{i=1}^3 [\![y^i_1 = 1]\!]}{\sum_{i=1}^3 [\![1]\!]} = \frac{2}{3}$$

Now for the second table, look at the first entry 2/5. We find all those values, where the next variable is 1, given the previous variable is 1. This occurs two times, and 1 occurs 5 times, giving the value 2/5. Note that this has occurred due to parameter sharing, i.e $\theta^{y_2}_{v:u} = \theta^{y_3}_{v:u} = \theta^{y_4}_{v:u}$. This allows us to use variable length sentences.

Neural Translation Models

Say X is a sentence in English having m tokens, an Y is a sentence in Hindi having n tokens. We want to create $P(Y|X)$. Say our Hindi dictionary has size 30000, then the number of sentences of length n you can create is $(30000)^n$. A simpler method would be

$$P(Y|X) = \prod_{j=1}^n P(y_j | y_1, \cdots, y_{j-1}, X) \tag{68}$$

The above gives a complete factorization. To learn the potentials, we would have to come up with a way to make the given set capable of handling variable length sentences. We can parameterize using a neural network that can handle variable length inputs such as RNNs and Transformers. For an RNN, we have s_t as an embedding for $y_1, \cdots, y_{j-1}$ and is computed recursively.

$$\begin{aligned} s_0 &\longleftarrow \text{ initial state} \\ s_t &\longleftarrow \text{ LSTMcell}(\theta, s_{t-1}, y_{t-1}) \\ v_t &\longleftarrow \text{ embedding of } X \end{aligned}$$

$$P(y_j|y_1, \cdots, y_{j-1}, X) \equiv Softmax(\{y_j\}, NN_\theta[s_t, v_t])$$

Thus, we get a standard classification problem, i.e given a X, find the Y for which $P(Y|X)$ is maximized. The is intractable for chain graph, and thus in practice greedy inference algorithms such as Beam Search are used.

Learning with Hidden Variables

If we suppose only a subset of variables, how do we learn the parameters of the graphical model?

Say we have the HMM shown in Figure 37. Note that we have changed the notation from $x \to y$ and $y \to z$, since x is not a random variable for now. In CRF, we try to learn $P(Y|X)$ with $\mathcal{D} = \{\mathbf{x}^i, \mathbf{y}^i\}$, where all variables $y_1^i, \cdots, y_n^i$ are present in the dataset, but here in addition some variables $z_1^i, \cdots z_m^i$ are not present in $\mathcal{D}$.

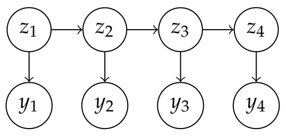

Figure 37: Sample HMM

If we denote θ as the parameters of the graphical model, then

$$P_{\theta,G}(y_1, \cdots y_n, z_1, \cdots, z_m|\mathbf{x}) = \frac{1}{Z_{\theta(\mathbf{x})}} \exp\Big(\sum_C F_\theta(\mathbf{y}_C, \mathbf{z}_C, \mathbf{x})\Big) \tag{69}$$

where C is the set of cliques in the graph. Suppose $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i) : i = 1, \cdots, N\}$ is our dataset, we want to find

$$\begin{aligned} \theta^{ML} &= \arg\max_\theta \sum_{i=1}^N \log P_\theta(\mathbf{y}^i|\mathbf{x}^i) \\ &= \arg\max_\theta \sum_{i=1}^N \log\Big(\sum_{\mathbf{z}} P_{\theta,G}(\mathbf{y}^i, \mathbf{z}|\mathbf{x}^i)\Big) \end{aligned} \tag{70}$$

The sum over $\mathbf{z}$ makes the optimization difficult (because the space of values is large) and thus we want to approximate this, and for this we use a variational approach.

In the variational approach, we introduce auxiliary variables and write

$$\begin{aligned} &\max_\theta \sum_{i=1}^N \log \sum_{\mathbf{z}:z_1,\cdots z_m} P(\mathbf{y}^i, \mathbf{z}|\theta, \mathbf{x}^i) \\ &\equiv \max_\theta \sum_{i=1}^N \max_{q_{i,\mathbf{z}}:\sum_{\mathbf{z}} q_{i,\mathbf{z}}=1} \sum_{\mathbf{z}} q_{i,\mathbf{z}} \log P(\mathbf{y}^i, \mathbf{z}|\theta, \mathbf{x}^i) - \sum_{\mathbf{z}} q_{i,\mathbf{z}} \log q_{i,\mathbf{z}} \end{aligned} \tag{71}$$

This rewriting makes us avoid the summation within the log, and we have two maximization problems - over θ and all q variables. The inner one can be solved in a closed form for fix θ, and the outer one can be solved like a maximum likelihood problem without hidden variables. We now prove the above result.

Proof. We will show that

$$\begin{aligned}\log \sum_{z=1}^{k} g(y,z) &= \max_{q_1,\cdots,q_k} \sum_{z=1}^{k} q_z \log g(y,z) - \sum_z q_z \log q_z \\ &\quad s.t. \sum_{z=1}^{k} q_z = 1 \text{ and } q_z \geq 0\end{aligned} \tag{72}$$

where the variables q_1 to q_k are auxiliary, and

$$Q(q,g) \triangleq \sum_{z=1}^{k} q_z \log g(y,z) - \sum_z q_z \log q_z \tag{73}$$

We use the Lagrangian multipliers method to do so.

Let $L(Q,\lambda) = Q(q,g) + \lambda(\sum_z q_z - 1)$. Then we can write

$$\begin{aligned}\frac{\partial L(Q,\lambda)}{\partial q_j} &= \log g(y,j) - \log q_j - 1 + \lambda = 0 \\ &\implies q_j^* \equiv \alpha g(y,j) \\ \sum_z q_z = 1 : q_j^* &= \frac{g(y,j)}{\sum_z g(y,z)}\end{aligned}$$

If we put this value of q^* in the original equation, we get the expression to be equal to $\log \sum_{z=1}^{k} g(y,z)$ and get the required result. Note that for all other q, we have

$$\log \sum_{z=1}^{k} g(y,z) \geq \sum_{z=1}^{k} q_z \log g(y,z) - \sum_z q_z \log q_z$$

□

Now coming back, the above optimization algorithm is solved using the EM Algorithm.

EM Algorithm

◇ **E-Step:** Solve for $q_{i,\mathbf{z}}$ keeping θ fixed at θ^t

$$q_{i,\mathbf{z}}^t = \frac{P(\mathbf{y}^i, \mathbf{z}|\theta^t, \mathbf{x}^i)}{\sum_{\mathbf{z}'} P(\mathbf{y}^i, \mathbf{z}'|\theta^t, \mathbf{x}^i)} \tag{74}$$

Here $q_{i,\mathbf{z}}$ is the posterior distribution of hidden variable at time t.

◇ **M-Step:** Solve for θ, keeping $q_{i,\mathbf{z}}$ fixed to $q_{i,\mathbf{z}}^t$. The problem becomes

$$\max_{\theta} \sum_i \sum_{\mathbf{z}} q_{i,\mathbf{z}}^t \log P(\mathbf{y}^i, \mathbf{z}|\theta, \mathbf{x}^i) \tag{75}$$

The above is concave in θ and can be often solved in a closed form (eg: HMM).

We can now use the above algorithm for graphical models.

