Perspectives on Probabilistic Graphical Models

DONG LIU

Doctoral Thesis in Electrical Engineering
Stockholm, Sweden 2020

Division of Information Science and Engineering
TRITA-EE XXXX KTH, School of Electrical Engineering and and Computer Science
ISSN .... SE-100 44 Stockholm
ISBN .... SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i Elektroteknik onsdag den xx November 2020 klockan 13.15 i F3, Lindstedtsvägen 26, Stockholm.

Tryck: Universitetsservice US AB

To my beloved

Abstract

Sammanfattning

Acknowledgements

Contents

Acronyms and Notations

Notations

X	random variable
x	realization of the random variable X
$\mathcal{X}$	alphabet of the random variable X
X_i^k	random sequence $(X_i, \ldots, X_k)$
x_i^k	realization of the random sequence X_i^k
$\mathcal{X}_i^k$	alphabet of the random sequence X_i^k
X^k	random sequence $(X_1, \ldots, X_k)$
x^k	realization of the random sequence X^k
$\mathcal{X}^k$	alphabet of the random sequence X^k
$X_i^{k\backslash n}$	random sequence $(X_i, \ldots, X_{n-1}, X_{n+1}, \ldots, X_k)$
$x_i^{k\backslash n}$	realization of the random sequence $X_i^{k\backslash n}$
$\mathcal{X}_i^{k\backslash n}$	alphabet of the random sequence $X_i^{k\backslash n}$
$X^{k\backslash n}$	random sequence $(X_1, \ldots, X_{n-1}, X_{n+1}, \ldots, X_k)$
$x^{k\backslash n}$	realization of the random sequence $X^{k\backslash n}$
$\mathcal{X}^{k\backslash n}$	alphabet of the random sequence $X^{k\backslash n}$
$\lvert\lvert \cdot \rvert\rvert$	set cardinality
f_X	p.d.f. of the continuous random variable X
p_X	p.m.f. of the discrete random variable X
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with mean μ and variance σ^2

$\mathrm{D}(\cdot \Vert \cdot)$	Kullback-Leibler divergence
$\mathrm{D}_\tau(\cdot \Vert \cdot)$	τ-th order Rényi divergence
$\mathrm{C}(\cdot,\cdot)$	Chernoff information
$\mathrm{E}[\cdot]$	expectation
$\partial\cdot$	boundary of a closed set
$\hat{\partial}\cdot$	upper boundary of a two-dimensional closed set
$\check{\partial}\cdot$	lower boundary of a two-dimensional closed set
$\log(\cdot)$	natural logarithm

Chapter 1

Introduction

1.1 Motivations

Most tasks conducted by a person or an automated system requires a fundamental ability of *reasoning*, which is always about reaching a conclusion based on available information. At times, a conclusion is not enough and it is also required to know how reliable the conclusion is. Take the coronavirus (COVID-19) that started at the end of 2019 as example, a doctor needs to check the information about a person to reason if the person is infected by the coronavirus. The relevant information includes symptoms such as fever, cough, breathing difficulties and probably kidney failure in severe cases. Even after the doctor has reached a conclusion of positive or negative infection of coronavirus for the person, a natural question is why and how *confident* the diagnose is made.

Instead of randomly guessing, reasoning is to answer queries with preserving uncertainty, by making best of available information. Two fundamental problems are inevitable before a rational reasoning can be conducted.

- How should we specify the relationship between a conclusion and the available information? In the coronavirus example, the counterpart question to answer is how the doctor should relate coronavirus infection with the symptoms. This step is called *modeling* which represents a reasoning problem abstractly by specifying the relationship between known information and unknown parts, in preparation of answer query on it.

- With the modeled representation, how a conclusion should be made? This process of reaching a answer to the query from an abstracted representation is called *inference*. Assume the doctor knows that the candidate has fever and cough, but not any breathing problem, how likely does the candidate have be infected by coronavirus?

In the modeling and inference process, it is not likely that the very beginning assumptions are perfectly right about the truth and can be used for all instances of the same type of queries. On the contrary, we usually begin with simple (sometimes naive) assumptions of representation a problem, and come back to revise it later when more information or evidence is available, which is aligned with our learning process of new knowledge. In fact, this assumption-and-revision procedure can be more compact. Instead of fixing the model representation at the beginning when one might not be sure the correctness of the assumptions about the model, one can assume a set of models with the assumption that the 'correct' model is in this set. A typical strategy is to leave some freedom to the configuration of the model at the beginning. Each instantiating the configuration generates a model representation. By using available observations or information, the model is adjusted to be able to be best compatible with the observations. This procedure adds the following fundamental problem in reasoning.

- Instead of having a fixed model at the first step, a set of models (or hypothesis) is given. We then need to choose one model that best describes the available observations or information. This phase of choosing a model is called *learning.*

Afterwards, inference can be conducted on the learned model to answer queries.

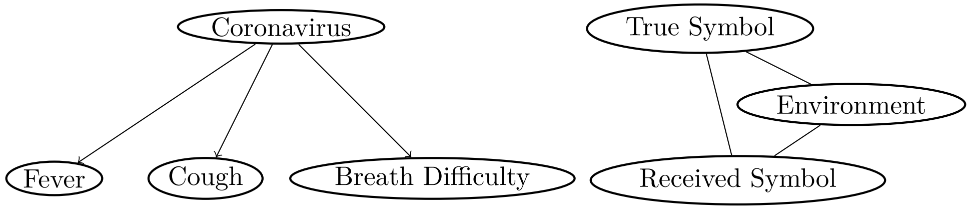
With all the discussed problems above, modeling, inference and learning, our purpose is to carry out reasoning with being aware of how confident a conclusion or answer is. These problems can be treated nicely with probabilistic models via Bayesian logic. Probabilistic model is built on the fundamental calculus of probability theory that is natural to accommodate the *uncertainty*, which is desired in reasoning. In additional, the probabilistic models offers rich space to modeling problems, where inference can be carried on either exactly or approximately. **More importantly, the modeling or modeling learning part is not necessarily coupled with inference algorithms**. This proper separation allows freedom that a certain family of general inference algorithms can be applies to a broad class probabilistic models. It offers the freedom of trying different model representation of a class without the need of replacing inference algorithm.

Example 1. *Consider the coronavirus infection problem. Using probabilistic model, we are able to model the problem in a rigid way. Additionally, we can make query more formally in probabilistic model framework. Assume each symptom among fever, cough and breathing difficulty can take value from* $\{\text{True}, \text{False}\}$*. Also the coronavirus infection is either true or false. One exemplified query can be*

$$P(\text{Infection} = \text{True}|\text{Fever} = \text{True}, \text{Cough} = \text{False}, \text{BreathingDifficulty} = \text{True}),$$

which is asking how likely the patient is infected by coronavirus if symptoms of both fever and breathing difficulty are observed but no sight of cough symptom.

Given the fact that probabilistic theory offers a rigid foundation to model and study the problems, which is used to answer query that we concern, it soon becomes intractable when dozens or hundreds of relevant attributes are joint considered.



(a) A directed graphical model for Example 1.

(b) An undirected graphical model.

Figure 1.1: Different perspectives on probabilistic graphic models. 1.1a A toy Bayesian network. 1.1b A toy Markov random field.

This can be exemplified by giving finer levels of each symptom in coronavirus infection, e.g. symptom fever is represented by the actual body temperature in integer instead of true-or-false binary state on the one hand. On the other hand, there could be more directly and indirectly relevant symptoms such as muscle pain and congestion. Together with the symptoms, the recent travel itinerary is also related. Additionally, season flu could also similarly bring up some symptoms listed above.

Probabilistic graphical model offers a general framework to encode random variable dependency of a complex probabilistic distribution into a structured graph, which is a powerful tool to compactly model relevant attributes and facts of a complex problem or a system. As show in Figure 1.1a that represents the problem of Example 1 into a directed graphical model (also called Bayesian network or generative model), the nodes (or vertexes) correspond to the variables that represents symptoms and infection state, whereas the edges between nodes correspond how one variable may influence others. In certain scenarios, it is natural to use directed graphical models (generative models) to represent that a observable variable is dependent on a latent variable that generates the observations. For instance, a noisy location sensor of a car keeps measuring the car's true location and reading noisy locations.

In contract to the directed graphical model, there are more scenarios that the interaction between related random variables is not directional and an undirected edge is used, which leads to the undirected graphical model (or Markov random field, Section 2.1) representation. Undirected graphical models are popularly used in computer vision, computational biology, digital communication, statistical physics, etc. Figure 1.1b illustrates an exemplified undirected graphical model in digital communication context. On the one hand, a receiver wants to know what is the true symbol by joint considering its communication environment and its received symbol. On the other hand, the symbol received by the receiver is jointly formulated by the true symbol and communication environment. The influence among them is apparently not directional since the impact along an edge can be bidirectional in this example.

Probabilistic graphical model offers a 'scientific language' to do reasoning with

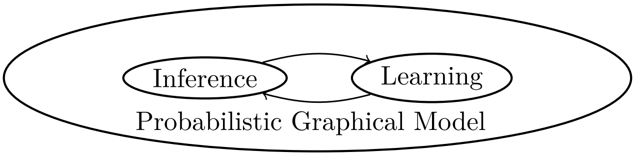


Figure 1.2: Two key aspects in practical graphical models.

uncertainty within framework of probabilistic theory. It is usually a nature representation for a complex system or problem and offers straightforward abstraction. The compact representation of probabilistic graphical model bridges the joint distribution of a complex system, and its graphical abstraction that captures the statistic dependency reflecting our understanding of the system. The advantage of its representation power is one of the reasons that leads to its popularity in difference disciplines.

Probabilistic graphical model coupled with its underlining distribution is a powerful tool for effective inference, apart from its advantage of representation power. It allows to answer queries with the help of the underlining distribution when practical inference algorithms are provided, which meets our need of reasoning with uncertainty. In addition to inference, probabilistic graphical model also supports learning from data. With certain amount of data available, a probabilistic graphical model can be learned to explain the observed data better in addition to align with our own understanding of a domain. The learned graphical model can serve to do inference with higher confidence in return. A diagram is illustrated in Figure 1.2. As would become clear in Part II, the inference may be needed to carry out model learning as well, apart from the above mutual-benefiting interaction.

1.2 Scope and Thesis Outline

We gives the intuition and motivation of probabilistic graphical model in last section, and the interaction between inference and learning in this framework. In this section, we would navigate further among the topics within this framework and states the ones that we would cover in the thesis.

Inference in probabilistic graphical model is about answering queries with help of its coupled distribution. These queries can be generally grouped into the following cases:

- Computing the likelihood of observed data or unobserved random variable.
- Computing the marginals distribution over a particular subset of random variables.
- Computing the conditional distribution a subset of variables given the configuration of another subset of variables.
- Computing the most likely configuration of (a subset of) variables.

The work of this thesis would be mainly related with the first three cases in inference part.

Due to either the requirement of efficiency in solving a problem or the structure of the problem's graphical model representation, it is not always that case that the above inference problem can solved exactly. Thus inference methods can be divided into

- exact inference,
- and approximate inference.

For a limited class of graphs, exact inference such as variable elimination and sum-product algorithm can be used. Some graphs also allow efficient inference after mild graph modification, e.g. junction tree method. However, the above listed inference problems can only be approximately solved in general graphs. The approximate inference family can be further divided into

- Stochastic Approximation (Particle methods),
- Deterministic Approximation (Variational methods).

Stochastic approximation mainly relies on sample instances to answer queries, where a major challenge lies in how to obtain samples efficiently from a target distribution. Gibbs sampling, importance sampling and Markov Chain Monte Carlo are within this family. On the other hand, deterministic approximations refer to the variational methods, such as mean field approximation, loopy belief propagation, expectation propagation etc. *From the perspectives of methodology, we related work in this thesis locates in the family of variational methods under approximate inference category.*

As for learning in probabilistic graphical models, there are two types of learning problems

- Structure learning,
- Parameter learning.

The first case refer to determine the structure of a graphical model from observations, which is usually reduced to the problem of whether there should be an edge between a pair of nodes in the graphical model. The parameter learning is about to determine the parameter of a probabilistic graphical model (or its coupled distribution), with its graphical structure known. Structure learning is out of the scope of this thesis. The term *learning* in thesis means the estimation of the parameters of a distribution. This problem is mainly discussed in Part II, where we would touch the topics about learning in both undirected and directed graphical models.

As for the learning techniques, the learning principles can categorized into

- Maximal likelihood estimation
- Maximal conditional likelihood

- Bayesian estimation
- Maximal-margin
- Maximum entropy

in general. We would touch techniques of the first three cases in Part II.

Publications

The following works ware done during the author's PhD education.

1. Dong Liu and Lars K Rasmussen. Region-based energy nerual network for approximate inference. Under review, 2020.
   Code: https://github.com/FirstHandScientist/renn (Private, publish later)

2. Dong Liu, Minh Thành Vu, Li Zuxing, and Lars K Rasmussen. α belief propagation for approximate beyeisian inference. Under review, 2020.
   Code: https://github.com/FirstHandScientist/alpha-bp

3. Dong Liu, Antoine Honoré, Saikat Chatterjee, and Lars K Rasmussen. Powering hidden markov model by neural network based generative models. In the 24th European Conference on Artificial Intelligence (ECAI), 2020.
   Code: https://github.com/FirstHandScientist/genhmm

4. Dong Liu, Minh Thành Vu, Saikat Chatterjee, and Lars K Rasmussen. Neural network based explicit mixture models and expectation-maximization based learning. In International Joint Conference on Neural Networks, Glasgow, UK, July 2020.
   Code: https://github.com/FirstHandScientist/EM-GM

5. Antoine Honoré, Dong Liu, David Forsberg, Karen Coste, Eric Herlenius, Saikat Chatterjee, and Mikael Skoglund. Hidden markov models for sepsis detection in preterm infants. In 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020.
   Code: https://github.com/FirstHandScientist/genhmm

6. D. Liu, N. N. Moghadam, L. K. Rasmussen, J. Huang, and S. Chatterjee. α belief propagation as fully factorized approximation. In 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pages 1-5, 2019.
   Code: https://github.com/FirstHandScientist/amp

7. Dong Liu, Minh Thành Vu, Saikat Chatterjee, and Lars K Rasmussen. Entropy-regularized optimal transport generative models. In 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3532-3536, 2019.
   Code: https://github.com/FirstHandScientist/eotgms

8. Dong Liu, Baptiste Cavarec, Lars K Rasmussen, and Jing Yue. On dominant interference in random networks and communication reliability. In ICC 2019 IEEE International Conference on Communications (ICC), pages 1-7, 2019.

9. Dong Liu, Chao Wang, and Lars K Rasmussen. Discontinuous reception for multiple-beam communication. IEEE Access, pages 46931-46946, 2019.

10. Dong Liu, Viktoria Fodor, and Lars K Rasmussen. Will scale-free popularity develop scale-free geo-social networks? IEEE Transactions on Network Science and Engineering, 2018.

The material presented in this thesis is based on the author's works which are partially published in 1, 2, 3, 4, 5, 6, 7 listed above.

Outline of Thesis

To be written after main content is finished.

Chapter 2

Background

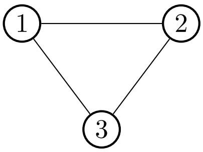
In this chapter, we review some background knowledge that is going be used in this thesis. We begin with the introduction to probabilistic graphical models. Then a divergence measure is introduced. Common inference tasks and methods are discussed before the learning problems in probabilistic graphical models are reviewed, which are interpreted as minimization of the divergence measure.

2.1 Graphical Models

Graphical models provide a formal graph representation of statistical dependency of complex problems or systems. The conditional independence of random variables can be conveniently encoded and analyzed by a graphical model. More importantly, query problems can be resolved by interactions of local regions of a graphical model in exact or approximate ways, which are usually unfeasible to solve directly.

More formally, a graphical model is a graphical representation of a collection of random variables (along their domains) where their statistical dependency encoded into a set of non-negative functions and the graphical structure. Let $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ be a vector of random variables with N as a positive integer, where an element variable x_i can be either discrete or continuous random variable and takes values from its domain $\mathcal{X}_i$. Note that the domain of a random variable is not necessary be the same as that of another. With some abuse of notation, we might use $\boldsymbol{x}$ to denote its assignment when there is no cause of ambiguity in context. The joint probability is denoted by $p(\boldsymbol{x}) = p(x_1, x_2, \cdots, x_N)$. We denote $\boldsymbol{\mathcal{X}} = \prod_{x=1}^{N} \mathcal{X}_i$ and then $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$.

As motivated in Chapter 1.1, a graphical model can be directed or undirected. A directed graphical model also is also know as a Bayesian network or generative model in literature [7, Chapter 8]. We might use the names alternatively. The non-negative functions in graphical models encodes the local compatibility of states of random variables. In directed graphical models, i.e. Bayesian networks, the local functions are conditional probability functions. The joint probability distribution



x_1	x_2	$\varphi(x_1, x_2)$
0	0	10
0	1	1
1	0	1
1	1	10

$\cdots$

x_2	x_3	$\varphi(x_1, x_2)$
0	0	5
0	1	3
1	0	3
1	1	5

Figure 2.1: A Markov random field with three binary nodes. Potential factors are represented by tables.

is represented as the product of these conditional probability functions,

$$p(\boldsymbol{x}) = \prod_{n=1}^{N} p(x_n|\mathcal{P}(x_n)), \tag{2.1}$$

where $\mathcal{P}(\cdot)$ denotes the set of parent nodes in the directed graph. In an directed graphical model, the local functions, i.e. the conditional probability distributions, e.g. $\{p(x_n|\mathcal{P}(x_n))\}$, are normalized and proper distributions. Additionally, sampling from a underlining distribution $p(\boldsymbol{x})$ of a directed graphical model is efficient. Due to acyclic property of directed graphical models, by the well know *ancestral sampling*, a sample $(x_1, x_2, \cdots, x_N)$ can be drawn sequentially via following the directed edges. In another word, x_n is always sampled after $\mathcal{P}(x_n)$. This process might be viewed as the 'generative' process of signal $\boldsymbol{x}$, i.e. how $\boldsymbol{x}$ is generated from the graphical model.

A Bayesian network (generative model) is usually easier to be interpreted due to the fact that its local functions are conditional probabilities and it is natural to decompose the joint underlining distribution into conditional probability distributions. But Bayesian networks can only be applied to the limited cases where influence between variables is directional. In many practical cases, interaction between variables can not be naturally described by impact with directionality. A problem of this kind can be represented by an undirected graphical model, i.e. a Markov random field (MRF). Under certain condition, a Bayesian network can be perfectly represented by a Markov random field without loss of independence information by moralizing edges [45, Chapter 4.5]. Instead of conditional probability distributions, the local functions of MRF represents the compatibility of states of different variables, which are termed as *potential factors*. Different from conditional probabilities in a Bayesian network, a potential factor in a MRF is not necessary normalized (not necessary to be summed to one). We provide a toy example of MRF with three variable nodes as follows.

Example 2. *As shown in Figure 2.1, the MRF encodes dependency of three random variables x_1, x_2, and x_3, where node i is associated with variable x_i and each has a binary domain, i.e. $\mathcal{X}_i = \{0, 1\}$ for $i = 1, 2, 3$. Three potential factors of the MRF* *together define the joint distribution*

$$p(\boldsymbol{x}) = \frac{1}{Z}\varphi_{1,2}(x_1, x_2)\varphi_{2,3}(x_2, x_3)\varphi_{1,3}(x_1, x_3)$$

where $Z = \sum_{x_1,x_2,x_3} \varphi_{1,2}(x_1, x_2)\varphi_{2,3}(x_2, x_3)\varphi_{1,3}(x_1, x_3)$ *normalizes the potential factors such that* $p(\boldsymbol{x})$ *is a proper distribution. The exemplified potential factors in Figure 2.1 demonstrate that it is more compatible or likely when* x_1*,* x_2 *and* x_3 *are in the same state (either* 0 *or* 1*) than they are configured into different states.*

From the above example to a formal statement, a MRF over random vector $\boldsymbol{x}$ can be represented by a undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, with each node $i \in \mathcal{V}$ is associated with a random variable x_i and undirected edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. This MRF encodes a collection of distributions that factorize as

$$p(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{a \in \mathcal{F}} \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}), \tag{2.2}$$

where $\mathcal{F}$ is the set of indexes of potential factors, and each factor φ_a for $a \in \mathcal{F}$ is defined on subset of $\boldsymbol{x}$, i.e. $\varphi_a : \mathcal{X}_a \to \mathbb{R}^+ \cup \{0\}$, where $\mathcal{X}_a = \prod_{i \in a} \mathcal{X}_i$ is the domain of potential factor φ_a. The scope of factor a is $\boldsymbol{x}_a = \{x_i | i \in a\}$ where $i \in a$ stands for that the variable x_i associated with node i is an argument of potential factor φ_a. In (2.2),

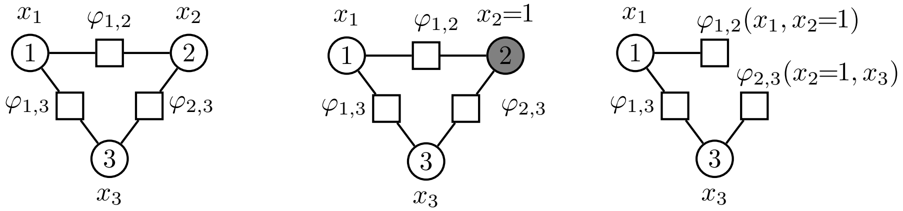
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{a \in \mathcal{F}} \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}) \tag{2.3}$$

is the *partition function.* Apparently, the partition function normalizes the potential factors such that $p(\boldsymbol{x}; \boldsymbol{\theta})$ is a proper probability.

Remark 1. *We can compare directed and undirected graphical models with regarding to the following aspects.*

- *Representation: The structure and the parameterizaiton in directed graphical models provide a natural representation for many types of real-world domains. MRF representation is not usually as intuitive as that of directed graphical models. But the acyclic property of directed graphical models limits their representation power. On the other hand, MRFs can be either cyclic or acyclic, which offers the flexibility of graph structure and can simplify the graphical representation. Due to the weaker requirement of potential factors and the weaker requirement of graphical structure in MRFs than local functions and acyclic constraint in directed graphical models, respectively, the representation of MRFs are richer.*

- *Local nonnegative functions: The local functions are conditional probability functions in directed graphical models, but potential factors (nonnegative) in undirected cases.*



(a) A factor graph representation.

(b) Conditioning on $x_2 = 1$ in factor graph.

(c) The reduced graph of Figure 2.2b

Figure 2.2: A Markov random field is represented by a factor graph, i.e. 2.2a, conditioning of the MRF 2.2b, the reduced MRF 2.2c.

- *Sampling: Sampling is more straightforward within generative models (directed graphs) than that in MRFs.*

- *Normalization: Since each local function is a conditional probability function in directed graphical models, partition function for normalization is not needed. A MRF in general comes with partition functions, since potential factors are not necessarily normalized.*

Alternative Representation of MRF

The representation of a MRF by $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as explained above is compact, but the potential factors are not present in the graphical representation. An alternative representation to MRF is *factor graph* [46], which is a bipartite graph topology. In a factor graph, a potential factor is explicitly represented as a factor node, as counterpart of variable node associated with a random variable.

Definition 1. *A factor graph $\mathcal{G}_F$, is a bipartite graph that represents the factorization structure of* (2.2)*. A factor graph has two types of nodes: i) a variable node for each variable x_i; ii) a factor node for each potential function φ_a. An edge between a variable node i and factor node a if and only if x_i is argument of φ_a. We would denote a factor graph by $\mathcal{G}_F(\mathcal{V} \cup \mathcal{F}, \mathcal{E}_F)$ with $\mathcal{V}$ as the set of variable nodes, $\mathcal{F}$ as the set of factor nodes, and $\mathcal{E}_F$ the set of undirected edges.*

Example 3. *Let us represent the Example 2.2a by a factor graph, which is shown in Figure 2.2a. Different from the representation by $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in Figure 2.1, factor nodes are explicitly represented by square nodes.*

Conditioning on Observations in MRFs

It is not rare that a graphical model may contain observed variable. The node set of a MRF can be separated into a subset $\mathcal{V}_O$ of nodes, that are associated with observed variable $\boldsymbol{x}_O$, and a subset $\mathcal{V}_U$ of nodes associated with unobserved variable $\boldsymbol{x}_U$. For an evidence is observed,

$$p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta}) = \frac{p(\boldsymbol{x}_U,\boldsymbol{x}_O;\boldsymbol{\theta})}{p(\boldsymbol{x}_O;\boldsymbol{\theta})} = \frac{Z(\boldsymbol{x}_O,\boldsymbol{\theta})}{Z(\boldsymbol{\theta})}, \tag{2.4}$$

where

$$\begin{aligned} \tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) &= \prod_{a\in\mathcal{F}} \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}), \\ Z(\boldsymbol{x}_O,\boldsymbol{\theta}) &= \sum_{\boldsymbol{x}_U} \tilde{p}(\boldsymbol{x};\boldsymbol{\theta}), \\ Z(\boldsymbol{\theta}) &= \sum_{\boldsymbol{x}_O}\sum_{\boldsymbol{x}_U} \tilde{p}(\boldsymbol{x};\boldsymbol{\theta}). \end{aligned} \tag{2.5}$$

This means that a condition probability can be computed by partition function and sub-partition functions. Alternatively, when an evidence $\boldsymbol{e}_O$ (an sample instance of $\boldsymbol{x}_O$) is observed, the conditional probability can be written as

$$p(\boldsymbol{x}_U|\boldsymbol{x}_O=\boldsymbol{e}_O;\boldsymbol{\theta}) = \frac{\tilde{p}(\boldsymbol{x}_U,\boldsymbol{x}_O=\boldsymbol{e}_O;\boldsymbol{\theta})}{\sum_{\boldsymbol{x}_U}\tilde{p}(\boldsymbol{x}_U,\boldsymbol{x}_O=\boldsymbol{e}_O;\boldsymbol{\theta})} \propto \tilde{p}(\boldsymbol{x}_U,\boldsymbol{x}_O=\boldsymbol{e}_O;\boldsymbol{\theta}) \tag{2.6}$$

where $\propto$ stands for propositional to. 2.6 shows an interesting phenomenon for MRF including evidence. It can be understood as clamping nodes in $\mathcal{V}_O$ of the MRF to configuration $\boldsymbol{e}_O$, i.e. the domain of $\boldsymbol{x}_O$ becomes a set containing only one instance $\boldsymbol{e}_O$. For instance, an example of conditioning on a variable node for Example 2 is shown in Figure 2.2b.

In addition to the above intuitions, conditioning can also be understood as a process of reducing the graph of a MRF. When a MRF is conditioned on $\boldsymbol{x}_O$, the variables nodes of set $\mathcal{V}_O$ are removed from $\mathcal{G}$, along with their edges. The potential factors with regarding to $\mathcal{V}_U$ are modified accordingly [45, Chapter 4.2.3]. For instance, the graph including evidence node 2 in Figure 2.2b can be further reduced into a Figure 2.2c. Then any inference applicable to a MRF applies to the MRF with nodes clamped as well. A MRF with several nodes clamped to some evidence can be seen either as a manipulation of its domain or the graph itself.

It can be seen that MRF framework is capable to handle conditioning as well. Therefore, in the following part of the thesis, it might or might not have bee based on conditioning observed variables when a MRF is mentioned.

2.2 Divergence

Before we get into more discussion about inference and learning topics, we firstly introduce the concept of *divergence* measures since principles of both learning and inference are closed related with divergence measure. A divergence measure play a fundamental role when we try to use a probability distribution (over discrete or continue variable) q to approximate another probability distribution p. A divergence measure is used to formally quantity how much information is lost when p is represented by q. Denote $\mathbb{P}$ as the space of measures p and q, i.e. $p, q \in \mathbb{P}$.

Definition 2. *Given the space $\mathbb{P}$ of probability distribution for a random variable $\boldsymbol{x}$, a divergence on this space is defined as a function $D(p\|q) : \mathbb{P} \times \mathbb{P} \to \mathbb{R}^+ \cup \{0\}$ such that $D(p\|q) \geqslant 0$ for all $p, q \in \mathbb{P}$ and $D(p\|q) = 0$ if and only if $p = q$.*

Here we introduce the classic *Kullback-Leibler divergence* [49,50], KL divergence for short, which is one of the most widely used divergence measures in machine learning, statistics and information theory.

Definition 3. *The Kullback-Leibler (KL) divergence on $\mathbb{P}$ is defined as a function $KL(\cdot\|\cdot) : \mathbb{P} \times \mathbb{P} \to \mathbb{R}^+ \cup 0$ with the following form*

$$\mathrm{KL}(p\|q) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}, \tag{2.7}$$

where log *is the natural loggarithm. Note the sum in 2.7 should be replaced by intergral when p and q are probability density functions.*

KL divergence is not symmetric. In another word, there is no equivalence between $\mathrm{KL}(p\|q)$ and $\mathrm{KL}(q\|p)$ in general.

2.3 Inference Tasks

Given a probability distribution $p(\boldsymbol{x})$ as the underline distribution of a graphical model, inference in general can be divided into four kinds of tasks, as brought up in Chapter 1.2. Our work in this thesis would be closely involved with the problems

- computing the likelihood of observed data or unobserved random variable;
- computing the marginals distribution over a particular subset of nodes, i.e. $p(\boldsymbol{x}_A)$ for $A \subset \mathcal{V}$. Note that single-node marginal distribution $p(x_i)$ also belongs to this case;
- computing the conditional distribution a subset of nodes given the configuration of another subset of nodes, i.e. $p(\boldsymbol{x}_A|\boldsymbol{x}_B)$ for $A, B \in \mathcal{V}$ and $A \cap B = \emptyset$;

in MRFs. The above tasks are also close related with the inference of partition function

- Computation of $Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{a \in \mathcal{F}} \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta})$, or sub-partition functions.

In the following section, we would introduce the variational methods for the above tasks in high-level.

2.4 Variational inference

In solving inference tasks, one important technique is based on a variational approach. With $p(\boldsymbol{x};\boldsymbol{\theta})$ as the underlining probability distribution of a graphical model, directly inference with $p(\boldsymbol{x};\boldsymbol{\theta})$ is often unfeasible due to the system represented by the graphical model is too large or complex. It can also be the case that even we know the form of $p(\boldsymbol{x};\boldsymbol{\theta})$, the computation in inference tasks can be prohibitive. In variational approaches, a 'trial' probability distribution $b(\boldsymbol{x})$ is introduced to approximate $p(\boldsymbol{x};\boldsymbol{\theta})$. The trial distribution should be intuitively simpler than $p(\boldsymbol{x};\boldsymbol{\theta})$. *Variational free energy* [71] is a quantity used to find such a approximation. The variational free energy is defined by

$$\begin{aligned} F_V(b) &= \mathrm{KL}(b(\boldsymbol{x})||p(\boldsymbol{x};\boldsymbol{\theta})) - \log Z(\boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \frac{b(\boldsymbol{x})}{p(\boldsymbol{x};\boldsymbol{\theta})} - \log Z(\boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \frac{b(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x};\boldsymbol{\theta})}, \end{aligned} \tag{2.8}$$

where $\tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) = \prod_{a\in\mathcal{F}} \psi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a)$. Since $\mathrm{KL}(b(\boldsymbol{x})\|p(\boldsymbol{x};\boldsymbol{\theta}))$ is always non-negative and is zero if and only if $b(\boldsymbol{x}) = p(\boldsymbol{x};\boldsymbol{\theta})$, we have $F_V(b) \geqslant -\log Z(\boldsymbol{\theta})$, with equality when $b(\boldsymbol{x}) = p(\boldsymbol{x};\boldsymbol{\theta})$.

