Interim Report of Undergraduate Thesis

SPN and Its Application

Author: Yilin ZHENG

11510506@mail.sustech.edu.cn

Southern University of Science and Technology

April 3, 2019

Abstract

Sum-product network(SPN) is a new kind of probabilistic graphical model(PGM) first proposed in 2011. The classical PGMs suffers some weaknesses such as larger computation burden and intractability of the inference. SPN uses a very different syntax and semantics to lower the computations and make the inference tractable. In this report, we will review the background of PGMs and give out high-level motivation. Then we will give a general review on SPN with its properties as well as the learning and inference on it. Later we will describe our experiment of reproducing the image completion as the goal of this project and the related experiment program. At last, as an interim report, we will summarize the current progress.

Contents

Nomenclature

$\boldsymbol{\lambda}$ Vector of indicator variables

$\emptyset$ Empty set

λ Indicator variables

$\boldsymbol{\Theta}$ Parameters in PGMs

ance(N) Ancestors of node N

chi(N) Children of node N

C Child node

decs(N) Descendants of node N

E Set of edges

F Father node

nance Non-ancestors of node N

ndesc Non-descendants of node N

nei(N) Neighbours of node N

N Node in SPN, either sum node or product node

par(N) Parents of node N

pa(N) Paths to node N

P Product node in SPN

$\mathbf{sc}(N)$ Scope of node N

$\mathbf{S}$ Sum node in SPN

$\mathbf{val}(X)$ Image of random variable X, set of its values

$\mathbf{V}$ Set of nodes

$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ Random variables

$\mathcal{A}$ Sigma-algebra

$\mathcal{B}$ Bayesian network

$\mathcal{D}$ Dataset

$\mathcal{G}$ Graph, directed or undirected graph, structure of PGMs

$\mathcal{L}$ Likelihood

Ω Sample space

ω Elementary event

$\overline{X}$ Negation of random variable X for binary variables

$\Phi(\mathbf{X})$ Parameters of random variables $\mathbf{X}$

A Events

E Edge

l Length of a path

P Probability distribution

p Probability mass function(PMF)

V Vertex or node

x, y, z Values of random variable

Chapter 1

Introduction

This thesis is to implement an SPN and reproduce the results of the application on image completion for further application. SPN is a newly proposed deep architecture in [1], which is a kind of probabilistic graphic model. Probabilistic graph models(PGMs) are a type of probabilistic tool to model the probability distribution, enabling us to assess and analyze the random quantities. By using probabilistic models, we can do reasoning and make decisions with uncertainty. This chapter will roughly summary the concepts of probability theory and graph theory as a background in Section 1.1, since PGMs combine both probability and graph, and deduce to a good motivation finally. This section is adapted from [2]. In Section 1.2, the motivation of this project will be stated. In Section 1.3, the targets of this thesis are proposed, following with the roadmap in Section 1.4.

1.1 Background

1.1.1 Probability Theory

In this part, we will describe the basic formalism of *probability space*, *Sigma-algebra*, *conditional probability*, *Bayes' rule*, and *random variables*(RVs). RVs are the most important and interesting object in probabilistic modelling like machine learning, so, some concepts dealing with RVs will also be presented like *expectations*, *marginal* and *conditional distributions*.

Sigma-algebra and Probability Space To describe the probability in a mathematical way, let's first introduce some symbols representing the components of probability theory. To describe the randomness, we first need a space containing all the possible events, which is called *sample space*, denoted as Ω. For any elementary event in the space, denoted as ω, we pick it randomly, which is regarded as a random trial. We can find that for any trials such as $\omega = \omega_1$, $\omega = \omega_2$ and $\omega_1 \neq \omega_2$, the elementary events are mutually exclusive. To assign the probability to the randomness, we can say, probability "0" means "impossible" while probability "1" means "certain". However, considering the probability of a random trial makes no sense, so we further define a set of elementary events $A(\omega \in A)$, called event, and assign the probability to the event. So far, according to our definition, all the possibilities of all the elementary event $\omega \in A$ sum up to be the probability of event A, and for any sub-events of A, the sum of all the sub-events are equal to the probability of the event A. Besides, the probability of our sample space Ω is 1. If we use a function P to measure the probability, we have

$$P(A) = \sum_i \omega_i > 0 \quad \text{for} \quad \forall \omega_i \in A$$

$$P(A) = \sum_i A_i > 0 \quad \text{for} \quad \forall A_i \in A$$

$$P(\Omega) = 1$$

This definition is accessible for discrete case but will be tough for continuous situation, so we then introduce simga-algebra.

Definition 1.1 (Sigma-algebra) *For Ω be any set, a sigma-algebra $\mathcal{A}$ over Ω is a closed system containing Ω, complements and countable unions. The sigma-algebra has following properties:*

1. $\Omega \in \mathcal{A}$
2. $A \in \mathcal{A} \implies A^c \in \mathcal{A}$
3. $A_n \in \mathcal{A}, \forall n \in \mathbb{N} \implies \cup_{n \in \mathbb{N}} A_n \in A$

Example Let Ω be the event of a die, which is $\Omega = \{1, 2, 3, 4, 5, 6\}$. One of the sigma-algebra over Ω can be $\mathcal{A} = \{\emptyset, \{1, 3, 5\}, \{2, 4, 6\}, \Omega\}$.

After we have sigma-algebra, we can further define the probability space.

Definition 1.2 (Probability Space) *A triple $(\Omega, \mathcal{A}, P)$ is defined as a probability space where:*

1. $\Omega \neq \emptyset$
2. $\mathcal{A}$ *is a sigme-algebra over* Ω.
3. P *is a probability function mapping* $\mathcal{A} \mapsto \mathbb{R}$.

Under probability space, we can now further think about more of the probability distribution.

Conditional Probability, Bayes' Rule and Independence Now we can easily define conditional probability.

Definition 1.3 (Conditional Probability) *Given a probability space $(\Omega, \mathcal{A}, P)$, let A, B be events where $A, B \in \Omega$, the conditional probability is described as:*

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(AB)}{P(B)}$$

, where $P(B) > 0$.

and if we exchange the position of A and B and combine both, we can finally get the Bayes' rule.

Definition 1.4 (Bayes' Rule)

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

.

Definition 1.5 (Independence of Events) *In a probability space $(\Omega, \mathcal{A}, P)$, if events $A \in \mathcal{A}$ and $B \in \mathcal{A}$ are independent, then for their probability, there exists the relation:*

$$P(AB) = P(A)P(B)$$

.

To introduce random variable, we need to first introduce measurable space and measurable function.

Definition 1.6 (Measure Space) *Given Ω is any sey, $\mathcal{A}$ is a sigma-algebra over Ω, a tuple $(\Omega, \mathcal{A})$ is a measurable space. We define a function f: $\mathcal{A} \mapsto [0, \infty]$ with the properties:*

1. $f(\emptyset) = 0$
2. $f(\cup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} f(A_i), \forall A_i \in A, i \neq j \Leftrightarrow A_i \cap A_j = \emptyset$.

. We say f is a measure on $(\Omega, \mathcal{A})$. The triple $(\Omega, \mathcal{A}, f)$ is called a measure space.

Compared to probability space, we can know that probability is a special measurable space with constraint that $P(\Omega) = 1$.

Definition 1.7 (Measure Function) *Let $(\Omega_1, \mathcal{A}_1)$, $(\Omega_2, \mathcal{A}_2)$ are two measurable space. Define a function f: $\Omega_1 \mapsto \Omega_2$ which is called $\mathcal{A}_1 - \mathcal{A}_2$-measurable.*

Definition 1.8 (Random Variable) *Given a probability space $(\Omega, \mathcal{A}, P)$, a random variable X defined on the probability space $(\Omega, \mathcal{A}, P)$ is a $\mathcal{A} - \mathcal{B}(\mathbb{R})$-measurable function X: $\Omega \mapsto \mathbb{R}$. We use $\mathbf{val}(X)$ to represent the set of all the values of X, defined as the image of Ω under X.*

Definition 1.9 (Distribution of Random Variables) *Given a random variable X defined on the probability space $(\Omega, \mathcal{A}, P)$, we can define the distribution of the random variable X:*

$$P_X(B) = P(X^{-1}(B)), \text{ for } B \in \mathcal{B}(\mathbb{R})$$

.

Definition 1.10 (Probability Mass Function) *Given a RV X defined on the probability space $(\Omega, \mathcal{A}, P)$, the probability mass function(PMF) is defined as:*

$$p_X(x) = P_X(x) = P_X(X = x), x \in \mathbf{val}(X).$$

Definition 1.11 (Joint Probability Distribution) *Given a probability space* $(\Omega, \mathcal{A}, P)$*,* X_n *be n RVs defined on the probability space,* $X_n : \Omega \mapsto \mathbb{R}^N$*, with* $\mathbf{X}(\omega) = (X(\omega_1), \ldots, X(\omega_n))^T$*, we have the joint probability distribution:*

$$P_{\mathbf{X}}(B) = P(\mathbf{X}^{-1}(B)), \text{ for } B \in \mathcal{B}(\mathbb{R}^N)$$

.