EM Algorithm for Graphical Models

If we have a large number of possible values of $\mathbf{z}$, we'd need a lot of computations, but for graphical models, we don't need to directly compute all $q_{i,\mathbf{z}}$. We revisit the **M-Step** as follows

$$\begin{aligned}
&\max_\theta \sum_i \sum_z q^t_{i,\mathbf{z}} \log P(\mathbf{y}^i, \mathbf{z}|\theta, \mathbf{x}^i) \\
&= \max_\theta \sum_i \sum_{\mathbf{z}} q^t_{i,\mathbf{z}} \left(\log \exp \sum_C F_\theta(\mathbf{y}^i_C, \mathbf{z}_C|\mathbf{x}^i) - \log Z_\theta(\mathbf{x}^i) \right) \\
&= \max_\theta \sum_i \sum_{\mathbf{z}} q^t_{i,\mathbf{z}} \left(\sum_C F_\theta(\mathbf{y}^i_C, \mathbf{z}_C|\mathbf{x}^i) \right) - \sum_i \log Z_\theta(\mathbf{x}^i) \sum_{\mathbf{z}} q^t_{i,\mathbf{z}} \\
&= \max_\theta \sum_i \sum_C \sum_{\mathbf{z}_C} F_\theta(\mathbf{y}^i_C, \mathbf{z}_C|\mathbf{x}^i) \sum_{\mathbf{z}-\mathbf{z}_C} q^t_{i,\mathbf{z}} - \sum_i \log Z_\theta(\mathbf{x}^i) \\
&= \max_\theta \sum_i \sum_C \sum_{\mathbf{z}_C} q^t_{i,\mathbf{z}_C} F_\theta(\mathbf{y}^i_C, \mathbf{z}_C|\mathbf{x}^i) - \sum_i \log Z_\theta(\mathbf{x}^i)
\end{aligned}$$

Example 78. Consider the address

$$\underset{x_1}{\overset{z_1}{\text{Hostel}}}\ \underset{x_2}{\overset{z_2}{\text{11}}},\ \underset{x_3}{\overset{y_3}{\text{IIT}}}\ \underset{x_4}{\overset{y_4}{\text{Bombay}}},\ \underset{x_5}{\overset{z_5}{\text{Powai}}},\ \underset{x_6}{\overset{y_6}{\text{Mumbai}}}$$

Here x_i denotes the tokens in the sequence. z_i denotes that we are unaware of the label, and y_i denotes that we are aware of the label. Say our label set is

$$y \in \{House\#, InstName, Area, City, Other\}$$

Also, consider a chain graph of the variables

$$z_1 - z_2 - y_3 - y_4 - z_5 - y_6$$

We want $q^t_{i:\mathbf{z}_C}$. First see $q_{1:z_1,z_2}$.

$$q_{1:z_1,z_2} \equiv P(z_1, z_2|\theta^t, y_3, y_4, y_6)$$

$$q^t_{1:z_2}$$

$$q^t_{1:z_3}$$

To be continued...

Sampling

It is often needed to sample from a joint distribution, of the form $P(y_1, \cdots, y_n)$ or $P(\mathbf{x} = x_1, \cdots, x_n)$ or $P(x_1, \cdots, x_r | x_{r+1} = E_{r+1}, \cdots, x_n = E_n)$. Some scenarios might be

1. Solving an intractable inference during training
2. Showing a diverse set of outputs instead of just the most likely value
3. Calculating the expected value of some arbitrary function $f(\mathbf{x})$ under distribution $P(\mathbf{x})$.

Motivation

◇ Say we have a deep language model, we might want to generate sample sentences, questions or expected distribution of first word for sentences ending with '?'.

◇ We can use VAEs for missing value imputation. This can be done by fixing values of some of the outputs, and generate most likely values of others.

We had seen in VAEs how to approximate the expected value of a function with sampling. Say we have a function $f(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$ and we want $\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})] = \sum_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})P(\mathbf{x})$. In case of continuous $\mathbf{x}$, we can take the integral. Our space $\mathcal{X}$ is very large, and we cannot compute the integral exactly in closed form. Hence we want to sample and approximate the expectation. Say we have samples $\mathbf{x}^1, \cdots, \mathbf{x}^M \sim P(\mathbf{x})$, we write

$$\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})] = \sum_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})P(\mathbf{x}) \approx \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}^i) \tag{76}$$

As $M \to \infty$, this approximation matches exact expected value.

Sampling scalar distributions

Let $p(x)$ be a distribution, and we want to draw samples $x^1, \cdots, x^M$. Say we can sample u from $U(0, 1)$ (uniform distribution). Let $F(x)$ be the CDF of $p(x)$.

```
For $i = 1, \cdots, M$
  ▷ Sample $u_i \sim U(0, 1)$
  ▷ Find $x_i = F^{-1}(u)$
```

Now, say we a multinomial distribution. Say x is discrete and $\in \{1, \cdots, m\}$. We have $p(x) \sim Mult(p_1, \cdots, p_m)$, $u_i \sim U(0, 1)$, and if u_i is between $\sum_{j=0}^{k-1} p_j$ and $\sum_{j=0}^{k} p_j$, choose k.

Now, as $M \to \infty$, the fraction of times we encounter a sample in the interval $[x, x+\Delta)$ would be proportional to the true probability of that in the interval in $p(x)$, i.e $F(x+\Delta) - F(x)$.

Sampling multivariate distributions

One option to sample from multivariate distributions is to factorize the distribution as a Bayesian Network, and perform *forward sampling*. Such a method is used in autoregressive language models. Assume

$$P(\mathbf{x}) = \prod_{j=1}^{n} P(x_j | \text{Pa}(x_j))$$

Now, $y_j \sim P(y_j | y_1, \cdots, y_{j-1}, \mathbf{x}) = P(y_j | s_j, \mathbf{x})$. The s_j is called the state in RNNs, and the calculation of probability is called softmax.

```
1 $x_1, \cdots, x_n \longleftarrow$ topologically sorted according to BN
2 for $i = 1$ to $M$ do
3     $\xi^i = [0, \cdots, 0]$
4     for $j = 1$ to $n$ do
5         $\xi^i_j \sim P(x_j | \xi^i_{\text{Pa}(x_j)})$
6     end
7 end
8 return $\xi^1, \cdots, \xi^M$
```

Algorithm 8: Forward Sampling Algorithm

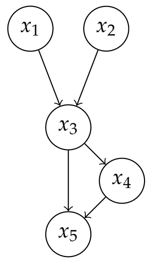

Figure 38: BN Example

Example 79. Consider the BN shown in Figure 38. We start with x_1 as it has no parents, and we want to get ξ^1.

1. $x_1 \sim P(x_1)$. Say x_i were binary, and we get $\xi^1_1 = 0$.
2. $x_2 \sim P(x_2)$. Say we get $\xi^1_2 = 1$.
3. $x_3 \sim P(x_3 | x_1 = 0, x_2 = 1)$. Say we get $\xi^1_3 = 1$.

We can proceed similarly and get our sample ξ^1.

There are drawbacks of forward sampling - mainly being it will not be consistent when we have conditions imposed. Say we want to get the probability of x_1 being 'what' when ! is the last token. Forward sampling would have most of the sampled sentences wasted since they don't end with '!'. Another example could be when we want to complete a missing attribute in a VAE network for object generation and in this case forward sampling wouldn't match the given values most of the time.

In such cases we use **importance sampling**. It is useful when it is hard to sample from $P(\mathbf{x})$ or to lower the error in computation of the expected value of the function, i.e $\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})]$, where $f(\mathbf{x})$ has zeros at a large number of $\mathbf{x}$. Importance sampling samples from the important regions.

In importance sampling, we get to choose $Q(\mathbf{x})$, called the proposal distribution, from which it is easy to generate samples. Designing such a $Q(\mathbf{x})$ is problem-dependent. For example, in a language modeling task, $Q(\mathbf{x})$ is the reverse language model.