Remark 2. *Note from 2.8, the minimization w.r.t. b of variational free energy is equivalent to the divergence minimization, i.e.* $\mathrm{KL}(b(\boldsymbol{x})||p(\boldsymbol{x};\boldsymbol{\theta}))$*, since* $\log Z(\boldsymbol{\theta})$ *does not depend on b. By observing $F_V(b) = \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \frac{b(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x};\boldsymbol{\theta})}$, the free energy minimization guides the choice of b such that b is close to an unnormalized measure $\tilde{p}(\boldsymbol{x};\boldsymbol{\theta})$ in its space.*

Another benefit of 2.8 is that we are able to approximate $p(\boldsymbol{x};\boldsymbol{\theta})$ without inference of the true marginal distributions $\{p(\boldsymbol{x}_a;\boldsymbol{\theta}), a \in \mathcal{F}\}$. Since $\log \tilde{p}(\boldsymbol{x};\boldsymbol{\theta})$ *can be formulated as sum of log-potential-factors that are all local functions, the computation of $F_V(v)$ can be done by inference of marginals $\{b_a(\boldsymbol{x}_a), a \in \mathcal{F}\}$ of the approximate distribution, which are tractable.*

Remark 3. *Discussion: Since we are essentially approximating distribution p by a distribution b, can we minimize* $\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{\theta})\|b(\boldsymbol{x}))$ *instead?*

It might be feasible by instinct. But a further check would reveal its infeasibility. The $\mathrm{KL}(p(\boldsymbol{x};\boldsymbol{\theta})\|b(\boldsymbol{x}))$ *would inevitable requires the marginals of p and therefore requires the exact inference in p, which are what we are trying to avoid. But it does not mean this divergence is useless. As shall be seen in section 2.5, this type of divergence measure is what we need in model learning.*

Variational Free Energy and Mean Field

In mean field approach, a fully-factorized approximation is used,

$$b_{MF}(\boldsymbol{x}) = \prod_{i=1}^{N} b_i(x_i). \tag{2.9}$$

Substituting (2.9) into the variational free energy gives

$$F_{MF}(b) = -\sum_{a\in\mathcal{F}}\sum_{\boldsymbol{x}_a} \log\varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta})\prod_{i\in a} b_i(x_i) + \sum_{i\in\mathcal{V}}\sum_{x_i} b_i(x_i)\log b_i(x_i), \tag{2.10}$$

where $i \in a$ stands for the node i's associated x_i is argument of φ_a. Solving the minimization of F_{MF} w.r.t. $b_{MF}(\boldsymbol{x})$ gives the update rule of mean field

$$\log b_i(x_i) \propto \sum_{a\in\mathrm{ne}_i}\sum_{\boldsymbol{x}_a\setminus x_i} \log\varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a)\prod_{j\in a\setminus i} b_j(x_j), \tag{2.11}$$

where $\mathrm{ne}_i = \{a|i \in a, a \in \mathcal{F}\}$, i.e. the potential factors that has x_i as argument.

Bethe Free Energy and (Loopy) Belief Propagation

Different from the mean field approximation, Bethe approximation also includes the non-single-node beliefs $\{b_a(\boldsymbol{x}_a)\}$ apart from the single-node beliefs $\{b_i(x_i)\}$ [101]. In this case, the Bethe free energy is given by

$$F_{Bethe}(b) = \sum_a\sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a)\log\frac{b_a(\boldsymbol{x}_a)}{\varphi_a(\boldsymbol{x}_a)} - \sum_{i=1}^{N}(|\mathrm{ne}_i| - 1)\sum_{x_i} b_i(x_i)\log b_i(x_i), \tag{2.12}$$

where $|\cdot|$ stands for carnality. Due to the non-single-node beliefs, there are consistency constrains $\sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a) = \sum_{x_i} b_i(x_i) = 1,\ \forall\ i \in a$ to obey. Then, solving the Bethe free energy minimization problem

$$\begin{aligned} \min_{\{b_a(\boldsymbol{x}_a)\},\{b_i(x_i)\}} \quad & F_{Bethe}(b) \\ \text{s.t.} \quad & \sum_{\boldsymbol{x}_a\setminus x_i} b_a(\boldsymbol{x}_a) = b_i(x_i), \\ & \sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a) = \sum_{x_i} b_i(x_i) = 1, \\ & 0 \leqslant b_i(x_i) \leqslant 1, \\ & i \in \mathcal{V}, a \in \mathcal{F}, \end{aligned} \tag{2.13}$$

where $\mathcal{V}$ and $\mathcal{F}$ are the set of variable nodes and the set of factor nodes in factor graph as defined in Definition 1, gives the (loopy) BP message-passing rule

$$m_{a\to i}(x_i) \propto \sum_{\boldsymbol{x}_a\setminus x_i} \varphi_a(\boldsymbol{x}_a)\prod_{j\in a\setminus i}\prod_{a'\in\mathrm{ne}_j\setminus a} m_{a'\to j}(x_j). \tag{2.14}$$

2.5 Learning principles

We have touched the learning topic in chapter 1, which is to find the 'best' probability distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ in its space $\mathbb{P}$. To make the discussion more concrete, we assume the domain is governed by a underlying distribution p^* that is induced by a (directed or undirected) graphical model, $\mathcal{M}^* = \{\mathcal{K}^*, \boldsymbol{\theta}^*\}$ with $\mathcal{M}^*$ representing its structure and $\boldsymbol{\theta}^*$ representing its parameter. Here we discuss about *model learning* (parameter learning only). For notation simplicity, we use $p^*(\boldsymbol{x})$ to denote this distribution. We are given a dataset $\mathcal{D} = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^M\}$. Following the standard assumption, these sample instances are *independent and identically distributed (i.i.d.).* The task is then to use the information from the dataset to learn a distribution p within its space $\mathbb{P}$, since the governing distribution $p^*(\boldsymbol{x})$ is not known.

The problem of learning a distribution in $\mathbb{P}$ to approximate p^* can be formulated as density estimation. With the concept of KL divergence in section 2.2, learning of p can be formulated as minimizing the KL divergence

$$\begin{aligned}&\mathrm{KL}(p^*(\boldsymbol{x})\|p(\boldsymbol{x};\boldsymbol{\theta}))\\ =&\mathbb{E}_{\boldsymbol{x}\sim p^*}\left[\log\frac{p^*(\boldsymbol{x})}{p(\boldsymbol{x};\boldsymbol{\theta})}\right]\\ =&-H(p^*) - \mathbb{E}_{\boldsymbol{x}\sim p^*}\left[\log p(\boldsymbol{x};\boldsymbol{\theta})\right],\end{aligned} \tag{2.15}$$

where $H(p^*)$ is the entropy of p^*. Due to the property of divergence, the KL divergence in 2.15 is zero if and only if $p(\boldsymbol{x};\boldsymbol{\theta}) = p^*(\boldsymbol{x})$. The last line of 2.15 shows that the negative entropy term does not depends on $p(\boldsymbol{x};\boldsymbol{\theta})$. Thus we can just focus on the expectation term $\mathbb{E}_{\boldsymbol{x}\sim p^*}\left[\log p(\boldsymbol{x};\boldsymbol{\theta})\right]$, which is *expected log-likelihood.* Therefore, we can just use the expected log-likelihood to do model learning instead of minimizing the KL divergence.

Note although we can use the expected log-likelihood for model learning task and even model comparison (comparing a trained model with another one), we loss the information of how close a trained model is to p^*. This is due to the omitting of $H(p^*)$, which is not available.

Since it is not possible to know p^* (otherwise we do not need to learn it), the expected log-likelihood is approximated by sample instances of p^*,

$$\mathcal{L}(\mathcal{D};\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}}\log p(\boldsymbol{x};\boldsymbol{\theta}), \tag{2.16}$$

and

$$\mathbb{E}_{\boldsymbol{x}\sim p^*}(\log p(\boldsymbol{x};\boldsymbol{\theta})) \approx \mathcal{L}(\mathcal{D};\boldsymbol{\theta}). \tag{2.17}$$

Log-likelihood $\mathcal{L}(\mathcal{D};\boldsymbol{\theta})$ is one of the most widely used loss for model learning. However, $\mathcal{L}(\mathcal{D};\boldsymbol{\theta})$ is not always a feasible loss to compute due to:

- exact computation of $p(\boldsymbol{x})$ is not possible;

- there are some elements of $\boldsymbol{x}$ which are not observable (latent variables).

For the first case, the typical treatment is to approximate the exact log-likelihood. This is done by approximation with employing inference methods or making simplified assumptions on dependency structure of the graphical model of $p(\boldsymbol{x})$. Then, optimization is carried out with regarding to the approximated log-likelihood. These methods include surrogate likelihood [58, 91], pseudo-likelihood [51, 77], piecewise likelihood [56, 84], saddle-point approximation [83, 97].

Apart from the above case where all variables are observable, the partial observed models,the latent variable case, are equally import in inference and learning with uncertainty. This class of models includes (but not limited to) classic Gaussian mixture models (GMMs) and hidden Markov models (HMMs). There are latent variables because:

- Use of abstract variable to model the generative process (usually a directed graph) of observation data, such as HMMs.
- A practical true attribute of an object may be difficult or impossible to measure exactly. For instance, the disease infection can only be diagnosed via the relevant symptoms, e.g. Example 1; In the position tracking of a car with noisy sensors, the true position of the car might only be inferred via noisy data of sensors.
- No measurement on an attribute of an object of interest is made. For instance, the velocity sensor in the car tracking example might not be stable and might read not quantity now and then.

In general, latent variables are commonly used to deal with partial observation problems, data clustering, data manipulation, etc. Let us denote the observable variable and latent variable by $\boldsymbol{x}_O$ and $\boldsymbol{x}_U$, respectively. We can see that the log-likelihood $p(\boldsymbol{x}_O, \boldsymbol{x}_U; \boldsymbol{\theta})$ is not available any more as it is in the fully-observed case. To deal with the latent variables, we can try to optimize the partial log-likelihood

$$\begin{aligned} l(\boldsymbol{x}_O; \boldsymbol{\theta}) =& \log p(\boldsymbol{x}_O; \boldsymbol{\theta}) \\ =& \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \left[\log \frac{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}{p(\boldsymbol{x}_U|\boldsymbol{x}_O; \boldsymbol{\theta})} \cdot \frac{p(\boldsymbol{x}_U, \boldsymbol{x}_O; \boldsymbol{\theta})}{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \right] \\ =& \mathrm{KL}(q(\boldsymbol{x}_U|\boldsymbol{x}_O) \| p(\boldsymbol{x}_U|\boldsymbol{x}_O; \boldsymbol{\theta})) + F(q, \boldsymbol{\theta}) \end{aligned} \tag{2.18}$$

with

$$F(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \left[\log p(\boldsymbol{x}_U, \boldsymbol{x}_O; \boldsymbol{\theta}) \right] + H(q(\boldsymbol{x}_U|\boldsymbol{x}_O)) \tag{2.19}$$

where $H(q(\boldsymbol{x}_U|\boldsymbol{x}_O))$ is the entropy of $q(\boldsymbol{x}_U|\boldsymbol{x}_O)$, and q can be any distribution over $\boldsymbol{x}_U$. Due to the non-negative property of KL divergence, we have

$$l(\boldsymbol{x}_O; \boldsymbol{\theta}) \geqslant F(q, \boldsymbol{\theta}), \tag{2.20}$$

with equality when $q(\boldsymbol{x}_U|\boldsymbol{x}_O) = p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta})$. $F(q,\boldsymbol{\theta})$ is also called *variational lower bound.*

One of the most wide used methods in learning with latent variable is *expectation maximization (EM)* [13]. In EM method, the posterior of $\boldsymbol{x}_U$ is computed exactly from p, $q(\boldsymbol{x}_U|\boldsymbol{x}_O) = \operatorname{argmax}_q F(q,\boldsymbol{\theta}) = p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta})$, which is the optimal solution to q. Then the parameter $\boldsymbol{\theta}$ of p is optimized. The two steps are optimized iteratively.

In cases the posterior of $\boldsymbol{x}_U$ is not feasible to compute, the variational EM [90, section 6.2.2] or Monte Carlo EM (need sampling technique) [69]. There are also neural network based methods with Monte Carlo estimator to cope with the latent variable problems, see [28, 42, 47, 51].

In above discussion, we have assumed that the model's distribution q is explicitly defined, i.e. the distribution (along its density function or mass function) is available.

In an alternative track, the deep generative models grow popularity in literature and can be applied to different areas such as high-dimensional data representation, reinforcement learning and semi-supervised learning, because of its efficient sampling of multi-mode distributions [27]. A generator in a deep generative model induces a distribution that can be either explicit or implicit distribution. The former problem dates back to [12] and receives more attention in recent years with latest work such as varational autoencoder [42], Real-NVP [15], normalizing flow model [41] and neural ordinary difference equations [8]. The training of these models are still based on the maximum likelihood principle or varational likelihood bound.

The later case brings an implicit distribution, where the maximum likelihood principle is not applicable any more. In this case, a state-of-art method is the generative adversarial model that employs a discriminator to play the role of divergence measure [3, 28, 80], which is essentially explained by a process of the minimization of the Jensen-Shannon divergence. Additionally, other sample-test based distances are employed as alternative methods for implicit model learning. Among this family, optimal transport is receiving more attention in recent years [10, 82] in this track, which has also been applied for training of Boltzmann machine [64] autoencoders [85] and generative adversarial networks [4].

Part I

Inference

Chapter 3

An alternative view of belief propagation

Belief propagation (BP) is a meta message-passing algorithm for inference problems in probabilistic graphical models. BP answers queries by locally exchanging beliefs (statistical information) between nodes in a graphical model [7, 46]. In section 2.4, we introduced the classic belief propagation as the minimization of a free energy, instead of a iterative message-passing routine. Interestingly to note, the message-passing rule of BP was developed as early as 1986 [74] and had been popularly used in different fields before the free energy optimization intuition was developed in literature [101].

BP can solve inference problems in linear-time exactly when graphs are loop-free or tree-structured [46]. The message-passing routine of BP can be boiled down to variable elimination in tree-structured graphs[1], the message scheduling of which corresponds to variable elimination order. The message scheduling can be omitted and equivalent exact inference results can be obtained, when beliefs are updated via message passing with division [45, section 10.3]. BP and its variants are widely applied in large computation systems due to their 'magic' of reducing the exponential number of operation for inference with enumeration into linear complexity. This is possible because

- A underline distribution of a graphical model is usually factorized, each sub-expressions (factors) depends only on a small number of variables.
- The intermediary results are computed once and cached as messages, which are reused in coming computations.

Inevitably, many real-world signals are naturally modeled by graph representations with loops. Surprisingly, although lost its original guarantees in loopy graphs,

[1] This applies to cases where systems themselves can be represented by tree-structured graphs, or cases where original graphs are not trees but becoming tree-structured after reorganized (such as clustering).

BP is still a practical method and gives reasonable good inference results by running it as if there was no loop, i.e. loopy BP. But its performance can vary from case to case and its behavior is not well understood in general.

In this chapter, to gain more insights of BP in general graphs, we take the path of variational methods to develop an interpretable variant of BP, which is termed as α-BP. The intuition of α-BP starts with a surrogate distribution $q(\boldsymbol{x})$, i.e. an approximate distribution. $q(\boldsymbol{x})$ is assumed to be fully factorized and each factor of $q(\boldsymbol{x})$ actually represents a message in the graphical model with an underlining distribution $p(\boldsymbol{x})$. We derive a message-passing rule that is induced by minimizing a localized α-divergence. The merits of α-BP are as follows: i). α-BP is derived intuitively as localized minimization of α-divergence between original distribution p and surrogate distribution q; ii). α-BP generalizes the standard BP, since the message-passing rule of BP is a special case of α-BP. iii). α-BP can outperform BP in full-connected graphs while still maintaining simplicity of BP for inference.

Apart from the algorithmic perspective, another common issue of BP and its variants in general graphs is convergence. We devote section 3.4 to convergence study of α-BP. Sufficient conditions that guarantee the convergence of α-BP to a unique fixed point, are studied and obtained. It turns out that the derived convergence conditions of α-BP depend on both the graph and also the value of α. This result suggests that proper choice of α can help to guarantee the convergence of α-BP.

3.1 α Divergence

Before we get into the algorithmic discussion, we firstly provide some preliminaries for the algorithmic intuition. As an extended discussion to section 2.2, we firstly introduce a more generalized divergence than KL divergence.

Apart from the KL divergence, another divergence measure that generalizes KL divergence is α-divergence. In fact, α-divergence appeared in literature just one year later than KL divergence when Herman Chernoff initially defined it for likelihood-ratio test [9]. Around a decade later, Alfréd Rényi proposed his version of divergence as well [78]. In the 80s of last century, Amari extended Chernoff's version of α-divergence [2], which is widely used now in study the geometry of distribution manifolds. α-divergence, similar to KL divergence, is a typical way to measure how different two measures characterized by densities p and q are. By following the notation [106], the definition of α-divergence (Amari's version, with correction term to accommodate unnormalized measure) is as follows,

$$\mathcal{D}_\alpha(p\|q) = \frac{\int \alpha p(\boldsymbol{x}) + (1-\alpha) q(\boldsymbol{x}) - p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha} d\boldsymbol{x}}{\alpha(1-\alpha)}, \tag{3.1}$$

where α is the parameter of this divergence. Difference from the KL divergence definition in section 2.2, p and q here are not necessary to be normalized measure.

In section 2.2, KL divergence was defined over two normalized measure. Here we extend that definition to a generalized case where p and q are not necessarily normalized, as another way of characterizing difference of measures, is closely related with α-divergence. KL divergence is defined as

$$KL(p\|q) = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x} + \int q(\boldsymbol{x}) - p(\boldsymbol{x}) d\boldsymbol{x}, \tag{3.2}$$

where the $\int q(\boldsymbol{x}) - p(\boldsymbol{x}) d\boldsymbol{x}$ is a correction factor to accommodate possibly unnormalized p and q.

Remark 4. *Note in* (3.1) *and* (3.2)*, the integral should be replaced by sum for discrete* $\boldsymbol{x}$*.*

Remark 5. *The KL divergence can be seen as a special case of α-divergence, by observing* $\lim_{\alpha \to 1} \mathcal{D}_\alpha(p\|q) = KL(p\|q)$ *and* $\lim_{\alpha \to 0} \mathcal{D}_\alpha(p\|q) = KL(q\|p)$ *(applying L'Hôpital's rule to* (3.1)*).*

Regarding basic properties of divergence measures, both α-divergence and KL divergence are zero when $p = q$, and they are non-negative.

Denote KL-projection by

$$\text{proj}[p] = \underset{q \in \mathcal{Q}}{\text{argmin}}\, KL(p\|q), \tag{3.3}$$

where $\mathcal{Q}$ is a family of distribution q. According to the stationary point equivalence Theorem in [63], $\text{proj}[p^\alpha q^{1-\alpha}]$ and $\mathcal{D}_\alpha(p\|q)$ have same stationary points (gradient is zero). This equivalence holds by assuming θ is parameter of $q(x)$ and observing

$$\frac{\partial \text{KL}(p\|q)}{\partial \theta} = \int \frac{\partial q(\boldsymbol{x})}{\partial \theta} - \int \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \frac{\partial q(\boldsymbol{x})}{\partial \theta} d\boldsymbol{x},$$
$$\frac{\partial \mathcal{D}_\alpha(p\|q)}{\partial \theta} = \frac{1}{\alpha} \left(\int \frac{\partial q(\boldsymbol{x})}{\partial \theta} - \int \frac{p'(\boldsymbol{x})}{q(\boldsymbol{x})} \frac{\partial q(\boldsymbol{x})}{\partial \theta} d\boldsymbol{x} \right),$$

with $p'(\boldsymbol{x}) = p^\alpha(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})$. Then it gives $\frac{\partial \mathcal{D}_\alpha(p\|q)}{\partial \theta} = \frac{1}{\alpha} \frac{\partial \text{KL}(p\|q)}{\partial \theta}|_{p=p'}$.

A heuristic scheme to find q^* minimizing $\mathcal{D}_\alpha(p\|q)$ starts with an initial q, and repeatedly updates q via the projection on $\mathcal{Q}$

$$q(\boldsymbol{x})^{\text{new}} = \text{proj}[p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha}]. \tag{3.4}$$

This heuristic scheme is a fixed-point iteration, which does not guarantee to converge.

3.2 α Belief Propagation Algorithm

Pairwise MRF

We consider a probability distribution over random vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$, where each x_i takes values in a discrete finite set $\mathcal{A}$. Let us denote the undirected

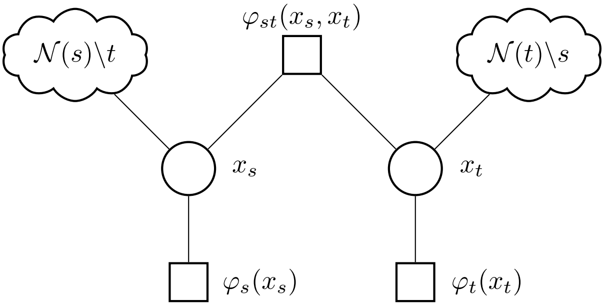


Figure 3.1: Graphic model illustration of $p(\boldsymbol{x})$ in (3.5).

graph of a pairwise MRF by $\mathcal{G} := (\mathcal{V}, \mathcal{E})$. $\mathcal{V} = [1 : N]$ is the node set associated with the index set of entries of $\boldsymbol{x}$. The graph contains undirected edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, where a pair of $(s, t) \in \mathcal{E}$ if and only if nodes v and u are connected by an edge. In addition to the undirected edge set, let us also define the directed edge set induced from $\mathcal{G}$ by $\vec{\mathcal{E}}$. We have $|\vec{\mathcal{E}}| = 2|\mathcal{E}|$, where $|\cdot|$ denotes the carnality. These directed edges serve the purpose of convergence analysis only.

The joint distribution of $\boldsymbol{x}$ can be formulated into a pairwise factorization form in a pairwise MRF as

$$p(\boldsymbol{x}) \propto \prod_{s \in \mathcal{V}} \varphi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \varphi_{st}(x_s, x_t), \tag{3.5}$$

where $\varphi_s : \mathcal{A} \to (0, \infty)$ and $\varphi_{st} : \mathcal{A} \times \mathcal{A} \to (0, \infty)$ are factor potentials. Relation $\propto$ in (3.5) indicates that a normalized factor is needed to turn the right-hand side into a distribution. Here the normalization is omitted. Note the joint of node set and edge set, i.e. $\mathcal{V} \cup \mathcal{E}$, instantiates the factor index set $\mathcal{F}$ of a general MRF in (2.2) here.

The factor graph representation of (3.5) is shown in Figure 3.1. In the figure, $\mathcal{N}(s)$ is the set of variable nodes neighboring x_s via pairwise factors, i.e. $\mathcal{N}(s) = \{t | (t, s) \in \mathcal{E}\}$, and $\backslash$ denotes exclusion.

From α-divergence minimization to α-BP

We start with defining a surrogate distribution and then use the surrogate distribution to approximate a given distribution. The message passing rule of α-BP is derived by solving the distribution approximation problem.

We begin with defining a distribution

$$q(\boldsymbol{x}) \propto \prod_{s \in \mathcal{V}} \tilde{\varphi}_s(x_s) \prod_{(s,t) \in \mathcal{E}} \tilde{\varphi}_{st}(x_s, x_t), \tag{3.6}$$

that is similarly factorized as the joint distribution $p(\boldsymbol{x})$. The distribution $q(\boldsymbol{x})$ acts as a surrogate distribution of $p(\boldsymbol{x})$. The surrogate distribution would be used to estimate inference problems of $p(\boldsymbol{x})$. We further choose $q(\boldsymbol{x})$ such that it can be fully factorized, which means that $\tilde{\varphi}_{s,t}(x_s, x_t)$ can be factorized into product of two independent functions of x_s, x_t respectively. We denote this factorization as

$$\tilde{\varphi}_{s,t}(x_s, x_t) := m_{st}(x_t) m_{ts}(x_s). \tag{3.7}$$

We use the notation $m_{ts}(x_s)$ to denote the factor as a function of x_s. $m_{ts} : \mathcal{A} \to (0, \infty)$, serves as the message along directed edge $(t \to s)$ in our algorithm. Similarly we have factor or message $m_{st}(x_t)$. Then the marginal can be formulated straightforwardly as

$$q_s(x_s) \propto \tilde{\varphi}_s(x_s) \prod_{w \in \mathcal{N}(s)} m_{ws}(x_s). \tag{3.8}$$

Now, we are going to use the heuristic scheme as in (3.4) to minimize the information loss by using a fully factorized $q(\boldsymbol{x})$ to represent $p(\boldsymbol{x})$. The information loss is measured by α-divergence $\mathcal{D}_\alpha(p(\boldsymbol{x}) \| q(\boldsymbol{x}))$.

We perform a factor-wise refinement procedure to update the factors of $q(\boldsymbol{x})$ such that $q(\boldsymbol{x})$ approximates $p(\boldsymbol{x})$. This approach is similar to the factor-wise refinement procedure of assumed density filtering [24, 72] and expectation propagation [61, 63]. Without loss of generality, we begin to refine the factor $\tilde{\varphi}_{ts}(x_t, x_s)$ via α-divergence characterized by α-parameter assigned with α_{ts}. Define $q^{\backslash(t,s)}(\boldsymbol{x})$ as the product of all other factors excluding $\tilde{\varphi}_{ts}(x_t, x_s)$

$$q^{\backslash(t,s)}(\boldsymbol{x}) = q(\boldsymbol{x}) / \tilde{\varphi}_{ts}(x_t, x_s) \propto \prod_{s \in \mathcal{V}} \tilde{\varphi}_s(x_s) \prod_{(v,u) \in \mathcal{E} \backslash (t,s)} \tilde{\varphi}_{vu}(x_v, x_u). \tag{3.9}$$

We also exclude the factor $\varphi_{ts}(x_t, x_s)$ in $p(\boldsymbol{x})$ to obtain $p^{\backslash(t,s)}(\boldsymbol{x})$. Instead of updating $\tilde{\varphi}_{ts}(x_t, x_s)$ directly by solving

$$\operatorname*{argmin}_{\tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s)} \mathcal{D}_{\alpha_{ts}}\Big(p^{\backslash(t,s)}(\boldsymbol{x}) \varphi_{ts}(x_t, x_s) \| q^{\backslash(t,s)}(\boldsymbol{x}) \tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s)\Big), \tag{3.10}$$

we consider the following tractable problem

$$\operatorname*{argmin}_{\tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s)} \mathcal{D}_{\alpha_{ts}}\Big(q^{\backslash(t,s)}(\boldsymbol{x}) \varphi_{ts}(x_t, x_s) \| q^{\backslash(t,s)}(\boldsymbol{x}) \tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s)\Big), \tag{3.11}$$

which searches for new factor $\tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s)$ such q can approximate p better. In (3.11), $\mathcal{D}_{\alpha_{ts}}(\cdot)$ denotes the α-divergence with the correcponding parameter α_{ts}. Note that the approximation (3.11) is accurate when $q^{\backslash(t,s)}(\boldsymbol{x})$ is equal to $p^{\backslash(t,s)}(\boldsymbol{x})$. Using fixed-point update in (3.4), the problem in (3.11) is equivalent to

$$q^{\backslash(t,s)}(\boldsymbol{x}) \tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s) \propto \operatorname{proj}\Big[q^{\backslash(t,s)}(\boldsymbol{x}) \varphi_{ts}(x_t, x_s)^{\alpha_{ts}} \tilde{\varphi}_{ts}(x_t, x_s)^{1-\alpha_{ts}}\Big]. \tag{3.12}$$

Without loss of generality, we update m_{ts} and define

$$\tilde{\varphi}^{\text{new}}_{ts}(x_t, x_s) = m^{\text{new}}_{ts}(x_s) m_{st}(x_t). \tag{3.13}$$

Since KL-projection onto a fully factorized distribution reduces to matching the marginals [45, Proposition 8.3], substituting (3.13) into (3.12), we obtain

$$\sum_{\boldsymbol{x}\backslash x_s} q^{\backslash(t,s)}(\boldsymbol{x})\tilde{\varphi}_{ts}^{\mathrm{new}}(x_t,x_s) \propto \sum_{\boldsymbol{x}\backslash x_s} q^{\backslash(t,s)}(\boldsymbol{x})\varphi_{ts}(x_t,x_s)^{\alpha_{ts}}\tilde{\varphi}_{ts}(x_t,x_s)^{1-\alpha_{ts}}. \tag{3.14}$$

We use summation here. But it should be replaced by integral if $\mathcal{A}$ is a continuous set. Solving (3.14) gives the message passing rule as

$$m_{ts}^{\mathrm{new}}(x_s) \propto m_{ts}(x_s)^{1-\alpha_{ts}}\left[\sum_{x_t}\varphi_{ts}(x_t,x_s)^{\alpha_{ts}} m_{st}(x_t)^{1-\alpha_{ts}}\tilde{\varphi}_t(x_t)\prod_{w\in\mathcal{N}(t)\backslash s} m_{wt}(x_t)\right]. \tag{3.15}$$

As for the singleton factor $\tilde{\varphi}_t(x_t)$, we can do the refinement procedure on $\tilde{\varphi}_t(x_t)$ in the same way as we have done on $\tilde{\varphi}_{ts}(x_t,x_s)$. This gives us the update rule of $\tilde{\varphi}_t(x_t)$ as

$$\tilde{\varphi}_t^{\mathrm{new}}(x_t) \propto \varphi_t(x_t)^{\alpha_t}\tilde{\varphi}_t(x_t)^{1-\alpha_t}, \tag{3.16}$$

which is the belief from factor $\varphi_t(x_t)$ to variable x_t. Here α_t is the local assignment of parameter α in α-divergence in refining factor $\tilde{\varphi}_t(x_t)$. Note, if we initialize $\tilde{\varphi}_t(x_t) = \varphi_t(x_t)$, then it remains the same in all iterations, which makes

$$m_{ts}^{\mathrm{new}}(x_s) \propto m_{ts}(x_s)^{1-\alpha_{ts}}\left[\sum_{x_t}\varphi_{ts}(x_t,x_s)^{\alpha_{ts}} m_{st}(x_t)^{1-\alpha_{ts}}\varphi_t(x_t)\prod_{w\in\mathcal{N}(t)\backslash s} m_{wt}(x_t)\right]. \tag{3.17}$$

In our notations, a factor potential is undirected, i.e. $\varphi_{ts}(x_t,x_s) = \varphi_{st}(x_s,x_t)$ for all $(t,s)\in\mathcal{E}$. When refining factors with α-BP, each factor potential (corresponding to an edge of $\mathcal{G}$) can be associated with a difference setting of α value. In addition we also have $\alpha_{ts} = \alpha_{st}$.