Expectation of an RV is a very common concepts used to assess the distribution of the RV.

Definition 1.12 (Expectation) *Given a random variable* X *of distribution* p_X*, we define the expectation of the* X *in discrete and continuous respectively:*

$$\mathbb{E}(X) = \begin{cases} \sum_{x \in \mathbf{val}(X)} x p_X(x), & x \text{ is discrete} \\ \int_{x \in \mathbf{val}(X)} x p_X(x) \mathbf{d}x, & x \text{ is continuous} \end{cases} \tag{1.1}$$

1.1.2 Graph Theory

Probabilistic graphical models are graphs to model the joint distribution of random variables. This part will summarize some concepts from graph theory as another background.

Graph, Directed/Undirected Graph To describe some definition of graph theory, we need to introduce some symbols usually used in the definitions. Let's use $\mathcal{G}$ to denote a graph, either a directed or an undirected graph, with $\mathbf{V}$, $\mathbf{E}$ denoting the vertices and edges in the graph respectively.

Definition 1.13 (Graph) *A graph* $\mathcal{G}$ *is defined as a tuple* $(\mathbf{V}, \mathbf{E})$ *consisting of two sets* $\mathbf{V}$ *denoting nodes or vertices and* $\mathbf{E}$ *denoting edges.*

The nodes in a graph can contain any useful information, such as the probability of a variable, the order between all other nodes or the reachability of a path. For edges in a graph, we focus the direction of the edge, for an example, edge (V_i, V_j), if the order of both nodes matters, we said the edge is directed, otherwise we recognize both edge (V_i, V_j), (V_j, V_i) are contained in $\mathbf{E}$.

Definition 1.14 (Directed Graph) *A graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is a directed graph if all the edges $(V_i, V_j) \in \mathbf{V} (i \neq j)$ are directed.*

In a directed graph $\mathcal{G}$, $(V_i, V_j)(i \neq j)$ and (V_j, V_i) are regarded as two different edges in $\mathbf{V}$.

Definition 1.15 (Undirected Graph) *A graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is an undirected graph if all the edges $(V_i, V_j) \in \mathbf{V} (i \neq j)$ are undirected.*

Neighbours, Parents and Children Given a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, we can further define the neighbours, parents, and children.

Definition 1.16 (Neighbours) *Neighbours are defined in undirected graphs. Given an undirected graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, in a edge (V_i, V_j), we said V_i is the neighbour of V_j and vice versa. We use $\mathbf{nei}(V)$ to denote the neighbour of vertex V.*

Definition 1.17 (Parents and Children) *Parents and children are defined in directed graphs to distinguish the direction of the edge. Given a directed graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, in a edge (V_i, V_j), also denoted as $(V_i \to V_j)$, we said V_i is a parent of V_j, and V_j is a child of V_i. We use $\mathbf{par}(V)$ to denote the parents of vertex V and $\mathbf{chi}(V)$ to denote the children of vertex V.*

Path, Trial and Cycles Edges and vertices can make form path, trial and cycles in graphs.

Definition 1.18 (Path and Trial) *A path $\mathbf{pa}_n$ is defined as a tuple containing n nodes $\{V_1, V_2, \ldots, V_n\}$. We said $\mathbf{pa}_n = \{V_1, V_2, \ldots, V_n\}$ is a path from V_1 to V_n for each edge $(V_i, V_{i+1}) \in \mathbf{E}$ $(i = 1, \ldots, n-1)$. In a directed graph, the tuple $\{V_1, V_2, \ldots, V_n\}$ is defined as a trail between V_1 and V_n if for each edge $(V_i \to V_{i+1})$ or $(V_{i+1} \to V_i) \in \mathbf{E}$ $(i = 1, \ldots, n-1)$.*

Definition 1.19 (Cycles) *A node V in a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ always has a path of length 1 from V to itself. Further, we define that if there is a path of length $l \geq 2$ from node V to itself, then there is a cycle in the graph passing the node V.*

Directed Acyclic Graphs, Ancestor and Descendant This part we will review the concepts of DAGs, ancestor and descendant.

Definition 1.20 (Directed Acyclic Graphs) *Given a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, it is defined as a directed acyclic graph(DAG) if there are no cycles.*

Definition 1.21 (Ancestor and Descendant) *In a DAG, we can define that in a path from $V_i \in \mathbf{V}$ to $V_j \in \mathbf{V}$, V_i is the ancestor of V_j, and V_j is one of the descendants of V_i. We use* $\mathbf{ance}(V)$ *and* $\mathbf{decs}(V)$ *to represent the ancestors and descendants of node V respectively. For non-ancestor and non-descendant, we use* **nance** *and* **ndesc**.

Rooted DAG, Directed/Undirected Tree We than summary the definition of rooted DAG, and directed/undirected tree.

Definition 1.22 (Rooted DAG) *A DAG is a rooted DAG if there exists a unique node V_1 with all other nodes $V_i (i \neq 1)$, all the paths from V_1 to V_i have the length $l \geq 1$ but all the paths from V_i to V_1 have the length of $l = 0$.*

Definition 1.23 (Undirected Tree) *An undirected graph is an undirected graph tree.*

Definition 1.24 (Directed Tree) *A rooted DAG is a directed tree if for all nodes V_i that the paths from V_i to itself are of length $l \leq 1$.*

Subgraph, Supergraph and Induced Graph This part we review the concepts of sub-graph, super-graph and induced graph.

Definition 1.25 (Subgraph and Supergraph) *A graph $\mathcal{G}' = (\mathbf{V}', \mathbf{E}')$ is a subgraph of graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ if $\mathcal{G}'$ is formed by the vertex set $\mathbf{V}' \subseteq \mathbf{V}$ and the edge set $\mathbf{E}' \subseteq \{(V_i, V_j) | i \neq j, V_i, V_j \in \mathbf{V}\}$. Then the grapg $\mathcal{G}$ is called supergraph of $\mathcal{G}'$.*

Definition 1.26 (Induced Graph) *An induced graph $\mathcal{G}' = (\mathbf{V}, \mathbf{E})$ is graph formed by the set $\mathbf{V}' \subseteq \mathbf{V}$ and the edges set $\mathbf{E}' \subseteq \mathbf{E}$.*

The difference between the subgraph and the induced graph is whether the edges are contained in the original graph. A subgraph may contain edges that not exist in the original graph, while an induced graph can only contain the subset of edges in the original graph.

Cliques, Complete Graph Last, we review the definitions of cliques in undirected graphs and complete graph.

Definition 1.27 (Clique) *Given an undirected graph* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$*, we define a set of nodes* $\mathbf{C} \subseteq \mathbf{V}$ *as a clique such that between all pairs of distinct nodes, there exists an edge. In mathematical formalism, a clique is:*

$$\forall V_i, V_j \in \mathbf{C} \subseteq \mathbf{V} \ (i \neq j), (V_i, V_j) \in \mathbf{E}$$

.

Further, we can define the maximal clique as a clique which becomes no longer a clique if adding a node $V \in \mathbf{V} \setminus \mathbf{C}$.

Given the definition of clique, we can derive the definition of complete graph.

Definition 1.28 (Complete Graph) *An undirected graph* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *is a complete graph if* $\mathbf{V}$ *is a clique itself, which means that for any pairs of distinct nodes* $V_i, V_j \in \mathbf{V} \ (i \neq j)$*, the edge* $(V_i, V_j) \in \mathbf{E}$*.*

1.1.3 Classical Probabilistic Graphic Models

Probabilistic graphical models(PGMs) are a combination of probability theory and graph theory. In this part, we will briefly review the popular classical PGMs, Bayesian networks, and Markov random fields followed by the learning and inference methods on these classical PGMs. PGMs are trying to use a graph to describe the probability distributions. However, compared to the next section about Sum-product network, regarded as a new PGM, they are very different from syntax and semantics. Besides, classical PGMs suffer some problems such as larger computations and intractability.

If we consider the dependencies between random variables, there are two extreme cases easy to realize: one is full independence between RVs, the other is full dependence between RVs. Here we focus more on the common situations which are conditional independence. PGMs are used to model such situations for avoiding the large computation.

Bayesian Network

Bayesian networks have factorization properties and can model the conditional independence between variables.