Say we generate $S_Q = \{\mathbf{x}^1, \cdots, \mathbf{x}^M\}$ from $Q(\mathbf{x})$. In general $\forall$ functions, $\mathbb{E}_{Q(\mathbf{x})}[f(\mathbf{x})] \neq \mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})]$. We use the following trick

$$\begin{aligned}
\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})] &= \sum_{\mathbf{x}} f(\mathbf{x})P(\mathbf{x}) \\
&= \sum_{\mathbf{x}} Q(\mathbf{x})\frac{P(\mathbf{x})}{Q(\mathbf{x})}f(\mathbf{x}) \qquad \text{let } W(\mathbf{x}) = \frac{P(\mathbf{x})}{Q(\mathbf{x})} \\
\mu_P(S_Q) &= \frac{1}{M}\sum_{i=1}^{M}\left[f(\mathbf{x}^i)W(\mathbf{x}^i)\right] \qquad \text{(Importance weighted estimate)}
\end{aligned}$$

```
1 Given: $M, Q(\mathbf{x}), P(\mathbf{x})$
2 for $i = 1$ to $M$ do
3     $\xi^i \longleftarrow$ Sample from $Q(\mathbf{x})$
4     $W^i \longleftarrow \frac{P(\xi^i)}{Q(\xi^i)}$
5 end
6 return $(\xi^1, W^1), \cdots, (\xi^M, W^M)$
```

Algorithm 9: Importance Sampling Algorithm

The user can decide what to do with the samples, for example

$$\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})] = \frac{1}{M}\sum_{i=1}^{M} f(\xi^i)W^i \tag{77}$$

The limitations for the above algorithm is that it is not applicable for $P(\mathbf{x})$ where we have an intractable normalizer (such as a CRF with a large tree width graph). In such cases, we do normalized importance sampling. We assume that

$$P(\mathbf{x}) = \frac{\widetilde{P}(\mathbf{x})}{Z} \quad \text{where } Z \text{ is intractable} \tag{78}$$

Then, we do

$$\begin{aligned}
\mathbb{E}_{P(\mathbf{x})}[f(\mathbf{x})] &= \frac{1}{Z}\sum_{\mathbf{x}} Q(\mathbf{x})\frac{\widetilde{P}(\mathbf{x})}{Q(\mathbf{x})}f(\mathbf{x}) \\
&= \frac{1}{Z}\sum_{\mathbf{x}} Q(\mathbf{x})\widetilde{W}(\mathbf{x})f(\mathbf{x}) \\
Z &= \sum_{\mathbf{x}} \widetilde{P}(\mathbf{x})
\end{aligned}$$

The applications for the above are

- Undirected Graphical Model
- Given a BN, we want to sample of a subset of variables conditioned on fixed value of others, i.e $P(x_1, \cdots, x_r | x_{r+1}, \cdots, x_n = evidence)$

Given $S_Q = \{\mathbf{x}^1, \cdots, \mathbf{x}^M\}$, we have

$$\mathbb{E}_P[f(\mathbf{x})] \approx \frac{1}{ZM} \sum_{i=1}^{M} f(\mathbf{x}^i) \widetilde{W}(\mathbf{x}^i) \tag{79}$$

where

$$Z = \sum_{\mathbf{x} \in \mathcal{X}} \widetilde{P}(\mathbf{x}) = \mathbb{E}_Q[\widetilde{W}(\mathbf{x})] \approx \frac{1}{M} \sum_{i=1}^{M} \widetilde{W}(\mathbf{x}^i) \tag{80}$$

Thus,

$$\mathbb{E}_P[f(\mathbf{x})] \approx \frac{\sum_{i=1}^{M} f(\mathbf{x}^i) \widetilde{W}(\mathbf{x}^i)}{\sum_{i=1}^{M} \widetilde{W}(\mathbf{x}^i)} \tag{81}$$

The choice of $Q(\mathbf{x})$ for which the expected square error of the estimate from the true expected value is minimum when

$$Q(\mathbf{x}) \propto |f(\mathbf{x})| P(\mathbf{x}), \quad Q(\mathbf{x}) > 0 \text{ whenever } P(\mathbf{x}) > 0 \tag{82}$$

Normalized importance sampling is biased when M is small, and designing a good $Q(x)$ for which the sampling is efficient is not always easy.

Markov Chain Monte Carlo Sampling

In forward and importance sampling, each $\mathbf{x}^i$ was independent, and hence we could just run all instances in parallel. But MCMC sampling is a more general case, where we have dependencies in variables. It is useful when we cannot easily design the proposal distribution. MCMC sampling is more broadly applicable as we don't need very accurate proposal distribution or we don't know need to assume BN type factorization.

Such sampling is applicable when either of the following holds:

1. It is easy to calculate conditional probability of 1 variable, i.e $P(x_i | \mathbf{x}_{-i})$ when the rest of the variables have fixed values.
2. It is easy to calculate $\dfrac{P(\mathbf{x})}{P(\mathbf{x}')}$ and normalizer is not required. Maybe a neural network is an example here.

MCMC sampling is useful when everything else fails and guaranteed to converge to optimal when we take infinite samples.

Given $P(\mathbf{x} = x_1, \cdots, x_n)$ where $x_i \in \{1, \cdots, m\}$ is intractable to sample from but easy to evaluate.

1. Gibbs Sampling - $P(x_i|\mathbf{x} - x_i) = P(x_i|\mathbf{x}_{-i})$
2. Metropolis Hastings Sampling - $\frac{P(\mathbf{x})}{P(\mathbf{x}')}$

We need to design the *MCMC Sampling Transition Function*. It is designed much like the proposal distribution. Thus,

$$\sum_{x \in X} T(x|x') = 1 \quad \text{where } T(x|x') \geq 0 \; \forall \; x, x' \in X \text{ and } |X| = m^n \tag{83}$$

where X is the space of all x.

```
1 Start with an initial sample $x^0$
2 **for** $t = 1$ *to L where L is large* **do**
3     $x^{t+1} \sim T(x|x' = x^t)$
4     $x^0 \to x^1 \to x^2 \to \cdots \to x^L$
5 **end**
6 Actually perform the sampling for $t = L + 1$ to $t = L + Mk$
7 $x^t \sim T(x|x^{t-1})$
8 **return** $x^{L+k}, \cdots, x^{L+Mk}$
```

Algorithm 10: MCMC Sampling Algorithm

In Gibbs Sampling, $T(x|x')$ is defined as
$T(x|x') = 0$ if x and x' differ in more than one co-ordinate
$T(x|x') = \frac{1}{n}\sum_{i=1}^{n} P(x_i|x'_{-i})$ if $x = x'$
$T(x|x') = P(x_i|x'_{-i})$ if $x \neq x'$

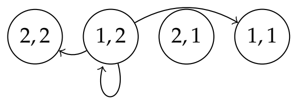

Figure 39: Markov Chain

Example 80. For $n = 2$ and $m = 2$. Thus each x_i can either be 1 or 2. Thus,
$T(x|x' = [1,2]) = 0$ if $x = [2,1]$
$T(x|x' = [1,2]) = \frac{1}{2}P(x_1 = 1|x_2 = 2) + \frac{1}{2}P(x_2 = 2|x_1 = 1)$ if $x = [1,2]$
$T(x|x' = [1,2]) = P(x_1 = 2|x_2 = 2)$ if $x = [2,2]$
$T(x|x' = [1,2]) = P(x_2 = 2|x_1 = 1)$ if $x = [1,2]$
For the above transition probabilities, we can have a graph as shown in Figure 39.

Consider finite state space X, where we have states 1 to m^n and say we choose an arbitrary initial state x^0 at $t = 0$. We want to calcualte $T(x|x^0)$. This $T(x|x^0)$ is a multinomial distribution over all states. We sample from this. At $t = 0$, $P_{t=0}(x) = 1$ if $x = x^0$ and 0 else. At $t = 1$, we want $P_{t=1}(x) = T(x|x^0)$. At $t = 2$, the probability of being in state x is $P_{t=2}(x) = \sum_{x'} T(x|x')T(x'|x^0) = \sum_{x'} T(x|x')P_{t-1}(x')$. Thus in general

$$P_t(x) = \sum_{x'} T(x|x')P_{t-1}(x') \tag{84}$$

As $t \to \infty$, $P_{t+1}(x) \approx P_t(x)$ for convergence as $t \to \infty$ and this is equal to the stationary distribution $\pi(x)$. We are interested in $T(x|x')$ for which $P_{t+1}(x) = \sum_{x'} P_t(x')T(x|x')$ and as $t \to \infty$

$$\pi(x) = \sum_{x'} \pi(x')T(x|x') \tag{85}$$

has a unique solution for given $T(x|x')$ and $\pi(x)$ should be reachable from any initial state x^0 via Markov walks using $T(x|x')$.