3.3 Remarks on α Belief Propagation

As discussed in Section 3.1, $\mathrm{KL}(p\|q)$ is the special case of $\mathcal{D}_\alpha(p\|q)$ when $\alpha\to 1$. When restricting $\alpha_{st} = 1$ for all $(s,t)\in\mathcal{E}$, the message-passing rule in (3.17) becomes

$$m_{ts}^{\mathrm{new}}(x_s) \propto \sum_{x_t}\varphi_{st}(x_s,x_t)\varphi_t(x_t)\prod_{w\in\mathcal{N}(t)\backslash s} m_{wt}(x_t), \tag{3.18}$$

which is exactly the messages of standard BP [7]. From this point of view, we can say α-BP is a generalization of BP. Additionally, the BP update rule in (3.18) actually corresponds to the fixed-point iteration assignment by solving the Bethe free energy minimization problem in (2.12).

Note although mean field method also uses fully-factorized approximation, it is obtained differently from α-BP and its factorization differs from that of α-BP.

From another perspective, mean field methods is actually using information project from $p(\boldsymbol{x})$ to a fully-factorized space via KL divergence as explained in section 2.4. In addition, α-BP is different from standard BP with damping technique. The later case uses message update rule that differs from (3.18) slightly by the way of assigning updated message.

Additionally, α-BP differs from the tree-reweighted belief propagation [90] by the way of message update rule and also how algorithm is derived. The tree-reweighted BP shares some similarity with α-BP in formula of the message-passing rule, namely the pairwise log-potential functions are scaled by a weight and reweighted old messages appear in computation of new messages. But different from α-BP, tree-reweighted BP is derived by obtaining an upper bound of log-partition function of $p(\boldsymbol{x})$ first via a Jensen's inequality and minimize the upper bound. The upper bound is

$$\begin{aligned} F_T(q) = &\sum_{s\in\mathcal{V}}\sum_{x_s} q_s(x_s)\ln\frac{q_s(x_s)}{\varphi_s(x_s)} + \sum_{(s,t)\in\mathcal{E}}\mu_{st}\sum_{x_s,x_t} q_{st}(x_s,x_t)\ln\frac{q_{st}(x_s,x_t)}{q_s(x_s)q_t(x_t)} \\ &- \sum_{(s,t)\in\mathcal{E}}\sum_{x_s,x_t} q_{st}(x_s,x_t)\ln\varphi_{st}(x_t,x_t), \end{aligned} \tag{3.19}$$

where $0 \leqslant \mu_{st} \leqslant 1$ is defined as the appearance probability of edge $(s,t)\in\mathcal{E}$, which denotes the appearance rate of edge (s,t) among all spanning trees of graph $\mathcal{G}$. Denote the set of all spanning trees of $\mathcal{G}$ by $\mathcal{T}(\mathcal{G})$. μ_{st} is the probability that edge (s,t) exists in a randomly selected spanning tree from $\mathcal{T}(\mathcal{G})$. The appearance rate can be expensive to compute as it is defined on all spanning trees of a graph.

The upper bound F_T can be reduced into the Bethe free energy (2.12) when $\mu_{st} = 1, \forall\ (s,t)\in\mathcal{E}$. The message-passing updates of the tree-reweighted algorithm corresponds to the minimization of F_T with marginalization constraints, which can be written as

$$m_{ts}^{\text{new}}(x_s) \propto \sum_{x_t}\varphi_{st}(x_s,x_t)^{1/\mu_{st}}\varphi_t(x_t)\frac{\prod_{w\in\mathcal{N}(t)\backslash s} m_{wt}(x_t)^{\mu_{wt}}}{m_{st}(x_t)^{1-\mu_{st}}}. \tag{3.20}$$

In the message update rule, both pairwise potential factor and old messages are reweighted, which are different from the way of how pairwise potential factor and old message are reweighted in message update in (3.17). Nevertheless, α-BP is derived in the way that is different from tree-reweighted BP.

From the practical perspective of view, α-BP as a meta algorithm can be used with other methods in hybrid way. Inspired by [26] and assembling methods [35], we can modify the graphical model shown in Figure 3.1 by adding an extra factor potential $\hat{p}_s(x_s)$ to each x_s. The extra factor potential $\hat{p}_s(x_s)$ acts as prior information that can be obtained from other methods. In other words, this factor potential stands for our belief from exterior estimation. Then we can run our α-BP on the modified graph. The modified graph is shown in Figure 3.2.

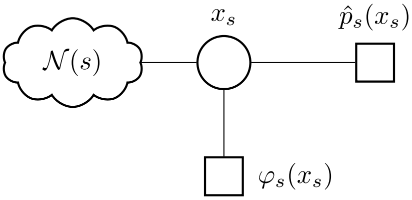


Figure 3.2: Modified graphical model with prior factor.

3.4 Convergence of α-BP with a Binary State Space

As pointed earlier, a key issue of BP and its variants is whether and when they converge. In this section, we discuss when α-BP converges. From the high-level perspective, we are going to use *contraction* property to show when α-BP does converge.

Definition 4. *For a number $c \in [0, 1)$, an operator G over a metric space $(\Delta, d(;))$ is c-contraction relative to the distance function $d(,)$ if for any $\boldsymbol{z}, \boldsymbol{z}' \in \Delta$, we have*

$$d(G(\boldsymbol{z}), G(\boldsymbol{z}')) \leqslant c d(\boldsymbol{z}, \boldsymbol{z}'). \tag{3.21}$$

Definition 4 tells us that an operator is a contraction if its application to two points in the space is guaranteed to decrease the distance between them by at least a constant $c < 1$. *Thus, we essentially are going to show that the message update rule of α-BP is actually a contraction, under which condition we would show that α-BP converges.*

We consider the case of binary $\mathcal{A}$, i.e. $\mathcal{A} = \{-1, 1\}$ to gain insight of α-BP behaviors. The factor potentials are further detailed as

$$\begin{aligned} \varphi_{st}(x_s, x_t) &= \exp\left\{\theta_{st}(x_s, x_t)\right\}, \\ \varphi_s(x_s) &= \exp\left\{\theta_s(x_s)\right\}. \end{aligned} \tag{3.22}$$

Further assume the symmetric property of potentials

$$\begin{aligned} \theta_{ts}(x_t, x_s) &= -\theta_{ts}(x_t, -x_s) = -\theta_{ts}(-x_t, x_s), \\ \theta_s(x_s) &= -\theta_s(-x_s). \end{aligned} \tag{3.23}$$

For notation simplicity, we use $\theta_{ts} = \theta_{ts}(1, 1)$ and $\theta_s = \theta_s(1)$. Denote by $\boldsymbol{\alpha}$ the vector of all local assignments of parameter α, i.e. $\boldsymbol{\alpha} = (\alpha_{ts})_{(t,s)\in\mathcal{E}}$, by $\boldsymbol{\theta}$ the vector of all parameters of potentials, i.e. $\boldsymbol{\theta} = (\theta_{ts})_{(t,s)\in\mathcal{E}}$. Define a matrix $\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})$ of size $|\vec{\mathcal{E}}| \times |\vec{\mathcal{E}}|$, in which its entries are indexed by directed edges $(t \to s)$, as

$$M_{(t\to s),(u\to v)} = \begin{cases} |1 - \alpha_{ts}|, & u = t, v = s, \\ |1 - \alpha_{ts}| \tanh|\alpha_{ts}\theta_{ts}|, & u = s, v = t, \\ \tanh|\alpha_{ts}\theta_{ts}|, & u \in \mathcal{N}(t)\backslash s, v = t, \\ 0, & \text{otherwise}. \end{cases} \tag{3.24}$$

Theorem 1. *For an arbitrary pairwise Markov random field over binary variables, if the largest singular value of matrix $\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})$ is less than one, α-BP converges to a fixed point. The associated fixed point is unique.*

Proof. Let us define z_{ts} as the log ratio of belief from node t to node s on two states of $\mathcal{A}$, i.e.

$$z_{ts} = \log \frac{m_{ts}(1)}{m_{ts}(-1)}. \tag{3.25}$$

By combining the local message passing rule in (3.17) with (3.25), we obtain a local update function $F_{ts} : \mathbb{R}^{|\vec{\mathcal{E}}|} \to \mathbb{R}$ that maps $\boldsymbol{z} = (z_{ts})_{(t \to s) \in \mathcal{E}}$ to updated z_{ts}, which can be expressed as

$$F_{ts}(\boldsymbol{z}) = (1 - \alpha_{ts}) z_{ts} + f_{ts}(\boldsymbol{z}), \tag{3.26}$$

where

$$f_{ts}(\boldsymbol{z}) = \log \frac{\exp\{2\alpha_{ts}\theta_{ts} + \Delta_{ts}(\boldsymbol{z})\} + 1}{\exp\{\Delta_{ts}(\boldsymbol{z})\} + \exp\{2\alpha_{ts}\theta_{ts}\}}, \tag{3.27}$$

with

$$\Delta_{ts}(\boldsymbol{z}) = 2\theta_s + (1 - \alpha_{ts}) z_{st} + \sum_{w \in \mathcal{N}(u) \backslash t} z_{wt}. \tag{3.28}$$

In the following, we use superscript (n) to denotes the n-th iteration. Since f_{ts} is continuous on $\mathbb{R}^{|\vec{\mathcal{E}}|}$ and differentiable, we have

$$\begin{aligned} & z_{ts}^{(n+1)} - z_{ts}^{(n)} \\ = & (1 - \alpha_{ts})(z_{ts}^{(n)} - z_{ts}^{(n-1)}) + f_{ts}(\boldsymbol{z}^{(n)}) - f_{ts}(\boldsymbol{z}^{(n-1)}) \\ \overset{(a)}{=} & (1 - \alpha_{ts})(z_{ts}^{(n)} - z_{ts}^{(n-1)}) + \nabla f_{ts}(\boldsymbol{z}^{\lambda})^T (\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)}), \end{aligned} \tag{3.29}$$

where (a) follows by the mean-value theorem, $\boldsymbol{z}^{\lambda} = \lambda \boldsymbol{z}^{(n)} + (1 - \lambda) \boldsymbol{z}^{(n-1)}$ for some $\lambda \in (0, 1)$, and $\nabla f_{ts}(\boldsymbol{z}^{\lambda})$ denotes the gradient of f_{ts} evaluated at $\boldsymbol{z}^{\lambda}$. In further details, ∇f_{ts} is given by

$$\frac{\partial f_{ts}}{\partial z} = \begin{cases} (1 - \alpha_{ts}) \frac{\partial f_{ts}}{\partial \Delta_{ts}}, & z = z_{st}, \\ \frac{\partial f_{ts}}{\partial \Delta_{ts}}, & z = z_{wt}, w \in N(t) \backslash s. \\ 0, & \text{otherwise.} \end{cases} \tag{3.30}$$

Our target here is to find the condition to make sequence $\left(z_{ts}^{(n+1)} - z_{ts}^{(n)} \right)$ to converge. To this aim we need to bound the term $\nabla f_{ts}(\boldsymbol{z}^{\lambda})^T (\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)})$ in (3.29). For this purpose, we need two auxiliary functions $H, G : \mathbb{R}^2 \to \mathbb{R}$ from lemma 4 in [79], which are cited herein for completeness

$$\begin{aligned} H(\mu;\kappa) &:= \log \frac{exp(\mu+\kappa)+1}{exp(\mu)+exp(\kappa)}, \\ G(\mu;\kappa) &:= \frac{exp(\mu+\kappa)}{exp(\mu+\kappa)+1} - \frac{exp(\mu)}{exp(\mu)+exp(\kappa)} \\ &= \frac{\sinh\kappa}{\cosh\kappa+\cosh\mu}, \end{aligned} \tag{3.31}$$

where it holds that $\frac{\partial H(\mu;\kappa)}{\partial\mu} = G(\mu;\kappa)$. Further, it holds that $|G(\mu;\kappa)| \leqslant |G(0,\kappa)| = \tanh(|\kappa|/2)$. Then we have

$$\begin{aligned} f_{ts}(z) &= H(\Delta_{ts}(\boldsymbol{z}); 2\alpha_{ts}\theta_{ts}), \\ \frac{\partial f_{ts}}{\partial \Delta_{ts}} &= G(\Delta_{ts}(\boldsymbol{z}); 2\alpha_{ts}\theta_{ts}), \end{aligned} \tag{3.32}$$

which implies

$$\left|\frac{\partial f_{ts}}{\partial \Delta_{ts}}\right| \leqslant \tanh(\alpha_{ts}\theta_{ts}). \tag{3.33}$$

Combining (3.29), (3.30), (3.32) and (3.33), we have

$$\begin{aligned} &|z_{ts}^{(n+1)} - z_{ts}^{(n)}| \\ &= |(1-\alpha_{ts})(z_{ts}^{(n)} - z_{ts}^{(n-1)}) + \nabla f_{ts}(\boldsymbol{z}^{\lambda})^T(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)})| \\ &\leqslant |(1-\alpha_{ts})(z_{ts}^{(n)} - z_{ts}^{(n-1)})| + |\nabla f_{ts}(\boldsymbol{z}^{\lambda})^T(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)})| \\ &= |1-\alpha_{ts}||z_{ts}^{(n)} - z_{ts}^{(n-1)}| + |\nabla f_{ts}(\boldsymbol{z}^{\lambda})|^T|\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)}| \\ &\overset{(a)}{\leqslant} |1-\alpha_{ts}||z_{ts}^{(n)} - z_{ts}^{(n-1)}| + |1-\alpha_{ts}|\tanh(|\alpha_{ts}\theta_{ts}|)|z_{st}^{(n)} - z_{st}^{(n-1)}| \\ &+ \sum_{w\in N(t)\setminus s} \tanh(|\alpha_{ts}\theta_{ts}|)|z_{wt}^{(n)} - z_{wt}^{(n-1)}|, \end{aligned} \tag{3.34}$$

where step (a) holds by applying (3.30) and (3.33).

Concatenating all $(t \to s) \in \vec{\mathcal{E}}$ for inequality (3.34) gives

$$|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}| \leqslant \boldsymbol{M}(\alpha,\boldsymbol{\theta})|\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1}|, \tag{3.35}$$

where $\boldsymbol{M}(\boldsymbol{\alpha},\boldsymbol{\theta})$ is defined in (3.24), and $\leqslant$ in (3.35) denotes the element-wise inequality. From (3.35), we could further have

$$\|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}\|_p \leqslant \|\boldsymbol{M}(\alpha,\boldsymbol{\theta})(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1})\|_p, \tag{3.36}$$

where $1 \leqslant p < \infty$, and $\|\cdot\|_p$ denotes the ℓ^p-norm.

When applying $p = 2$ to (3.36), we have

$$\begin{aligned}\|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{(n)}\|_2 &\leqslant \|\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)})\|_2 \\ &\leqslant \lambda^*(\boldsymbol{M})\|\boldsymbol{z}^{(n)} - \boldsymbol{z}^{(n-1)}\|_2, \end{aligned} \tag{3.37}$$

where $\lambda^*(\boldsymbol{M})$ denotes the largest singular value of matrix $\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})$. If the largest singular value of $\boldsymbol{M}$ is less than 1, the sequence $\left(|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{(n)}|\right)$ converges to zero in ℓ^2-norm as $n \to \infty$. Therefore, for $\lambda^*(\boldsymbol{M}) < 1$, ℓ^2-norm $\left(\boldsymbol{z}^{(n)}\right)$ is a Cauchy sequence and must converge.

By concatenating local update function (3.26), we have a global update function $\boldsymbol{F} = (F_{ts})_{(t\to s)\in\vec{\mathcal{E}}}$, which defines a mapping from $\mathbb{R}^{|\vec{\mathcal{E}}|}$ to $\mathbb{R}^{|\vec{\mathcal{E}}|}$. $\boldsymbol{F}$ is a continuous function of $\boldsymbol{z}$, we have

$$\boldsymbol{F}(\lim_{n\to\infty} \boldsymbol{z}^{(n)}) = \lim_{n\to\infty} \boldsymbol{F}(\boldsymbol{z}^{(n)}). \tag{3.38}$$

Assume that $\left(\boldsymbol{z}^{(n)}\right)$ converges to $\boldsymbol{z}^*$. Then

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{z}^*) - \boldsymbol{z}^* &= \lim_{n\to\infty} \boldsymbol{F}(\boldsymbol{z}^{(n)}) - \lim_{n\to\infty} \boldsymbol{z}^{(n)} \\ &= \lim_{n\to\infty} (\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{(n)}) \\ &= 0. \end{aligned} \tag{3.39}$$

Thus $\boldsymbol{z}^*$ must be a fixed point.

In what follows we show that the fixed point is unique when $\lambda^*(\boldsymbol{M}) < 1$. Assume that there are two fixed points $\boldsymbol{z}_0^*$ and $\boldsymbol{z}_1^*$ for sequence $\left\{\boldsymbol{z}^{(n)}\right\}$. Then we have

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{z}_0^*) &= \boldsymbol{z}_0^*, \\ \boldsymbol{F}(\boldsymbol{z}_1^*) &= \boldsymbol{z}_1^*. \end{aligned} \tag{3.40}$$

Applying (3.37) gives

$$\|\boldsymbol{F}(\boldsymbol{z}_0^*) - \boldsymbol{F}(\boldsymbol{z}_1^*)\|_2 \leqslant \lambda^*(\boldsymbol{M})\|\boldsymbol{z}_0^* - \boldsymbol{z}_1^*\|_2. \tag{3.41}$$

Substituting (3.40) into (3.41) gives

$$\|\boldsymbol{z}_0^* - \boldsymbol{z}_1^*\|_2 \leqslant \lambda^*(\boldsymbol{M})\|\boldsymbol{z}_0^* - \boldsymbol{z}_1^*\|_2, \tag{3.42}$$

which gives us $\boldsymbol{z}_0^* = \boldsymbol{z}_1^*$ and completes the uniqueness of the fixed point. □

Remark 6. *From Theorem 1 we can see that, the sufficient condition for convergence of α-BP is $\lambda^*(\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})) < 1$. It is interesting to notice that $\lambda^*(\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta}))$ is a function of $\boldsymbol{\alpha}$ from α-divergence and $\boldsymbol{\theta}$ from joint distribution $p(\boldsymbol{x})$. This means that whether α-BP can converge depends on the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the problem $p(\boldsymbol{x})$ and also the choice of $\boldsymbol{\alpha}$. Therefore, proper choice of $\boldsymbol{\alpha}$ can guarantee the convergence of α-BP if the sufficient condition can possibly be achieved for given $\boldsymbol{\theta}$.*

Given the fact that α-BP would converge if the condition in Theorem 1 is fulfilled, the largest singular value computation for large-sized graph could be non-trivial. We give alternative sufficient conditions for the convergence of α-BP.

Corollary 1. *α-BP would converge to a fixed point if the condition*

$$\max_{u\to v} |1-\alpha_{uv}| + |1-\alpha_{vu}| \tanh\left(|\alpha_{vu}\theta_{vu}|\right) + \sum_{w\in\mathcal{N}(v)\backslash u} \tanh\left(|\alpha_{vw}\theta_{vw}|\right) < 1, \tag{3.43}$$

is fulfilled or the condition

$$\max_{t\to s} |1-\alpha_{ts}|(1+\tanh(|\alpha_{ts}\theta_{ts}|)) + (|\mathcal{N}(t)|-1)\tanh(|\alpha_{ts}\theta_{ts}|) < 1. \tag{3.44}$$

is achieved, where $|\mathcal{N}(t)|$ denotes the carnality of the set $\mathcal{N}(t)$. The associated fixed point is unique.

Proof. Setting $p = 1$ to (3.36), we have

$$\|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}\|_1 \leqslant \|\boldsymbol{M}(\alpha,\boldsymbol{\theta})(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1})\|_1. \tag{3.45}$$

Furthermore, from (3.36), we also have

$$\|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}\|_\infty \leqslant \|\boldsymbol{M}(\alpha,\boldsymbol{\theta})(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1})\|_\infty, \tag{3.46}$$

where $\|\cdot\|_\infty$ denotes the ℓ^∞-norm. Then we have

$$\begin{aligned} \|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}\|_1 &\leqslant \|\boldsymbol{M}\|_1 \|(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1})\|_1, \\ \|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}\|_\infty &\leqslant \|\boldsymbol{M}\|_\infty \|(\boldsymbol{z}^{(n)} - \boldsymbol{z}^{n-1})\|_\infty, \end{aligned} \tag{3.47}$$

where we omit the parameters of $\boldsymbol{M}$ here for simplicity. We can expand the first multiplicand on the right hand side of (3.47) as follows

$$\begin{aligned} \|\boldsymbol{M}\|_1 =& \max_{u\to v} \sum_{t\to s} M_{(t\to s),(u\to v)} \\ =& \max_{u\to v} |1-\alpha_{uv}| + |1-\alpha_{vu}| \tanh|\alpha_{vu}\theta_{vu}| + \sum_{w\mathcal{N}(v)\backslash u} \tanh|\alpha_{vw}\theta_{vw}|, \\ \|\boldsymbol{M}\|_\infty =& \max_{t\to s} \sum_{u\to v} M_{(t\to s),(u\to v)} \\ =& \max_{t\to s} |1-\alpha_{ts}|(1+\tanh|\alpha_{ts}\theta_{ts}|) + (|\mathcal{N}(t)|-1)\tanh|\alpha_{ts}\theta_{ts}|. \end{aligned} \tag{3.48}$$

When condition $\|\boldsymbol{M}\|_1 < 1$ is met, sequence $\left(|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}|\right)$ approaches to zero as $n \to \infty$. Similarly, condition $\|\boldsymbol{M}\|_\infty < 1$ can also guarantee the convergence to zero of sequence $\left(|\boldsymbol{z}^{(n+1)} - \boldsymbol{z}^{n}|\right)$. The analysis for uniqueness of converged fixed point is similar to that in proof of Theorem 1. □

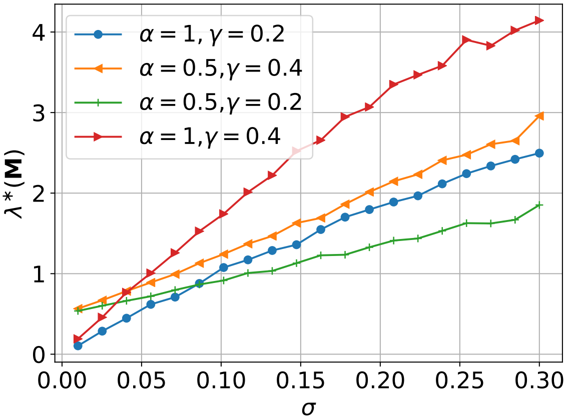


Figure 3.3: The largest singular value of $\boldsymbol{M}$ defined in (3.24) versus variance of potential parameter $\boldsymbol{\theta}$. A value of each curve is the mean of 100 graph realizations.

3.5 Experiments

In this section, we firstly give the simulations for convergence condition of α-BP explained in Theorem 1. Then the application of α-BP to a MIMO detection problem is demonstrated.

Simulated Results on Random Graphs

In this section simulations on random graphs are carried out to gain some insights on the α-BP. The random graphs used here are generated by Erdos-Rényi (ER) model [20]. In generating a graph by EP model, an edge between any two nodes is generated with probability γ, $\gamma \in (0, 1)$.

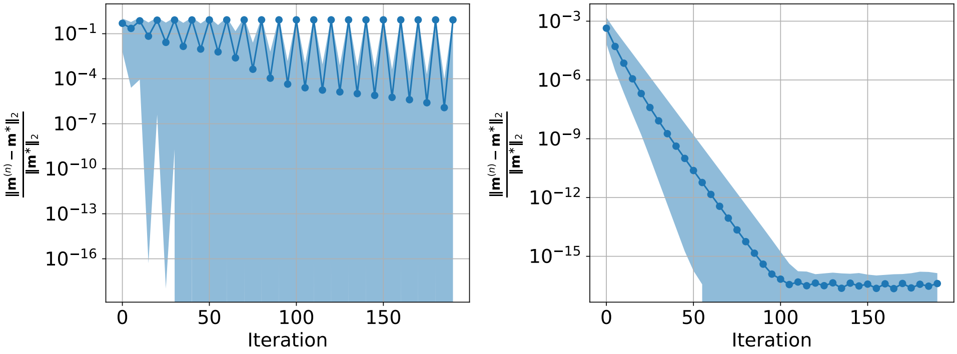
Note that the MRF joint probability in (3.5) can be reformulated into

$$p(\boldsymbol{x}) \propto \exp\{-\boldsymbol{x}^T \boldsymbol{J} \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x}\}, \boldsymbol{x} \in \mathcal{A}^N, \tag{3.49}$$

with $\varphi_{ts}(x_t, x_s) = e^{-2J_{t,s}x_t x_s}$ and $\varphi_s(x_s) = e^{-J_{s,s} - b_s x_s}$. J here is the weighted adjacency matrix. In our experiments, we generate a random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with γ by ER model and then associate potential factors to the graph. Specifically, factor $\varphi_s(x_s)$ is associated to node x_s, $s \in \mathcal{V}$, and φ_{ts} to edge $(t, s) \in \mathcal{E}$. J_{ts} is zero if these is no edge (t, s).

For this set of experiments, we set $\mathcal{A} = \{-1, 1\}$ and $N = 16$. To specify (3.49), the non-zero entries of $\boldsymbol{J}$ is sampled from a Gaussian distribution $\mathsf{N}(0, \sigma^2)$, i.e. $J_{ts} \sim \mathsf{N}(0, \sigma^2)$ if $J_{ts} \neq 0$. For entries of $\boldsymbol{b}$, we use $b_t \in \mathsf{N}(0, (\sigma/4)^2)$. For each edge $(t, s) \in \mathcal{E}$, we set $\alpha_{ts} = \alpha$, i.e. the edges share a global value α.

Figure 3.3 illustrate how the largest singular value of $\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})$ as defined in (3.24) changes when the standard deviation σ of potential factors increases. The behavior is illustrated with different values of α and the edge probability γ. For



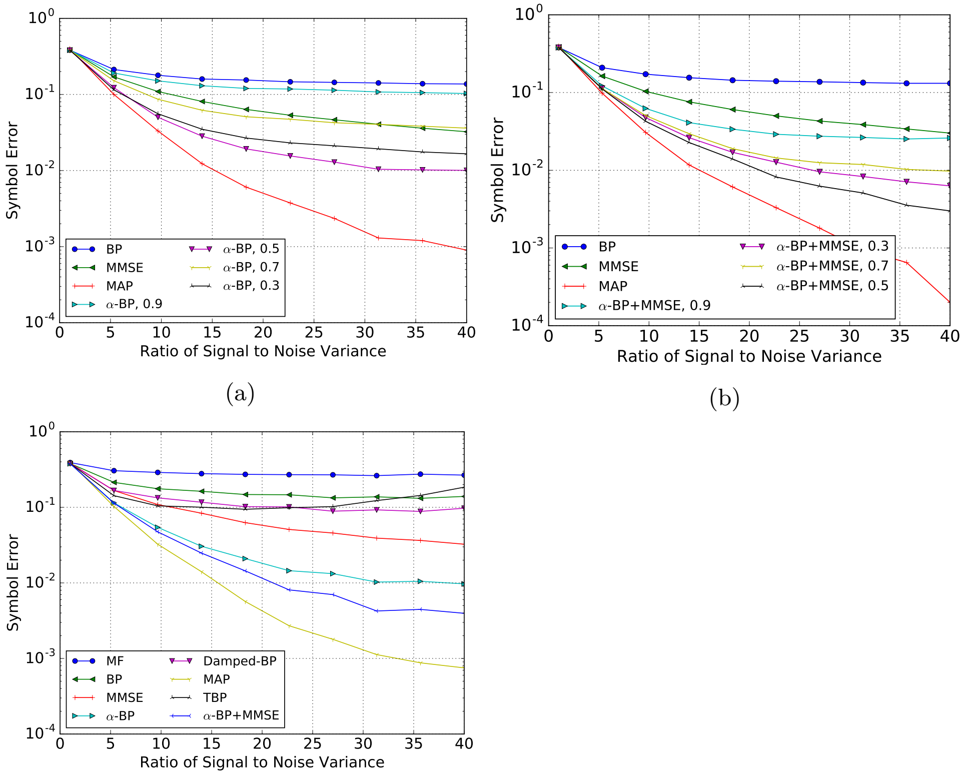
(a) (b)

Figure 3.4: Numerical illustration on convergence, with normalized error $\|\boldsymbol{m}^{(n)} - \boldsymbol{m}^*\|_2/\|\boldsymbol{m}^*\|_2$ versus the number of iterations. Number of nodes $N = 16$. 3.4a parameter setting: $\gamma = 0.4$, $\alpha = 1$ (equivalent to standard BP), $\sigma = 0.5$. 3.4b Parameter setting: $\gamma = 0.2$, $\alpha = 0.5$, $\sigma = 0.1$. Blue region denotes the range from minimum to maximum of the normalized error of 100 graph realizations, whereas the curve stands for mean error of the 100 realized graphs.

each curve, a point on the curve is the mean of 100 realizations of random graphs as described above. The curves of Figure 3.3 show in general that a larger standard deviation of potential factors of graph edges makes it more difficult to fulfill the convergence condition in Theorem 1. This is also the case when a graph is denser as we raise the edge probability γ in generating random graphs, by comparing the green and orange curves. The comparison between green and blue curves indicates that choice of α value in α-BP also makes a difference, and its effect depends on the graph itself. How to tune α value to fulfill condition of Theorem 1 depends not only how dense (γ) the graph is, but also how potential factors spread out from each other.

To illustrate our developed convergence condition for α-BP, we also observe how messages in a graph changes along belief propagation iterations. To be specific, we run our α-BP with 200 iterations on a graph, after which the messages in the graph are denoted by $\boldsymbol{m}^*$. $\boldsymbol{m}^*$ can be the converged messages if α-BP has converged within the 200 iterations. Then we measure the quantity $\|\boldsymbol{m}^{(n)} - \boldsymbol{m}^*\|_2/\|\boldsymbol{m}^*\|_2$ during the iterations. In Figure 3.4a, we generate 100 random graphs by ER model with parameter setting as $\gamma = 0.4$, $\alpha = 1^2$, $\sigma = 0.5$. By referring to the curves in Figure 3.3, it can be inferred that this setting does not fulfill the condition in Theorem 1. The log error changes versus iteration number n for the 100 graphs are shown in Figure 3.4a, in which the blue region indicates the range and the solid curve indicates the mean of the normalized errors. It is clear that Figure 3.4a does not show any sign of convergence within 200 iterations.

[2] $\alpha = 1$ in α-BP corresponding to standard BP.



(a)

(b)

(c)

Figure 3.5: Numerical results of α-BP: symbol error of MIMO detection.

We then carry out a set of experiments in Figure 3.4b similar to our experiments in Figure 3.4a. The only difference lies in the graph generating process. Here we set the parameters to be $\gamma = 0.2$, $\alpha = 0.5$, $\sigma = 0.1$. According to our curves in Figure 3.3, a graph generated with this parameter setting should fulfill the condition in Theorem 1. Due to randomness of both graph generating by ER and potential factors, we regenerate a graph if the initial generated graph does not satisfy $\lambda^*(\boldsymbol{M}) < 1$. Therefore the 100 graphs used in experiments for Figure 3.4b all fulfill the Theorem 1. The result in Figure 3.4b is consistent with our analysis on the convergence of α-BP.