Definition 1.29 (Bayesian Network) *A Bayesian network $\mathcal{B}$ over RVs $\mathbf{X}$ is defined as a tuple $(\mathcal{G}, \{p_{\mathbf{X}}\})$ where $\mathcal{G}$ is a DAG over RVs $\mathbf{X}$ and $p_{\mathbf{X}}$ is the conditional probability distribution of RVs $\mathbf{X}$. For every RV $X \in \mathbf{X}$, $p_{\mathbf{X}}$ represents its conditional probability $p(X|\mathbf{par}(X))$. We also called the graph $\mathcal{B}$ forming Bayesian network over RVs as Bayesian structure. A Bayesian network defines the following joint probability distribution:*

$$p_{\mathcal{B}}(X) = \prod_{X \in \mathbf{X}} p(X|\mathbf{par}(X))$$

Example of A Bayesian Network Here we use an example to illustrate the Bayesian network. In Figure 1.1, there are five random variable A, B, C, E, D, with each edge from an RV to another RV indicating their conditional independence. According to the definition of BN, we can easily yield the joint probability distribution:

$$p_{\mathcal{B}}(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|C)p(E|B, C)$$

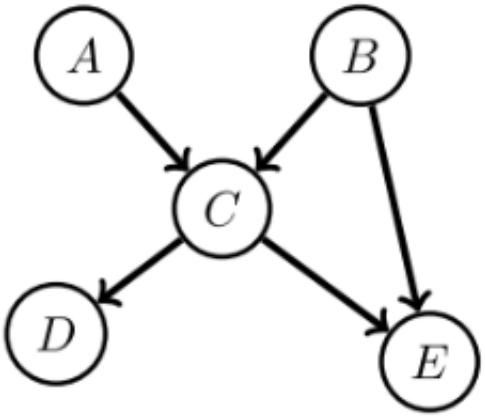

Figure 1.1: Example of a Bayesian Network

Conditional Independence of BNs From the Bayes' rule, we can sum out that conditional independence is connected with the factorization properties in BN structures. For further learning and inference on BNs, conditional independence is the most important part which will decide the efficiency of the processes. Here we refer to the local conditional independencies of BN in [3]:

Definition 1.30 (Local Conditional Independence) *Given a Bayesian network* $\mathcal{B} = (\mathcal{G}, p_X)$*, then the joint conditional distribution* p_X *satisfied the property:*

$$X \perp\!\!\!\perp (Ndesc(X) \setminus \mathbf{par}(X)) | \mathbf{par}(X).$$

We said that RV X *is conditional independent of its non-descendants if given its parents.*

Markov Random Field

Markov random fields, or called Markov networks, also represent the distribution by factorization and conditional independence.

Definition 1.31 (Markov Random Field) *A Markov random field(MRF)* $\mathcal{M}$ *over RVs* $\mathbf{X}$ *is defined as a tuple* $(\mathcal{G}, \{\Psi_{\mathbf{C}_l}\}_{l=1}^{L})$*. In the tuple, the* $\mathcal{G}$ *is an undirected graph with a maximal clique* $\mathbf{C}_1, \ldots, \mathbf{C}_L \subseteq \mathbf{X}$ *and the potential* Ψ_{C_i} *are nonnegative functions over the maximal clique:* $\Psi_{\mathbf{C}_l} : \mathbf{val}(\mathbf{C}_l) \mapsto [0, \infty)$*. The mathematical formalism of MRF is given as:*

$$p_{\mathcal{M}}(\mathbf{X}) = \frac{1}{\mathcal{Z}_{\mathcal{M}}} \Phi_{\mathcal{M}}(\mathbf{X}) = \frac{1}{\mathcal{Z}_{\mathcal{M}}} \prod_{l=1}^{L} \Psi_{\mathbf{C}_l}(\mathbf{X}_{\mathbf{C}_l})$$

where $\mathcal{Z}_{\mathcal{M}}$ *is a normalization factor:*

$$\mathcal{Z}_{\mathcal{M}} = \int_{\mathbf{val}(X_1)} \cdots \int_{\mathbf{val}(X_N)} \Phi_{\mathcal{M}}(X_1, \ldots, X_N) \mathbf{d}X_1 \ldots \mathbf{d}X_N$$

for continuous variables or

$$\mathcal{Z}_{\mathcal{M}} = \sum_{X_1}^{X_N} \Phi_{\mathcal{M}}(X_1, \ldots, X_N)$$

for discrete variables.

We also call normalization factor $\mathcal{Z}_{\mathcal{M}}$ *the partition function and call* $\Phi_{\mathcal{M}}$ *the unnormalized MN. The graph encoding the distribution is called MN structure.*

Example of MRF We also give a simple example of MRF. Figure 1.2 is a MRF with five variables X_1, X_2, X_3, X_4, X_5, assume that RVs are discrete. Apparently, there two cliques circle in red or blue dotted line respectively:

$$\begin{aligned} \mathbf{C}_1 &= \{X_1, X_2, X_4\} \text{ and} \\ \mathbf{C}_2 &= \{X_2, X_3, X_5\}. \end{aligned} \tag{1.2}$$

According to the definition, we can yield the probability distribution:

$$p_{\mathcal{M}}(X_1, X_2, X_3, X_4, X_5) = \frac{1}{\mathcal{Z}} \Psi_{\mathbf{C}_1}(X_1, X_2, X_4) \Psi_{\mathbf{C}_2}(X_2, X_3, X_5)$$

and the partition function:

$$\mathcal{Z}_{\mathcal{M}} = \sum_{\mathbf{val}(X_1)}^{\mathbf{val}(X_5)} \Phi_{\mathcal{M}}(x_1, x_2, x_3, x_4, x_5).$$

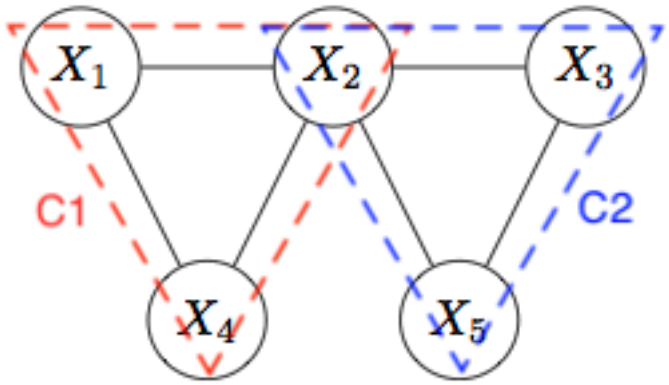

Figure 1.2: Example of a Markov Random Field

Conditional Independence of MRF To review the conditional independence of MRF, we need to introduce the separation in MRF.

Definition 1.32 (Separation in MRF) *Given a MRF $\mathcal{M} = (\mathcal{G}, \mathbf{X})$, RVs $X_i, X_j \in \mathbf{X}$ $(i \neq j)$ are separated by $\mathbf{Z} \subset \mathbf{X}$ $(X_i, X_j \notin \mathbf{Z})$ if all paths between X_i and X_j are passed the RVs $X_k \in \mathbf{Z}$. Similarly, any two subsets $\mathbf{X}_1, \mathbf{X}_2 \subset \mathbf{X}$ are separated by $\mathbf{Z}$ if following conditions are met:*

1. $\mathbf{X}_1 \cap \mathbf{Z} = \emptyset$ *and* $\mathbf{X}_2 \cap \mathbf{Z} = \emptyset$,

2. $\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$,

3. $\forall X \in \mathbf{X}_1, X' \in \mathbf{X}_2$ *are separated by* $X_z \in \mathbf{Z}$.

Definition 1.33 (Conditional Independence of MRF) *Conditional independence of MRF can be summarized into three properties:*

1. ***Local Markov Property.*** *A random variable X is conditionally independent from all other nodes except its neighbours when given its neighbour. The mathematical formalism is:*

$$X \perp\!\!\!\perp (\mathbf{X} \setminus (X \cup \mathbf{nei}(X))) | \mathbf{nei}(X).$$

2. ***Pairwise Markov Property.*** *If any two nodes X_i and X_j, $i \neq j$, are non-adjacent in an MRF, then they are conditionally independent given all other nodes excluding themselves. Formalism is:*

$$X_i \perp\!\!\!\perp X_j | (\mathbf{X} \setminus X_i, X_j), i \neq j \text{ and } (X_i, X_j) \notin \mathbf{E}.$$

3. ***Global Markov Property.*** *Given any two sets $\mathbf{X}_1, \mathbf{X}_2 \subset \mathbf{X}$, if they are separated by $\mathbf{Z} \subset \mathbf{X}$, and all these sets are mutually disjoint, then $\mathbf{X}_2, \mathbf{X}_2$ are conditionally independent. This property can be formalized as:*

$$\mathbf{X}_1 \perp\!\!\!\perp \mathbf{X}_2 | \mathbf{Z},\ \mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset,\ \mathbf{X}_1 \cap \mathbf{Z} = \emptyset,\ \text{and } \mathbf{X}_2 \cap \mathbf{Z} = \emptyset.$$

Learning

This section will roughly discuss the methods of constructing a model for a probability distribution. PGMs are used to model the joint distribution of data via combing both probability theory and graph theory. For a very simple probability distribution, a feasible way might construct a BN by hand since BNs are more accessible for its syntax and semantics. However, in the machine learning field, we are going to learn PGMs automatically, which means the learning process contains both learning the structure and learning the parameters by some algorithms. Generally, there are two common approaches in machine learning: discriminative learning and generative learning.