Remark 81. $P_t(x)$ is the probability of being in state x after T MCMC steps, but it is conditioned on x_0, i.e the initial state.

Remark 82. It is not necessary that $\pi(x)$ is unique, but it can be proven that for the transition matrix $T(\cdot)$, we will at least get 1 stationary distribution.

Example 83 (Single Solution)**.** Consider a state space such that $|\mathcal{X}| = 3$ and we have

$$T = \begin{bmatrix} 0.25 & 0.5 & 0 \\ 0.75 & 0 & 0.75 \\ 0 & 0.5 & 0.25 \end{bmatrix}$$

In the matrix, the columns denote x' and rows denote x, thus we should sum to 1 over all columns.

$$\pi(x) = [\pi_1, \pi_2, \pi_3]$$

$$\begin{aligned} \pi(x) &= \sum_{x'} \pi(x')T(x|x') \\ \pi_1 &= 0.25\pi_1 + 0.5\pi_2 \\ \pi_2 &= 0.75\pi_1 + 0.75\pi_3 \\ \pi_3 &= 0.5\pi_2 + 0.25\pi_3 \\ \pi_1 + \pi_2 + \pi_3 &= 1 \end{aligned}$$

We get

$$\pi_1 = \frac{2}{7}, \pi_2 = \frac{3}{7}, \pi_3 = \frac{2}{7}$$

Notice that 1 and 3 are symmetric, as seen in T too.

Example 84 (Multiple Solutions)**.** Say that

$$T(x|x') = \begin{cases} 1 \text{ if } x = x' \\ 0 \text{ otherwise} \end{cases}$$

Any stationary distribution is valid, and T is an identity matrix. Thus, infinite number of solutions and

$$P_t = \delta(x^0)$$

Example 85 (Unreachable Solution)**.** Even when $T(x|x')$ has a unique stationary distribution $\pi_T(x)$, this distribution may not be the one we can reach or converge to via MCMC walks.

$$T(x|x') = \begin{cases} 1 \text{ if } x \neq x' \\ 0 \text{ otherwise} \end{cases}$$

$\pi_1 = \pi_2, \pi_2 = \pi_1$ and $\pi_1 + \pi_2 = 1$. Thus we get

$$\pi_1 = \pi_2 = \frac{1}{2}$$

This is an unreachable solution since we aren't guaranteed to reach it via MCMC sampling. Say

$$\begin{aligned} t &= 0 \quad x^0 = v_1 \quad t \text{ is even } x^t = v_1 \\ t &= 1 \quad x^1 = v_2 \quad t \text{ is odd } x^t = v_2 \end{aligned}$$

Definition 86 (Ergodicity)**.** Markov chains with a unique $\pi(x)$ that can be reached via MCMC steps irrespective of starting point are called ergodic.

Definition 87 (Regular)**.** A Markov chain is regular if there exists a k such that x to x' can be reached in exactly k steps.

Theorem 88. *When number of states is finite, then a Markov chain is ergodic iff it is regular.*

Proof. Skipped. □

Example 89. In the single solution example above, let us see what k means. For $k = 1, v_1 \to v_3$ not possible in 1 step and thus, it is not ergodic for $k = 1$. Now for $k = 2$ we need to check all possibile transitions and we find that the Markov chain is regular with $k = 2$.

Manually checking might get cumbersome, and we have a simpler sufficient condition as follows -

Lemma 90. *A finite state Markov chain is regular when*

1. *every state has a self loop with non-zero probability*
2. *between any two states $\exists$ a path of non-zero probability*

Note that the condition above is *when* and not iff.

Theorem 91. *Markov chains defined by Gibbs sampling is ergodic, and has a stationary distribution on $P(\mathbf{x})$ when $P(\mathbf{x})$ is positive.*

Proof. Recall that

$$T(x|x') = \begin{cases} 0 \text{ if } x \text{ and } x' \text{ differ in more than one coordinate} \\ \frac{1}{n}\sum_{i=1}^{n} P(x_i|x'_{-i}) \text{ if } x = x' \\ P(x_i|x'_{-i}) \text{ if } x \neq x', \text{ i.e } x_{-i} = x'_{-i} \end{cases}$$

Since $P(\mathbf{x})$ is positive, self-loop probability is positive as $\forall$ $\mathbf{x}$ $T(\mathbf{x}|\mathbf{x}') > 0$. We can reach from $\mathbf{x}' \to \mathbf{x}$ with non-zero probability in n steps at maximum. Now to show $\pi(\mathbf{x}) = P(\mathbf{x})$.

$$\begin{aligned} \pi(\mathbf{x}) &= \sum_{\mathbf{x}'} T(\mathbf{x}|\mathbf{x}') \\ &= \sum_{\mathbf{x}'} P(\mathbf{x}')T(\mathbf{x}|\mathbf{x}') \\ &= \sum_{x'_1} P(x'_1, x_2, \cdots, x_n)\frac{1}{n}P(x_1, \cdots, x_n) + \cdots \\ &\quad + \sum_{x'_n} P(x_1, \cdots, x_{n-1}, x'_n)\frac{1}{n}P(x_n|x_1, \cdots, x_{n-1}) \\ &= P(x_2, \cdots, x_n)\frac{1}{n}P(x_1, \cdots x_n) + \cdots \\ &\quad + \frac{1}{n}P(x_1, \cdots x_{n-1})P(x_n|x_1, \cdots x_{n-1}) \\ &= \frac{1}{n}P(x_1, \cdots, x_n) + \cdots + \frac{1}{n}P(x_1, \cdots, x_n) \\ &= P(x_1, \cdots, x_n) \end{aligned}$$

□

In undirected graphical models, it is easy to find the probabilities.

$$P(x_1, \cdots, x_n) = \frac{1}{Z}\prod_C \psi_C(\mathbf{x}_C)$$

$$\begin{aligned} P(x_i|\mathbf{x}_{-i}) &= \frac{P(x_1, \cdots, x_n)}{\sum_{x_i} P(x_1, \cdots, x_n)} \\ &= \frac{\prod_C \psi_C(\mathbf{x}_C)}{\sum_{x_i}\prod_C \psi_C(\mathbf{x}_C)} = \frac{\prod_{C:i\in\mathcal{C}} \psi_C(\mathbf{x}_C)}{\sum_{x_i}\prod_{C:i\in\mathcal{C}} \psi_C(\mathbf{x}_C)} \end{aligned} \tag{86}$$

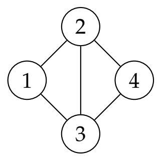

Figure 40: Sample UGM

Example 92. Say we have the undirected graphical model as shown in Figure 40. In this case, we can write

$$\begin{aligned} P(x_1|\mathbf{x}_{-1}) &= \frac{\psi_{123}(x_1, x_2, x_3)\psi_{234}(x_2, x_3, x_4)}{\sum_{x_1} \psi_{123}(x_1, x_2, x_3)\psi_{234}(x_2, x_3, x_4)} \\ &= \frac{\psi_{123}}{\sum_{x_1} \psi_{123}(x_1, x_2, x_3)} \end{aligned}$$

Limitations of Gibbs

1. For a continuous distribution, $P(x_i|x_{-i})$ may not be easy to obtain
2. When variables are highly correlated, Gibbs sampling which only had local moves might have high probability states whose neighbors are mostly low probability leading to poor mixing.

Example 93 (Poor mixing)**.** Say we have

$$P(x_1, x_2) = \begin{cases} \dfrac{1-\epsilon}{2} \text{ if } x_1 = x_2,\ x_i \in \{0,1\} \\ \dfrac{\epsilon}{2} \text{ if } x_1 \neq x_2 \text{ and } \epsilon \text{ is small} \end{cases}$$

Say we compute $T([0,0]|[0,0])$.

$$T([0,0]|[0,0]) = \frac{1}{2}\Big[P(x_1 = 0|x_2 = 0) + P(x_2 = 0|x_1 = 0)\Big] = 1 - \epsilon$$

By symmetry $T([1,0]|[0,0]) = T([0,1]|[1,1]) = \frac{\epsilon}{2}$ and when $\epsilon \to 0$, we mostly will get [0, 0] and [1, 1] states. To go to [1, 1] from [0, 0] we need to take a low probability path, and thus we have high probability islands with no direct transition between them.