Full-connected Graph Case: Application to MIMO Detection

In this section, we show the application of α-BP to a MIMO detection problem. For a MIMO system, the observation $\boldsymbol{y}$ is a linear function of channel $\boldsymbol{H} \in \mathbb{R}^{N \times N}$ when unknown signal $\boldsymbol{x}$ need to be estimated,

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{e}, \boldsymbol{x} \in \mathcal{A}^N, \tag{3.50}$$

where $\boldsymbol{e}$ is noise modeled as Gaussian noise $\boldsymbol{e} \sim \mathsf{N}\left(\mathbf{0}, \sigma_w^2 \boldsymbol{I}\right)$. Here $\boldsymbol{I}$ is unitary matrix. In this case, the posterior of $\boldsymbol{x}$ can be written as:

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{y}) &\propto e^{-\frac{1}{2\sigma_w^2}\|\boldsymbol{Hx}-\boldsymbol{y}\|_2^2} \\ &= e^{-\frac{1}{2\sigma_w^2}\left[\boldsymbol{x}^T\boldsymbol{H}^T\boldsymbol{Hx}-2\boldsymbol{y}^T\boldsymbol{Hx}+\boldsymbol{y}^T\boldsymbol{y}\right]}. \end{aligned} \tag{3.51}$$

Denote $\boldsymbol{S} = \boldsymbol{H}^T\boldsymbol{H}$, $\boldsymbol{h}_i$ as the i-th column of $\boldsymbol{H}$, and

$$\begin{aligned} \varphi_i(x_i) =& e^{-\frac{S_{i,i}x_i^2}{2\sigma_w^2}+\frac{\langle \boldsymbol{h}_i,\boldsymbol{y}\rangle x_i}{\sigma_w^2}}, \\ \varphi_{ij}(x_i,x_j) =& e^{-\frac{x_i S_{i,j} x_j}{\sigma_w^2}}. \end{aligned} \tag{3.52}$$

Then it can be seen that (3.51) is an instance of (3.5). We set $\mathcal{A} = \{-1, 1\}$, $N = 8$, and $\boldsymbol{H} \in \mathbb{R}^{8\times 8}$ sampled from Gaussian.

We test the application of α-BP to the MIMO signal detection numerically. We run the α-BP, without the prior trick (Subsection 3.3) in Figure 3.5a and with the prior in Figure 3.5b (legend "α-BP+MMSE") as estimation of minimum mean square error (MMSE). The reference results of MMSE and maximum a posterior (MAP, exhausted search) are also reported under the same conditions. MMSE estimator depends on Gaussian posterior $\mathsf{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$, where $\hat{\boldsymbol{\mu}} = (\boldsymbol{H}^T\boldsymbol{H}+\sigma_w^2\boldsymbol{I})^{-1}\boldsymbol{H}^T\boldsymbol{y}$ and $\hat{\boldsymbol{\Sigma}} = (\boldsymbol{H}^T\boldsymbol{H}+\sigma_w^2\boldsymbol{I})^{-1}\sigma_w$. Detection of MMSE carried out by $\mathrm{argmin}_{x_i\in\mathcal{A}} |x_i - \hat{\mu}_i|$.

Figure 3.5a shows that BP even underperforms MMSE but α-BP can outperform MMSE by assigning smaller value of α. Note that MMSE requires the matrix inverse computation whose complexity is proportional to N^3, while the complexity of α-BP increases linearly with N. Therefore α-BP is superior to MMSE both performance-wise and complexity-wise. However, there is still a big gap between α-BP (even for $\alpha = 0.5$) and MAP. This gap can be decreased further by using the prior trick discussed in Subsection 3.3. Figure 3.5b exemplifies this effects by using prior belief from MMSE, $\hat{p}_i(x_i) \propto \exp\{-(x_i - \hat{\mu}_i)^2/(2\hat{\boldsymbol{\Sigma}}_{i,i})\}$, by modifying the graph as shown in Figure 3.2, which comes with legend "α-BP+MMSE". It is shown that larger performance gain is observed when α-BP runs with prior belief.

Additional, we also carry out the experiments where the proposed α-BP is compared with mean field (legend 'MF'), BP with damping technique [76] (with legend 'Damped-BP'), and Tree-reweighted belief propagation [90] (with legend 'TBP') in Figure 3.5c. As expected, mean field method performs no better than BP. Damping technique improves BP's performance with a noticeable difference but still falls behind MMSE. The performance of tree-reweighted BP reaches that of MMSE in low ratio range of signal-to-noise variance but degenerates a lot in the high ratio range. The old message and potential factors are reweighted by the edge appearance probability in TBP to compute new messages. In TBP, the edge appearance probability is the probability that the edge exists in a randomly chosen spanning tree from all possible spanning trees of graph $\mathcal{G}$, which is usually expensive to compute.

3.6 Summary

In this chapter, we went through an alternative development of belief propagation. The alternative method was connected and compared with the classic approximate iterative inference methods, namely mean filed, loopy belief propagation and tree-reweighted belief propagation. The connection and comparison were made with regarding to both the high-level optimization objectives and the practical iterative update rules. All methods share a common methodology, *inference an optimization.* Although they started from different optimization cost functions, the derived iterative update rules share similarity. The study enriches our insight of deterministic approximate inference approaches.

Given the wide application of the family of belief propagation methods, when an implementation of them can have guaranteed convergence is a fundamental question, which is of practical interest in general. This issue was attempted in binary support case in this chapter via the method of contraction condition. The derived sufficient conditions for α-BP gives us the preview about whether we can expect it to converge.

Another essential question is about the error of deterministic approximate inference, compared with the exact answer. This problem is surely challenging. There is little work offering error-bounded methods in deterministic approximate inference family. Due to the lack of knowledge here, manual turning or trial-and-error is still not able to be avoided in practical system implementations. The development towards automated inference methods with at least less manual work is always an important direction in this track.

3.7 Relevant Literature

Approximate inference are applicable to wide range of settings. The wide applications include but not limited to imaging processing [105], multi-input-multi-output (MIMO) signal detection in digital communication [11,36], inference on structured lattice [21], machine learning [55, 65, 103]. The empirical success of belief propagation (BP) rules came much earlier than its theoretical examination, which dates back to 80s in last century [74].

As a result of the representation power of probabilistic graphical models, graphs with loops are inevitable in real-world problems. Before there was any justification, problems represented by graphs with loops simply employ BP as if there was no loop, i.e. loopy BP. Although loopy BP is still a practical method to do inference approximately, its performance varies from case to case and its behavior is not well understood in general. A direct workaround is to propagate messages on a manipulated graph instead of the original graph. The representative methods of this family are junction tree (clique tree) method [45, section 10] and generalized belief propagation (GBP) [100]. Although they both cluster multiple nodes of the original graph into a node in a hyper-graph (clique tree or region graph) and propagate message in the new graph. Junction tree provides a exact inference method while GBP is an approximate inference method. How to convert a general graph to a junction tree or region graph is not trivial, and structure of the converted graph makes significant difference in inference performance. Additionally, the former's complexity relies on tree-width (the size of the largest clique minuses one), which means that there is not too much gain by using junction tree than enumerating configurations in very dense graphs. For the latter, constructing a region graph itself is a challenging task and still need further study.

Apart from the work of transforming the problem represented by a loopy graph into one of a hyper-graph, research is more active in approximate methods. Starting from the stationary point explanation of Bethe free energy in [102], variants of BP have been derived to improve BP in general graph. Fractional BP in [96] applies a correction coefficient to each factor and obtain a message passing rule similarly as minimization of Bethe free energy. Generalized BP in [102] propagates belief between different regions of a graph; and damping BP in [76] updates beliefs by combining old and new beliefs. [90] relaxes a Bethe free energy into an upper bound of partition function and the tree-reweighed BP is obtained. Technique such as damping is also explored to seek convergence of BP and its variants [76]. Another track falls to the variational method framework, introduced by Opper and Winther [73] and Minka [61, 62], namely expectation propagation (EP). In EP, a simpler factorized distribution defined in exponential distribution family is used to approximate the original complex distribution, and an intuitive factor-wise refinement procedure is used to find such an approximate distribution. The method intuitively minimizes a localized Kullback-Leibler (KL) divergence. This is discussed further in [63] and shows unifying view of message passing algorithms. The following work, stochastic EP by [54], explores EP's variant method for applications to large dataset.

Due to the fundamental role of BP for probabilistic inference and related applications, research of seeking insight of BP performance and study on its convergence have been constantly carried out. [92] presents the convergence condition of BP in graphs containing a single loop. Work in [32] analyzes the Bethe free energy and offers sufficient conditions on uniqueness of BP fixed point. Closely related to the content of this chapter, [67] studies the sufficient conditions for BP convergence to a unique fixed point (as shall be seen in our paper, our convergence analysis is on a message-passing method that generalizes BP). [70] proposes a BP algorithm for high-dimensional discrete space and gives the convergence conditions of it. [44] shows that BP can converge to global optima of Bethe energy when BP runs in Ising models that are ferromagnetic (neighboring nodes prefer to be aligned). There are also works trying to give insight on variant methods of BP. Namely, [18, 59] studies the convergence condition of Gaussian BP inference over distributed linear Gaussian models. [79] gives the convergence analysis of a reweighted BP algorithm, and offers the necessary and sufficient condition for subclasses of homogeneous graphical models with identical potentials.

Chapter 4

Inference as Optimization: An Region-based Energy Method

In last chapter, we explained an iterative message passing algorithm and discussed the connection with mean field, belief propagation and tree-reweighted belief propagation. These methods fellow different message-passing rules (fix point iterations) that are manually developed. The manual efforts in implementation for a practical inference problem includes both the message-passing rule definition and also message update schedule, both of which make a significant difference in performance of approximate inference. However, on our way towards automated inference methods, we intend to reduce the manual efforts without degenerating their performance. In this chapter, we would discuss one promising way to realize this target. The principle idea is to do inference as solving an optimization problem, i.e. inference as optimization.

In fact, we have touched this topic in section 2.4, where we were trying to interpret what message-passing updates of mean field and belief propagation are actually doing. It turns out that the message passing rules are fix-point iterations of optimization problems. Therefore mean field and belief propagation get the intuition of minimization of variational free energy. Take the most widely used belief propagation method as example, since the message-passing rule can be obtained from minimization of Bethe free energy cost, we may just as well solve the optimization problem by other optimization techniques such as gradient descent. But early attempt on this track showed that it might suffer from the stability for peaky potential functions compared to iterative message-passing method [95].

Another limitation lies on the Bethe approximation itself. When representing Bethe approximation in a factor graph (not necessary to be pairwise case), a factor node associated with a potential function is related with more than one random variable, while a variable node is associated with one random variable. When beliefs are propagated on the graph, only a univariate marginal distribution (associated with a variable node) can be propagated from one neighboring factor to another

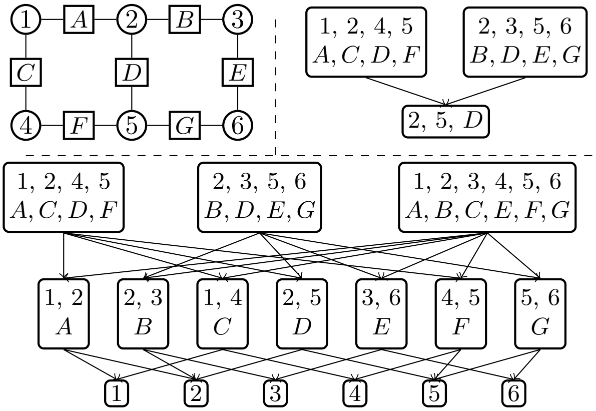


Figure 4.1: Illustration of a factor graph for 2-by-3 grid (top left, variable nodes are indexed by number and factor nodes by letters), and two alternative regions graphs (two levels for the top right one and three levels for bottom one) constructed from the factor graph.

neighboring factor of the variable node. The interactions between variables can not be directly propagated as messages in the graph. This limitation actually affects the performance of Bethe approximation in general graphs, especially the loopy graphs.

In this chapter, we seek to use larger clusters in graphical representation to overcome the above mentioned limitation. Along with that, we would bring the concept of inference as optimization into the alternative representation.

4.1 Region Graph and Generalized Belief Propagation

In a MRF, the underlining probability distribution of its N-dimensional random vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ can be written as

$$p(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{a \in \mathcal{F}} \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a), \tag{4.1}$$

where potential functions' index set is instantiated as $\mathcal{F} = \{A, B, \cdots, M\}$. Here we explicitly denote that potential φ_a has its own parameter $\boldsymbol{\theta}_a$, and $\boldsymbol{\theta} = \{\boldsymbol{\theta}_a, a \in \mathcal{F}\}$. For notation simplicity, we define that each variable has K states, i.e. $x_i \in \mathcal{X}_i$ with $|\mathcal{X}_i| = K$, where $|\cdot|$ denotes the cardinality.

We have discussed that loopy BP as a message-passing algorithm operates on factor graphs (as the top-left example in Figure 4.1) in section 2.4. We then can compute the marginal distributions or most likely $\boldsymbol{x}$ of (4.1) with collection of messages after their propagation under rule (2.14) or its variants discussed in chapter 3. The fundamental limitation of methods in this family that a message over a single variable is communicated instead of over pairs of variables or multiple variables (as stated at the beginning of this chapter). This issue can be relieved by using hyper-graph or cluster graph where a node is associated with multiple variables. Region graph is such kind of graphs, which was proposed by [93, 100] in assistance of scheduling messages for generalized belief propagation (GBP). For illustration purpose, two alternative region graphs constructed from the same factor graph are shown in Figure 4.1.

Before giving definition of region graph, we need to define *region* first, which is on the basis of factor graph $\mathcal{G}_{\mathcal{F}}(\mathcal{V} \cup \mathcal{F}, \mathcal{E}_{\mathcal{F}})$ (definition 1).

Definition 5. *A region R is a set V_R of variables nodes and a set A_R of factor nodes, such that if a factor node a belong to A_R, all the variables nodes neighboring a are in V_R.*

Then, a region graph is defined as bellow.

Definition 6. *A region graph is a directed graph $\mathcal{G}_R(\mathcal{R}, \mathcal{E})$, where each vertex $R \in \mathcal{R}$ is defined as the joint set of variable and factor nodes in this region, i.e. $R = \{i \in V_R, a \in A_R | i \in \mathcal{V}, a \in \mathcal{F}\}$. Each edge $e \in \mathcal{E}$ in $\mathcal{G}_R$ is directed from R_p to R_c such that $R_c \subset R_p$.*

We define some notations before going further. Since $\mathcal{G}_R$ is a hierarchical directed graph, $\mathcal{R}_l$ denotes regions in level l, and $R_i^{[l]} \in \mathcal{R}_l$ denotes $R_i^{[l]}$ is the i-th region node in level l. This means $\mathcal{R}_0$ is set of the top root regions that have no parents, i.e. the level 0 regions. Also, $R^{[l]}$ means it can be any node in $\mathcal{R}_l$ and R denotes a region node when it is not clear or doesn't matter which level it locates. Lastly, due to a region may be associated with both variable and factor, we denote the scope of R by $\boldsymbol{x}_R = \{x_i | i \in R\}$.

GBP operates on region graph $\mathcal{G}_R(\mathcal{R}, \mathcal{E})$. A message is always sent from a parent region P to a child region R, i.e. directed edge $(P, R) \in \mathcal{E}$. Let us define the factors in region R as $A_R = \{a | a \in R\}$. The same as notation in paper, $\mathcal{P}(R)$ denotes the set of parent regions of R. The descendants of R is denoted by $\mathcal{D}(R)$ (excluding R). The descendants of R including R is denoted by $\hat{\mathcal{D}}(R) = \mathcal{D}(R) \cup R$. The message update rule from the parent region P to the child region R is

$$m_{P \to R} \propto \frac{\sum_{\boldsymbol{x}_P \backslash \boldsymbol{x}_R} \prod_{a \in A_P \backslash A_R} \varphi_a(\boldsymbol{x}_a) \prod_{(I,J) \in \mathcal{N}(P,R)} m_{I \to J}(\boldsymbol{x}_J)}{\prod_{(I,J) \in \mathcal{H}(P,R)} m_{I \to J}(\boldsymbol{x}_J)}, \tag{4.2}$$

where

$$\begin{aligned} \mathcal{N}(P,R) &= \left\{ (I,J) \in \mathcal{E} | J \in \hat{\mathcal{D}}(P) \backslash \hat{\mathcal{D}}(R), I \notin \hat{\mathcal{D}}(P) \right\}, \\ \mathcal{H}(P,R) &= \left\{ (I,J) \in \mathcal{E} | J \in \hat{\mathcal{D}}(R), I \in \hat{\mathcal{D}}(P) \backslash \hat{\mathcal{D}}(R) \right\}. \end{aligned} \tag{4.3}$$

The belief for each region R is given by

$$b_R(\boldsymbol{x}_R) \propto \prod_{a \in A_R} \varphi_a(\boldsymbol{x}_a) \prod_{P \in \mathcal{P}(R)} m_{P \to R}(\boldsymbol{x}_R) \prod_{D \in \mathcal{D}(R)} \prod_{P' \in \mathcal{P}(D) \backslash \hat{\mathcal{D}}(R)} m_{P' \to D}(\boldsymbol{x}_D). \tag{4.4}$$

4.2 Region-based free energy

Recall that BP corresponds to minimization of Bethe free energy, it is interesting to note that GBP also corresponding to a free energy defined over region graphs, i.e. *region-based free energy.* For each region R in a region graph, it has its own region energy, defined as follows.

Definition 7. *Given a region R in $\mathcal{G}$ and $\boldsymbol{\theta}_R = \{\boldsymbol{\theta}_a, a \in A_R\}$, the region energy is defined to be*

$$E_R(\boldsymbol{x}_R; \boldsymbol{\theta}_R) = -\sum_{a \in A_R} \ln \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a). \tag{4.5}$$

The region-based free energy is defined accordingly.

Definition 8. *For any region graph $\mathcal{G}_R$, the region-based free energy is defined as*

$$F_R(\mathcal{B}; \boldsymbol{\theta}) = \sum_{R \in \mathcal{R}} c_R \sum_{\boldsymbol{x}_R} b_R(\boldsymbol{x}_R)(E_R(\boldsymbol{x}_R; \boldsymbol{\theta}_R) + \ln b_R(\boldsymbol{x}_R)), \tag{4.6}$$

where $b_R(\boldsymbol{x}_R)$ is the belief to region R, $\mathcal{B}$ is the set of region beliefs $\mathcal{B} = \{b_R | R \in \mathcal{R}\}$, and integer $c_R \in \mathbb{N}$ is the counting number for region R.

Region-based free energy generalizes the well known Bethe free energy. Loopy BP is equivalent to minimizing of Bethe free energy.

Recover Bethe Free Energy from Region-based Free Energy

As shown in [100], if we define two types of regions (large regions and small regions) directly from a factor graph $\mathcal{G}_F(\mathcal{V} \cup \mathcal{F}, \mathcal{E}_F)$, by defining the large regions and small regions as

$$\begin{aligned} \mathcal{R}_L &= \{a, \boldsymbol{x}_a | a \in \mathcal{F}\}, \\ \mathcal{R}_S &= \{i | i \in \mathcal{V}\}. \end{aligned} \tag{4.7}$$

For the large regions, we set counting number $c_{R,a} = 1$, and for small regions each node i, set counting number $c_{R,i} = 1 - |\mathrm{ne}_i|$. Then we can recover the Bethe free energy from region-based free energy defined in (4.6). To be specific, for large regions,

$$\begin{aligned} F_{R,L} &= \sum_{R \in \mathcal{R}_L} c_{R,a} \sum_{\boldsymbol{x}_R} b_R(\boldsymbol{x}_R)(E_R(\boldsymbol{x}_R; \boldsymbol{\theta}_R) + \ln b_R(\boldsymbol{x}_R)) \\ &= \sum_{a \in \mathcal{F}} \sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a) \ln \frac{b_a(\boldsymbol{x}_a)}{\varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a)}. \end{aligned} \tag{4.8}$$

And for the small regions, the free energy can be similarly obtained as

$$F_{R,S} = \sum_{i=1}^{N} (1 - |\mathrm{ne}_i|) \sum_{x_i} b_i(x_i) \ln b_i(x_i). \tag{4.9}$$

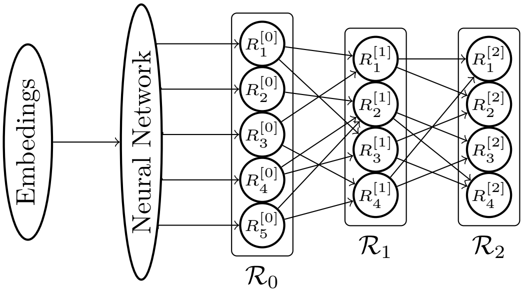


Figure 4.2: Illustration of a RENN with three levels of regions ($\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$).

Putting (4.8) and (4.9) together gives the Bethe free energy 2.12.

4.3 Region-based Energy Neural Network

In this section, we explain how the proposed region-based energy neural network (RENN) works. We are interested in finding marginal probabilities such as $p(\boldsymbol{x}_R)$. The belief $b_R(\boldsymbol{x}_R)$ can be understood as the estimation to $p(\boldsymbol{x}_R)$. Instead of directly optimize w.r.t. variables $b_R(\boldsymbol{x}_R)$ of interests, we use the reparameterization technique that are also used by recent works [1, 42, 83, 97], to model the values of interests to be the output of a neural network. We then optimize w.r.t. the parameters of the neural network.

Note that in RENN, the neural network only need to directly model beliefs on root regions $\mathcal{R}_0$. The beliefs of non-root regions $\mathcal{R}_l,\ l > 0$ could be obtained from root region beliefs according to the structure of the region graph. This can reduce the amount of neural network parameters compared to directly modeling beliefs of all regions.

Inference by RENN

For a root region $R^{[0]} \in \mathcal{R}_0$, our RENN has a corresponding vector representing its score $\boldsymbol{f}(\mathcal{G}_R, R^{[0]}; \boldsymbol{\omega}) \in \mathbb{R}^{|\boldsymbol{x}_{R^{[0]}}| \times K}$, where $\boldsymbol{\omega}$ is the parameter of mapping $\boldsymbol{f}$ that is modeled by a neural network. We define the predicted belief on the root region node $R^{[0]}$ as

$$b_{R^{[0]}}(\boldsymbol{x}_{R^{[0]}}; \boldsymbol{\omega}) = \sigma(\boldsymbol{f}(\mathcal{G}_R, R^{[0]}; \boldsymbol{\omega})), \forall\ R^{[0]} \in \mathcal{R}, \tag{4.10}$$

where $\sigma(\cdot)$ is the softmax function. The softmax function guarantees $b_{R^{[0]}} \in (0,1)^{|\boldsymbol{x}_{R^{[0]}}| \times K}$.

The representation maping $\boldsymbol{f}$ followed by the softmax function in a RENN only needs to directly output the beliefs on root regions $\mathcal{R}_0$. For the rest regions $\{R \in \mathcal{R} \backslash \mathcal{R}_0\}$ that are not root regions in the region graph, where $\backslash$ denotes the set exclusion, the RENN computes the belief as

$$b_{R^{[l]}}(\boldsymbol{x}_{R^{[l]}}; \boldsymbol{\omega}) = \frac{1}{|\mathcal{P}(R^{[l]})|} \sum_{R_p \in \mathcal{P}(R^{[l]})} \sum_{\boldsymbol{x}_{R_p} \backslash \boldsymbol{x}_{R^{[l]}}} b_{R_p}(\boldsymbol{x}_{R_p}; \boldsymbol{\omega}), \tag{4.11}$$

where $\mathcal{P}(R^{[l]})$ is the set of parent regions of $R^{[l]}$ in region graph $\mathcal{G}_R$. The non-root region belief of RENN defined in this way comes with the intuition of typical iterative belief propagation methods. In BP and its variants, messages are passed to a variable node to reduce the mismatch of beliefs w.r.t. the variable node, which are sent from this node's neighbors in a factor graph. The message passing iteration of BP or its variants stops when this kind of mismatch w.r.t. every variable node is eliminated in the factor graph.

In RENN, we directly cast the mismatch between a non-root region belief $b_{R^{[l]}}(\boldsymbol{x}_{R^{[l]}};\boldsymbol{\omega})$ and the marginalization from its parent region $\sum_{\boldsymbol{x}_{R_p}\backslash\boldsymbol{x}_R^{[l]}} b_{R_p}(\boldsymbol{x}_{R_p};\boldsymbol{\omega})$ as a penalty in the cost function. As the mismatch penalty is close to zero, the non-root region belief gets close to marginalization calculated from its parent regions. Matching a region's belief with marginalization from its parent regions' beliefs is termed as region belief consistency in region graph.

Different from GBP that minimizes region-based free energy by iterative message-passing, RENN minimizes the region-based free energy by optimizing w.r.t. the neural network parameter $\boldsymbol{\omega}$. Considering the region belief consistency, we summarize the cost function of RENN to include both the region-based free energy and mismatch penalty on non-root regions. This gives the optimization problem:

$$\min_{\boldsymbol{\omega}} F_R(\mathcal{B};\boldsymbol{\theta}) + \lambda \sum_{R\in\mathcal{R}\backslash\mathcal{R}_0} \sum_{R_p\in\mathcal{P}(R)} d(b_R, \sum_{\boldsymbol{x}_{R_p}\backslash\boldsymbol{x}_R} b_{R_p}(\boldsymbol{x}_{R_p};\boldsymbol{\omega})), \tag{4.12}$$

where $d(\cdot,\cdot)$ is distance metric or divergence to measure the mismatch between the beliefs (L_2 distance is used in our experiments), λ is the regulation parameter.

As shown in Figure 4.2, a three-level RENN takes embedding vectors as input and outputs the beliefs on $\mathcal{R}_0$ directly (embedding vectors would be explained in subsection 4.4, although not explicitly included in the objective function (4.12)). The beliefs in other levels $\{\mathcal{R}_1, \mathcal{R}_2\}$ are computed as in (4.11). Then the region-based free energy along with the penalty of region belief consistency is minimized w.r.t. $\boldsymbol{\omega}$.

Region Graph Construction for RENN

In this section we would detail how to construct the region graph $\mathcal{G}_R$ for RENN. Informally, a region graph can be generated by firstly clustering the nodes in a factor graph in any way and then connect the clusters with directed edges. But it does not mean we can rely on an arbitrary region graph to do our inference. Conditions such as *valid* region graph (would be discussed in the following subsection), and *maxent-normality* [94, 100] have been proposed for region graphs. But these conditions do not gives rules of how to construct "good" region graphs. Our idea for this issue is to combine the cluster variation method [40, 68] with *tree-robust* condition [22] (that was originally developed to improve accuracy of GBP) for practical region graph construction.

Determining the counting numbers

In Definition 8, region-based free energy is a function of counting numbers $\{c_R\}$. The counting numbers here are used to balance each region's contribution to the free energy. According to [100], the region-base free energy is valid if the following 1-balanced conditions holds

$$\sum_{R \in \mathcal{R}} c_R \delta_R(i) = 1, \forall \text{ node } i \text{ in } \mathcal{G}_F, \tag{4.13}$$

where $\delta_R(i)$ is the indicator function, equal to 1 if and only if node i defined in factor graph $\mathcal{G}_F$ is in region R of region graph $\mathcal{G}_R$ and equal to 0 otherwise. Note that node i can be either a variable or factor node here. It can be seen that each node would be counted exactly once if the condition (4.13) holds.

Given a region graph $\mathcal{G}_R$, the counting numbers $\{c_R\}$ can be constructed recursively as:

$$c_R = 1 - \sum_{R_i \in \mathcal{A}(R)} c_{R_i}, \forall R, \tag{4.14}$$

where $\mathcal{A}(R)$ denotes the ancestor set of region node R in $\mathcal{G}_R$. This rule means the counting numbers of root regions are always 1, since they do not have any ancestors.

Generating graph by Cluster Variation Method

Cluster variation method was in introduced by Kikuchi and other physcists [40,68], which started with the intuition of approximating free energy by using larger set of variable nodes instead of the single-node factorization in mean field approximation.

Cluster variation method starts with the root regions $\mathcal{R}_0$. There are two requirements for $\mathcal{R}_0$: i) every variable node i of factor graph $\mathcal{G}_F$ is included in at least one region $R^{[0]} \in \mathcal{R}_0$; ii) there should be no region $R^{[0]} \in \mathcal{R}_0$ being a subregion of any other region in $\mathcal{R}_0$. With $\mathcal{R}_0$ ready, the other sets of regions are generated hierarchically. To construct level-1 regions $\mathcal{R}_1$ from $\mathcal{R}_0$, we find out all the intersections between regions in $\mathcal{R}_0$, but omit any that are subregion of other intersection regions. Then level-2 regions $\mathcal{R}_2$ can be similarly constructed from $\mathcal{R}_1$. Assume there are L such sets, then $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_{L-1}$. The construction rule can be formulated as

$$\begin{aligned} \mathcal{R}_l = \{ & R_i^{[l]} = R_j^{[l-1]} \cap R_k^{[l-1]} | R_i^{[l]} \not\subset R_n^{[l]}, \ \forall i \neq n, \\ & R_j^{[l-1]}, R_k^{[l-1]} \in \mathcal{R}_{l-1}, j \neq k\}, l = 1, 2, \cdots, L-1. \end{aligned} \tag{4.15}$$

With the hierarchical region sets built, we need to draw the edges. The directed edges are always connected from regions in $\mathcal{R}_{l-1}$ to these in $\mathcal{R}_l$. For one region $R^{[l]}$ in $\mathcal{R}_l$, a directed edge is drawn from any superregion of $R^{[l]}$ in $\mathcal{R}_l$. This can be represented as

$$\mathcal{E} = \{e = (R^{[l-1]}, R^{[l]}) | R^{[l]} \subset R^{[l-1]}, R^{[l]} \in \mathcal{R}_l, R^{[l-1]} \in \mathcal{R}_{l-1}, \forall l\}. \tag{4.16}$$

Root Region Construction

In section 4.3, we detailed the region graph construction steps by cluster variation method, starting from $\mathcal{R}_0$. In this section, we explain how to build the root region set $\mathcal{R}_0$. This is important since it totally decides from which we start building region graph. We take the path of [22, 94] for this issue.

Specifically, we use the *tree-robust* condition [22] to build the root regions for our RENN. These root regions are then used to grow the other hierarchical levels of region graphs by cluster variation method in subsection 4.3. Tree-robust condition was developed original for GBP to gain better approximations. GBP has better accuracy on tree-robust graphs than on non-tree-robust graphs. We would show in our experiment section that RENN outperforms GBP even in tree-robust graphs.

To explain the concept of *tree-robust*, we need explain the concepts of *cycle basis* and *tree exact*, based on which the tree-robust is defined.