Generative Learning Generative learning approach gives out the model by learning the process of generating the data. It tries to find the way how the data can be generated, which leads to model the joint probability distribution directly. In principle, the marginal and conditional probability can be derived from the joint probability distribution.

Discriminative Learning Discriminative learning approach learns the joint distribution indirectly. In common situations, the data or samples used to learning PGMs are drawn from the real distribution which might not be modelled exactly. So, the discriminative learning approach focus on the final goal of modeling the joint distribution, which is to solve some practical problems such as classification.

For learning PGMs, there are two key parts learned: structure and parameters. **Structure learning** is to learn the structure of RVs, which is to find the relations or associations between RVs. One approach to learn the structure is minimum description length(MDL) principle proposed in [4], which is to find the shortest and most compact representation of data $\mathcal{D}$.

Parameter learning, however, is to learn the weight or the values of RVs in the model. Maximum a-posterior(MAP) approach is a method used to learning parameters. MAP gives out the model according to the likelihood $\mathcal{L}$ of the model given data $\mathcal{D}$.

If we use $\mathcal{G}$ to represent the structure of the PGM and use $\mathbf{\Theta}$ to denote the parameters, combining the learning approaches mentioned above, we can have various learning methods, either learning the structure or the parameter using either generative learning or discriminative learning approach, even hybrid learning methods are possible.

Inference

After PGMs are constructed, the following process to perform reasoning on the model is called inference. This part will summarize the common inference scenarios.

Given a probability distribution p_X over RVs $\mathbf{X}$, we perform the following inference:

Marginalization Given certain query RVs $\mathbf{X}^q \subset \mathbf{X}$, we want to calculate the marginal distribution $p(X^q)$, which is to marginalize $\mathbf{X} \setminus \mathbf{X}^q = \mathbf{X}^m = \{X_1^m, X_2^m, \ldots, X_K^m\}$. The

mathematical formalism is:

$$p(\mathbf{X}^q) = \int_{\mathbf{val}(X_1^m)} \cdots \int_{\mathbf{val}(X_N^m)} p(\mathbf{X}^q, X_1^m, X_2^m, \ldots, X_K^m) \mathbf{d}X_1 \ldots \mathbf{d}X_K$$

for continuous RVs, and

$$p(\mathbf{X}^q) = \sum_{X_1^m}^{X_K^m} p(\mathbf{X}^q, X_1^m, X_2^m, \ldots, X_K^m)$$

for discrete RVs.

Conditions Given RVs $\mathbf{X} = \mathbf{X^q} \cup \mathbf{X}^o \cup \mathbf{X}^m$, the inference goal is to compute the posterior distribution of the query $\mathbf{X}^q$ conditioned on the observation $\mathbf{x}^o \in \mathbf{X}^o$. And the formalism is:

$$p(\mathbf{X}^q | \mathbf{x}^o) = \frac{p(\mathbf{X}^q, \mathbf{x}^o)}{p(\mathbf{x}^o)}$$

where $p(\mathbf{X}^q, \mathbf{x}^o)$ and $p(\mathbf{x}^o)$ are determined by the marginalization $\mathbf{X}^m$ and $\mathbf{X}^m \cup \mathbf{X}^q$.

Most Probable Explanation MPE is a process to find the most probable explanation of query RVs $\mathbf{X}^q$ given the observed RVs $\mathbf{X}^o$, where $\mathbf{X} = \mathbf{X}^q \cup \mathbf{X}^o$. To find the MPE, we should compute:

$$\mathbf{x}^{q*} = \arg\max_{\mathbf{X}^q \in \mathbf{val}(\mathbf{X}^q)} p(\mathbf{x}^q | \mathbf{x}^o) = \arg\max_{\mathbf{X}^q \in \mathbf{val}(\mathbf{X}^q)} p(\mathbf{x}^q, \mathbf{x}^o)$$

.

Maximum A-Posterior MAP is a process similar to MPE but ignores $\mathbf{X}^m$. The RVs $\mathbf{X}$ are still split into query RVs $\mathbf{X}^q$, observed RVs $\mathbf{X}^o$ and marginalized RVs $\mathbf{X}^m$. Here we will compute:

$$\begin{aligned}
\mathbf{x}^{q*} &= \arg\max_{\mathbf{X}^q \in \mathbf{val}(\mathbf{X}^q)} p(\mathbf{x}^q | \mathbf{x}^o) = \arg\max_{\mathbf{X}^q \in \mathbf{val}(\mathbf{X}^q)} p(\mathbf{x}^q, \mathbf{x}^o) \\
&= \arg\max_{\mathbf{X}^q \in \mathbf{val}(\mathbf{X}^q)} \int_{\mathbf{val}(X_1^m)} \cdots \int_{\mathbf{val}(X_K^m)} p(\mathbf{x}^q, x_1^m, x_2^m, \ldots, x_K^m) \mathbf{d}x_1 \ldots \mathbf{d}x_K \text{ (continuous)} \\
&= \sum_{\mathbf{x}_1^m}^{\mathbf{x}_K^m} p(\mathbf{x}^q, x_1^m, x_2^m, \ldots, x_K^m) \text{ (discrete).}
\end{aligned} \tag{1.3}$$

These inferences are all NP-hard according to [3]. The goal of targeting the tractable models finally leads to the design of the algorithms adapted to the complexity of PGMs. One feasible solution is to relax the aim of exact inference which uses the factorization properties, and swift to model an approximate inference.

1.2 Motivation

In the previous section, we review the probability and graph theory as well as the classical PGMs so as to raise a high-level motivation in this section.

Classical PGMs suffers some weaknesses. The classical PGMs separate the inference from learning since inference is a sub-process of learning actually. This separation often causes the exact learning improper. Besides, using approximate learning to construct the PGMs has some disadvantages [2]:

1. Too many trials due to the unknown what kind of approximate learning is correct.
2. Even we find the correct approximate learning and the right algorithm, the results will still be unpredictable since the learning itself is not exact.
3. The approximate learning is a trade-off of time and accuracy. This approach turns the original difficulty to the difficulty of finding a good trade-off, which is still hard.

To avoid the problems caused by approximate learning, we should reconsider the exact learning and find the tractable models. However, in most time, a tractable classical PGM is either sparsely connected or too simplistic which cannot give out a good representation of data. Then in [1], Poon and Domingos proposed Sum-Product Network(SPN), which is a tractable model contains only sum and product operations. In Chapter 2 we will introduce SPN in details.

1.3 Target

This section will state the target of this project and gives out the expectation of final results. The original target is a new application of SPN to cancer dataset as a representation learning problem. Then to adjust to the requirement of the undergraduate thesis,

the target is changed to reproduce the application of image completion via Poon's architecture in [1]. The original target is optional if time permits. The final results contain the source code of the experiment program, the reproduction of image completion on SPN, the analysis of the results, and a paper as an undergraduate thesis.

1.4 Roadmap

The roadmap is a clear way to show this project on what stages will be conducted in order. The roadmap of this project including paper reading for concepts learning and reviewing, coding for the experiment program, reproducing the experiment, optimization of the program, and the analysis of the results, with the final stage of final paper writing and presentation preparation. The optional target is not set into the whole roadmap since the timeline might not permit and the requirements of graduation are more important.

Chapter 2

Sum-Product Network

In Chapter 1, we mentioned the classical PGMs and pointed out the weaknesses which lead us to think about what characteristics of PGMs should have for better representation of data. A feasible solution is to design the models with less conditional independencies among RVs [2]. The mixture of distributions is proposed to support this model. In the mixture of distributions, RVs are augmented by marginalized latent RVs. Sum-Product Network(SPN) is such type of PGM. In this chapter, we will first review the important concepts, evidence, and network polynomials in refsec:2.1, then further introduce the definition of SPN with its key properties in Section 2.2, Section 2.3 respectively. In Section 2.4, we will discuss the representation of SPN. Last in Section 2.5, we will finally summarize the learning and inference in SPN.

2.1 Evidence and Network Polynomials

Definition 2.1 (Evidence) *An evidence is interpreted as the* $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ *over RVs* $\mathbf{X}$ *with values assigned. If each* $\mathbf{x} \in \mathbf{val}$ *is assigned a value, then we call it complete evidence. Given a subset of* $\mathbf{Y} \in \mathbf{X}$*, if we have complete evidence* $\mathbf{y}$*, then we can evaluate the probability of the partial evidence by marginalizing* $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$*. Typically, to evaluate the probability of evidence, is to compute* $p(\mathbf{x})$*.*

In [5], network polynomials is introduced to describe the distribution over finite-state RVs and then generalized in [1] for unnormalized distributions.