Metropolis Hastings Sampling

This is motivated by the need to design moves that go from one high probability state to another without passing through low probability states. In this algorithm, $T(\cdot)$ isn't the sole determiner of transition, but is a proposal distribution for transitions. The Metropolis Hastings Algorithm has that

1. Choose any proposal distribution for transferring from x to x' i.e $T^Q(x \to x')$
2. Use T^Q to propose a transition from $x \to x'$. We accept the proposal with probability $A(x \to x')$ and transition or stay in x. Then we define

$$T(x \to x') = T^Q(x \to x')A(x \to x') \quad x \neq x' \tag{87}$$
$$T(x \to x') = T^Q(x \to x) + \sum_{x' \neq x} T^Q(x \to x')(1 - A(x \to x')) \tag{88}$$

Definition 94 (Reversible chain)**.** A finite state Markov Chain T is reversible, if $\exists$ a unique π such that $\forall\ x, x' \in \mathcal{X}$

$$\pi(x')T(x' \to x) = \pi(x)T(x \to x') \tag{89}$$

The above equation is called the **Detailed Balance Equation (DBE)**.

Theorem 95. *If $\pi(x)$ satisfies the above equation, then $\pi(x)$ is a stationary distribution of T.*

Proof. $\sum_{x'} \pi(x')T(x' \to x) = \pi(x) \sum_{x'} T(x \to x') = \pi(x)$ □

Question 96. *Show that the transition function for Gibbs Sampling satisfies* ***DBE****, i.e*

$$P(\mathbf{x})T(\mathbf{x}'|\mathbf{x}) = P(\mathbf{x}')T(\mathbf{x}|\mathbf{x}')$$

Answer 97. Note that since we are working with the Gibbs transition function, $\mathbf{x}$ and $\mathbf{x}'$ can differ only in one position. Let that position be i. Then for $\mathbf{x}' = x_i'$ we have

$$\begin{aligned} P(\mathbf{x})T(\mathbf{x}'|\mathbf{x}) &= \frac{1}{n}P(\mathbf{x})T(x_i'|\mathbf{x}) \\ &= \frac{1}{n}P(\mathbf{x})\frac{P(x_i', \mathbf{x}_{-i})}{\sum_{x_i'} P(x_i', \mathbf{x}_{-i})} \\ &= \frac{1}{n}P(\mathbf{x})\frac{P(\mathbf{x}')}{\sum_{x_i'} P(x_i', \mathbf{x}_{-i})} \\ &= P(\mathbf{x}')T(\mathbf{x}|\mathbf{x}') \end{aligned}$$

Hence, notice that $P(\mathbf{x})$ is the stationary distribution under Gibbs Sampling.

Now, a task is to choose A. We need to design A to satisfy **DBE** for $x \neq x'$.

$$\pi(x)T^Q(x \to x')A(x \to x') = \pi(x')T^Q(x' \to x)A(x' \to x)$$

$$A(x \to x') = \min\left[1, \frac{\pi(x')T^Q(x' \to x)}{\pi(x)T^Q(x \to x')}\right] \tag{90}$$

Given a desired stationary distribution $P(\mathbf{x})$, designing the $A(\cdot)$ just requires the user provided T^Q and the ratio of probabilities $\frac{P(\mathbf{x}')}{P(\mathbf{x})}$. The proof that the above equation works is not tough to see. It is easy to see that either $\pi(x')T^Q(x' \to x) \geq \pi(x)T^Q(x \to x')$ or $\pi(x')T^Q(x' \to x) < \pi(x)T^Q(x \to x')$. In either case if $A(x \to x') < 1$, then $A(x' \to x) = 1$. Hence, just substituting with this result in **DBE** gives the required result.

Example 98. Say we want a stationary distribution

$$\pi = [\pi_1, \pi_2, \pi_3] = \left[\frac{2}{Z}, \frac{3}{Z}, \frac{2}{Z}\right]$$

In this, it is trivial to see $Z = 7$, but in more complex cases, it might not be. So we leave it as Z. Say we choose an arbitrary T^Q and compute A. Say we have a uniform one, i.e

$$T^Q(x \to x') = \frac{1}{3}$$

We can see that

$$A(1 \to 2) = \min\left[1, \frac{\pi(2)T^Q(1|2)}{\pi(1)T^Q(2|1)}\right] = \min\left[1, \frac{3}{2}\right] = 1$$

Similarly, $A(2 \to 3) = 2/3$.

Langevin Monte-Carlo

Sampling from an arbitrary differential function, say a neural network representing $P(\mathbf{x})$. For example in audio, images. We write

$$P(\mathbf{x}) = \frac{e^{-E_\theta(\mathbf{x})}}{Z} \tag{91}$$

where $E_\theta(\mathbf{x}) \mapsto \mathbb{R}$ is an arbitrary differentiable function in $\mathbf{x}$ and Z_θ is intractable to compute. Given two $\mathbf{x}$ and $\mathbf{x}'$, it is easy to compute the ratio

$$\frac{P(\mathbf{x})}{P(\mathbf{x}')} = \frac{e^{-E_\theta(\mathbf{x})}}{e^{-E_\theta(\mathbf{x}')}} \tag{92}$$

To design $T^Q(\mathbf{x}'|\mathbf{x})$, we use the intuition that transitioning from any $\mathbf{x}$ to $\mathbf{x}'$ along directions of maximum increase in $\log P(\mathbf{x})$ by using gradients.

$$\mathbf{x}' = \mathbf{x} + \tau\nabla_{\mathbf{x}} \log P(\mathbf{x}) = \mathbf{x} - \tau\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) \tag{93}$$

To make the transitions probabilistic, we add small Gaussian noise.

$$\mathbf{x}' = \mathbf{x} - \tau\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) + \sqrt{2\tau}\xi \tag{94}$$

where $\xi \sim \mathcal{N}(\mathbf{0}, I_d)$. With this, we can write

$$T^Q(\mathbf{x} \to \mathbf{x}') = \frac{1}{\sqrt{2\pi 2\tau}} \exp\left(-\frac{1}{4\tau}\|\mathbf{x}' - \mathbf{x} + \tau\nabla_{\mathbf{x}} E_\theta(\mathbf{x})\|^2 \right) \tag{95}$$

$$A(\mathbf{x} \to \mathbf{x}') = \min\left\{1, \frac{e^{-E_\theta(\mathbf{x}')}T^Q(\mathbf{x}' \to \mathbf{x})}{e^{-E_\theta(\mathbf{x})}T^Q(\mathbf{x} \to \mathbf{x}')}\right\} \tag{96}$$

Generative Adversarial Networks

It is seen that PixelCNNs define a tractable density function like BNs, optimizing

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i|x_1, \cdots, x_{i-1}) \tag{97}$$

and VAEs define an intractable density function with latent $\mathbf{z}$ as

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz \tag{98}$$

We optimize the lower bound on likelihood in this case. If we want the ability to just sample and not want an explicit modeling density, we can use GANs.

GANs don't work with any explicit density function but instead take a game theory approach and learn to generate from a training distribution through a 2-player game.

We want to sample from a complex, high dimensional training distribution. Although there is no direct way to do this, we can solve the problem by sampling from a simple distribution like random noise and learning transformation to training distribution. This complex transformation can be represented using a neural network. The two players are

- **Generator:** Tries to fool the discriminator by generating real-looking images
- **Discriminator:** Tries to distinguish between real and fake images

We train both jointly in a minimax game.