Definition 9. *A cycle basis of the cycle space of a graph $\mathcal{G}$ is a set of simple cycles $\mathcal{CB} = \{C_1, C_2, \cdots, C_\mu\}$ such that for every cycle C in graph $\mathcal{G}$, there exists a unique subset $\mathcal{CB}_C \subseteq \mathcal{CB}$ such that the set of edges appearing an odd number of times in $\mathcal{CB}_C$ comprise the cycle C.*

Definition 10. *Let T be a spanning tree of graph $\mathcal{G}$. A cycle basis $\mathcal{CB}$ is tree exact w.r.t. T if there exists an ordering π of the cycles in $\mathcal{CB}$ such that*

$$\left\{C_{\pi(i)} \backslash C_{\pi(1)} \cup C_{\pi(2)} \cup \cdots \cup C_{\pi(i-1)}\right\} \neq \emptyset \quad \text{for} \quad i = 2, \cdots, \mu.$$

Definition 10 tells us that if a cycle basis is tree exact w.r.t. T and ordered properly, there is at least one edge of C_π that has not appeared in any cycles preceding it, and meanwhile this edge does not appear in the spanning tree T. With the above concepts, we are ready to give the definition of tree-robust.

Definition 11. *A cycle basis $\mathcal{CB}$ is tree-robust if it is tree exact w.r.t. all spanning trees of $\mathcal{G}$.*

We use two important theorems from [22] for choosing cycle basis in two specific graph cases, i.e. planar graphs and complete graphs. A planar graph is a graph that can be embedded in the two-dimensional plain (it can be drawn on the plane in such a way that its edges intersect only at their nodes). In a complete graph, every pair of distinct nodes is connected by an unique edge.

Theorem 2. *In a planar graph $\mathcal{G}$, the cycle basis comprised of the faces of graph $\mathcal{G}$ is tree-robust.*

Theorem 3. *In a complete graph $\mathcal{G}$, construct a cycle basis as follows. Choose a node i as the root. Create a 'star' spanning tree rooted at i. Then construct cycles of form (i, j, k) from each off-tree edge (j, k). The constructed basis is tree-robust.*

Algorithm 1 Construct Root Regions from General Graphs.

```
Input: Pairwise Markov random field $p(\boldsymbol{x})$
Draw the factor graph $\mathcal{G}_F$ of $p(\boldsymbol{x})$
Obtain graph $\mathcal{G}$ by preserving the variable nodes as they are and converting the factor nodes of $\mathcal{G}_F$ into edges
Find the subgraph $\mathcal{G}_s$ of $\mathcal{G}$, such that $\mathcal{G}_s$ is planar or complete graph
Add the tree-robust basis $\mathcal{CB}(\mathcal{G}_s)$ of $\mathcal{G}_s$ into $\mathcal{R}_0$
Marked all nodes as visited and edged as used in $\mathcal{G}_s$
repeat
  Choose an unused edge $e = (s, t)$ from a visited node $s$
  if $t$ is visited then
    Set path1 $= e$
    Find the shortest path path$_2$ from $s$ to $t$ via used edges
  else
    Find path from $s$ to a visited $u$ that contains edge $e$, this path is set as path1.
    Find the shortest path path$_2$ from $s$ to $u$ via used edges
  end if
  Add cycle $C$ consisting of path1 and path2 to $\mathcal{R}_0$.
  Mark all nodes as visited and edges as used in $C$
until $\nexists$ unused edge $e = (s, t)$ from a visited node $s$
```

Tree-robust root regions can also be constructed for general graphs, which can be seen as an extension from Theorem 2 and 3.

Root regions of region graph $\mathcal{G}_R$ from planar and compete graphs are explained in section 4.3. For general graph case, it basically is to find a subgraph that is planar or compote graph, and then extract the corresponding tree-robust basis, after which extra cycles are added in by following Algorithm 1.

4.4 Experimental Results

We conducted a series of experiments to validate the proposed RENN model. The experiments are designed to verify RENN in inference problems. Analysis and experiments on MRF learning would be reserved to explain in Part learning.

Experiment Setting and Evaluation Metrics

Without loss of generality, our experiments are carried out on binary pairwise MRF (Ising model). This gives us $p(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp\left(\sum_{(i,j)\in\mathcal{E}_F} J_{ij}x_ix_j + \sum_{i\in\mathcal{V}} h_ix_i\right)$, $\boldsymbol{x} \in \{-1,1\}^N$, where J_{ij} is the pairwise log-potential between node i and j, h_i is the node log-potential for node i. Then $\boldsymbol{\theta} = \{J_{ij}, h_i | (i,j) \in \mathcal{E}_F, i \in \mathcal{V}\}$. J_{ij} is always sampled from standard normal distribution, i.e. $J_{ij} \sim \mathcal{N}(0,1)$, meanwhile $h_i \sim \mathcal{N}(0,\gamma^2)$.

Table 4.1: Inference on grid graph ($\gamma = 0.1$). ℓ_1 error and correlation ρ between true and approximate marginals, and $\log Z$ error.

Metric	n	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
ℓ_1 error	25	0.271 ± 0.051	0.086 ± 0.078	0.084 ± 0.076	0.057 ± 0.024	0.111 ± 0.072	**0.049** ± 0.078
	100	0.283 ± 0.024	0.085 ± 0.041	0.062 ± 0.024	0.064 ± 0.019	0.074 ± 0.034	**0.025** ± 0.011
	225	0.284 ± 0.019	0.100 ± 0.025	0.076 ± 0.025	0.073 ± 0.013	0.073 ± 0.012	**0.046** ± 0.011
	400	0.279 ± 0.014	0.110 ± 0.016	0.090 ± 0.016	0.079 ± 0.009	0.083 ± 0.009	**0.061** ± 0.009
Correlation ρ	25	0.633 ± 0.197	0.903 ± 0.114	0.905 ± 0.113	0.923 ± 0.045	0.866± 0.117	**0.951** ± 0.112
	100	0.582 ± 0.112	0.827 ± 0.134	0.902 ± 0.059	0.899 ± 0.043	0.903± 0.049	**0.983** ± 0.012
	225	0.580 ± 0.080	0.801 ± 0.078	0.863 ± 0.088	0.869 ± 0.037	0.873 ± 0.037	**0.949** ± 0.022
	400	0.596 ± 0.054	0.779 ± 0.059	0.822 ± 0.047	0.852 ± 0.024	0.841 ± 0.028	**0.912** ± 0.025
$\log Z$ error	25	2.512 ± 1.060	0.549 ± 0.373	0.557 ± 0.369	**0.169** ± 0.142	0.762 ± 0.439	0.240 ± 0.140
	100	13.09 ± 2.156	1.650 ± 1.414	1.457 ± 1.365	**0.524** ± 0.313	2.836 ± 2.158	1.899 ± 0.495
	225	29.93 ± 4.679	3.348 ± 1.954	3.423 ± 2.157	**1.008** ± 0.653	3.249 ± 2.058	4.344 ± 0.813
	400	51.81 ± 4.706	5.738 ± 2.107	5.873± 2.211	**1.750** ± 0.869	3.953 ± 2.558	7.598 ± 1.146

Table 4.2: Inference on grid Graph. ($\gamma = 1$)

Metric	n	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
L1	25	0.131 ± 0.080	**0.022** ± 0.017	0.022 ± 0.018	0.137 ± 0.026	0.043 ± 0.017	0.027 ± 0.014
	100	0.130 ± 0.041	0.025 ± 0.014	0.025 ± 0.014	0.146 ± 0.020	0.046 ± 0.009	**0.017** ± 0.002
	225	0.135 ± 0.024	0.024 ± 0.010	0.023 ± 0.009	0.154 ± 0.012	0.052 ± 0.010	**0.017** ± 0.003
	400	0.131 ± 0.020	0.020 ± 0.003	0.020 ± 0.003	0.158 ± 0.007	0.052 ± 0.007	**0.017** ± 0.001
Correlation	25	0.849 ± 0.159	**0.992** ± 0.011	0.991 ± 0.012	0.798 ± 0.088	0.980 ± 0.015	0.988 ± 0.025
	100	0.841 ± 0.087	0.988 ± 0.013	0.988 ± 0.012	0.788 ± 0.051	0.976 ± 0.013	**0.997** ±0.001
	225	0.824 ± 0.057	0.989 ± 0.010	0.990 ± 0.010	0.764 ± 0.022	0.966 ± 0.016	**0.996** ± 0.001
	400	0.828 ± 0.043	0.993 ± 0.002	0.993 ± 0.002	0.759 ± 0.018	0.967 ± 0.013	**0.997** ± 0.001
$\log Z$ error	25	2.113 ± 1.367	**0.170** ± 0.199	0.194 ± 0.188	0.605 ± 0.611	2.214 ± 0.775	0.649 ± 0.363
	100	8.034 ± 2.523	**0.372** ± 0.427	0.415 ± 0.422	1.545 ± 1.081	11.14 ± 0.954	3.129 ± 0.520
	225	17.923 ± 3.474	0.952 ± 1.037	**0.917** ± 0.922	3.143 ± 2.122	25.55 ± 2.025	7.473 ± 0.906
	400	31.74 ± 4.766	**0.919** ± 0.684	1.011 ± 0.685	3.313 ± 1.872	46.61 ± 3.094	12.77 ± 0.991

Table 4.3: Inference with the *infinite face* on grid, $n = 25$.

γ	Metric	GBP	RENN
0.1	ℓ_1 Error	0.061 ± 0.025	**0.025** ± 0.020
	ρ	0.913 ± 0.049	**0.984** ± 0.021
	$\log Z$ Error	3.564 ± 2.823	0.384 ± 0.223
1	ℓ_1 Error	0.145 ± 0.028	**0.016** ± 0.010
	ρ	0.783 ± 0.091	**0.995** ± 0.010
	$\log Z$ Error	0.825 ± 0.841	0.364 ± 0.201

In the inference experiments, we are interested into how well beliefs from RENN approximate true marginal distributions of $p(\boldsymbol{x};\boldsymbol{\theta})$. We quantity this by both the ℓ_1 error (ℓ_1-norm distance) and Pearson correlation coefficient ρ, between true marginals and beliefs of RENN. The marginal evaluations include both $p(x_i)$ and $p(x_i, x_j)$. In addition, we also quantify the $\log Z$ error as the absolute difference between true partition function and estimated value.

Apart from the inference experiments, we also carried out the modeling learning experiments. We use the negative log-likelihood (NLL) to evaluate how well the model is learned, which is then compared with NLL of true model.

In all experiments, for each evaluation of RENN, mean field, (loopy) BP [66], damped BP [76] with damping factor 0.5, and GBP [100] as benchmarks, are all evaluated on the same MRF, which are then compared with RENN. Additionally, a recent neural network benchmark model, saddle-point Inference Net [97] targeting at Bethe free energy, is also used as comparison. To make the comparison with Inference Net fair, RENN and Inference Net use the same neural network structures and hidden dimension. Each variable x_i is associated with a learnable embedding vector $\boldsymbol{e}_i$. A transform layer [89] consumes $\boldsymbol{e}_i$ and output a hidden representation $\boldsymbol{h}_i$. The transform layer is shared by all embeddings. Then an affine layer followed by softmax consumes $[\boldsymbol{h}_1, \cdots, \boldsymbol{h}_N]$ and outputs beliefs.

Inference on Grid Graphs

We first evaluate how well RENN can estimate the marginal distributions, compared with benchmark algorithms/models, w.r.t. marginal ℓ_1 errors and Pearson correlation ρ, at different graph size n and standard deviation γ of $\{h_i\}$. At each evaluation for a given size n, 20 MRFs are generated by sampling $\{J_{ij}\}$ and $\{h_i\}$. Then RENN and other candidate algorithms perform inference on these MRFs. Then the ℓ_1 error and correlation ρ between true and estimated marginal distributions are evaluated. The $\log Z$ error is also recorded. Experiments are carried out in large and small standard deviation of $\{h_i\}$ ($\gamma = 0.1$, $\gamma = 1$), which reflects the relative strength of standalone node log-potentials to pairwise log-potentials. The results are reported as 'mean ± standard deviation'.

Table 4.4: Inference on complete graph of size 9.

Metric	γ	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
ℓ_1 error	0.1	0.294 ± 0.061	0.120 ± 0.038	0.118 ± 0.034	0.237 ± 0.061	**0.109** ± 0.025	0.130 ± 0.085
	1	0.233 ± 0.133	0.200 ± 0.098	0.201 ± 0.098	0.246 ± 0.135	0.196 ± 0.061	**0.137** ± 0.117
	2	0.187 ± 0.131	0.176 ± 0.114	0.177 ± 0.113	0.247 ± 0.117	0.182 ± 0.084	**0.067** ± 0.045
	3	0.155 ± 0.120	0.145 ± 0.112	0.146 ± 0.112	0.204 ± 0.107	0.152 ± 0.079	**0.060** ± 0.038
	4	0.124 ± 0.115	0.120 ± 0.103	0.121 ± 0.102	0.194 ± 0.076	0.129 ± 0.071	**0.051** ± 0.050
Correlation ρ	0.1	0.262 ± 0.177	0.695 ± 0.104	0.698 ± 0.099	0.446 ± 0.196	0.720 ± 0.065	**0.741** ± 0.220
	1	0.465 ± 0.349	0.538 ± 0.292	0.538 ± 0.292	0.461 ± 0.331	0.639 ± 0.159	**0.769** ± 0.313
	2	0.587 ± 0.300	0.619 ± 0.284	0.619 ± 0.282	0.457 ± 0.257	0.645 ± 0.175	**0.929** ± 0.118
	3	0.657 ± 0.289	0.697 ± 0.267	0.697 ± 0.265	0.582 ± 0.218	0.697 ± 0.162	**0.936** ± 0.076
	4	0.758 ± 0.257	0.778 ± 0.221	0.776 ± 0.221	0.597 ± 0.177	0.753 ± 0.178	**0.941** ± 0.099
$\log Z$ error	0.1	8.402 ± 4.369	34.61 ± 2.439	34.74 ± 2.195	**1.763** ± 1.176	35.46 ± 1.651	3.171 ± 1.259
	1	6.473 ± 3.737	45.91 ± 6.888	45.96 ± 6.927	**1.826** ± 2.024	51.87 ± 6.150	2.796 ± 1.194
	2	5.830 ± 2.979	75.35 ± 14.58	75.46 ± 14.57	3.080 ± 2.958	81.23 ± 12.939	**2.577** ± 1.845
	3	4.401 ± 2.522	111.0 ± 22.20	111.1 ± 22.17	3.205 ± 3.720	116.1 ± 19.76	**2.645** ± 1.507
	4	3.037 ± 2.122	142.9 ± 25.58	143.1 ± 25.56	5.167 ± 5.249	147.2 ± 23.38	**1.820** ± 1.306

Table 4.5: Inference on complete graph of size 16.

Metric	γ	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
ℓ_1-error	0.1	0.303 ± 0.056	0.176 ± 0.039	0.174 ± 0.038	0.244 ± 0.047	0.174 ± 0.044	**0.169** ± 0.052
	1	0.273 ± 0.086	0.239 ± 0.059	0.239 ± 0.059	0.260 ± 0.086	0.249 ± 0.067	**0.181** ± 0.092
	2	0.231 ± 0.079	0.222 ± 0.064	0.221 ± 0.064	0.249 ± 0.078	0.232 ± 0.069	**0.170** ± 0.109
	3	0.218 ± 0.042	0.204 ± 0.038	0.204 ± 0.038	0.247 ± 0.065	0.213 ± 0.051	**0.138** ± 0.106
	4	0.197 ±0.049	0.181 ± 0.035	0.180 ± 0.034	0.210 ± 0.070	0.174 ± 0.030	**0.125** ± 0.050
Correlation ρ	0.1	0.231 ± 0.196	0.509 ± 0.056	0.510 ± 0.055	0.316 ± 0.207	0.506 ± 0.063	**0.539** ± 0.235
	1	0.381 ± 0.255	0.514 ± 0.185	0.515 ± 0.185	0.445 ± 0.223	0.533 ± 0.150	**0.756** ± 0.187
	2	0.535 ± 0.207	0.569 ± 0.180	0.570 ± 0.179	0.480 ± 0.186	0.559 ± 0.176	**0.750** ± 0.261
	3	0.586 ± 0.142	0.618 ± 0.134	0.619 ± 0.134	0.502 ± 0.144	0.613 ± 0.128	**0.853** ± 0.159
	4	0.622 ± 0.166	0.658 ± 0.133	0.660 ± 0.132	0.564 ± 0.165	0.693 ± 0.060	**0.868** ± 0.053
$\log Z$ error	0.1	24.45 ± 7.560	143.7 ± 9.297	145.5 ± 6.096	166.3 ± 11.98	148.5 ± 3.522	**12.57** ± 3.689
	1	20.66 ± 5.451	178.7 ± 22.18	178.9 ± 21.88	153.3 ± 25.29	213.6 ± 12.75	**14.41** ± 4.135
	2	16.04 ± 4.352	296.3 ± 44.41	296.9 ± 44.24	116.9 ± 32.72	335.1 ± 32.86	**13.37** ± 4.531
	3	13.87 ± 6.554	432.7 ± 66.44	433.4 ± 66.30	100.2 ± 39.62	462.9 ± 53.61	**12.56** ± 6.046
	4	**10.74** ± 7.385	565.7 ± 73.33	566.1 ± 73.13	106.0 ± 54.43	588.3 ± 62.58	14.72 ± 4.155

Marginal approximation can be reflected by ℓ_1 error and correlation coefficient ρ. The results are reported in Table 4.1 and 4.2. In all cases except one, beliefs of RENN outperform benchmark algorithms with large marginals. As expected, performances of loopy BP and its variant damped BP are similar in general while damped BP sometime gets better estimations. Both loopy BP and damped BP have better marginal estimations than mean field method in all of our considered scenarios. GBP outperforms loopy BP and damped BP at case $\gamma = 0.1$, $h_i \sim \mathcal{N}(0, 0.1^2)$, agreeing with the results at [100], but performs poorly at case of $\gamma = 1$. Similar phenomena happen to Inference Net, which has better estimations than loopy BP and damped BP in some cases of $\gamma = 0.1$ but falls behind in all cases of $\gamma = 1$.

As for the error of partition function values, GBP gets the most accurate estimations when $\gamma = 0.1$. $\log Z$ estimated by loopy BP and damped BP is better for $\gamma = 1$. Partition function estimation by RENN is acceptable at different considered cases.

Note the region graphs in this set of experiments uses all faces of grid graph but the *infinite face* (the perimeter circle). For instance, the region $\{1, 2, 3, 4, 5, 6, A, B, C, E, F, G\}$ is obtained from the infinite face in the 2-by-3 grid in Figure 4.1. By comparing Table 4.3 with the $n = 25$ cases of Table 4.1 and 4.2, performance of RENN can be further better when we include the infinite face in building region graphs from grid. On the contrary, performance of GBP drops slightly after including the infinite face. But number of nodes in the region built from the infinite face would scale with the perimeter of grid graph. Since RENN already has reasonable good accuracy outperforming benchmark methods as shown in Table 4.1 and 4.2, we suggest to drop the infinite face in constructing region graphs from grids.

Inference on Challenging Complete Graphs

In this subsection, we evaluate RENN in comparison with benchmark methods on more challenging graphs, i.e. complete graphs in which every two nodes are connected by an unique edge. Due the high complexity, we carry out the inference experiments on complete graphs of size $n = 9$ and $n = 16$ but with richer setting of γ, to be able the track the true marginals and partition functions exactly, which are used to evaluate candidate methods.

For marginal distribution estimations, RENN still outperforms all other benchmark methods except for one case at $\gamma = 0.1$ in size-9 graph, as shown in Table 4.4 and 4.5. In the case of $\gamma = 0.1$ in Table 4.4, Inference Net outperforms RENN w.r.t. ℓ_1 error, i.e. 0.109 versus 0.130, but falls behind RENN w.r.t. correlation ρ (0.720 versus 0.741) and $\log Z$ estimation significantly (35.46 versus 3.171).

In the complete graphs, GBP does not have privilege over loopy BP and damped BP any more, RENN operating on the same region graphs as those for GBP, gives consistently better marginal distribution estimations. Also, generally speaking, the performance of Inference Net is close to loopy BP and damped in most cases of complete graphs.

As for partition function evaluations of complete graphs, the results are quite different from those of grid graphs, by observing Table 4.4 and 4.5. Loopy BP, damped BP, and Inference Net are getting very large error of partition function as node log-potentials are more different from each other, i.e. γ gets larger. GBP has reasonable good estimation of $\log Z$ in smaller sized complete graph in Table 4.4, but gets large $\log Z$ error in a bit larger complete graph as in Table 4.5. Mean field methods gives much better estimation of $\log Z$ in compete graphs than loopy BP, damped BP and Inference Net, but it has poorer marginal distribution estimations.

4.5 Discussion

In this chapter, we present an alternative way to do inference in MRFs. Difference from iteratively propagating messages as classical message-passing algorithms, we directly minimize the region-based free energy defined over a region graph. We, therefore, are able to performance inference by solving the energy minimization problem. The region-based free energy is a function of variational distribution b. We amortize the distribution b by a neural network in solving the energy minimization problem.

A region in a region graph usually contains more than one variable, allowing the region's parental regions to enforce marginal distribution over multiple variable. This is in contrast to propagating belief of a single variable in mean field and Bethe approximations (loopy BP, damped BP and Inference Net). This advantage allows the performance gain with RENN, which we observed in experiments. Nevertheless, RENN need to handle the region graph construction. In construction of region graphs, we select the tree-robust condition that is based on cycle basis for root regions. Loops in graphs are known to be difficulty for classical message-passing methods. The construction principle of RENN gives it the chance to relieved the difficulty due conflicting potentials in a loop (such a loop may be clustered into a region).

The other aspect is the feasible space of variational distribution b. For a general graphical model, we can rarely formulate the true feasible space. Therefore the belief consistency, also named as marginalization constraints, is used to formulate a relaxed feasible space. The belief consistency is enforced via the fixed-point update rules in message-passing methods. In RENN, this belief consistency is enforced via: i) via the average in definition of a child region's belief from the marginalization of its parental regions's beliefs; ii) the penalty cost of inconsistency in objective function.

In the next Part, we would discuss the topic of learning MRF via the inference methods referred to in this section.

4.6 Literature

The well-known standard belief propagation (BP) algorithm [46, 75] has been popularly used in exact inference problems of tree-structures graphs and approximate inference in general graphs (loopy BP), which was explained by the Bethe free energy minimization later on [101]. The attempts in improving the approximate inference with BP and gaining better insight has been made, where the representative works are fractional BP [96], tree-reweighted BP [90], generalized BP (GBP, , also known as parent-to-child algorithm) [100, 102], etc. GBP propagates messages between regions (clusters of nodes), is generally more accurate than loopy BP but is more complex than BP. Fix points of GBP that operate on regions graphs, correspond to stationary points of region-based free energy of the region graphs. Depending on the graph size and potential functions, the iterative message-passing algorithms can take long time to converge (if they can converge) before returning inference results. Inference results of these message-passing methods can degenerate significantly in dense graphs (graphs with too many circles).

Neural network based inference methods are getting popularly in recent years, where they have success in areas of deep graphical generative models for structured data modeling [38, 53, 77]. Along with a neural network based generative model, a separate recognitive neural network has to be trained for inference. In these directed graphical models built on neural networks, training of inference networks need sampling which brings in the trade-off between training speed and estimation variance. These issues also lies in the one of most successful model, variational autoencoders [42, 60, 85], and other variational methods [48, 87].

Apart from the directed graphical models, there is also a track of work on using neural network to model the message passing functions. [1] models the intractable message update functions by a Gaussian dissertations with its parameters as output of a neural network, and then follows the typical message passing rules to do iterative message updates of standard BP. [31, 37] also similarly learn a neural network to model the message update functions of expectation propagation methods.

In another track, message-passing functions are implemented neural networks with parameters that need to be trained. In fact, these methods still do iterative message propagation analogous to standard BP. Neural message passing methods [25, 103] use a graph network update messages and a separate network to map messages into targeted results. Training of these models need true marginal distributions which might be an issue for cases where it is too expensive or prohibitive to collect the ture marginals.

Closely related to in this section, Bethe free energy is directed minimized in [95, 97] instead of iterative message passing methods, which can be treated as special cases of RENN. [95] uses gradient decent method to alternative minimize single variable marginals. [97] parameterizes marginals directly by a neural network and minimizes Bethe free energy.

Part II

Learning

Chapter 5

Learning with inference

In Part I, we have discussed different ways for inference, assuming a probabilistic graphical model is given. As introduced in chapter 2, learning graphical models includes structure learning and parameter learning. We restrict our discussion to the parameter learning of graphical models. From this chapter onward, we would focus on answering the question of how to decide the parameters of a probabilistic graphical model. In this chapter, we mainly discuss the learning of undirected graphical models and leave the learning of directed graphical models for the coming chapters.

As explained in section 2.5, the most essential learning principle is *maximal likelihood* that is derived from the minimization of KL-divergence. Since we do not have access to the true distribution $p^*(\boldsymbol{x})$, practical maximal likelihood learning is via tuning parameter $\boldsymbol{\theta}$ of model $p(\boldsymbol{x};\boldsymbol{\theta})$ using information of samples of $p^*(\boldsymbol{x})$, i.e. $\mathcal{D} = \left\{\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^M\right\}$.

We would begin with the explanation of why inference is required in learning, after which MRF learning via RENN is explained. Numerical comparisons with classic message-passing methods w.r.t. learning is demonstrated afterwords. Topics on learning with hidden variables are then discussed.

5.1 Why does learning MRF requires inference?

Let us continue the discussion on learning in section 2.5. Given a sample $\boldsymbol{x}$, the log-likelihood of this evidence is

$$l(\boldsymbol{x};\boldsymbol{\theta}) = \log \tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}). \tag{5.1}$$

where $\tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) = \prod_{a\in\mathcal{F}} \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a)$. Without loss of generality, computing the gradient w.t.r. $\boldsymbol{\theta}_a$ gives

$$\frac{\partial l(\boldsymbol{x};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_a} = \frac{\partial \log \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a} - \mathbb{E}_{p(\boldsymbol{x}_a;\boldsymbol{\theta})}\left[\frac{\partial \log \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a}\right]. \tag{5.2}$$

Applying all sample from dataset $\mathcal{D}$, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} l(\boldsymbol{x}; \boldsymbol{\theta}), \tag{5.3}$$

we have

$$\frac{\partial \mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_a} = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \frac{\partial \log \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a} - \mathbb{E}_{p(\boldsymbol{x}_a; \boldsymbol{\theta})} \left[\frac{\partial \log \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a} \right]. \tag{5.4}$$

In general, there is no closed-form solution in maximizing this log-likelihood $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta})$. But it is intuitive to observe that the stationary point of $\mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta})$ is

$$\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{\partial \log \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a} = \mathbb{E}_{p(\boldsymbol{x}_a; \boldsymbol{\theta})} \left[\frac{\partial \log \varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta}_a)}{\partial \boldsymbol{\theta}_a} \right]. \tag{5.5}$$

The left-hand-side of (5.5) is empirical expectation w.r.t. to the gradient of potential function φ_a, while on the right-hand-side of (5.5) the expectation is computed by support of the marginal distribution $p(\boldsymbol{x}_a; \boldsymbol{\theta})$. This stationary point is intuitively telling us that the maximum likelihood estimation is trying to enforce the equality of empirical expectation of gradient with each $\boldsymbol{\theta}_a$ and the model's expectation of that.

Remark 7. *If we instantiate $p(\boldsymbol{x}; \boldsymbol{\theta})$ from exponential family and it is canonically parameterized, its potential $\varphi_a(\boldsymbol{x}_a; \boldsymbol{\theta})$ is log-linear w.r.t. the sufficient statistic. The corresponding equation of* (5.5) *in this case is reduced into moment matching.*

Apart from the classical techniques such as iterative proportional fitting [90, section 6.1] [98], the wide used approach is gradient decent method by using (5.4), in maximizing $\mathcal{L}(\mathcal{D}; \boldsymbol{\theta})$. Either way, the inference on marginal distribution $\{p(\boldsymbol{x}_a; \boldsymbol{\theta}), a \in \mathcal{F}\}$ is inevitable.

Take the gradient decent method as example, we need to use one of the inference approaches introduced in Part I to approximate the marginals $\{p(\boldsymbol{x}_a; \boldsymbol{\theta}), a \in \mathcal{F}\}$, then do update of the parameter

$$\boldsymbol{\theta}_a \leftarrow \boldsymbol{\theta}_a + r \cdot \frac{\partial \mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_a}, \forall \;\; a \;\in\; \mathcal{F}. \tag{5.6}$$

Remark 8. *When the MRF scales up, the number of parameters, i.e. elements of $\boldsymbol{\theta}$, could be large. To reduce overfitting, a typical trick is to add a zero-mean Gaussian prior to $\boldsymbol{\theta}$ as regularization. Then log-likelihood of evidence $\boldsymbol{x}$ becomes*

$$l(\boldsymbol{x}; \boldsymbol{\theta}) = \log \tilde{p}(\boldsymbol{x}; \boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}) - \frac{\|\boldsymbol{\theta}\|^2}{2\sigma^2}, \tag{5.7}$$

where σ^2 is variance of the Gaussian prior, which is free to determine. The rest analysis fellows.

Table 5.1: NLL of grid graphical models training using different inference methods.

n	True	Exact	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
25	9.000	9.004	9.811	9.139	9.196	10.56	9.252	**9.048**
100	19.34	19.38	23.48	19.92	20.02	28.61	20.29	**19.76**
225	63.90	63.97	69.01	66.44	66.25	92.62	68.15	**64.79**

5.2 Model Learning with Inference of RENN

In section 4.3, we explained how to do inference with RENN when parameter $\boldsymbol{\theta}$ of $p(\boldsymbol{x}; \boldsymbol{\theta})$ is assumed to be known. As the continuation, we consider the case of learning parameter $\boldsymbol{\theta}$ of $p(\boldsymbol{x}; \boldsymbol{\theta})$ with inference by RENN here.

The likelihood minimization of (5.1) can be written as minimization of the negative log-likelihood

$$\min_{\boldsymbol{\theta}} -\log \tilde{p}(\boldsymbol{x}; \boldsymbol{\theta}) + \log Z(\boldsymbol{\theta}). \tag{5.8}$$

Instead of explicitly formulating gradients as in section 5.1, we can make use of the autodiff methods in PyTorch or Tensorflow and minimize the above cost directly. Note this does not mean that inference is not required. Instead, we need to deal with the intractability of $\log Z(\boldsymbol{\theta})$, which is expensive or prohibitive to solve (5.8) directly. Approximation to $\log Z(\boldsymbol{\theta})$ requires inference, which can be done by RENN.

The region-based free energy $F_R(\mathcal{B}; \boldsymbol{\theta})$ would exactly be negative partition function of $p(\boldsymbol{x}; \boldsymbol{\theta})$, i.e. $-\log Z(\boldsymbol{\theta})$, if each belief is exactly the corresponding marginalization, $b_R(\boldsymbol{x}_R) = p(\boldsymbol{x}_R), \forall\, R \in \mathcal{R}$. Otherwise, $F_R(\mathcal{B}^*; \boldsymbol{\theta})$ can always be an approximation of $-\log Z(\boldsymbol{\theta})$, where $\mathcal{B}^* = \{b_R(\boldsymbol{x}_R; \boldsymbol{\omega}^*), R \in \mathcal{R}\}$ with $\boldsymbol{\omega}^*$ being the solution to problem (4.12).