Definition 2.2 (Network Polynomial) *A network polynomial of a network $\mathcal{N}$ over RVs $\mathbf{X}$ is defined as a polynomial function mapping the network $\mathcal{N}$ to RVs $\mathbf{X}$.*

Given a finite-state RVs $\mathbf{X}$, we can define the indicator variable(IV) $\lambda_{\mathbf{X}=x} \in \mathbb{R}$ to represent the state of RVs. IVs can be assigned values for corresponding RVs' state. The vector form of IV is donated by $\boldsymbol{\lambda}$.

For Bayesian network, the network polynomials in Darwiche [5] are:

$$f_{\mathcal{B}}(\mathbf{x}) = \sum_{\mathbf{x} \in \mathbf{val}(\mathbf{X})} \prod_{X \in \mathbf{X}} \lambda_{\mathbf{X}=\mathbf{x}[X]} \theta^{\mathbf{X}}_{\mathbf{x}[X],[\mathbf{par}(X)]}$$

where $p_{\mathcal{B}}(\mathbf{x}) = \prod_{X \in \mathbf{X}} \theta^{\mathbf{X}}_{\mathbf{x}[X],[\mathbf{par}(X)]}$ is the probability distribution of BN.

For unnormalized distribution, the network polynomials in [1] are defined as:

$$f_{\Phi}(\boldsymbol{\lambda}) = \sum_{\mathbf{x} \in \mathbf{val}(\mathbf{X})} \Phi(\mathbf{x}) \prod_{X \in \mathbf{X}} \lambda_{\mathbf{X}=\mathbf{x}[X]}$$

where $\Phi(\mathbf{x})$ is the probability distribution with the conditions that $\forall \mathbf{x} \in \mathbf{val}(\mathbf{X}), \Phi(\mathbf{x}) \geq 0$ and $\exists \mathbf{x} \in \mathbf{val}(\mathbf{X}), \Phi(\mathbf{x}) > 0$.

2.2 Definition

Before we introduce the definition of SPN, we will mention the arithmetic circuits(AC) since it has a compact representation of the network polynomials and it contains more operations than the SPN.

Definition 2.3 (Arithmetic Circuit) *An arithmetic circuit is a rooted acyclic directed graph with its numeric inputs and the internal arithmetic operation nodes. The arithmetic operations includes addition($+$), subtraction($-$), multiplication($\times$), and division($\div$). The values are computed in the root.*

Figure 2.1 is an example of AC with two variables X_1, X_2, and a constant 1. From the definition we can derive the network polynomial of this AC:

$$f(X_1, X_2, 1) = (X_1 + X_2)X_2(X_2 + 1).$$

which is computed in the product node.

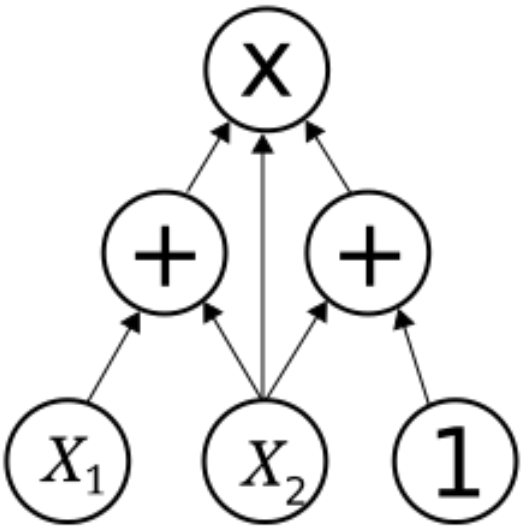

Figure 2.1: Example of an arithmetic circuit

In [1], Poon and Domingos proposed Sum-Product Network(SPN) as a new type of PGMs to overcome the weakness of intractability of classical PGMs. SPNs is an undirected acyclic graph over finite-state RVs with numeric inputs and the only two operations sum and product in the internal nodes.

Definition 2.4 (Sum-Product Network) *A Sum-product network $\mathcal{S}$ over finit-state RVs X is a tuple $(\mathcal{G}, \Phi(\mathbf{X}))$ where $\mathcal{G}$ is a rooted DAG and $\Phi(\mathbf{X})$ is a set of nonegative parameters. In SPN, leaves are numeric input represented by indicator variables, the internal nodes are sum nodes or product nodes which appear alternatively, and the root is always sum node. The edges between nodes are weights or parameters which will be learned during parameter learning process.*

Network Polynomials of SPN The network polynomials of SPN are similar to that in AC. We can easily formalize the form:

$$f_{\mathcal{S}} = \sum_{X \in \mathbf{X}} \Phi(X) \prod_{X \in \mathbf{X}} (X).$$

Example of SPN Figure 2.2 is a simple example of SPN with three RVs X_1, X_2, X_3 and their negations $\overline{X}_1$, $\overline{X}_2$, and $\overline{X}_3$. With the parameters in the edges, we can yield the

network polynomials:

$$
\begin{aligned}
&f_{\mathcal{S}}(X_1, X_2, X_3, \overline{X}_1, \overline{X}_2, \overline{X}_3) = \\
&\quad 0.95X_1(0.1X_2 + 0.9\overline{X}_2)(0.5X_3 + 0.5\overline{X}_3) + 0.05(0.2X_2 + 0.8\overline{X}_2)(0.3X_3 + 0.7\overline{X}_3)\overline{X}_1 \\
&= (0.95 \times 0.1 \times 0.5)X_1X_2X_3 + (0.95 \times 0.1 \times 0.5)X_1X_2\overline{X}_3 \ldots ...
\end{aligned}
\tag{2.1}
$$

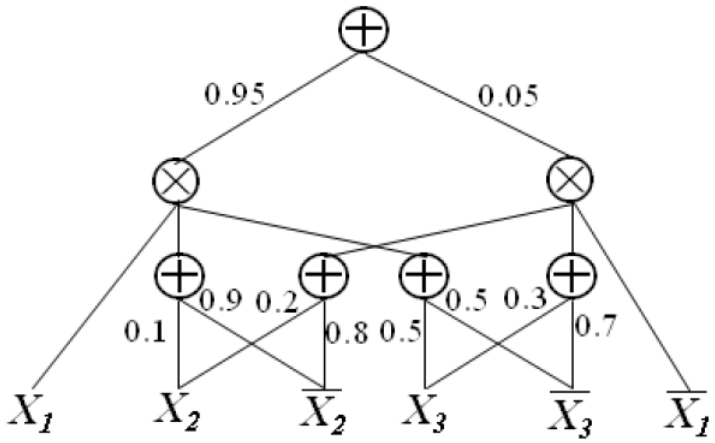

Figure 2.2: Example of a Sum-Product Network

Definition 2.5 (Scope of Node) *Given a SPN* $\mathcal{S} = (\mathcal{G}, \Phi(\mathbf{X}))$ *over RVs* $\mathbf{X}$*, for a node* $N \in \mathbf{N}$*, we define the scope of node* N*, denoted as* $\mathbf{sc}(N)$*:*

$$
\mathbf{sc}(N) \begin{cases} \{X\} & \text{if } N \text{ is a leaf} \\ \cup_{\mathbf{C} \in \mathbf{chi}(N)} \mathbf{sc}(\mathbf{C}) & \text{if } N \text{ is an internal node} \end{cases}
\tag{2.2}
$$

2.3 Properties

This section we will review the properties of SPN defined in [1].

Theorem 2.6 (Completeness) *A sum node* $\mathbf{S}$ *is complete if and only if all its children have the same scope. A sum-product network* $\mathcal{S}$ *is complete if and only if all its sum nodes are complete.*

Theorem 2.7 (Consistency) *A product node* $\mathbf{P}$ *is consistent if and only if there is no negation of a variable in* $\mathbf{P}$ *appears in one child of another product node* $\mathbf{P}'$*. A sum-product network* $\mathcal{S}$ *us consistent if and only if all its product node are consistent.*

Theorem 2.8 (Validity) *A sum-product network* $\mathcal{S}$ *is defined to be validate if and only it is both complete and consistent.*

Besides, we said an SPN is valid means that it always computes the correct probability of evidence.

Theorem 2.9 (Decomposability) *A product node* $\mathbf{P}$ *is decomposable if and only if there is no variables appear in more than one scope of its children. A sum-product network* $\mathcal{S}$ *is decomposable if and only if all its product nodes are decomposable.*

Completeness requires an SPN has the same scope for the children of the same sum node. Consistency requires that each variable and its negation should be in the same scope of a product node. Decomposability is to constrain the scopes of the children of the same product node no overlap.