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right] \tag{99}$$

where $D_{\theta_d}(\cdot)$ is the discriminator output for data and $G_{\theta_g}(\cdot)$ is the generated data. The discriminator wants to maximize the objective such that $D(x)$ is close to 1 and $D(G(z))$ is close to 0 while the generator wants to minimize objective such that $D(G(z))$ is close to 1. The solving ov the following problem is alternatively, first we maximize entire objective over θ_d and second we minimize the second term over θ_g. Thus we do gradient ascent on discriminator but gradient descent on generator. Theoretically this works, but in practice the convergence is slow. Hence, an alternate objective is taken for the generator which is maximized, i.e

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z))) \tag{100}$$

Algorithm 11: GAN training algorithm

Gaussian Processes

The major applications for Gaussian processes are

1. Bayesian Regression where we have a joint distribution over multiple predictions
2. Optimizing functions for which gradients are not available such as hyper-parameter optimization of deep models

In normal regression, we have for $\mathbf{x} \in \mathbb{R}^d$, say we have $f(\mathbf{x}) = \sum_i w_i x_i + b$ where $w_1, \cdots, w_d, b$ are parameters. We can try to claim $y \sim \mathcal{N}(f(\mathbf{x}), \sigma^2)$ where σ is independent of $\mathbf{x}$. In this case, we can train the parameters using MLE.

Now say for $d = 1$, say given y for the given x, we will for two points give independent predictions in the above case. But with Gaussian process, over the two points, we give the mean prediction value and the joint distribution over the values of the two points. In Gaussian process, we have a distribution over functions and not just a single function. Thus we have a mean, and will have a variance which varies with $\mathbf{x}$.

Definition 99 (Gaussian Processes)**.** In a Gaussian process, we assume that for $\mathcal{X} \subseteq \mathbb{R}^d$ being the space of inputs, we pick N inputs $x^1, \cdots, x^N$ and for these, the values of the function $f(x^1), \cdots, f(x^N)$ are random variables and we would like to get

$$\mathbb{P}(\begin{bmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^N) \end{bmatrix}) \sim \mathcal{N}(\mu, \Sigma) \tag{101}$$

where elements of the covariance matrix are given as $\Sigma_{ij} = \kappa(x^i, x^j)$ where $\kappa(\cdot)$ is a kernel function. This can also be written as for $f(x^i) = y_i$, let $\mathbf{y} = [y_1, \cdots, y_n]^\top$. Then

$$\mathcal{N}(\mathbf{y}, \mu, \Sigma) = \frac{1}{(2\pi|\Sigma|)^{\frac{k}{2}}} e^{-\frac{1}{2}(\mathbf{y}-\mu)^\top \Sigma^{-1}(\mathbf{y}-\mu)} \tag{102}$$

The kernel functions tells us how the value changes from one x to the other. An example would be the RBF kernel given as

$$\kappa(x^i, x^j) \propto e^{-\frac{\|x^i - x^j\|^2}{\ell}} \tag{103}$$

where ℓ is the length parameter. As we decrease ℓ, the correlation decreases.

Properties of Multivariate Gaussians

Say we have one set of Gaussian distributed variables $y_A \sim \mathcal{N}(\mu_A, \Sigma_{AA})$ and $y_B \sim \mathcal{N}(\mu_B, \Sigma_{BB})$. If $y_A \perp\!\!\!\perp y_B$, then

$$y_A + y_B \sim \mathcal{N}(\mu_A + \mu_B, \Sigma_{AA} + \Sigma_{BB}) \tag{104}$$

Say a random variable $Y = [Y_A Y_B]^\top$, where Y is an n-dimensional vector split into two groups. Say $Y \sim \mathcal{N}(\mu, \Sigma)$, which can be written as

$$\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix} \quad \Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}$$

Say we want $P(Y_A | Y_B = O_B)$, then

$$P(Y_A | Y_B = O_B) = \mathcal{N}(\mu_{A|B}, \Sigma_{A|B}) \tag{105}$$

where

$$\mu_{A|B} = \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(O_B - \mu_B) \tag{106}$$
$$\Sigma_{A|B} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA} \tag{107}$$

To calculate the posterior distribution of the function, notice that in many settings, we are interested in $Y = [y_1, \cdots, y_N, y^*]^\top$ found for $[x_1, \cdots, x_N, x^*]$ and we want to calculate $P(y^* | f(x^1) = y_1, \cdots, f(x^N) = y_N, x^*)$. This follows $\mathcal{N}(\mu_*, \overrightarrow{\sigma}^*)$. Denoting $\mathbf{Y} = [y_1, \cdots, y_N]^\top$ and $\mathbf{X} = [x_1, \cdots, x_N]^\top$, we write (assuming $\mu_A = 0$)

$$\mu_* = 0 + \kappa(x^*, \mathbf{X})[\kappa(\mathbf{X}, \mathbf{X})]^{-1}\mathbf{Y} \tag{108}$$
$$\sigma_*^2 = \kappa(x^*, x^*) - \kappa(x^*, \mathbf{X})[\kappa(\mathbf{X}, \mathbf{X})]\kappa(\mathbf{X}, x^*) \tag{109}$$

In the above case, x^* is a single point, but we can extend it to more dimensions. σ^2 will be large if x^* is far from majority of the training data, and small if it is close to them.

Now say we want to get distribution over $x^{N+1}, \cdots, x^{N+M}$ and these are jointly called $\mathbf{X}^*$. Hence we need $P(\mathbf{Y}^* | f(x_1) = y_1, \cdots, f(x_N) = y_N) \sim \mathcal{N}(\mu^*, \Sigma^*)$ where the expressions are same as above, just that the kernel gives matrices instead of scalars.

Hyperparameter Optimization

Say we want to do hyperparameter optimization of a deep model. Denote $\mathcal{X}$ as the space of hyperparameters such as number of layers, vocabulary size, learning rate etc. Given a kernel $\kappa(x^i, x^j)$ and $f(X)$ could be the validation loss/error of the deep model $\mathcal{M}$ with hyperparameters X. It is expensive to search over the entire space and hence we want to find the hyperparameter X for which $f(X)$ is minimum. Our function $f(X)$ is not differentiable.

Say we start with an X, and then the next step can be taken in two ways - either find another X for which the function value is expected to be small, or find an X where there is lot of uncertainty in the function value. Clearly, as GP give a distribution over the values, it can guide in choosing such an X. This is similar to the exploitation-exploration dilemma in reinforcement learning/bandit settings. GP provide control in choosing exploration over exploitation. We choose an acquisition function through which we choose x. A particular one is lowest confidence bound given as

$$a_{LCB}(y_{best}, \mu, \sigma) = \mu - \kappa\sigma \tag{110}$$

where κ is the trade-off between exploitation and exploration. Small $\kappa \implies$ more exploitation and larger κ explores more high variance points.

Algorithm 12: Bayesian Optimization with GP Prior

Time Series Forecasting

Temporal Sequences

They are of two types

1. Regular time-series
   - Daily traffic on individual webpages from different regions in Wikipedia
   - Hourly load on various servers of different services in a data center
   - Monthly demand for products from different regions in a company
2. Irregular event sequences
   - User visits to a music service and the song played
   - Event logs of a system
   - Attacks on a system

Definition 100 (Temporal Sequence)**.** A three-tuple (t_k^i, x_k^i, y_k^i) with t_k^i denoting the time of the k^{th} event of the i^{th} sequence, x denoting the input features and y denoting the value. The sequences are related to each other, and i could span a multi-dimensional space.

One of the major tasks on temporal sequences is probabilistic forecasting (either long-term or short-term). In case of event sequences, we would want to predict the time and type of the event. Other tasks denote outlier detection and missing value imputation.

Probabilistic Forecasting

Early time-series forecasting methods include

1. Statistical time-series methods: ARIMA, Box-Jenkins. They are local and cannot handle features x_j, non-stationarity and interactions.
2. State space models: Kalman Filters. They simplify assumption (linear Gaussian) which may not hold in practice.
3. Classical ML: Regression methods. Domain experts featurize history using wavelets or Fourier coefficients, and powerful regressors such as Boosted regression trees to predict y.