Combining the model learning and inference, we have

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\omega}} -\log \tilde{p}(\boldsymbol{x}; \boldsymbol{\theta}) - F_R(\mathcal{B}; \boldsymbol{\theta}) - \lambda \sum_{R \in \mathcal{R} \setminus \mathcal{R}_0} \sum_{R_p \in \mathcal{P}(R)} d(b_R, \sum_{\mathcal{S}(R_p) \setminus \mathcal{S}(R)} b_{R_p}(\boldsymbol{x}_{R_p}; \boldsymbol{\omega})). \tag{5.9}$$

Then the difficulty of computing $Z(\boldsymbol{\theta})$ is dealt with by joint learning of MRF and inference by RENN in (5.9).

Learning MRF with RENN doesn't need iterative message propagation. Additionally, RENN can be implemented with modern toolboxes such as PyTorch or TensorFlow that enjoys the GPU computation capacity. Consequently, learning MRF with RENN can be much faster, which would be shown in the following section.

5.3 Numerical Comparisons in MRF Learning

To make the comparison more concrete, we do the learning of MRFs, i.e. learning the model parameter $\boldsymbol{\theta}$, via different inference methods discussed.

Table 5.2: NLL of complete graphical models training using different inference methods.

n	True	Exact	Mean Field	Loopy BP	Damped BP	GBP	Inference Net	RENN
9	3.276	3.286	9.558	5.201	5.880	10.06	5.262	**3.414**
16	4.883	4.934	28.74	13.64	18.95	24.45	13.77	**5.178**

Table 5.3: Average consumed time per epoch (unit: second) for two training cases in Table 5.1 and 5.2.

	Grid Graph	Complete Graph
	$n = 225$	$n = 16$
Mean Field	40.09	2.499
Loopy BP	335.1	12.40
Damped BP	525.1	5.431
GBP	12.37	1.387
Inference Net	19.49	0.882
RENN	16.03	2.262

We do learning on two types of MRF graphs, grid (Table 5.1) and complete graphs (Table 5.2). For both cases, we firstly sample the parameter set $\boldsymbol{\theta}'$, then sample training and testing dataset from $p(\boldsymbol{x}; \boldsymbol{\theta}')$. The true NLL of sampled datasets can be computed by $p(\boldsymbol{x}; \boldsymbol{\theta}')$. We then train a randomly-initialized model with the obtained training dataset by using RENN (section 5.2). The trained model by RENN is evaluated with testing dataset w.r.t. NLL value, which is compared with trained models by other methods. We also include the comparison with exact inference where $Z(\boldsymbol{\theta})$ is computed exactly. In the grid graphs, there are 4000 samples for training and 1000 for testing. In the complete graph case, there are 2000 samples for training and 1000 samples for testing.

In the cases of grid graphs, the NLLs of most methods are close to the true NLL for small-sized graphs ($n = 25, 100$), with RENN reaching the lowest NNL. In the case of $n = 255$, RENN outperforms all other methods significantly. Additionally, RENN is much faster. As shown in Table 5.3, loopy BP needs almost 335s and damped BP needs about 525s per epoch iteration, while RENN takes 16s per epoch. Neural network based methods parameterize the beliefs or marginal distributions and thus can do new inference estimations much faster when model parameter $\boldsymbol{\theta}$ is updated in optimization steps.

In the cases of complete graphs, the advantage of RENN is significant, compared with other methods as shown in Table 5.2. Other benchmark methods fall behind RENN by a distinct difference, given the size of graphs is relatively small. The results here actually agree with inference experiments shown in Table 4.4 and 4.5, where partition function estimations of other benchmark methods have much larger errors. As for the average time per epoch, neural network based models still are faster than iterative message-passing methods in general.

5.4 Discussion on Learning

The parameter learning becomes more challenge when variable vector $\boldsymbol{x}$ is *partial observed.* Denote $\boldsymbol{x} = [\boldsymbol{x}_U, \boldsymbol{x}_O]$, where $\boldsymbol{x}_O$ is the observed part and $\boldsymbol{x}_U$ is unobserved part that is also known as latent variable. Then the joint distribution becomes $p(\boldsymbol{x}; \boldsymbol{\theta}) = p(\boldsymbol{x}_U, \boldsymbol{x}_O; \boldsymbol{\theta})$. Since it is only partially observed, we can not maximize the complete evidence log-likelihood as what we did in section 5.1. Instead, we marginalize the latent variable $\boldsymbol{x}_U$ out first

$$l(\boldsymbol{x}_O; \boldsymbol{\theta}) = \log \sum_{\boldsymbol{x}_U} p(\boldsymbol{x}_U, \boldsymbol{x}_O; \boldsymbol{\theta}) = \log Z(\boldsymbol{x}_O; \boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}), \tag{5.10}$$

where $Z(\boldsymbol{x}_O; \boldsymbol{\theta}) = \sum_{\boldsymbol{x}_O} \tilde{p}(\boldsymbol{x}; \boldsymbol{\theta})$. In general, there are two categories of methods in dealing with learning in partial observed scenarios:

- Directly optimize $l(\boldsymbol{x}_O; \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$. We need to be able to compute or estimate $l(\boldsymbol{x}_O; \boldsymbol{\theta})$. Essentially, it requires the gradient of $l(\boldsymbol{x}_O; \boldsymbol{\theta})$ which is a function of both $p(\boldsymbol{x}_a | \boldsymbol{x}_O; \boldsymbol{\theta})$ and $p(\boldsymbol{x}_a; \boldsymbol{\theta})$. Inference methods that we previously discussed, e.g., mean field, BP (and its variants), GBP, and RENN, would do the job.

- Optimize the a lower bound of $l(\boldsymbol{x}_O; \boldsymbol{\theta})$. Methods such as (variational) EM, variational Bayes and variational auto-encoder (VAE) [43] belongs to this category.

The first track is straightforward since it is to estimate the sub-partition function $Z(\boldsymbol{x}_O; \boldsymbol{\theta})$ and partition function $Z(\boldsymbol{x}; \boldsymbol{\theta})$ as done in section 5.2 via RENN or other inference methods. Then do the maximization step sequentially. Note different from partition function $Z(\boldsymbol{\theta})$, the sub-partition function $Z(\boldsymbol{x}_O; \boldsymbol{\theta})$ is evidence-dependent, i.e. the inference has to be done per given evidence or sample $\boldsymbol{x}_O$.

Alternative, we may work with a lower bound of $l(\boldsymbol{x}_O; \boldsymbol{\theta})$ instead. In this track, the most classical framework is EM, which can be well explained by our target of maximizing $l(\boldsymbol{x}_O; \boldsymbol{\theta})$. The intuition can be obtained by firstly observing that

$$\begin{aligned} F_V(q(\boldsymbol{x}_U|\boldsymbol{x}_O)) &= \mathrm{KL}(q(\boldsymbol{x}_U|\boldsymbol{x}_O)||p(\boldsymbol{x}_U|\boldsymbol{x}_O; \boldsymbol{\theta})) - \log Z(\boldsymbol{x}_O; \boldsymbol{\theta}) \\ &= \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \left[\log \frac{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}{p(\boldsymbol{x}_U|\boldsymbol{x}_O; \boldsymbol{\theta})} \right] - \log Z(\boldsymbol{x}_O; \boldsymbol{\theta}) \\ &= \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \left[\log \frac{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}{\tilde{p}(\boldsymbol{x}_U, \boldsymbol{x}_O; \boldsymbol{\theta})} \right], \end{aligned} \tag{5.11}$$

similar to the variational free energy (2.8). Here $\boldsymbol{x}_O$ is the clamped value of observation. Since

$$\mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log\frac{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}{\tilde{p}(\boldsymbol{x}_U,\boldsymbol{x}_O;\boldsymbol{\theta})}\right]\geqslant -\log Z(\boldsymbol{x}_O;\boldsymbol{\theta}), \tag{5.12}$$

we have

$$\begin{aligned} l(\boldsymbol{x}_O;\boldsymbol{\theta}) &\geqslant \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log\frac{\tilde{p}(\boldsymbol{x}_U,\boldsymbol{x}_O;\boldsymbol{\theta})}{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\right]-\log Z(\boldsymbol{\theta}) \\ &= \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log\frac{p(\boldsymbol{x}_U,\boldsymbol{x}_O;\boldsymbol{\theta})}{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\right] \\ &= \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log p(\boldsymbol{x}_U,\boldsymbol{x}_O;\boldsymbol{\theta})\right]+H(q(\boldsymbol{x}_U|\boldsymbol{x}_O)) \\ &:= F(q,\boldsymbol{\theta}), \end{aligned} \tag{5.13}$$

which is exactly the lower bound (2.19). This give the intuition of viewing EM as a coordinate ascent method:

$$\text{E step}: \quad q^{(t+1)} = \underset{q}{\operatorname{argmax}}\ F(q,\boldsymbol{\theta}^{(t)}), \tag{5.14}$$

$$\text{M step}: \quad \boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ F(q^{(t+1)},\boldsymbol{\theta}). \tag{5.15}$$

It is interesting to note that VAE maximizes the same lower bound of $l(\boldsymbol{x}_O;\boldsymbol{\theta})$ as EM and variational Bayes, by slightly rewriting the bound as

$$F(q,\boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log p(\boldsymbol{x}_O|\boldsymbol{x}_U;\boldsymbol{\theta})+\log p(\boldsymbol{x}_U;\boldsymbol{\theta})\right]+H(q(\boldsymbol{x}_U|\boldsymbol{x}_O)), \tag{5.16}$$

with encoder $q(\boldsymbol{x}_U|\boldsymbol{x}_O)$ and decoder $p(\boldsymbol{x}_O|\boldsymbol{x}_U;\boldsymbol{\theta})$. The prior on latent variable $\boldsymbol{x}_U$, i.e. $p(\boldsymbol{x}_U;\boldsymbol{\theta})$ is usually assumed as a known Gaussian distribution.

Remark 9. *The lower bound $F(q,\boldsymbol{\theta})$ of $l(\boldsymbol{x}_O;\boldsymbol{\theta})$ can also be obtained from a importance sampling trick with Jensen's inequality.*

$$\begin{aligned} l(\boldsymbol{x}_O;\boldsymbol{\theta}) &= \log\sum_{\boldsymbol{x}_U} p(\boldsymbol{x}_O,\boldsymbol{x}_U;\boldsymbol{\theta}) \\ &= \log\sum_{\boldsymbol{x}_U} q(\boldsymbol{x}_U|\boldsymbol{x}_O)\frac{p(\boldsymbol{x}_O,\boldsymbol{x}_U;\boldsymbol{\theta})}{q(\boldsymbol{x}_U|\boldsymbol{x}_O)} \qquad (5.17) \\ &\geqslant \mathbb{E}_{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\left[\log\frac{p(\boldsymbol{x}_O,\boldsymbol{x}_U;\boldsymbol{\theta})}{q(\boldsymbol{x}_U|\boldsymbol{x}_O)}\right], \qquad (5.18) \end{aligned}$$

where (5.17) *can be seen as an evaluation of importance sampling with proposal distribution q and* (5.18) *uses Jensen's inequality* $\log\mathbb{E}_{p(x)}[f(x)]\geqslant\mathbb{E}_{P(x)}[\log f(x)]$.

Remark 10. *Among all the methods discussed in the section, a key piece of the methods is computing or estimating $p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta})$. Apart from direct optimization of $l(\boldsymbol{x}_O;\boldsymbol{\theta})$ with exact inference methods and EM method that uses exact $q(\boldsymbol{x}_U|\boldsymbol{x}_O)=p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta})$, the rest employs an approximation of the posterior, i.e. $q(\boldsymbol{x}_U|\boldsymbol{x}_O)\approx p(\boldsymbol{x}_U|\boldsymbol{x}_O;\boldsymbol{\theta})$.*

Chapter 6

Equip Expectation Maximization with Normalizing Flows

When learning with partial observed varaibles in graphical models, EM is one of the most widely used frameworks in practices. In directed graphical models, an immediate relieve is that we do not to deal with the partition function $Z(\boldsymbol{\theta})$ any more. However, it does not mean directed graphical models with partial observed variables are trivial to learn. The 'missing' data (corresponding to unobserved variables) still poses challenges in model learning, since we want to have the partial evidence (observed) well explained by our model while keeping its ability of tracking uncertainty of the 'unobserved' part.

As discussed in section 2.5, the presence of latent or hidden variables is either due to the data availability or abstraction of data generating. Once the dataset is given, the option left to us is the choice of the model that we plan to use to approximate the true distribution p^* which actually generated the dataset. This process is to select one model from a set of models, i.e. hypothesis space, based on the likelihood criteria. If the hypothesis space is very limited, the ability to represent our true p^* is limited, leading to inherent error. This type of inherent limitation brings learning *bias* before we start to fit the parameters. Therefore, given that the dataset is large and representative enough, we hope that the hypothesis space is highly expressive such that we are more likely to be able to represent p^* with a selected model from the hypothesis space. This naturally requires our model to be flexible and expressive.

Gaussian mixture model has been successfully applied widely with EM framework, which also enjoys the closed-form update rule in iterations. The advantage comes with limitation of linear dependency and free parameters that can be tuned, i.e. limitation of flexibility. To explore further, we discuss more flexible models beyond Gaussian in this chapter. We start with inducing the normalizing flows that are going to used as probabilistic models in place of Gaussian models. A normalizing flow can consist of as many non-linear transformations as needed, and thus is highly flexible. In addition, likelihood computation ability remains and it allows efficient sampling in normalizing flow. But closed-form iterative update solutions are not available any more. We therefore need to design the practical algorithms to model learning in EM framework.

content: Neural Network based Explicit Mixture Models and Expectation-maximization based Learning, under review

section/chapter transition text: mixture model could be obtained from clamping and condition on a discrete variable, ref to Geier, Locally conditional belief propagation. Weller, clamping variables and approximate inference

Remark 11. *Theory interpretation of EM and variational EM, see Section 6.2, Wainwright, Graphical Models, Exponential Families, and Variational Inference.*

Remark 12. *RELATIONSHIP TO K-MEANS CLUSTERING Big picture: The EM algorithm for mixtures of Gaussians is like a soft version of the K-means algorithm.*

Remark 13. *EM lower bound + entropy of posterior of latent variable if a free energy. ref to 10-708 lecture6 note. EM using posterior of latent variable is equivalent to fully observable MLE where statistics are replaced by their expectations w.r.t the posterior.*

Can be viewed as two-node graphical model learning. 10-708lecture5-note

6.1 Normalizing Flow

Normalizing flow is practical approach to improve the flexibility of probabilistic model while still keeping the likelihood computation tractable and the efficiency of generating samples. The general idea is to start with an initial random variable following a relatively simple distribution with known probability density function. Then a chain of invertible parameterized transformations are applied to the initial random variable such that the output follows a more flexible distribution.

Denote the chain of transformations by $\tilde{\boldsymbol{g}}$ and the output by $\tilde{\boldsymbol{x}}$. Then we have

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{g}}(\boldsymbol{z}) \tag{6.1}$$

where $\boldsymbol{z}$ is the initial random variable with know density function $p(\boldsymbol{z})$. To explicitly denote the chain,

$$\tilde{\boldsymbol{g}} = \tilde{\boldsymbol{g}}^{[L]} \circ \tilde{\boldsymbol{g}}^{[L-1]} \circ \cdots \circ \tilde{\boldsymbol{g}}^{[1]}, \tag{6.2}$$

which is invertible $\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{g}}^{-1}$. Then the signal flow can be depicted as

$$\begin{tikzcd}
\boldsymbol{z} = \boldsymbol{h}_0 \arrow[r, shift left, "\tilde{\boldsymbol{g}}^{[1]}"] & \boldsymbol{h}_1 \arrow[l, shift left, "\tilde{\boldsymbol{f}}^{[1]}"] \arrow[r, shift left, "\tilde{\boldsymbol{g}}^{[2]}"] & \cdots\cdots\cdots\cdots \arrow[l, shift left, "\tilde{\boldsymbol{f}}^{[2]}"] \arrow[r, shift left, "\tilde{\boldsymbol{g}}^{[L]}"] & \tilde{\boldsymbol{x}} = \boldsymbol{h}_L \arrow[l, shift left, "\tilde{\boldsymbol{f}}^{[L]}"]
\end{tikzcd}$$

,

where $\tilde{\boldsymbol{g}}^{[l]}$ and $\tilde{\boldsymbol{f}}^{[l]}$ are the l-th transformation in $\tilde{\boldsymbol{g}}$ and $\tilde{\boldsymbol{f}}$, respectively. In practice, $\tilde{g}$ can be implemented by feed-forward neural networks, which we will detail later. If every transformation in the chain is invertible, the full mapping is invertible.

The inverse function is given by $\boldsymbol{z} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}})$. Flow-based network, proposed in [16], is such a feed-forward neural network, which is further improved in subsequent works [15, 41]. It also has additional advantages as efficient Jacobian computation and low computational complexity.

Introduce different flows and give details of normalizing flows.

6.2 expectation maximization of neural network based mixture models

6.3 An alternative construction method

6.4 Experiments

6.5 Summary

6.6 Raw material

6.7 Introduction

The paradigm of neural network based implicit distribution modeling has received a significant attention. In this paradigm, a neural network being a powerful nonlinear function acts as an efficient generator. Prominent examples of neural network based implicit distributions are generative adversarial networks (GANs) [29] and its variants [80,81,104]. GANs are efficient for generating samples and successful in several applications [52], [80]. For a GAN, a latent variable is used as an input to the generator neural network of the GAN and the output of the neural network is considered to be a data sample from the implicit distribution. In implicit distribution modeling by GANs, neither analytical form of the distribution nor likelihood for a data sample is available. Naturally the use of neural network based implicit distribution models like GANs is restricted to many applications where it is important to compute likelihood, for example, maximum likelihood based classification.

In this article, we focus on neural network based explicit distribution modeling. An explicit distribution model has an analytical functional form and we are able to compute likelihood. While use of neural network based generators in GANs for distribution modeling is powerful, we look for further improvements. In this regard, a standard practice is to use mixture models assuming that the underlying distribution is multi-modal, or data are spread over multiple manifolds and subspaces. Therefore we propose to design neural network based explicit mixture models such that they

(a) have analytical forms,

(b) offer the advantage of computing likelihood, and

(c) retain the advantage of generating samples efficiently.

With the advantage of computing likelihood, our proposed neural network based mixture models are suitable for maximum likelihood based classification.

An important question is how to design practical algorithms to learn parameters of the proposed mixture models. In literature, expectation-maximization (EM) [14] is a standard approach for learning parameters of an explicit mixture model in a maximum likelihood framework, such as learning parameters of a Gaussian mixture model (GMM) [7]. For realizing EM, computation of the posterior distribution of a hidden variable (related to identity of a mixture component) given the observation (visible signal/data) is required in the expectation step (E-step). In addition, it is required to compute the joint log-likelihood of the observation signal and the hidden variable in the maximization step (M-step). For example, EM for GMM can be realized due to fullfilment of the above two requirements. Typically it is challenging to fulfill these two requirements for many other mixture distribution models. We also face the challenge to realize EM for learning parameters of neural network based explicit mixture models. This is due to the fact that use of neural networks in design of a system/algorithm/method often leads to loss of required level of analytical tractability.

In pursuit of neural network based explicit mixture models, our contributions in this article are as follows.

(a) Proposing two mixture models - a high-complexity model and a low-complexity model. The low-complexity model uses shared parameters.

(b) Finding theoretical conditions for the models such that EM can be applied for their parameter learning. The theoretical conditions help to find explicit posterior and computation of expected likelihood.

(c) Designing practical algorithms for realization of EM where gradient search based optimization is efficiently embedded into M-step.

(d) Demonstrating efficiency of proposed mixture models through extensive experiments for generating samples and maximum likelihood based classification.

At this point we mention the conditions of realizing EM to learn neural network based explicit mixture models. The sufficient conditions are invertibility of associated neural networks and Jacobian computation of functional form of the neural networks. This helps to compute likelihood using change of variables. In practice we address the sufficient conditions using a flow-based neural network [41].

Related Work and Background

While GANs have high success in many applications, they are known to suffer in a mode dropping problem where a generator of a GAN is unable to capture all modes of an underlying probability distribution of data [39]. To address diversity in data and model multiple modes in a distribution, variants of generative models have been developed and usage of multiple generators has been considered. For instance, methods of minibatch discrimination [80] and feature representation [5] are used to construct new discriminators of GANs which encourage the GANs to generate samples with diversity. Multiple Wasserstein GANs [4] are used in [39] with appropriate mutual information based regularization to encourage the diversity of samples generated by different GANs. A mixture GAN approach is proposed in [34] using multiple generators and multi-classification solution to encourage diversity of samples. Multi-agent diverse GAN [23] similarly employs k generators, but uses a $(k+1)$-class discriminator instead of a typical binary discriminator to increase the diversity of generated samples. These works are implicit probability distribution modeling and thus prior distribution of generators can not be inferred when multiple generators are used.

Typically, for a GAN, the latent variable is assumed to follow a known and fixed distribution, e.g., Gaussian. The latent signal for a given data sample can not be obtained since generators which are usually based on neural networks are non-invertible. The mapping from a data sample to its corresponding latent signal is approximately estimated by neural networks in different ways. [17] and [19] propose to train a generative model and an inverse mapping (also neural network) from the data sample to the latent signal simultaneously, using the adversarial training method of GAN. Alternatively, [88] proposes to approximately minimize a Kullback-Leibler divergence to estimate the mapping from the data sample to the latent variable, which leads to a nontrivial probability density ratio estimation problem.

Another track of mixture modeling is based on ensembling method that combines weaker learners together to boost the overall performance [30, 86]. In this approach mixture models are obtained as follows. Based on how well the current-step mixture model captures the underlying distribution, a new generative model is trained to compensate the miss-captured part. However, measuring the difference between current-step mixture model and underlying distribution of dataset quantitatively is a nontrivial task. In addition, since incremental building components are used in the mixture modeling, parallel training of model components is not feasible.

6.8 Generator mixture model and EM

In this proposed generative model, we have K separate neural networks. All the K neural networks have a common input latent variable $\boldsymbol{z} \in \mathbb{R}^M$. Here, a neural network $\boldsymbol{g}_k(\boldsymbol{z}) : \mathbb{R}^M \rightarrow \mathbb{R}^N$ acts as the k-th generator and depends on a set of parameters $\boldsymbol{\theta}_k$ as $\boldsymbol{g}_k(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k)$. For simplicity, we assume that all K neural networks have the same signal-flow structure. Furthermore, the distribution of $\boldsymbol{z}$

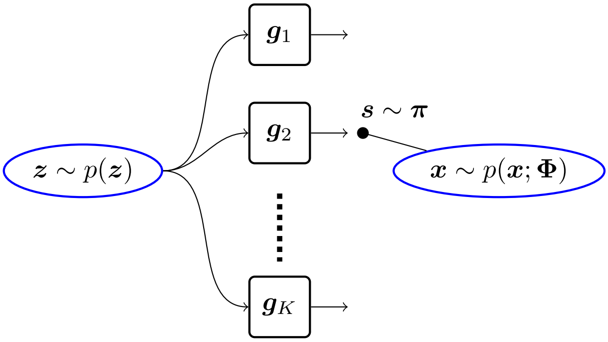

$\boldsymbol{g}_1$
$\boldsymbol{g}_2$
$\boldsymbol{g}_K$
$\boldsymbol{s} \sim \boldsymbol{\pi}$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$\boldsymbol{x} \sim p(\boldsymbol{x}; \boldsymbol{\Phi})$

Figure 6.1: Diagram of Generator Mixture Model (GenMM).

is fixed as Gaussian $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The induced probability density function (pdf) of $\boldsymbol{x} \in \mathbb{R}^N$ of the proposed mixture model with K mixture components is given as:

$$\begin{aligned} p(\boldsymbol{x}; \boldsymbol{\Phi}) &= \textstyle\sum_{k=1}^K \pi_k p_k(\boldsymbol{x}) \\ &= \textstyle\sum_{k=1}^K \pi_k p(\boldsymbol{g}_k(\boldsymbol{z})) \\ &= \textstyle\sum_{k=1}^K \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k)). \end{aligned} \tag{6.3}$$

Their parameters $\boldsymbol{\theta}_k$, however, are different. We use $\boldsymbol{\Phi}$ to denote the set of all parameters $\{\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K\}$, where $\boldsymbol{\pi} = [\pi_1, \dots, \pi_K]^T$ is the prior distribution of the generators. Note that $\pi_k \geqslant 0$ and $\sum_{k=1}^K \pi_k = 1$. The mixture model in (6.3) is called a generator mixture model (GenMM). The diagram of GenMM is illustrated in Figure 6.1. The GenMM can be considered as a high-complexity model because each mixture component $p_k(\boldsymbol{x})$ has its own parameter set $\boldsymbol{\theta}_k$.

The maximum likelihood estimation problem is

$$\hat{\boldsymbol{\Phi}} = \arg\max_{\boldsymbol{\Phi}} \log \textstyle\prod_i p(\boldsymbol{x}^{(i)}; \boldsymbol{\Phi}), \tag{6.4}$$

where the superscript (i) corresponds to the i'th data sample in a given dataset. We address the above maximum likelihood estimation problem using EM. Let us use a categorical variable $\boldsymbol{s} = [s_1, s_2, \cdots, s_K]$ for 1-of-K representation to be a hidden variable that indicates which generator is the actual one. Elements of $\boldsymbol{s}$ follow $s_k \in \{0, 1\}$, $\sum_{k=1}^K s_k = 1$, and $\mathbb{P}\{s_k = 1\} = \pi_k$. The variable $\boldsymbol{s}$ is the hidden variable in EM. We will use γ_k to denote the posterior probability $\mathbb{P}\{s_k = 1 | \boldsymbol{x}\}$ calculated as

$$\gamma_k = \mathbb{P}\{s_k = 1 | \boldsymbol{x}; \boldsymbol{\Phi}\} = \frac{\pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))}{\sum_{l=1}^K \pi_l p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_l))}. \tag{6.5}$$

The posterior probability γ_k is also known as responsibility in the EM literature. Assume that a value $\boldsymbol{\Phi}^{\text{old}}$ of the parameter set $\boldsymbol{\Phi}$ is given, the iterative steps in EM algorithm update $\boldsymbol{\Phi}$ as follows.

1. E-step: Evaluation of $\gamma_k^{(i)}$ is

$$\gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) = \frac{\pi_k^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z}^{(i)}; \boldsymbol{\theta}_k^{\text{old}}))}{\sum_{l=1}^{K} \pi_l^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z}^{(i)}; \boldsymbol{\theta}_l^{\text{old}}))}. \tag{6.6}$$

2. M-step: Evaluation of $\boldsymbol{\Phi}^{\text{new}}$ given by

$$\boldsymbol{\Phi}^{\text{new}} = \arg\max_{\boldsymbol{\Phi}} \mathcal{Q}(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\text{old}}), \tag{6.7}$$

where the expected likelihood is

$$\mathcal{Q}(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\text{old}}) = \sum_i \sum_k \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) \log \pi_k p_k(\boldsymbol{x}^{(i)}). \tag{6.8}$$

For the GenMM in (6.3), the main technical challenges in realizing EM are computation of γ_k in the E-step and computation of the joint likelihood $\log \pi_k p_k(\boldsymbol{x})$ in the M-step. They require explicit computation of the conditional density $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z})) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$. Thus, the problem statement is how to design the neural network $\boldsymbol{g}_k(\cdot) = \boldsymbol{g}(\cdot; \boldsymbol{\theta}_k)$ such that $p_k(\boldsymbol{x}) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$ can be computed.

On Theoretical Requirement

We provide sufficient conditions for realizing EM algorithm associated with learning parameters of GenMM.

Proposition 1 (Sufficient conditions). *EM algorithm for GenMM distribution* (6.3) *is realizable if every generator neural network is an one-to-one mapping function and $M = N$. That means $\boldsymbol{g}_k(\boldsymbol{z}) : \mathbb{R}^N \to \mathbb{R}^N, \forall k$ are invertible.*

Proof: We use the multivariate transformation method to prove the proposition. To realize EM for GenMM, it is required to compute $p_k(\boldsymbol{x}), \forall k$. Without loss of generality we consider computation of $\gamma_k = \mathbb{P}(s_k = 1 | \boldsymbol{x}; \boldsymbol{\Phi})$ and $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z})) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$. Let us denote the output of the k-th generator by $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{g}}(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k)$. We have $\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{g}}(\boldsymbol{z})$. Under the conditions $M = N$ and invertible $\boldsymbol{g}_k(\boldsymbol{z}) = \tilde{\boldsymbol{g}}(\boldsymbol{z})$, there exists an inverse function $\tilde{\boldsymbol{g}}^{-1}(\tilde{\boldsymbol{x}}) = [\tilde{g}_1^{-1}(\tilde{\boldsymbol{x}}), \tilde{g}_2^{-1}(\tilde{\boldsymbol{x}}), \ldots, \tilde{g}_N^{-1}(\tilde{\boldsymbol{x}})]^\top = \boldsymbol{z}$. Then, the Jacobian of this multivariate transformation is

Let $\det(\boldsymbol{J})$ denotes the determinant of $\boldsymbol{J}$. Then, the pdf is

$$p_k(\boldsymbol{x}) = p(\tilde{\boldsymbol{x}}) = p(\boldsymbol{z}) \left| \det(\boldsymbol{J}) \big|_{\boldsymbol{z} = \tilde{\boldsymbol{g}}^{-1}(\tilde{\boldsymbol{x}})} \right|. \tag{6.9}$$

Similarly, we can compute the pdf of other mixture components.

Algorithm for Learning

In this section we first discuss about a suitable neural network model for $\boldsymbol{g}_k(\boldsymbol{z})$ in GenMM and then design the EM algorithm for GenMM.