Example of a Valid and Decomposable SPN Figure 2.3 [1] is a valid and decomposable SPN.

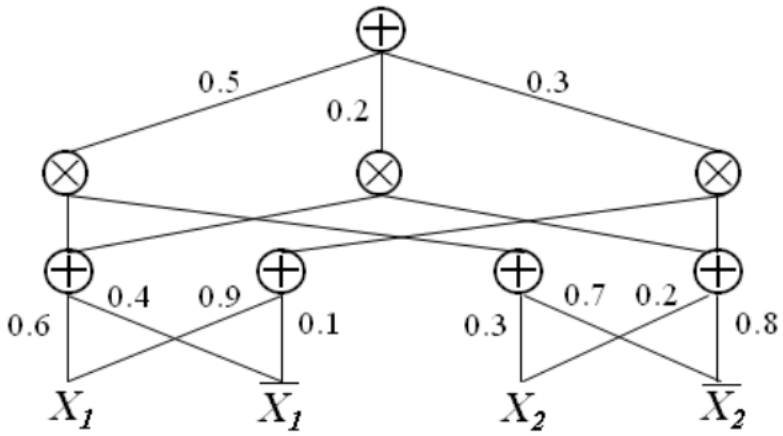

Figure 2.3: Example of a Valid and Decomposable SPN

2.4 Differential Approach of Inference

If we set the indicator variables(IVs), we can compute the marginalization via network polynomials, then we can interpret the probability distribution through the derivatives of the network polynomials. This process is called differential approach of inference.

Given an SPN $\mathcal{S}$, assume the nodes $\mathbf{N}$. If $\mathbf{N}$ is root, then we have the derivative:

$$\frac{\partial \mathcal{S}}{\partial \mathbf{N}}(\boldsymbol{\lambda}) = 1$$

.

If $\mathbf{N}$ is an internal node, then the derivative should be

$$\frac{\partial \mathcal{S}}{\partial \mathbf{N}}(\boldsymbol{\lambda}) = \sum_{\mathbf{F} \in \mathbf{pa(N)}} \frac{\partial \mathcal{S}}{\partial \mathbf{F}} \frac{\partial \mathbf{F}}{\partial \mathbf{N}}(\boldsymbol{\lambda})$$

Internal nodes have two types. If $\mathbf{F}$ is a sum node, which is

$$\mathbf{F} = \sum_{\mathbf{C} \in \mathbf{pa(F)}} \phi_{\mathbf{F},\mathbf{C}} \mathbf{C}(\boldsymbol{\lambda}),$$

then

$$\frac{\partial \mathbf{F}}{\partial \mathbf{N}}(\boldsymbol{\lambda}) = \phi_{\mathbf{F},\mathbf{N}}.$$

If $\mathbf{F}$ is a product node, which is

$$\mathbf{F} = \prod_{\mathbf{C} \in \mathbf{pa(F)}} \mathbf{C}(\boldsymbol{\lambda}),$$

then

$$\frac{\partial \mathbf{F}}{\partial \mathbf{N}}(\boldsymbol{\lambda}) = \prod_{\mathbf{C} \in \mathbf{pa(F)} \setminus \{\mathbf{N}\}} \mathbf{C}(\boldsymbol{\lambda}).$$

Usually, given numeric input, we will evaluate the nodes in bottom-up direction, sorting values for each node along the process.

Example of Inference on SPN We still take the SPN in Figure 2.3 for an example. First we gives out its network polynomials:

$$\begin{aligned} f_{\mathcal{S}}(X_1, \overline{X}_1, X_2, \overline{X}_2) =& 0.5(0.6X_1 + 0.4\overline{X}_1)(0.3X_2 + 0.7\overline{X}_2) \\ &+ 0.2(0.6X_1 + 0.4\overline{X}_1)(0.2X_2 + 0.8\overline{X}_2) \\ &+ 0.3(0.9X_1 + 0.1\overline{X}_1)(0.2X_2 + 0.8\overline{X}) \end{aligned} \tag{2.3}$$

And we unfold the brackets which finally gives out:

$$f_{\mathcal{S}}(X_1, \overline{X}_1, X_2, \overline{X}_2) = 0.0168X_1X_2 + 0.522X_1\overline{X}_2 + 0.082\overline{X}_1X_2 + 0.228\overline{X}_1\overline{X}_2 \tag{2.4}$$

If given a complete evidence $e = (X_1, \overline{X}_1, X_2, \overline{X}_2) = (1, 0, 1, 0)$, the network polynomials is computed to be 0.168.

We can also compute the derivatives according to the method mentioned above.

2.5 Learning

SPN has its syntax and semantics largely differ from classical PGMs, and makes itself feasible to incorporate the learning with inference. We call this inference-ware learning. In the case of inference-ware learning, the upper bound of learning is also the upper bound of inference, which indicates that we are possible to get a model with reasonable inference cost by a reasonable learning cost of the model. In Section 2.3 we have mentioned that only a valid SPN can give out the correct probability, and a valid SPN is further constrained by the structure. However, if we have no appropriate parameters, we still cannot obtain satisfying results. In this section, we will review some learning methods on SPN for structure learning and parameter learning.

2.5.1 Structure Learning

Structure learning is to construct the relations or associations among numeric input, internal sum, and product nodes for further conduction of parameter learning. The structure of SPNs are various for the different domain-specific problem, here we only

introduce the Poon-Domingos architecture for rectangle regions in [1], Dennis-Venture architecture for non-rectangle regions in [6], and a Top-down scheme in [7].

Poon-Domingos Architecture

The Poon-Domingos architecture proposed in [1] constructs the SPN by arranging data in rectangles since the application of the SPN is to the image data which are rectangle matrix. First, a single sum node is assigned to the image as the root of the SPN. Second, split the overall image into all possible regions of two-subrectangles along two dimensions. For each region, keep the same number of sum nodes, and then recursively repeat the second step to split more subrectangles as smaller regions. Later, all pairs of sum nodes from two sub-regions are connected as the children of a product node. The last is connecting the product nodes as the children of the sum nodes of their parent rectangles. Apparently, the size of the SPN grows with the size of the image data. The authors proposed to use coarser way for aggregations. The overall structure of the final SPN is a dense structure.

Figure 2.4 shows the architecture of Poon-Domingos architecture.

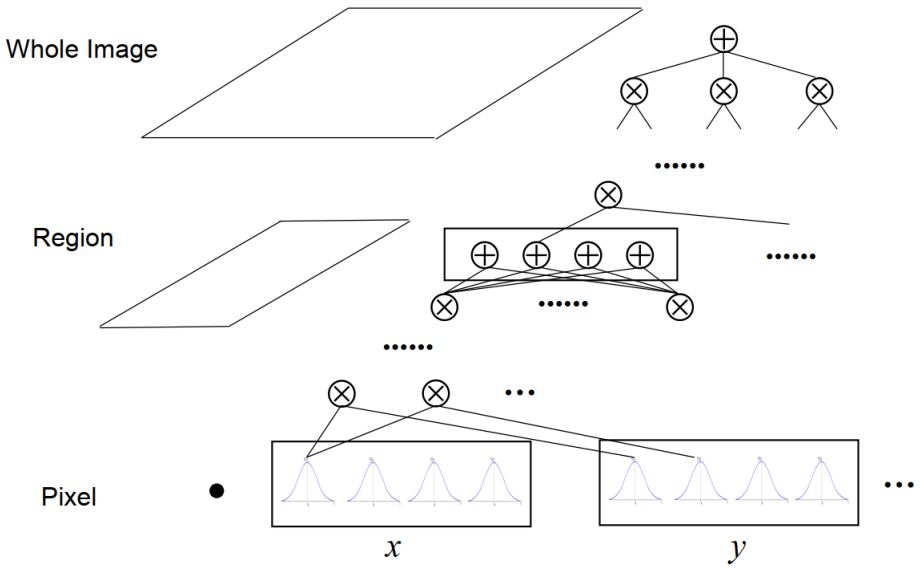

Figure 2.4: Poon-Domingos Architecture

Dennis-Venture Architecture

The Dennis-Venture architecture in [6] is to extend the Poon-Domingos architecture for non-rectangle data. The regions in this architecture are found by a k-mean clustering, which swift the original shape-driven method to data-driven way. By clustering methods, the data are no longer limited to the rectangle shape.

Top-Down Scheme

In [7], Gens et al. proposed a scheme for learning the structure from top to down. The uses probabilistic cluster on data for sum nodes and cluster on RVs for product nodes using independence tests. The scheme will split the input vector of the instance if more than unit length. And then the scheme recursively does so on submatrices which contain fewer rows or columns. If it can split the variables into mutually independent subsets, it recursively does so and finally returns the results as product nodes, otherwise, it will cluster the instances into similar subsets for the recursive split and finally returns the results as sum nodes.

Figure 2.5 shows the process of this scheme.