Neural models for time-series forecasting encode history using neural sequence model such as RNNs, CNNs, Transformers with self-attention. In the RNN encoder, say g_i is the hidden state at time-step i, and the inputs to g_i will be $g_{i-1}, y_{i-1}, embd(x_i)$ where $embd(\cdot)$ denotes an embedding and the output will be the next hidden state. In the RNN decoder, the inputs will be $g_{i-1}, y_{i-1}, embd(x_i)$ but the outputs will be g_{i+1} and $\sigma_i, \mu_i = NN(g_4)$, i.e we pass the hidden state through a feedforward neural network and get σ_i and μ_i, and finally write

$$P(y_j|\cdot) \sim \mathcal{N}(\widehat{\mu}_j, \widehat{\sigma}_j^2) \tag{111}$$

We can also use mixture of Gaussians or non-parameterized quantiles. Thus

$$\mathbf{g}_j^i = RNN([y_{j-1}^i, \mathbf{x}_j^i] : j = 1, \cdots, n|\theta_{enc}) \tag{112}$$
$$\mathbf{h}_j^i = FF([\mathbf{g}_j^i, \mathbf{x}_j^i] : j = n, \cdots, n+K|\theta_{dec}) \tag{113}$$
$$\mu_j^i = \theta_\mu[\mathbf{h}_j^i, 1] \tag{114}$$
$$\sigma_j^i = \log(1 + e^{\theta_\sigma[\mathbf{h}_j^i, 1]}) \tag{115}$$
$$\Pr(y_j^i|\mathbf{x}_j^i, (\mathbf{x}_1^i, y_1^i), \cdots, (\mathbf{x}_{j-1}^i, y_{j-1}^i)) = \mathcal{N}(\mu_j^i, \sigma_j^i) \tag{116}$$

Very often single Gaussian is insufficient, we use a mixture of Gaussians. Thus, the last layers outputs k mixture components, means, variance. In MoG, we define the weight of the component p_j and the mean, variance μ_j, σ_j. We define

$$P(y_j^i|H_t, x_j^i) = \sum_{k=1}^{K} p_{jk}^i \mathcal{N}(y_j^i|\mu_{jk}^i, \sigma_{jk}^i) \tag{117}$$

To train the above, we model the full joint distribution and maximize it, i.e

$$\sum_{i:series} \sum_{j:pos} \log P(y_j^i|\theta(x_j^i, (x_1^i, y_1^i), \cdots, (x_{j-1}^i, y_{j-1}^i))) \tag{118}$$

The parameters θ are trained end to end jointly over all series and training is efficient and easily parallelizable.

The limitation of the approach is that it does not output the joint distribution over multiple predictions, it makes both along time and along different items. We address this via GP and Gaussian Copula.

Multivariate Forecasting

The general setup for the problem is that given input features x_t, we want

$$z_1, \cdots, z_T \implies P(z_{T+1}, \cdots, z_{T+\tau}) \tag{119}$$

Multivariate time series is used to refer to forecasting independently values of multiple time series, and we want to model jointly the values of multiple series. If a future value of one series is higher, we would like the other to be higher if positively correlated or lower if negatively correlated. Thus, instead of scalar forecasting, we do

$$\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_T \implies P(\mathbf{z}_{T+1}, \cdots, \mathbf{z}_{T+\tau}) \tag{120}$$

where $\mathbf{z}_j \in \mathbb{R}^N$. A simple LSTM vectorial approach could learn an autoregressive model with an LSTM with state $\mathbf{h}_t$. The inputs will be $z_{1t}, \cdots, z_{Nt}$, and the output will be the normal distribution of next time-step i.e $P(\mathbf{z}_{t+1}|\mathbf{h}_t) = \mathcal{N}(\mu_t, \Sigma_t)$. The multivariate model has transition dynamics as

$$\mathbf{h}_t = \varphi_{\theta_h}(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}) \in \mathbb{R}^k \tag{121}$$

We can factorize the joint distribution as

$$p(\mathbf{z}_1, \cdots, \mathbf{z}_{T+\tau}) = \prod_{t=1}^{T+\tau} P(\mathbf{z}_t|\mathbf{z}_1, \cdots, \mathbf{z}_{t-1}) = \prod_{t=1}^{T+\tau} P(\mathbf{z}_t|\mathbf{h}_t) \tag{122}$$

The training is by minimizing the NLL of the multivariate Gaussian as

$$-\log P(\mathbf{z}_1, \cdots, \mathbf{z}_T) = -\sum_{t=1}^{T} \log P(\mathbf{z}_t|\mathbf{h}_t) \tag{123}$$

The downside of the method is that N is very large, and $\mathbf{h}_t \in \mathbb{R}^k$, thus $\mathcal{O}(Nk)$ parameters. Projecting $\mathbf{h}_t$ to likelihood parameters of $P(\mathbf{z}_t|\mathbf{h}_t)$ has $\mathcal{O}(N^2k)$ parameters.

The issues with this model are

1. Large number of parameters, expensive to compute
2. All time series at training time need to be available during inference
3. Different scales of time series and non-Gaussian data

Low-rank Gaussian Copula Process

The covariance matrix in GP is expressed through a kernel. Say $\mathbf{y}_{i,t} = [\mathbf{h}_{i,t}; \mathbf{e}_i]^\top \in \mathbb{R}^{p \times 1}$ with $\mathbf{e}_i$ as the learned embeddings of the feature, we parameterize a Gaussian process $g_t(\mathbf{y}_{i,t})$ as

$$g_t \sim GP(\widetilde{\mu}(\cdot), k(\cdot, \cdot)) \text{ with } k(\mathbf{y}, \mathbf{y}') = \mathbb{1}_{\mathbf{y}=\mathbf{y}'}\widetilde{d}(\mathbf{y}) + \widetilde{\mathbf{v}}(\mathbf{y})^\top \widetilde{\mathbf{v}}(\mathbf{y}') \tag{124}$$

where $\mu_i(\mathbf{h}_{i,t}) = \widetilde{\mu}(\mathbf{y}_{i,t})$, $d_i(\mathbf{h}_{i,t}) = \widetilde{d}(\mathbf{y}_{i,t})$ and $\mathbf{v}_i(\mathbf{h}_{i,t}) = \widetilde{\mathbf{v}}(\mathbf{y}_{i,t})$. Unlike before, now we can process each time series separately, with each one giving as output the μ_{it}, d_{it}, v_{it}. The covariance matrix is given as

$$\Sigma(\mathbf{h}_t) = \text{diag}(d_i(\mathbf{h}_i, t)) + \begin{bmatrix} \mathbf{v}_1(\mathbf{h}_{1,t}) \\ \vdots \\ \mathbf{v}_N(\mathbf{h}_{N,t}) \end{bmatrix} \begin{bmatrix} \mathbf{v}_1(\mathbf{h}_{1,t}) \\ \vdots \\ \mathbf{v}_N(\mathbf{h}_{N,t}) \end{bmatrix}^\top = D_t + V_t V_t^\top \tag{125}$$

where $D_t \in \mathbb{R}^{N \times N}$, $V_t \in \mathbb{R}^{N \times r}$ which has $\mathcal{O}(N \times r)$ parameters. Also, $V_t V_t^\top$ is a low-rank matrix with rank hyperparameter $r \ll N$. Thus, our low-rank likelihood evaluation is $\mathcal{O}(Nr^2 + r^3)$.

Often, data doesn't look Gaussian and we use a Gaussian copula.

$$C(F_1(z_1), \cdots, F_d(z_N)) = \phi_{\mu,\Sigma}(\Phi^{-1}(F_1(z_1)), \cdots, \Phi^{-1}(F_N(z_N))) \quad (126)$$

F_i is the empirical CDF of z_{it}.

Long-Horizon Forecasting

In this section, we review the paper

Coherent Probabilistic Aggregate Queries on Long-horizon Forecasts

Short-term forecasting typically deals with forecasting few tens of values or lesser while long-term forecasting deals with few hundred or thousands of values. The latter task is more challenging because of computational limitations and because we have to model dependencies over long range in both history and forecast-horizon. In general it is often useful to look at aggregated values of a window in a forecast horizon - such as looking at monthly forecasts for daily sales data. Aggregations can also be of different types, such as trend or difference of sum. As we choose higher level of windows in aggregation, noise cancels out. Distribution is forecasted at a base level and we can also aggregate such base-level distributions to obtain an aggregate.