Use of flow-based neural network

For a flow-based neural network architecture, let us assume that the feature $\boldsymbol{h}_l$ at the l'th layer has two subparts as $\boldsymbol{h}_l = [\boldsymbol{h}_{l,a}^T, \boldsymbol{h}_{l,b}^T]^T$ where $(\cdot)^T$ denotes transpose operation. Then considering $\boldsymbol{h}_0 = \boldsymbol{z}$, we have the following forward and inverse relations between $(l-1)$'th and l'th layers:

$$\begin{aligned} \boldsymbol{h}_{l-1} &= \begin{bmatrix} \boldsymbol{h}_{l-1,a} \\ \boldsymbol{h}_{l-1,b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_{l,a} \\ \boldsymbol{m}_a(\boldsymbol{h}_{l,a}) \odot \boldsymbol{h}_{l,b} + \boldsymbol{m}_b(\boldsymbol{h}_{l,a}) \end{bmatrix}, \\ \boldsymbol{h}_{l} &= \begin{bmatrix} \boldsymbol{h}_{l,a} \\ \boldsymbol{h}_{l,b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_{l-1,a} \\ (\boldsymbol{h}_{l-1,b} - \boldsymbol{m}_b(\boldsymbol{h}_{l-1,a})) \oslash \boldsymbol{m}_a(\boldsymbol{h}_{l-1,a}) \end{bmatrix}, \end{aligned} \tag{6.10}$$

where $\odot$ denotes element-wise product, $\oslash$ denotes element-wise division, and $\boldsymbol{m}_a(\cdot), \boldsymbol{m}_b(\cdot)$ can be complex non-linear mappings (implemented by neural networks). For the flow-based neural network, the determinant of Jacobian matrix is

$$\det(\boldsymbol{J})|_{\boldsymbol{z}=\tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}})} \quad = \textstyle\prod_{l=1}^{L} \det(\boldsymbol{J}_l)|_{\boldsymbol{h}_l}, \tag{6.11}$$

where $\boldsymbol{J}_l$ is the Jacobian of the transformation from the l-th layer to the $(l-1)$-th layer, i.e., the inverse transformation. We compute the determinate of the Jacobian matrix as

$$\begin{aligned} \det(\boldsymbol{J}_l)|_{\boldsymbol{h}_l} &= \det\left[\frac{\partial \boldsymbol{h}_{l-1}}{\partial \boldsymbol{h}_l}\right] \\ &= \det\begin{bmatrix} \boldsymbol{I}_a & \boldsymbol{0} \\ \frac{\partial \boldsymbol{h}_{l-1,b}}{\partial \boldsymbol{h}_{l,a}} & \mathrm{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a})) \end{bmatrix} \\ &= \det\left(\mathrm{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a}))\right), \end{aligned} \tag{6.12}$$

where $\boldsymbol{I}_a$ is identity matrix and diag$(\cdot)$ returns a square matrix with the elements of $(\cdot)$ on the main diagnal. Then the pdf is

$$\begin{aligned} p(\tilde{\boldsymbol{x}}) &= p(\boldsymbol{z})\left|\det(\boldsymbol{J})|_{\boldsymbol{z}=\tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}})}\right| \\ &= p(\boldsymbol{z}) \prod_{l=1}^{L} |\det\left(\mathrm{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a}))]\right)|. \end{aligned} \tag{6.13}$$

Equation 6.10 describes a *coupling* mapping between layers. Since the coupling has a partial identity mapping, direct concatenation of multiple such coupling mappings would result in a partial identity mapping of the whole neural network $\tilde{\boldsymbol{g}}$. Alternating the positions of identity mapping [15] or using 1 × 1 convolution operations [41] before each coupling mapping is used to treat the issue. Furthermore, [15] [41] split some hidden layer signal $\boldsymbol{h}$ and model a part of it directly as standard Gaussian to reduce computation and memory burden.

EM Algorithm for GenMM

The mixture model GenMM is illustrated in Figure 6.1, where K generators with a certain prior distribution share the same latent distribution $p(\boldsymbol{z})$. With a flow-based neural network used as the generator $\boldsymbol{g}_k$ for the k'th mixture component in GenMM, the pdf $p_k(\boldsymbol{x})$ for any $\boldsymbol{x}$ can be computed exactly. Recall that $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z})) = p(\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta}_k))$. Let $\boldsymbol{f}_k$ be the inverse of $\boldsymbol{g}_k$. Then, the posterior probability can be computed further from Equation 6.6 as

$$\begin{aligned}\gamma_k(\boldsymbol{\Phi}^{\text{old}}) =& \frac{\pi_k^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta}_k))}{\sum_{j=1}^{K} \pi_j^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta}_j))} \\ =& \frac{\pi_k^{\text{old}} p(\boldsymbol{f}_k(\boldsymbol{x}))\left|\det\left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|}{\sum_{j=1}^{K} \pi_j^{\text{old}} p(\boldsymbol{f}_j(\boldsymbol{x}))\left|\det\left(\frac{\partial \boldsymbol{f}_j(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|},\end{aligned} \tag{6.14}$$

and the objective function in the M-step can be written as

$$\begin{aligned}\mathcal{Q}\left(\boldsymbol{\Phi},\boldsymbol{\Phi}^{\text{old}}\right) = \sum_{i=1}^{n}\sum_{k=1}^{K} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}})\Big[\log\ \pi_k \\ + \log\ p(\boldsymbol{f}_k(\boldsymbol{x}^{(i)})) + \log\ \left|\det\left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}}\right)\right|\Big],\end{aligned} \tag{6.15}$$

where n denotes the number of data samples. We usually deal with a large dataset for model learning, i.e. n is large. In that case we implement the EM algorithm in batch fashion. Recall that $\boldsymbol{\Phi} = \{\boldsymbol{\pi},\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K\}$ and hence the M-step optimization problem $\arg\max_{\boldsymbol{\Phi}} \mathcal{Q}(\boldsymbol{\Phi},\boldsymbol{\Phi}^{\text{old}})$ is addressed in two steps: (a) optimization of $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$, and (b) optimization of $\boldsymbol{\pi}$.

Finding a closed-form solution for the problem $\arg\max_{\{\boldsymbol{\theta}_k\}_{k=1}^{K}} \mathcal{Q}(\boldsymbol{\Phi},\boldsymbol{\Phi}^{\text{old}})$ is challenging. Instead, we do the batch-size gradient decent to optimize w.r.t. $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$. Further, optimization in the batch fashion leads to a practical problem as follows. Since $\boldsymbol{\theta}_k$ is the parameter set of neural networks $\boldsymbol{g}_k$, one update step of gradient decent would update the generator $\boldsymbol{g}_k$ and we would lose the old mixture model parameter set $\boldsymbol{\Phi}^{\text{old}}$ that is needed to compute the posteriors $\gamma_k(\boldsymbol{\Phi}^{\text{old}})$ and to update $\boldsymbol{\pi}$. Thus, in learning GenMM, we maintain two such models with parameter sets $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}^{\text{old}}$, respectively. At the beginning of an EM step, $\boldsymbol{\Phi} = \boldsymbol{\Phi}^{\text{old}}$. While we optimize $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$ of $\boldsymbol{\Phi}$ with batch-size gradient decent, we use the model with old parameter set $\boldsymbol{\Phi}^{\text{old}}$ to do posterior computation and update of $\boldsymbol{\pi}$. At the end of the EM step, the old parameter set is replaced by the updated one: $\boldsymbol{\Phi}^{\text{old}} \leftarrow \boldsymbol{\Phi}$.

Then we discuss the optimization of the prior distribution $\boldsymbol{\pi}$. The optimization problem is

$$\boldsymbol{\pi}^{\text{new}} = \arg\max_{\boldsymbol{\pi}} \mathcal{Q}(\boldsymbol{\Phi},\boldsymbol{\Phi}^{\text{old}}),\ \text{s.t.} \sum_{k=1}^{K} \pi_k = 1. \tag{6.16}$$

Algorithm 2 EM for learning GenMM

```
Input: Latent distribution: $p(\boldsymbol{z})$. Empirical distribution $P_d(\boldsymbol{x})$ of the input dataset;
Set a total number of epochs $T$ for training, a prior distribution $\boldsymbol{\pi}$, EM update gap $t_{\mathrm{EM}}$;
Set a learning rate $\eta$.
Build two models with parameter sets:
$\boldsymbol{\Phi}^{\mathrm{old}} = \{\boldsymbol{\pi}^{\mathrm{old}}, \boldsymbol{\theta}_1^{\mathrm{old}}, \dots, \boldsymbol{\theta}_K^{\mathrm{old}}\}$,
$\boldsymbol{\Phi} = \{\boldsymbol{\pi}, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K\}$.
Initialize the generator prior distribution $\pi_k = 1/K$;
Initialize $\boldsymbol{\theta}_k$ of $\boldsymbol{g}_k$, for all $k = 1, \dots, K$ randomly.
$\boldsymbol{\Phi}^{\mathrm{old}} \leftarrow \boldsymbol{\Phi}$.
for epoch $t < T$ do
    for the iteration in epoch $t$ do
        Sample a batch of data $\{\boldsymbol{x}^{(i)}\}_{i=1}^{n_b}$ from the dataset $P_d(\boldsymbol{x})$
        Compute $\gamma_k^{(i)}(\boldsymbol{\Phi}^{\mathrm{old}})$ as in Equation 6.14, for all $\boldsymbol{x}^{(i)}$ and $k = 1, \dots, K$
        Compute $\mathcal{Q}\left(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\mathrm{old}}\right)$ as in Equation 6.15
        $\partial g_k \leftarrow \nabla_{\boldsymbol{\theta}_k} \frac{1}{n^b} \mathcal{Q}\left(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\mathrm{old}}\right), \forall \boldsymbol{\theta}_k \in \boldsymbol{\Phi}$
        $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k + \eta \cdot \partial g_k, \forall \boldsymbol{\theta}_k \in \boldsymbol{\Phi}$
    end for
    if $(t \mod t_{\mathrm{EM}}) = 0$ then
        $\pi_k \leftarrow \mathbb{E}_{P_d}[\gamma_k]$
        $\boldsymbol{\Phi}^{\mathrm{old}} \leftarrow \boldsymbol{\Phi}$.
    end if
end for
```

The update of prior follows the solution

$$\pi_k^{\mathrm{new}} = \frac{1}{n} \sum_{i=1}^{n} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\mathrm{old}}). \tag{6.17}$$

The detail to get the solution is derived in the subsection 6.8. For a given dataset with empirical distribution $P_d(\boldsymbol{x})$, γ_k is evaluated with batch data in order to calculate the cost $\mathcal{Q}\left(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\mathrm{old}}\right)$ and to update the parameter $\boldsymbol{\theta}_k$ of $\boldsymbol{g}_k$. We accumulate the values of γ_k of batches and average out for one epoch to update $\boldsymbol{\pi}$, i.e., $\pi_k \leftarrow \mathbb{E}_{P_d}[\gamma_k]$.

We summarize the EM algorithm for GenMM in Algorithm 2. In implementation, to avoid numerical computation problem, $\log p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$ is scaled by the dimension of signal $\boldsymbol{x}$ in order to compute γ_k.

Proof for update of π

The optimization of $\boldsymbol{\pi}$ is addressed in the following Lagrangian form

$$\mathcal{F}(\boldsymbol{\Phi}) = \mathcal{Q}(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\text{old}}) + \lambda \left(1 - \sum_{k=1}^{K} \pi_k \right), \tag{6.18}$$

where λ is the Lagrange multiplier. Then

$$\begin{aligned}
\boldsymbol{\pi}^{\text{new}} &= \arg\max_{\boldsymbol{\pi}} \mathcal{F}(\boldsymbol{\Phi}) \\
&= \arg\max_{\boldsymbol{\pi}} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) \Bigg[\log\ \pi_k + \log\ p(\boldsymbol{f}_k(\boldsymbol{x}^{(i)})) \\
&\quad + \log \left| \det \left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right) \right| \Bigg] + \lambda \left(1 - \sum_{k=1}^{K} \pi_k \right) \\
&= \arg\max_{\boldsymbol{\pi}} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) \log\ \pi_k + \lambda \left(1 - \sum_{k=1}^{K} \pi_k \right),
\end{aligned} \tag{6.19}$$

where $\boldsymbol{f}_k = \boldsymbol{g}_k^{-1}$. Then solving

$$\frac{\partial \mathcal{F}}{\partial \pi_k} = 0, k = 1, 2, \cdots, K, \tag{6.20}$$

we get $\pi_k = \frac{1}{\lambda} \sum_{i=1}^{n} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}), \forall k$. With condition $\sum_{k=1}^{K} \pi_k = 1$, we have $\lambda = \sum_{k=1}^{K} \sum_{i=1}^{n} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) = n$. Therefore, the solution is $\pi_k = \frac{1}{n} \sum_{i=1}^{n} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}), \forall k$. Note that the updated prior parameter π_k is non-negative due to the non-negativity of the posterior $\gamma_k(i)$.

On Convergence of GenMM

In general the convergence of EM is guaranteed only in some cases, cf. [**?**]. However, under some conditions our GenMM converges. In what follows we present the convergence arguments.

Proposition 2. *Assume that for all k, the parameters $\boldsymbol{\theta}_k$ are in a compact set such that the corresponding mapping $\boldsymbol{g}_k$ is invertible. Assume further that all generator mappings fulfill that $\boldsymbol{f}_k$ and $\frac{\partial \boldsymbol{f}_k}{\partial \boldsymbol{x}}$ are continuous functions of $\boldsymbol{\theta}_k$. Then GenMM converges.*

Proof. Assume that the assumption holds. Then the determinant term $\det(\boldsymbol{J})$ in Equation 6.9 is a continuous function of $\boldsymbol{\theta}_k$. Due to Equation 6.9 and the continuity of Gaussian density $p(\boldsymbol{z})$, the pdf $p_k(\boldsymbol{x})$ is a continuous function of $\boldsymbol{\theta}_k$. Therefore, $p(\boldsymbol{x})$ given in Equation 6.3 is a continuous function of $\boldsymbol{\Phi}$. Denote the likelihood in Equation 6.4 as $\mathcal{L}(\boldsymbol{\Phi}) = \log \prod_i p(\boldsymbol{x}^{(i)}; \boldsymbol{\Phi})$. The maximum value of $\mathcal{L}(\boldsymbol{\Phi})$ is bounded due to continuity of $p(\boldsymbol{x})$ w.r.t. $\boldsymbol{\Phi}$. Define $\mathcal{B}(\boldsymbol{\Phi}) = \mathcal{Q}\left(\boldsymbol{\Phi}, \boldsymbol{\Phi}^{\text{old}}\right) - \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}}) \log \gamma_k^{(i)}(\boldsymbol{\Phi}^{\text{old}})$. It is well known that $\mathcal{B}(\boldsymbol{\Phi})$ is a lower bound on the likelihood function $\mathcal{L}(\boldsymbol{\Phi})$, i.e. $\mathcal{L}(\boldsymbol{\Phi}) \geqslant \mathcal{B}(\boldsymbol{\Phi})$. Note that the essence of EM algorithm is that the likelihood function value is elevated by increasing the value of its lower bound $\mathcal{B}(\boldsymbol{\Phi})$. Since the maximum value of the log-likelihood $\mathcal{L}(\boldsymbol{\Phi})$ is finite, $\mathcal{B}(\boldsymbol{\Phi})$ can not grow unbounded.

□

6.9 A low-complexity model

There are K neural networks in GenMM, which makes GenMM a high-complexity model. We now propose a low-complexity model where parameters are shared. This is motivated by many machine learning setups where model parameters are shared across model components. For example, this techniques is applied as use of shared covariance matrices in a tied Gaussian mixture model, in linear discriminant analysis [?, ?, 7], and use of common subspace in non-negative matrix factorization [?]. Based on the idea of sharing parameters, we propose a low-complexity model which we refer to as latent mixture model as follow.

Latent mixture model

In this generative model, we use a latent variable $\boldsymbol{z}$ that has the following Gaussian mixture distribution

$$p(\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k p_k(\boldsymbol{z}), \tag{6.21}$$

where $p_k(\boldsymbol{z})$ is pdf of Gaussian distribution $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_k, \boldsymbol{C}_k)$ with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{C}_k$. The data $\boldsymbol{x}$ is assumed to be generated in the model using a single neural network $\boldsymbol{g}(\boldsymbol{z}) : \mathbb{R}^M \to \mathbb{R}^N$ as $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the set of parameters of the neural network. The diagram of this mixture model is shown in Figure 6.2. Similarly, we use $\underline{\boldsymbol{\Phi}}$ to denote the set of all parameters $\{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K, \boldsymbol{C}_1, \dots, \boldsymbol{C}_K, \boldsymbol{\theta}\}$. Furthermore, we also have a categorical variable $\boldsymbol{s}$ to indicate which underlying source is chosen. The density function of the proposed latent mixture model (LatMM) is given as

$$\begin{aligned} p(\boldsymbol{x}; \underline{\boldsymbol{\Phi}}) &= \textstyle\sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x}) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}) | s_k = 1) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_k, \boldsymbol{C}_k). \end{aligned} \tag{6.22}$$

The LatMM is illustrated in Figure 6.2 where the neural network $\boldsymbol{g}$ is shared. Learning of LatMM requires solving the maximum likelihood estimation problem

$$\hat{\underline{\boldsymbol{\Phi}}} = \arg\max_{\underline{\boldsymbol{\Phi}}} \log \prod_i p(\boldsymbol{x}^{(i)}; \underline{\boldsymbol{\Phi}}), \tag{6.23}$$

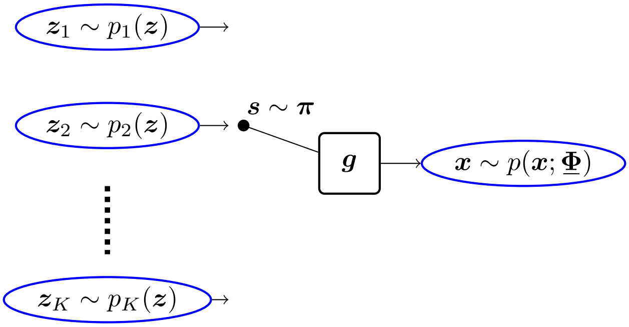

$\boldsymbol{z}_1 \sim p_1(\boldsymbol{z})$
$\boldsymbol{z}_2 \sim p_2(\boldsymbol{z})$
$\boldsymbol{z}_K \sim p_K(\boldsymbol{z})$
$\boldsymbol{s} \sim \boldsymbol{\pi}$
$\boldsymbol{g}$
$\boldsymbol{x} \sim p(\boldsymbol{x}; \underline{\boldsymbol{\Phi}})$

Figure 6.2: Diagram of Latent Mixture Model (LatMM).

which we address using EM. We have

$$\underline{\gamma}_k = \mathbb{P}(s_k = 1 | \boldsymbol{x}; \underline{\boldsymbol{\Phi}}) = \frac{\pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_k, \boldsymbol{C}_k)}{\sum_{l=1}^{K} \pi_l p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_l, \boldsymbol{C}_l)}. \tag{6.24}$$

Similar to the case of GenMM, realization of the corresponding EM algorithm associated with LatMM in Equation 6.22 also has technical challenges on computing the posterior distribution $\underline{\gamma}_k$ and the joint likelihood $\log \pi_k p_k(\boldsymbol{x})$. They require explicit computation of the conditional density function $p_k(\boldsymbol{x}) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}) | s_k = 1) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_k, \boldsymbol{C}_k)$. In LatMM, $\boldsymbol{g}(\boldsymbol{z}) : \mathbb{R}^N \to \mathbb{R}^N$ is also required to be invertible. We model $\boldsymbol{g}$ by a flow-based neural network as explained in section 6.8. Then, the problem is how to learn the parameters of LatMM.

EM Algorithm for LatMM

Algorithm 3 summarizes the EM algorithm for LatMM. LatMM is used to learn one generative model that gets input from a mixture latent source distribution with one single generator $\boldsymbol{g}$. For simplicity, we set the covariance matrix of each latent Gaussian source as a diagonal matrix, $\boldsymbol{C}_k = \mathrm{diag}(\boldsymbol{\sigma}_k^2)$. Each component $p_k(\boldsymbol{z})$ of the latent source $p(\boldsymbol{z})$ can be obtained by an affine transform from the standard Gaussian, i.e., $\boldsymbol{z}_k \sim p_k(\boldsymbol{z})$ can be obtained by a linear layer of neural network with $\boldsymbol{z}_k = \boldsymbol{\mu}_k + \boldsymbol{\sigma}_k \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. According to section 6.9, the posterior and objective function in M-step of LatMM can be computed as

$$\underline{\gamma}_k(\underline{\boldsymbol{\Phi}}^{\mathrm{old}}) = \left. \frac{\pi_k^{\mathrm{old}} p_k\left(\boldsymbol{z}\right)}{\sum_{j=1}^{K} \pi_j^{\mathrm{old}} p_j\left(\boldsymbol{z}\right)} \right|_{\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x})}, \tag{6.25}$$

$$\begin{aligned} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Phi}}, \underline{\boldsymbol{\Phi}}^{\mathrm{old}}\right) = \sum_{i=1}^{n} \log \left| \det\left(\frac{\partial \boldsymbol{f}(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right) \right| \\ + \sum_{k=1}^{K} \underline{\gamma}_k^{(i)}(\underline{\boldsymbol{\Phi}}^{\mathrm{old}}) \left[\log \pi_k + \log p_k\Big(\boldsymbol{f}(\boldsymbol{x}^{(i)}); \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2 \Big) \right], \end{aligned} \tag{6.26}$$

Algorithm 3 EM for learning LatMM

```
**Input:** Empirical distribution $P_d(\boldsymbol{x})$ of dataset;
Latent mixture distribution:
$\sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{z}; \boldsymbol{\mu}_k, \mathrm{diag}(\boldsymbol{\sigma}_k^2)\right)$
Set a total number of epochs $T$ of training, prior $\boldsymbol{\pi}$ update gap $t_\pi$, EM update
    gap $t_{\mathrm{EM}}$, a learning rate $\eta$; Set hyperparameter $a$ , $b$ for prior of $\boldsymbol{\sigma}_k^{-1}, \forall k$.
Build two models with parameter sets:
$\underline{\boldsymbol{\Phi}}^{\mathrm{old}} = \{\boldsymbol{\pi}^{\mathrm{old}}, \boldsymbol{\mu}_1^{\mathrm{old}}, \ldots, \boldsymbol{\mu}_K^{\mathrm{old}}, \boldsymbol{\sigma}_1^{\mathrm{old}}, \ldots, \boldsymbol{\sigma}_K^{\mathrm{old}}, \boldsymbol{\theta}^{\mathrm{old}}\}$,
$\underline{\boldsymbol{\Phi}} = \{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_K, \boldsymbol{\theta}\}$.
Initialize the generator prior distribution $\pi_k = 1/K$ and initialize its $\boldsymbol{\theta}$ for $\boldsymbol{g}$,
    $\boldsymbol{\mu}_k$, $\boldsymbol{\sigma}_k$, $\forall k$ randomly.
$\underline{\boldsymbol{\Phi}}^{\mathrm{old}} \leftarrow \underline{\boldsymbol{\Phi}}$
**for** epoch $t < T$ **do**
    **for** the iteration in epoch $t$ **do**
        Sample a batch of data $\left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{n_b}$ from dataset
        Compute $\underline{\gamma}_k(\underline{\boldsymbol{\Phi}}^{\mathrm{old}})$ by Equation 6.25, $\forall \boldsymbol{x}^{(i)}$ and $k = 1, 2, \cdots, K$
        Compute $\underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Phi}}, \underline{\boldsymbol{\Phi}}^{\mathrm{old}}\right)$ in Equation 6.26
        $\partial\boldsymbol{\theta}, \partial\boldsymbol{\mu}_k, \partial\boldsymbol{\sigma}_k \leftarrow \nabla_{\boldsymbol{\theta}, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k} \frac{1}{n^b} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Phi}}, \underline{\boldsymbol{\Phi}}^{\mathrm{old}}\right) + \frac{1}{K} \log \prod_{k=1}^{K} \Gamma(\boldsymbol{\sigma}_k^{-1}; a, b)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \cdot \partial\boldsymbol{\theta}$
        $\boldsymbol{\mu}_k \leftarrow \boldsymbol{\mu}_k + \eta \cdot \partial\boldsymbol{\mu}_k, \forall k$
        $\boldsymbol{\sigma}_k \leftarrow \boldsymbol{\sigma}_k + \eta \cdot \partial\boldsymbol{\sigma}_k, \forall k$
    **end for**
    **if** $(t \mod t_{\mathrm{EM}}) = 0$ **then**
        $\pi_k \leftarrow \mathbb{E}_{P_d}\left[\underline{\gamma}_k(\underline{\boldsymbol{\Phi}}^{\mathrm{old}})\right]$
        $\underline{\boldsymbol{\Phi}}^{\mathrm{old}} \leftarrow \underline{\boldsymbol{\Phi}}$
    **end if**
**end for**
```

where $\boldsymbol{f}$ is the inverse of $\boldsymbol{g}$. Similar to section 6.8, update of prior $\boldsymbol{\pi}$ follows $\pi_k \leftarrow \mathbb{E}_{P_d}[\underline{\gamma}_k(\boldsymbol{x})]$. However, we need to consider the following issue when learning the parameters of Gaussian mixture source $p(\boldsymbol{z}) = \sum_{k=1}^{K} \pi p_k(\boldsymbol{z})$. If a component of the mixture source overfits and collapses onto a data sample, the likelihood can be large but the parameter learning can be problematic. This problem is known as the singularity problem of Gaussian mixture [7]. We avoid this problem by using the following alternatives:

oitemsep, nolistsep Assume that for each $\forall k = 1, 2, \cdots, K$, there is a parameter prior distribution for $\boldsymbol{C}_k = \mathrm{diag}(\boldsymbol{\sigma}_k^2)$. To be specific, assume that the parameter prior distribution of the precision $\boldsymbol{\sigma}_k^{-1}$ is $\Gamma(\boldsymbol{\sigma}_k^{-1}; a, b)$, where $\Gamma(\cdot; a, b)$ is Gamma distribution with parameter a and b. Then, the objective function of the optimization problem w.r.t. $\boldsymbol{\Phi}$ is reformulated as

$$\max_{\underline{\boldsymbol{\Phi}}} \frac{1}{n} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Phi}}, \underline{\boldsymbol{\Phi}}^{\mathrm{old}}\right) + \frac{1}{K} \log \prod_{k=1}^{K} \Gamma(\boldsymbol{\sigma}_k^{-1}; a, b). \tag{6.27}$$

oitemsep, nolistsep Alternatively, we use an l_2 regularization on $\boldsymbol{\sigma}_k$, which formulates the optimization step as

$$\max_{\underline{\boldsymbol{\Phi}}} \frac{1}{n} \underline{\mathcal{Q}} \left(\underline{\boldsymbol{\Phi}}, \underline{\boldsymbol{\Phi}}^{\text{old}} \right) - \lambda \sum_{k=1}^{K} \frac{(1-\boldsymbol{\sigma}_k)^2}{K}, \tag{6.28}$$

where λ is the regulation parameter.

On complexity of models and new variant models

We have proposed two models, GenMM and LatMM. GenMM has a high complexity whereas LatMM has a low complexity. Due to their difference in model complexity as well as training complexity, their usage efficiency is highly application-dependent. For example, when the training data is limited, it may be advisable to use LatMM.

It is possible to combine GenMM and LatMM to obtain new models. A simple way is to replace the latent source $p(\boldsymbol{z})$ of GenMM by a LatMM model. This new combined model has a higher complexity than both GenMM and LatMM.

To get a less complex model than GenMM, another new model can be derived by modifying the architecture of LatMM. There are multiple latent sources $p_k(\boldsymbol{z})$, $k = 1, 2, \cdots, K$, in LatMM. If we assume that each such latent source $p_k(\boldsymbol{z})$ is induced by a latent generator network, we can obtain a new model that has a common-and-individual architecture. Each latent generator of its corresponding latent source has its own parameters and acts as an individual part. The common part of the new model transforms signal between observable signal $\boldsymbol{x}$ and latent signal $\boldsymbol{z}$ (generated by the latent generator networks). The common-and-individual technique is prevalently used in machine learning systems [**?**, **?**].

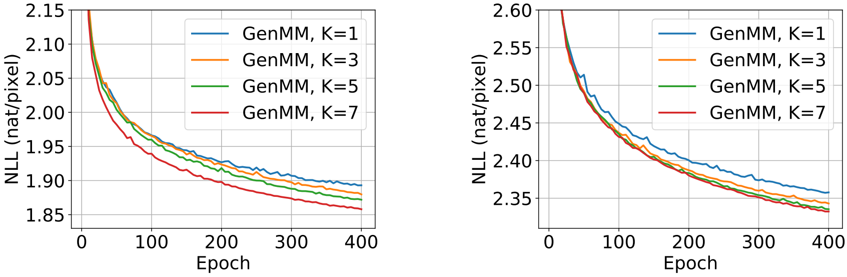
Therefore several new models can be derived using our proposed models, GenMM and LatMM. In spite of the scope and potential, development of analytical methodology to derive new model architectures turns out to be challenging. Traditionally the development is trial-and-error driven. Development of new model architectures by combining GenMM and LatMM will be investigated in future, and not to be pursued in this article.

6.10 Experiments Results

In this section, we evaluate our proposed mixture models for generating samples and maximum likelihood classification. We will show encouraging results.

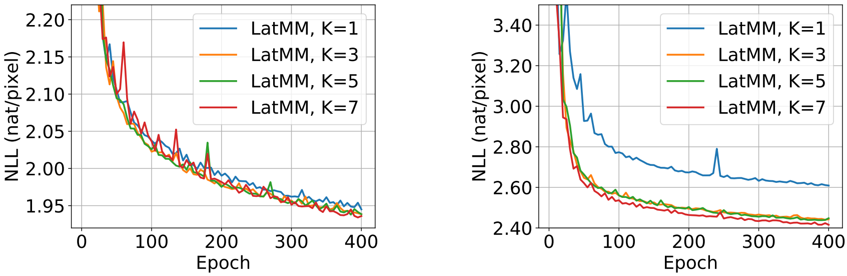
Experimental setup

We use the flow-based neural network for implementing generators $\{\boldsymbol{g}_k\}_{k=1}^{K}$ in GenMM and $\boldsymbol{g}$ in LatMM. Specifically, we use the Glow structure [41] that is developed based on RealNVP [15] and NICE [16]. As introduced in section 6.8, the operation in Equation 6.10 is a coupling layer. Since only a part of the input



(a) Dataset MNIST (b) Dataset Fashion-MNIST

Figure 6.3: NLL (Unit: nat/pixel) of GenMM versus training epochs with different number of mixture component k. (a) 10000 images from MNIST is used for training, (b) 10000 images from Fashion-MNIST is used for training.



(a) Dataset of MNIST (b) Dataset Fashion-MNIST

Figure 6.4: NLL (Unit: nat/pixel) of LatMM versus training epochs with different number of mixture component k. (a) 10000 images from MNIST is used for training, (b) 10000 images from Fashion-MNIST is used for training.

is mapped non-linearly after a coupling layer and the rest part remains the same, permutation [15] or 1×1 convolution operation [41] is used to alternate the part of signal that goes through identity mapping. In Glow structure, a basic *flow step* is the concatenation of three layers: Actnorm (element-wise affine mapping) → 1×1 Convolution (for permutation purpose) → Coupling layer. A *flow block* consists of: a squeeze layer, several flow steps, a split layer. A squeeze layer reshapes signal. A split layer allows flow model to split some elements of hidden layers out and model them directly as standard Gaussian, which relieves computation burden. In our experiments, there are also split layers that make dimension of $\boldsymbol{z}$ one fourth of dimension $\boldsymbol{x}$, and split signal in hidden layers are modeled as standard Gaussian.

All generators used in our experiments are randomly initialized before training. In addition, the prior distribution $\boldsymbol{\pi}$ update in both GenMM and LatMM is every 5 epochs, i.e., $t_\pi = 5$. For the training of LatMM, we adopt the Gamma distribution $\Gamma(\boldsymbol{\sigma}_k^{-1}; a, b)$ as the parameter prior for $\boldsymbol{\sigma}_k^{-1}, \forall k$, with shape parameter $a = 2$ and rate parameter $b = 1$. Our models are implemented using Pytorch and experiments

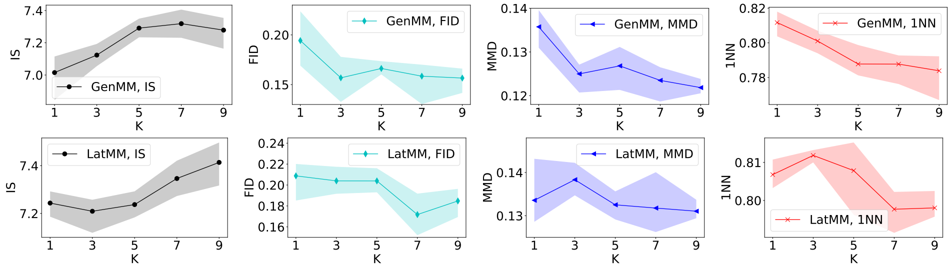


Figure 6.5: IS, FID, MMD and 1NN of GenMM and LatMM for MNIST dataset. GenMM and LatMM are trained on 60000 images of MNIST. The results are evaluated on 2000 samples per simulation point (1000 samples generated by GenMM or LatMM for corresponding K, 1000 samples from MNIST). 5 experiments are carried out for each assessed score at each setting of K. Curve with marker denotes mean score and shaded area denotes the range of corresponding score.

are carried out on Tesla P100 GPU. Code is available at github repository[1].