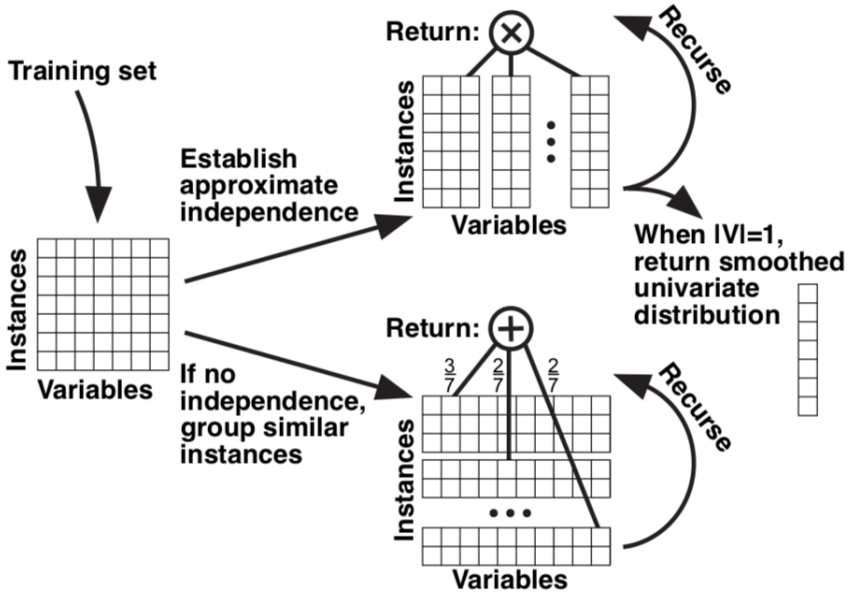

Figure 2.5: Top-Down Scheme

2.5.2 Parameter Learning

Given an SPN with a fixed structure, we further need to learn the parameters or weights. Here we will introduce two common approaches for parameter learning.

Gradient Methods

The first approach is the gradient method adapted from [2]. We have mentioned in Section 2.4 that we can easily derive the derivatives of the SPN from the network polynomials.

Given an SPN $\mathcal{S} = (\mathcal{G}, \Phi(\mathbf{X}))$ over RVs $\mathbf{X}$ and a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$ of L i.i.d. samples. The goal is set to maximize the log-likelihood:

$$\max \quad \log \mathcal{L} = \sum_{l=1}^{L} \mathcal{S}(\mathbf{x}^{(l)}) \tag{2.5}$$

$$\text{s.t.} \quad \sum_{\mathbf{C} \in \mathbf{chi}(\mathbf{S})} \phi_{\mathbf{S},\mathbf{C}} = 1, \forall \mathbf{S} \tag{2.6}$$

$$\phi_{\mathbf{S},\mathbf{C}} \geq 0, \forall \mathbf{S}, \mathbf{C} \in \mathbf{chi}(\mathbf{S}). \tag{2.7}$$

And for any node $\mathbf{N}$, the derivative of the log-likelihood of the l^{th} is:

$$\frac{\partial \log \mathcal{S}}{\partial \mathbf{N}}(\mathbf{x}^{(l)}) = \frac{1}{\mathcal{S}(\mathbf{x}^{(l)})} \frac{\partial \mathcal{S}}{\partial \mathbf{N}}(\mathbf{x}^{(l)}) \tag{2.8}$$

where the derivative $\frac{\partial \mathcal{S}}{\partial \mathbf{N}}$ can be computed by backpropagation.

The derivative w.r.t the sum weights is:

$$\frac{\partial \log \mathcal{S}}{\partial \phi_{\mathbf{S},\mathbf{C}}}(\mathbf{x}^{(l)}) = \frac{\partial \log \mathcal{S}}{\partial \mathbf{S}}(\mathbf{x}^{(l)}) \mathbf{C}(\mathbf{x}^{(l)}) = \frac{1}{\mathcal{S}(\mathbf{x}^{(l)})} \frac{\partial \mathcal{S}}{\partial \mathbf{S}}(\mathbf{x}^{(l)}) \mathbf{C}(\mathbf{x}^{(l)}) \tag{2.9}$$

and the derivative of the log-likelihood of Equation 2.5 is:

$$\frac{\partial \log \mathcal{L}}{\partial \phi_{\mathbf{S},\mathbf{C}}} = \sum_{l=1}^{L} \frac{\partial \log \mathcal{S}}{\partial \phi_{\mathbf{S},\mathbf{C}}}(\mathbf{x}^{(l)}). \tag{2.10}$$

At last, using a appropriate step size η, we can update the parameters by:

$$\phi \leftarrow \phi + \eta \nabla \log \mathcal{L}. \tag{2.11}$$

EM Algorithm

If we interpret the SPN as a latent RVs model, we can use the EM algorithm for parameter learning. In this thesis, we do not review the latent RVs interpretation of SPN so we will not go into the detail of the foundations of the EM process on SPN. In [1], Poon and Domingos first proposed the EM algorithm on SPN but later in [2], Peharz found the incorrection of the original EM algorithm. And therefore here we reference the correct EM algorithm refined by Peharz.

Algorithm 1 shows the EM process on SPN. The symbol $n_{\mathbf{S},\mathbf{C}}$ represents the accumulator variables for finally compute the sum-weights $\phi_{\mathbf{S},\mathbf{C}}$.

Algorithm 1 EM for SPN Weights

Input: An SPN $\mathcal{S}$ with fixed structure

Output: An SPN $\mathcal{S}$ with fixed structure and proper parameters.

Initialize Φ

while not converged **do**

$\forall \mathbf{S} \in \mathcal{S}, \forall \mathbf{C} \in \mathbf{chi}(\mathbf{S}) : n_{\mathbf{S},\mathbf{C}} \leftarrow 0$

for $l = 1, \ldots, L$ **do**

Input $\mathbf{x}^{(l)}$ to $\mathcal{S}$

Evaluate $\mathcal{S}$ (upward-pass)

Backprop $\mathcal{S}$ (backward-pass)

$\forall \mathbf{S} \in \mathcal{S}, \forall \mathbf{C} \in \mathbf{chi}(\mathbf{S}) : n_{\mathbf{S},\mathbf{C}} \leftarrow n_{\mathbf{S},\mathbf{C}} + \frac{1}{\mathcal{S}} \frac{\partial \mathcal{S}}{\partial \mathbf{S}} \mathbf{C} \phi_{\mathbf{S},\mathbf{C}}$

end for

$\forall \mathbf{S} \in \mathcal{S}, \forall \mathbf{C} \in \mathbf{chi}(\mathbf{S}) : n_{\mathbf{S},\mathbf{C}} \leftarrow \frac{n_{\mathbf{S},\mathbf{C}}}{\sum_{\mathbf{C}' \in \mathbf{ch}(\mathbf{S})} n_{\mathbf{S},\mathbf{C}'}}$

end while

return $\mathcal{S}$

Chapter 3

SPN for Image Completion

In this chapter, we will introduce the experiment this project is reproducing. The image completion is originally presented in [1] to verify the validity of the new PGM, SPN. This project aims to reproduce the results for validating the implementation of SPN. In Section 3.1, the hardware of this experiment will be mentioned. In Section 3.2, the dataset used in this experiment will be introduced.

3.1 Experiments

The experiment is originally performed on authors' cluster with 50 cores per node, for our conditions, this experiment can be conducted on the cluster TaiYi with 40 cores per node. Given to the configuration of the our cluster, the experiment will be run under 121 cores for **Caltech** and 81 cores for **Olivetti**, compared to the original 102 and 51 cores, respectively.

3.2 Datasets

This section we roughly introduce the two dataset used in this experiment: Caltech and Olivetti.

Caltech Caltech(`http://www.vision.caltech.edu/Image_Datasets/Caltech101/`) is a dataset containing the pictures of 101 objects. For each category, there are 40 to 800

images. Most of the categories have more than 50 images. The size of the image is about 300×200 pixels. In this experiment, the dataset is rescaled to 100×64 pixels.

Olivetti Olivetti(`https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`) is a dataset containing face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge with each image in size 64×64.

Chapter 4

Experiment Program

In Chapter 2, we reviewed the details of SPN from its definition to the learning and inference methods. SPN has been widely used in various applications: image completions [1], activity recognition [8], language modelling [9], speech modelling [10], facial attributes analysis [11], robot control [12], and other reasoning and inference scenarios. In this chapter, we will move to one of the application this project focus to reproduce: image completions. In Section 4.1, we will briefly mention the Poon-Domingos again since we had introduced in Section 2.5. Section 4.2 will introduce the source code of the experiment program from the code structure and the roles of the modules with the detailed document. Last Section 4.3 will show the callgraph of this program with a simple explanation of the procedure of this experiment program.