Auto-regressive models suffer from drift caused by cascading errors (in RNN - during prediction, the predicted y values are fed back which may be erroneous). Also, computing aggregate distributions using auto-regressive models require repeated sampling steps which is computationally expensive. Non auto-regressive models (NAR) offer an efficient way to calculate all values and has been shown to work well in practice. The limitation to NAR models is that it is difficult to capture top-level patterns when the time-series contains noise.

The idea of the paper is to use a NAR model to efficiently compute long-range probabilistic forecasts, and aggregate forecasts to guide potentially noisy base-level forecasts to more accurate forecasts, and finally establish coherency between the base-level and aggregate forecasts. The setup in the base-level is as follows: given H_T as $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_T, y_T)$ where $\mathbf{x}_t \in \mathbb{R}^d$, we need to predict y_j in the forecast horizon given as $(\mathbf{x}_{T+1}, y_{T+1}), \cdots, (\mathbf{x}_{T+R}, y_{T+R})$. In aggregation, the j^{th} value of the i^{th} aggregate series as

$$z_j^i = \mathbf{a}^i \mathbf{y}_{w_i,j} = \sum_{r=1}^{K_i} a_r^i y_{r+(j-1)K_i} \quad (127)$$

The average of a window

$$a_r^i = \frac{1}{K_i} \implies z_j^i = \sum_{r=1}^{K_i} \frac{1}{K_i} y_{r+(j-1)K_i} \tag{128}$$

The trend aggregate (1D regression) has

$$a_r^i = \frac{r}{K_i} - \frac{K_i+1}{2K_i} \implies z_j^i = \sum_{r=1}^{K_i} \left(\frac{r}{K_i} - \frac{K_i+1}{2K_i} \right) y_{r+(j-1)K_i} \tag{129}$$

Hence, essentially we have taken weighted average with constant-fixed weights.

The forecasting model is a transformer based architecture with a warm start window between encoder and decoder input to provide a context to the decoder so that it learns better representation. Say our dummy input is $(\mathbf{x}_i, y_i)_{i=1}^4$ and we want to predict y_5, y_6. Thus our encoder input will be $(\mathbf{x}_i, y_i)_{i=1}^4$, warm start window will be $(\mathbf{x}_i, y_i)_{i=3}^4$ and the decoder input will be $(\mathbf{x}_i, 0)_{i=5}^6$. All of these are passed through a convolution back applied on a small window to extract representations that can be fed to the transformer. Then, the encoder and window+decoder is passed through separate multi-head self attention layers followed by encoder-decoder cross attention layer which outputs $(\widehat{\mu}_i, \widehat{\sigma}_i)_{i=5}^6$. For each aggregate, a separate forecast model is trained and

$$\widehat{P}(z_j^i, H_T, \mathbf{x}_j) \sim \mathcal{N}(\widehat{\mu}(z_j^i), \widehat{\sigma}(z_j^i)) \tag{130}$$

Since all aggregates are trained independently, the forecast distributions across aggregates are incoherent. In order to get the coherence, a new consensus distribution $Q(\cdot, \cdot)$ is inferred over base-level forecasts as

$$Q \sim \mathcal{N}(\mu, \Sigma) \text{ where } \mu = [\mu_{T+1}, \cdots, \mu_{T+R}]^\top \tag{131}$$

and Σ is the covariance matrix of the joint distribution. With this tractable form, we can compute the marginal distribution for aggregate variable z_j^i as

$$Q_j^i = \mathcal{N}(\mu_{w_i,j}^\top \cdot \mathbf{a}^i, \mathbf{a}^{i^\top} \cdot \sum_{w_i,j} \mathbf{a}^i) \tag{132}$$

Now, for coherency, the KL distance between Q and $\widehat{P}$ is minimized as

$$\min_{\mu,\Sigma} \sum_{i \in \mathcal{A}} \sum_{j=T_i}^{T_i+R_i} \alpha_i D_{KL}\Big(Q_j^i(z_j^i | \mu, \Sigma) \| \widehat{P}(z_j^i | \cdot) \Big) \tag{133}$$

The KL distance between two Gaussians is given as (parameterized by p and q, i.e $D_{KL}(\mathcal{N}_q \| \mathcal{N}_p)$)

$$\frac{(\mu_q - \mu_p)^2 + \sigma_q^2}{2\sigma_p^2} - \log \frac{\sigma_q}{\sigma_p} - \frac{1}{2} \tag{134}$$

$$\begin{aligned} RHS &= D_{KL}\Big(\mathcal{N}(\mu_{w_i,j}^\top \mathbf{a}^i, \mathbf{a}^{i^\top} \sum_{w_i,j} \mathbf{a}^i) \| \mathcal{N}(\widehat{\mu}(z_j^i), \widehat{\sigma}(z_j^i)) \Big) \\ &= \frac{\Big(\mu_{w_i,j}^\top \mathbf{a}^i - \widehat{\mu}(z_j^i) \Big)^2 + \Big(\mathbf{a}^i, \mathbf{a}^{i^\top} \sum_{w_i,j} \mathbf{a}^i \Big)^2}{2(\widehat{\sigma}(z_j^i))^2} - \log \frac{\mathbf{a}^{i^\top} \sum_{w_i,j} \mathbf{a}^i}{(\widehat{\sigma}(z_j^i))^2} \end{aligned}$$

Substituting the above expression as D_{KL}, we get the complete objective function. We can split it into 2 optimization problems. The first one is

$$\min_{\mu} \sum_{i \in \mathcal{A}} \sum_{j=T_i}^{T_i+R_i} \frac{1}{(\widehat{\sigma}(z_j^i))^2} \left(\mu_{w_{i,j}}^\top \mathbf{a}^i - \widehat{\mu}(z_j^i) \right)^2 \tag{135}$$

can be solved in closed form. The second one is

$$\min_{\Sigma} \sum_{i \in \mathcal{A}} \sum_{j=T_i}^{T_i+R_i} \frac{\left(\mathbf{a}^i, \mathbf{a}^{i\top} \sum_{w_{i,j}} \mathbf{a}^i \right)^2}{2(\widehat{\sigma}(z_j^i))^2} - \log \mathbf{a}^{i\top} \sum_{w_{i,j}} \mathbf{a}^i \tag{136}$$

has R^2 parameters in Σ and cannot be solved in closed form. Hence to efficiently solve for Σ, we use low-rank aproximation as

$$\widehat{\Sigma} = \operatorname{diag}(\sigma_{T+i}^2)_{i=1}^R + \begin{bmatrix} v_{T+1} \\ \vdots \\ v_{T+R} \end{bmatrix} \begin{bmatrix} v_{T+1} \\ \vdots \\ v_{T+R} \end{bmatrix}^\top \tag{137}$$

where $v_{T+r} \in \mathbb{R}^k$. This has $\mathcal{O}(R)$ parameters and can be stored purely in form of diagonal and v vectors.

For training, the large time-series is split into chunks of size $T + R$. The training objective is given as (for θ^i to be the parameters of the i^{th} model)

$$\max_{\theta_i} \sum_{(x_j^i, \mathbf{z}_j^i)} \sum_{t=T_i+1}^{T_i+R_i} \log \mathcal{N}(z_t; (\mu_t, \sigma_t) = F(H_T, \mathbf{x}, t | \theta^i)) \tag{138}$$

The evaluation metrics for this task would be

1. Mean Absolute Error
2. Mean Square Error
3. Continuous Ranked Probability Score given by

$$\begin{aligned} \Lambda_\alpha(q, y_t) &= (\alpha - \mathcal{I}_{[y_t < q]})(y_t - q) \\ \mathrm{CRPS}(F_t^{-1}, y) &= \int_0^1 2\Lambda_\alpha(F^{-1}(\alpha), y_t) d\alpha \end{aligned} \tag{139}$$