4.1 Poon-Domingos Architecture

The Poon-Domingos architecture is a dense SPN. The structure is learned by splitting the rectangle data into two subrectangles recursively and construct the sum nodes and product nodes alternatively. For more details of this architecture, please refer to Section 2.5.

4.2 Document of Code

The experiment program is referenced from the Java program provided by authors on their website: `http://spn.cs.washington.edu/spn/`. In our experiment program, there are still three modules: common, evaluation and spn.

- **common**: Contain some helper functions to provide time management, messaging between progress(MPI), parameter settings for training on clusters, and some utilities.
- **evaluation**: Process the dataset, apply network to datatset to output models, and evaluate the results generated from the models.
- **spn**: Contain the SPN architecture, including node definition, computation functions, the learning, and inference.

For each modules, we provides details document for every source code file. (*.cpp* files are implementations while *.hpp* files store function declarations and class definition.)

- **common**
  1. *my_mpi*: Use OpenMPI to support the messaging in a parallel program. It means that this program will use parallel architecture to accelerate computing.
  2. *parameter*: Control parameters for EM algorithm, SPN, and evaluation.
  3. *timer*: Manage the time to help calculate the time spent on computation.
  4. *utils*: Some helper functions to access time, print log, and do some numeric process.
- **eval**
  1. *dataset*: Read and process data from the dataset.
  2. *eval*: Conduct evaluation over the dataset.
  3. *image_completion*: Conduct image completion, which is the application.
  4. *run*: Control the program, which is the main function.

- **spn**

    1. *decomposition*: Decompose the regions.
    2. *generative_learning*: Conduct the learning process to generative the model.
    3. *instance*: Record the mean and variance of an instance, which is calculated from the dataset.
    4. *node*: Define the node, to provide the base class of the sum node and the product node.
    5. *prod_node*: Define the product node, derived from *node*.
    6. *sum_node*: Define the sum node, derived from *node*.
    7. *region*: Compute the mean and variance of regions in the picture(for this image completion application), as well as the MAP.
    8. *SPN*: Define the Sum-product network, including a root, functions of learning and applications. These functions are implemented via calling other modules.

4.3 Callgraph

The following picture shows the call graph of this program. The program will start from *run*, which reads the arguments from the command line containing the domain we choose, then the program calls the corresponding processes for a specific dataset, the processes including reading data from data folder(*dataset*), setting instance one by one, learning to construct SPN structure(*generative_learning* and *SPN*), performing inference on the SPN(*SPN*), completing images(*image_completion*), and storing models and complete results to folders(*SPN* and *dataset*).

The callgraph is presented in Figure 4.1

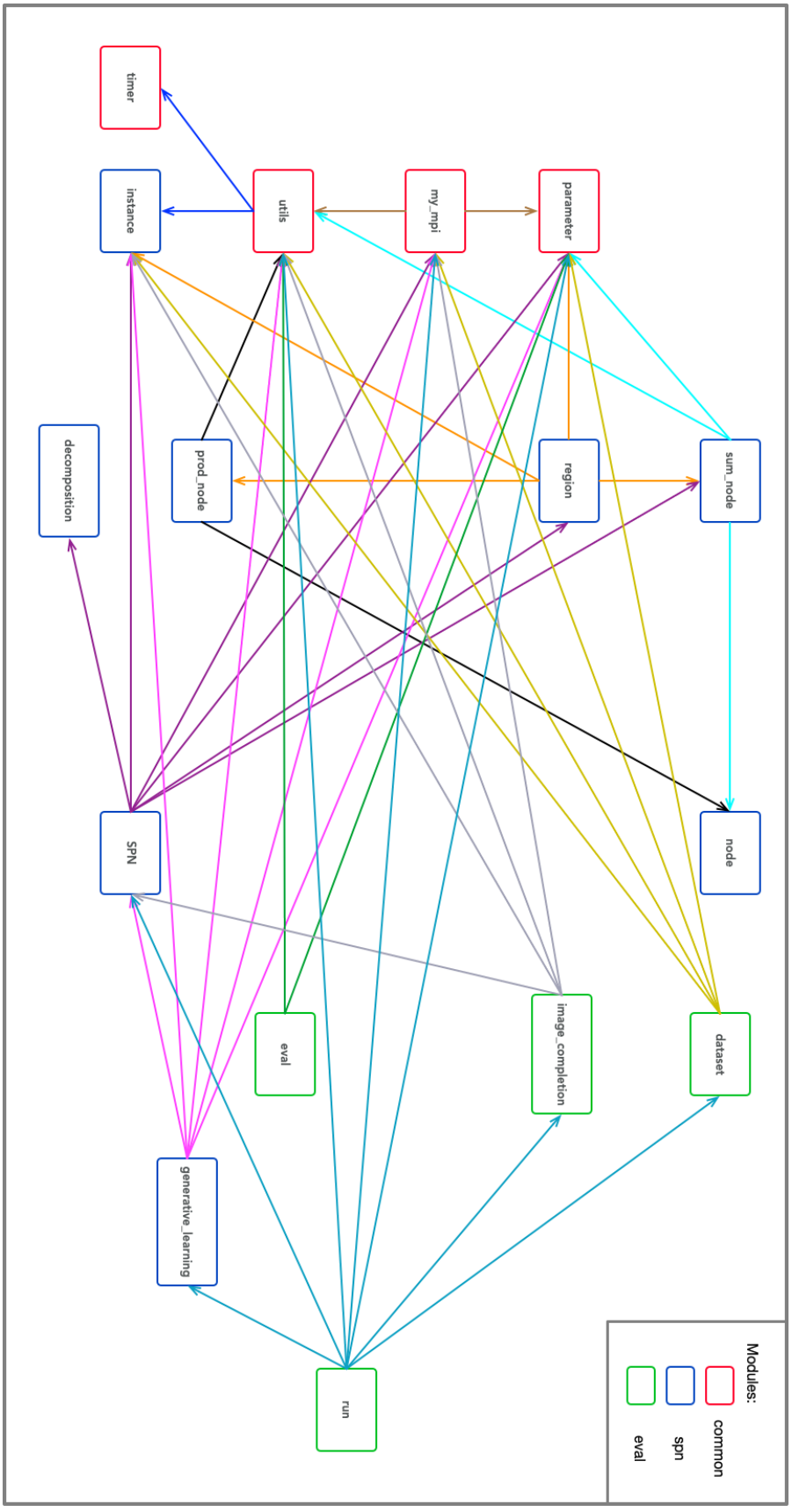

Figure 4.1: Callgraph of SPN

Chapter 5

Progress Feedback

This chapter will summarize the current progress of this project from what had done in Section 5.1, what is processing currently in Section 5.2, and what will be done next in Section 5.3. Also, this progress can be tracked in the GitHub: `https://github.com/users/Spacebody/projects/1`.

5.1 Done

The things done:

1. A brief review of the SPN
2. Opening report and Monthly report till Mar. 2019
3. Implementation of SPN architecture
4. Unit Tests of the Program
5. Interim Report

5.2 Processing

The things currently processing:

1. Debug of learning part of the program

2. Reproduce the results of image completions

3. Draft the thesis

5.3 ToDo

The things to do:

1. Optimization of codes

2. Compare and analysis the results

Here I will roughly explain the current problem I encounter. The program is a parallel program which means more difficult to debug and consumes more time. Currently, I am focusing on the debugging of this program even though I am afraid of the failure which might be an obstacle to my graduation.

References

[1] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, 2011, pp. 689–690.

[2] R. Peharz, "Foundations of sum-product networks for probabilistic modeling," Ph.D. dissertation, PhD thesis, Medical University of Graz, 2015.

[3] D. Koller, N. Friedman, and F. Bach, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[4] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[5] A. Darwiche, "A differential approach to inference in bayesian networks," *Journal of the ACM (JACM)*, vol. 50, no. 3, pp. 280–305, 2003.

[6] A. Dennis and D. Ventura, "Learning the architecture of sum-product networks using clustering on variables," in *Advances in Neural Information Processing Systems*, 2012, pp. 2033–2041.

[7] R. Gens and D. Pedro, "Learning the structure of sum-product networks," in *International conference on machine learning*, 2013, pp. 873–880.

[8] M. R. Amer and S. Todorovic, "Sum product networks for activity recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 4, pp. 800–813, 2016.

[9] W.-C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai, "Language modeling with sum-product networks," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[10] R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf, "Modeling speech with sum-product networks: Application to bandwidth extension," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 3699–3703.

[11] P. Luo, X. Wang, and X. Tang, "A deep sum-product architecture for robust facial attributes analysis," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2864–2871.

[12] A. Pronobis, F. Riccio, and R. P. Rao, "Deep spatial affordance hierarchy: Spatial knowledge representation for planning in large-scale environments," in *ICAPS 2017 Workshop on Planning and Robotics*, 2017.