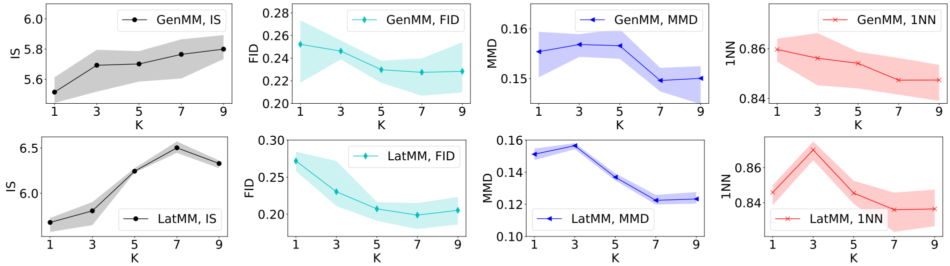


Figure 6.6: IS, FID, MMD and 1NN of GenMM and LatMM for Fashion-MNIST dataset. GenMM and LatMM are trained on 60000 images of Fashion-MNIST. The results are evaluated on 2000 samples per simulation point (1000 samples generated by GenMM or LatMM for corresponding K, 1000 samples from Fashion-MNIST). 5 experiments are carried out for each assessed score at each setting of K. Curve with marker denotes mean score and shaded area denotes the range of corresponding score.

Evaluation of Proposed Models

In order to see if the proposed algorithms of GenMM and LatMM help to improve probability distribution modeling capacity, we assess our proposed algorithms with varying number of mixtures (K). Since our models are explicit models, the negative log likelihood (NLL) is used for comparison of our models. Apart from NLL, another four different metrics are used in assessment of models. The metrics are Inception Score (IS) [6,80,99], Frechet Inception Distance (FID) [33], Maximum Mean

[1] https://github.com/FirstHandScientist/EM-GM

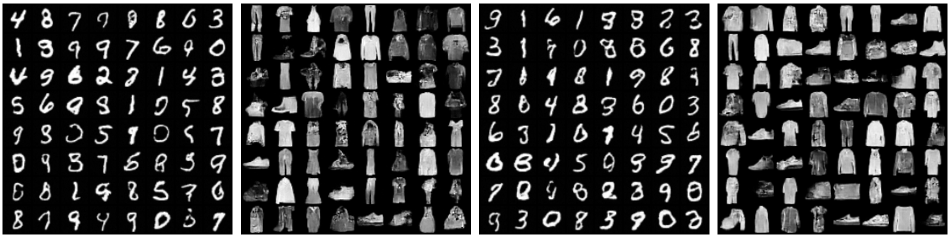
Table 6.1: The lowest NLL value of GenMM for curves in Figure 6.3 (nat/pixel).

Dataset	K=1	K=3	K=5	K=7
MNIST	1.8929	1.8797	1.8719	1.8579
FashionMNIST	2.3571	2.3429	2.3353	2.3323

Discrepancy (MMD) [99] and two-sample test based 1-Nearest Neighbor (1NN) score [57]. IS measures statistically if a given sample can be recognized by a classifier with high confidence. A high IS stands for high quality for generated samples. FID measures a divergence between two distributions under testing by assuming these two distribution are both Gaussian. We also use MMD with Gaussian kernel to test how dissimilar two distributions are. Small values of FID and MMD mean that the mixture distribution model is close to the underlying distribution of dataset. 1NN score measures if two given distributions are empirically close by computing 1NN accuracy on samples from two distributions under testing. The closer 1NN score is to 0.5, the more likely two distributions under testing are the same. Therefore, a high IS is good, low FID and MMD scores, and 1NN score close to 0.5 are good. We use the evaluation framework of [99] to compute these metrics scores, where we train a ResNet on datasets MNIST and Fashion-MNIST, respectively, as the feature extractor for evaluation of the four performance metrics.

The NLL curves of GenMM and LatMM models during model training phase are shown in Figure 6.3 and Figure 6.4, respectively. Subsets of MNIST and Fashion-MNIST are used to train our mixture models in order to assess their performance w.r.t. NLL when different number of mixture components K is used. All the curves in Figure 6.3 and Figure 6.4 show that NLL decreases as training epoch number increases in general. There is fluctuation of these decreasing NLL curves due to: (a) the iteration of E-step and M-step of EM, and (b) the use of batch-size gradient in optimization at M-step. In each figure of Figure 6.3 and Figure 6.4, NLL curve corresponding to larger total number of mixture components, K, reaches smaller NLL value after traning for same number of epochs. The results are consistent since as K increases, both GenMM and LatMM have smaller NLL. These results confirm our hypothesis that mixture models fit real data better. The lowest NLL values of curves in Figure 6.3 in training GenMM models are reported in Table 6.1.

As for the scores of IS, FID, MMD, and 1NN, we increase K for the proposed models and check how the four metrics vary. We do several trials of evaluation and report the results. The results are shown in Figure 6.5 for MNIST datset and Figure 6.6 for Fashio-MNIST dataset. Let us first address the results in Figure 6.5. It can be observed that IS increases with number of mixtures K. The IS improvement shows a saturation and decreasing trend for GenMM when $K = 9$. The FID, MMD and 1NN scores show a decreasing trend with increase in K. Their trends also saturate with increase in K. The trends obey a statistical knowledge that performance improves with increase in the model complexity, and then deteriorates if the model complexity continues to increase. As that in Figure 6.5, similar trends are also observed in Figure 6.6. In some cases, performance for $K = 3$ is poorer than



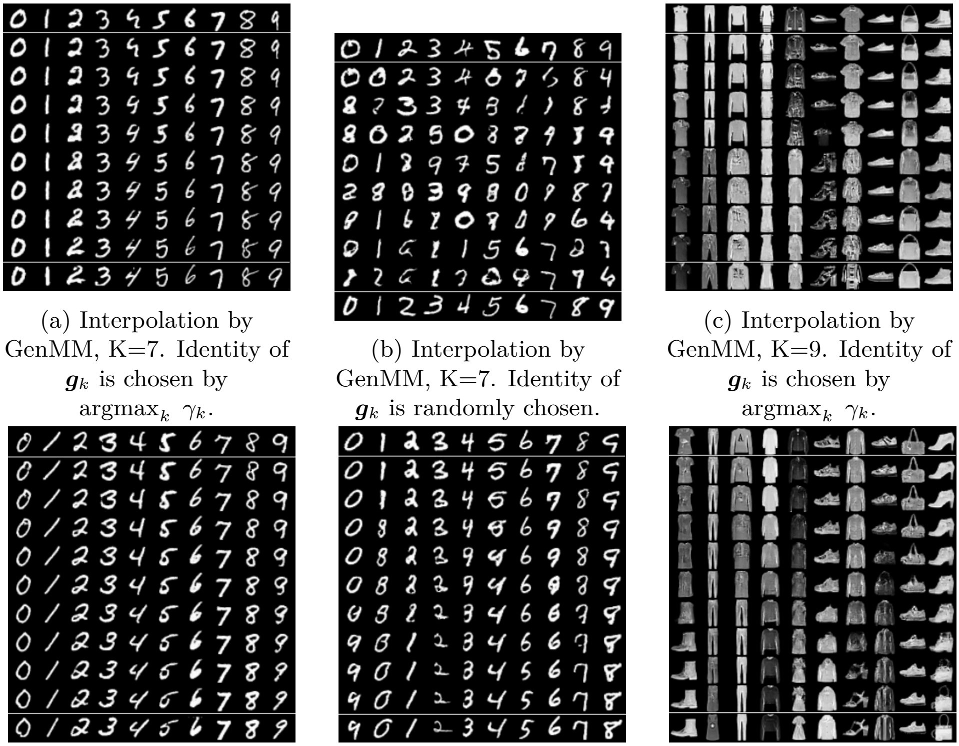
(a) Generated Samples. (GenMM, K=7)

(b) Generated samples. (GenMM, K=3)

(c) Generated samples. (LatMM, K=3)

(d) Generated samples. (LatMM, K=7)

Figure 6.7: Generated samples by GenMM and LatMM for MNIST and Fashion-MNIST datasets.



(a) Interpolation by GenMM, K=7. Identity of $\boldsymbol{g}_k$ is chosen by $\text{argmax}_k \ \gamma_k$.

(b) Interpolation by GenMM, K=7. Identity of $\boldsymbol{g}_k$ is randomly chosen.

(c) Interpolation by GenMM, K=9. Identity of $\boldsymbol{g}_k$ is chosen by $\text{argmax}_k \ \gamma_k$.

(d) Interpolation by LatMM, K=9.

(e) Interpolation by LatMM, K=9.

(f) Interpolation by LatMM, K=9.

Figure 6.8: Interpolation in latent space to generate samples . First and last rows are real samples from MNIST. For each row, images are generated by interpolating latent variables of empirical images in first and last rows.

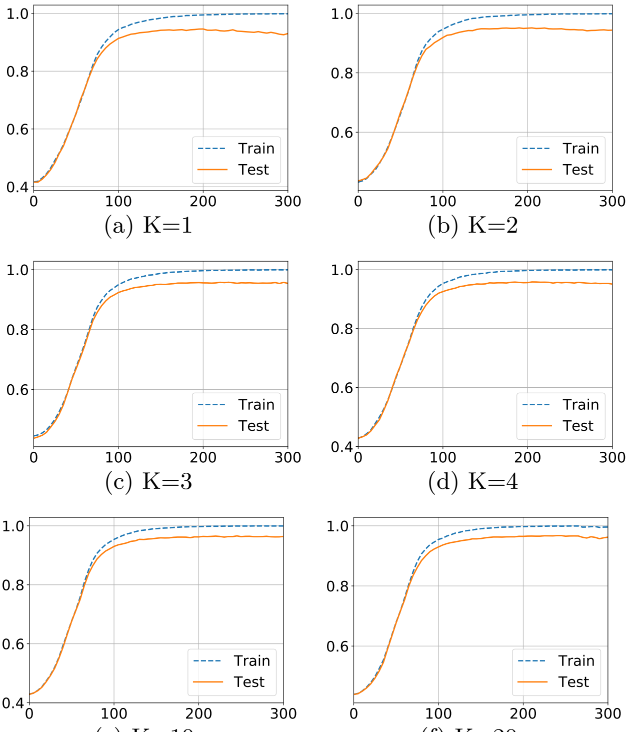
$K = 1$. We assume that the random initialization of parameters in mixture models

Table 6.2: Test Accuracy Table of GenMM for Classification Task

Dataset	K=1	K=2	K=3	K=4	K=10	K=20	State Of Art
Letter	0.9459	0.9513	0.9578	0.9581	0.9657	**0.9674**	0.9582 [?]
Satimage	0.8900	0.8975	0.9045	0.9085	0.9105	**0.9160**	0.9090 [?]
Norb	0.9184	0.9257	0.9406	0.9459	0.9538	**0.9542**	0.8920 [?]

has a high influence in this regard. Considering the trends in all the scores for both the figures, we can conclude that GenMM and LatMM can model the underlying distributions of data and the mixture models are good.

Sample Generating and Interpolation



(a) K=1 (b) K=2

(c) K=3 (d) K=4

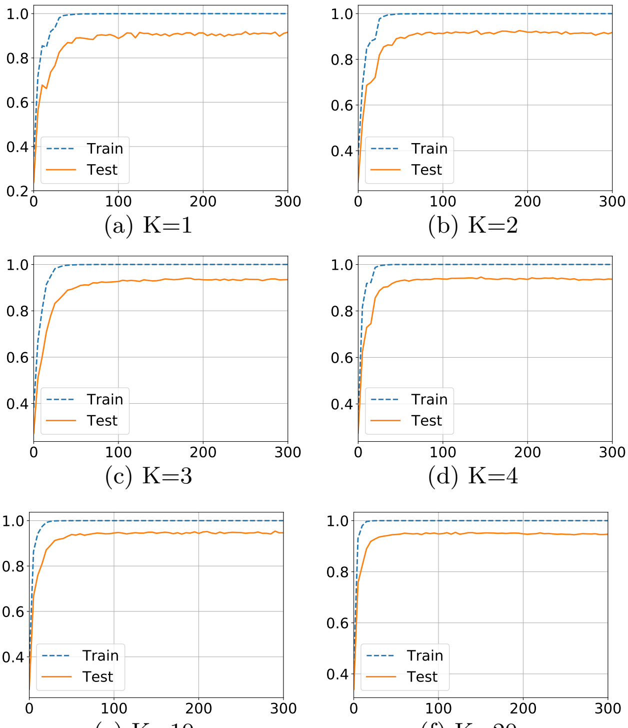
(e) K=10 (f) K=20

Figure 6.9: Train and Test Accuracy Curves versus Epochs on Dataset Letter.

Next we show generated samples from the proposed models trained with MNIST and Fashion-MNIST in Figure 6.7. In the figure, we show generated samples from GenMM and LatMM for MNIST and Fashion-MNIST datasets. We use different value of K to generate images. It can be observed that LatMM is able to produce good quality image samples as GenMM. While we argue that LatMM has a lower level of complexity than GenMM, it is seen that LatMM works good in practice.

In the second experiment, we explore power of invertibility for interpolation in the latent domain. We use samples from MNIST and Fashion-MNIST datasets for this 'interpolation' experiment. In Figure 6.8, we have six subfigures. For each subfigure, the first row and the last row are comprised of the real (true) data samples from MNIST and Fashion-MNIST dataset. In each column, we find latent codes corresponding to the real samples of the first row and the last row, $\boldsymbol{z}_1, \boldsymbol{z}_2$. This is possible as the neural networks are invertible. Then, we perform a convex combination of the two latent codes as $\alpha \boldsymbol{z}_1 + (1-\alpha)\boldsymbol{z}_2$, where $0 < \alpha < 1$. The latent code produced by the convex combination is used to generate a new sample using the trained models. All other rows except the first and the last rows of the figure are the generated samples by varying α. In Figure 6.8, we observe the change visually from the first row to last row - how the first row slowly changes to the last row. We use GenMM for Figure 6.8a, Figure 6.8b, Figure 6.8c, and LatMM for Figure 6.8d, Figure 6.8e, Figure 6.8f. Interpolation experiment for LatMM is easier than GenMM. GenMM has a set of neural network generators $\{\boldsymbol{g}_k(\boldsymbol{z})\}_{k=1}^{K}$ and a fixed Gaussian distribution for latent variable $\boldsymbol{z}$. We compute γ_k for a real image $\boldsymbol{x}$, and then find the latent code $\boldsymbol{z}$ of $\boldsymbol{x}$ using $\boldsymbol{g}_{k^*}^{-1}(\boldsymbol{x}) = \boldsymbol{f}_{k^*}(\boldsymbol{x})$, where $k^* = \arg\max_k \gamma_k$. For two real images (one image is in the first row and the second image in the last row), we find the corresponding latent codes, compute their convex combination as interpolation, and then pass the computed latent code through a generator $\boldsymbol{g}_k(\boldsymbol{z})$ to produce a generated sample $\boldsymbol{x}$. Identity of the generator of GenMM is chosen as k^* corresponding to the image of the first row if $\alpha < 0.5$, or to the image of the last row if $\alpha \geqslant 0.5$.

The second experiment on interpolation shows interesting result for modeling multi-modal data. The distribution of ten digits together in MNIST dataset is expected to be multi-modal. The aspect of multi-modal distribution is addressed using the experimental result shown in Figure 6.8b. We use similar experimental steps as that in Figure 6.8a but with modifications. It is evident that the generated digit images do not correspond well to the real images of the first row and the last row. For example, in the first column of Figure 6.8b, we observe presence of digits two and eight, while we expect that the column should be comprised of only images of digit zero. Natural question is why interpolation leads to generation of digits that are unexpected. The answer lies in the procedure of performing our experiment. The key difference for this experiment compared to the experiment in Figure 6.8a is that a sample is produced by a randomly selected generator $\boldsymbol{g}_k(\boldsymbol{z})$ from K possible choices. We compute interpolated latent code using the same procedure as that in Figure 6.8a, but use the generator where its identity k is randomly sampled from the prior $\boldsymbol{\pi}$ directly. The generated images in this interpolation experiment reveals a clue that each generator models a subset of the whole training dataset. We can qualitatively argue that use of multiple generators helps for modeling the multi-modal distribution.



(a) K=1 (b) K=2

(c) K=3 (d) K=4

(e) K=10 (f) K=20

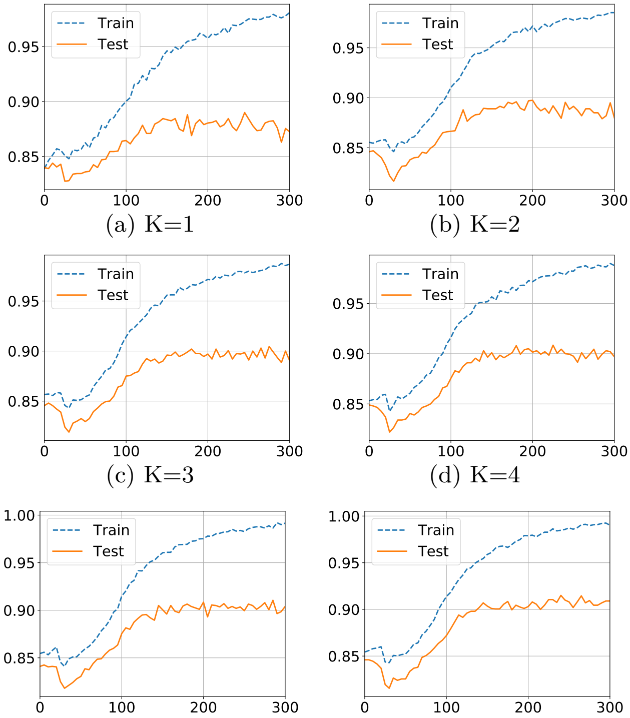
Figure 6.10: Train and Test Accuracy Curves versus Epochs on Dataset Norb

Application to Classification Task

In this subsection, we apply our proposed mixture models to classification tasks using the maximum likelihood criterion. We compare classification performance with the state-of-art results. The state-of-art results are produced by discriminative learning approaches. The major advantage of maximum likelihood based classification is that any new class can be accommodated on-the-fly. On the contrary a discriminative learning approach requires retraining whenever new classes appear.

For a given dataset with Y classes, we divide the dataset by sample labels and each subset has the same label y. Then we train one GenMM model per class of data, i.e. $p(\boldsymbol{x};\boldsymbol{\Phi}_y)$ is trained with the y-th class's data. After we have all $p(\boldsymbol{x};\boldsymbol{\Phi}_y)$, $\forall y = 1, 2, \cdots, Y$ trained, a new sample $\boldsymbol{x}$ is predicted by $\text{argmax}_y \, p(\boldsymbol{x};\boldsymbol{\Phi}_y)$.

The maximum likelihood based classification experiment as described above is carried out in three different datasets: Letter, Satimage, and Norb. For each dataset, we train our models for 300 epoches on the training data of the corresponding dataset, and the test accuracy is reported in Table 6.2. The state-of-art accuracy of each dataset in literature is also listed in this table for comparison. For each dataset, we increase the total number of mixture components K and the neural network generators have the same structure. The table shows that the



(a) K=1 (b) K=2

(c) K=3 (d) K=4

(e) K=10 (f) K=20

Figure 6.11: Train and Test Accuracy Curves versus Epochs on Dataset Satimage.

classification accuracy on each dataset is increased as we increase the number of generators in GenMM. When K is 10 or 20, maximum likelihood based classification by GenMM outperforms the state-of-art accuracy. The state-of-art accuracy results are obtained by using discriminative learning approaches. For dataset Norb, more significant performance gain is observed. Our classification accuracy is boosted from 0.9184 to 0.9542 when K is increased from 1 to 20 and a large improvement margin is obtained over reference accuracy. We also test LatMM on classification task, but its accuracy is more or less around the accuracy of GenMM with $K = 1$. Note that LatMM is a relatively low-complexity model than GenMM.

Figure 6.9 Figure 6.11 and Figure 6.10 show the train and test accuracy changing along with the training epoch on dataset Letter and Satimage, respectively. For each dataset, the accuracy curves versus epoch trained with GenMM at different value of K are shown. In these sets of figures, all accuracy curves climbs and flattens around some value, as training epoch increases. Train accuracy is either coincident with, or above test accuracy curve at different training phases. For each set of figures on a given dataset, the gap between train and test curve is smaller as a larger number of mixture components is used. As K increases, test curve flattens at a larger accuracy value. This again speaks for validation of our proposed models

and also the advantage of using our mixture models for practical tasks.

6.11 Conclusion

We conclude that the principal of expectation maximization can be used for neural network based probability distribution modeling. Our approach leads to explicit distribution modeling and the experimental results show an important aspect that the normal statistical behaviour of modeling performance versus model complexity remains valid. The proposed models are able to generate images which have good visual quality. This is also supported by several metric scores. Practical applications of our models for classification tasks are also carried out. The results confirm that our approach is good for modeling multi-modal distributions. Further extensions using variational inference for learning parameters of mixture models will be studied in the future.

Chapter 7

Powering Hidden Markov Model by Neural Network based Generative Models

For permeable part and notation of this chapter, refer to [45, Chapter 6.2]. Give a figure/illustration: Dynamic Bayesian Network > 2-TBN > HMM

A bit history of HMM, see [45, Chapter 6.8]

content:

1. Powering Hidden Markov Model by Neural Network based Generative Models, ECAI 2020

2. Antoine Honore, Dong Liu, Hidden Markov Models for sepsis detection in preterm infants, ICASSP, 2020

HMM is an instance of 2-time-slice Bayesian network(2-TBN) (section 6.2.2 Koller). Also, it can be argued from CRF.

7.1 Hidden Markov Model

7.2 GenHMM

7.3 Application to phone recognition

7.4 Application to sepsis detection in preterm infants

7.5 Summary

Chapter 8

An implicit probabilistic generative model

content: Entropy-regularized Optimal Transport Generative Models, ICASSP 2019

8.1 Modeling data without explicit probabilistic distribution

8.2 Employing EOT for modeling

8.3 Experimental results

8.4 Summary

Part III

Epilogue

Chapter 9

Conclusion and Discussions

Bibliography

[1] Semih Akbayrak and Bert de Vries. Reparameterization gradient message passing. In *EUSIPCO 2019 - 27th European Signal Processing Conference*, United States, 9 2019. Institute of Electrical and Electronics Engineers.

[2] Shun-Ichi Amari. Differential geometry of curved exponential families-curvatures and information loss. *Ann. Statist.*, 10(2):357–385, 06 1982. URL `https://doi.org/10.1214/aos/1176345779`.

[3] M. Arjovsky and L. Bottou. Towards Principled Methods for Training Generative Adversarial Networks. *ArXiv e-prints*, January 2017.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[5] Duhyeon Bang and Hyunjung Shim. Improved training of generative adversarial networks using representative features. *CoRR*, abs/1801.09195, 2018.

[6] S. Barratt and R. Sharma. A Note on the Inception Score. *ArXiv e-prints*, January 2018.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[8] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf`.

[9] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Annals of Mathematical Statistics*, 23: 409–507, 1952.

[10] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2292–2300. Curran Associates, Inc., 2013.

[11] J. CÃ©spedes, P. M. Olmos, M. SÃ¡nchez-FernÃ¡ndez, and F. Perez-Cruz. Expectation propagation detection for high-order high-dimensional mimo systems. *IEEE Transactions on Communications*, 62(8):2840–2849, Aug 2014. ISSN 0090-6778.

[12] Gustavo Deco and Wilfried Brauer. Higher order statistical decorrelation without information loss. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 247–254. MIT Press, 1995. URL `http://papers.nips.cc/paper/901-higher-order-statistical-decorrelation-without-information-loss.pdf`.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977. URL `http://web.mit.edu/6.435/www/Dempster77.pdf`.

[14] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[15] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *ArXiv e-prints*, May 2016.

[16] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. *CoRR*, abs/1410.8516, 2014.

[17] Jeff Donahue, Philipp KrÃ¤henbÃ¼hl, and Trevor Darrell. Adversarial feature learning. In *International Conference on Learning Representations*, 2017.

[18] Jian Du, Shaodan Ma, Yik-Chung Wu, Soummya Kar, and José M. F. Moura. Convergence analysis of distributed inference with vector-valued gaussian belief propagation. *J. Mach. Learn. Res.*, 18(1):6302–6339, January 2017. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=3122009.3242029`.

[19] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarial learned inference. In *International Conference on Learning Representations*, 2017.

[20] Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[21] N. Friel, A. N. Pettitt, R. Reeves, and E. Wit. Bayesian inference in hidden markov random fields for binary data defined on large lattices. *Journal of Computational and Graphical Statistics*, 18(2):243–261, 2009. ISSN 10618600. URL `http://www.jstor.org/stable/25651244`.

[22] Andrew E. Gelfand and Max Welling. Generalized belief propagation on tree robust structured region graphs. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI'12, pages 296–305, Arlington, Virginia, USA, 2012. AUAI Press. ISBN 9780974903989.

[23] Arnab Ghosh, Viveka Kulharia, Vinay P. Namboodiri, Philip H. S. Torr, and Puneet Kumar Dokania. Multi-agent diverse generative adversarial networks. *CoRR*, abs/1704.02906, 2017.

[24] Soumya Ghosh, Francesco Delle Fave, and Jonathan Yedidia. Assumed density filtering methods for learning bayesian neural networks, 2016. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12391`.

[25] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[26] J. Goldberger and A. Leshem. Pseudo prior belief propagation for densely connected discrete graphs. In *2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo)*, pages 1–5, Jan 2010.

[27] I. Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *ArXiv e-prints*, December 2017.

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[29] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[30] Aditya Grover and Stefano Ermon. Boosted generative models. *CoRR*, abs/1702.08484, 2017.

[31] Nicolas Heess, Daniel Tarlow, and John Winn. Learning to pass expectation propagation messages. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3219–3227, Red Hook, NY, USA, 2013. Curran Associates Inc.

[32] Tom Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Comput.*, 16(11):2379–2413, November 2004. ISSN 0899-7667. URL `https://doi.org/10.1162/0899766041941943`.

[33] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, *et al.* Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017.

[34] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018.

[35] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

[36] C. Jeon, R. Ghods, A. Maleki, and C. Studer. Optimality of large mimo detection via approximate message passing. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 1227–1231, June 2015.

[37] Wittawat Jitkrittum, Arthur Gretton, Nicolas Heess, S. M. Ali Eslami, Balaji Lakshminarayanan, Dino Sejdinovic, and Zoltán Szabó. Kernel-based just-in-time learning for passing expectation propagation messages. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pages 405–414, 2015.

[38] Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2946–2954. Curran Associates, Inc., 2016.

[39] M. Khayatkhoei, A. Elgammal, and M. Singh. Disconnected Manifold Learning for Generative Adversarial Networks. *ArXiv e-prints*, June 2018.

[40] Ryoichi Kikuchi. A theory of cooperative phenomena. *Phys. Rev.*, 81:988–1003, Mar 1951.

[41] D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *ArXiv e-prints*, July 2018.

[42] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[43] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and TrendsÂ® in Machine Learning*, 12(4):307–392, 2019. ISSN 1935-8237. URL `http://dx.doi.org/10.1561/2200000056`.

[44] Frederic Koehler. Fast convergence of belief propagation to global optima: Beyond correlation decay. In *Advances in Neural Information Processing Systems 32*, pages 8329–8339. Curran Associates, Inc., 2019.

[45] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning.* The MIT Press, 2009. ISBN 0262013193.

[46] F. R. Kschischang, B. J. Frey, and H. . Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2): 498–519, Feb 2001. ISSN 0018-9448.

[47] Volodymyr Kuleshov and Stefano Ermon. Neural variational inference and learning in undirected graphical models. *CoRR*, abs/1711.02679, 2017. URL `http://arxiv.org/abs/1711.02679`.

[48] Volodymyr Kuleshov and Stefano Ermon. Neural variational inference and learning in undirected graphical models. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6737–6746, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[49] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. URL `https://doi.org/10.1214/aoms/1177729694`.

[50] Solomon Kullback. *Information Theory and Statistics.* Wiley, New York, 1959.

[51] Miguel Lazaro-Gredilla, Wolfgang Lehrach, and Dileep George. Learning undirected models via query training, 2019.

[52] C. Ledig, L. Theis, F. HuszÃ¡r, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, July 2017.

[53] Chongxuan LI, Max Welling, Jun Zhu, and Bo Zhang. Graphical generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6069–6080. Curran Associates, Inc., 2018.

[54] Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Stochastic expectation propagation. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2323–2331. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5760-stochastic-expectation-propagation.pdf`.

[55] Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 361–369, Cambridge, MA, USA, 2015. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2969239.2969280`.

[56] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[57] D. Lopez-Paz and M. Oquab. Revisiting classifier two-sample tests. *ArXiv e-prints*, October 2016.

[58] You Lu, Zhiyuan Liu, and Bert Huang. Block belief propagation for parameter learning in markov random fields. *CoRR*, abs/1811.04064, 2018. URL `http://arxiv.org/abs/1811.04064`.

[59] Dmitry M. Malioutov, Jason K. Johnson, and Alan S. Willsky. Walk-sums and belief propagation in gaussian graphical models. *J. Mach. Learn. Res.*, 7:2031–2064, December 2006. ISSN 1532-4435.

[60] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. *ArXiv e-prints*, January 2017.

[61] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1. URL `http://dl.acm.org/citation.cfm?id=647235.720257`.

[62] Thomas P. Minka. *A Family of Algorithms for Approximate Bayesian Inference.* PhD thesis, Cambridge, MA, USA, 2001. AAI0803033.

[63] Tom Minka. Divergence measures and message passing. Technical Report MSR-TR-2005-173, January 2005. URL `https://www.microsoft.com/en-us/research/publication/divergence-measures-and-message-passing/`.

[64] Grégoire Montavon, Klaus-Robert Müller, and Marco Cuturi. Wasserstein training of restricted boltzmann machines. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3718–3726. Curran Associates, Inc., 2016.

[65] G. Montufar. Restricted Boltzmann Machines: Introduction and Review. *ArXiv e-prints*, June 2018.

[66] J. M. Mooij and H. J. Kappen. Sufficient conditions for convergence of the sum-product algorithm. *IEEE Transactions on Information Theory*, 53(12): 4422–4437, Dec 2007. ISSN 1557-9654.

[67] Joris M. Mooij and Hilbert J. Kappen. Sufficient conditions for convergence of loopy belief propagation. *CoRR*, abs/1207.1405, 2012. URL `http://arxiv.org/abs/1207.1405`.

[68] Tohru Morita. Cluster Variation Method for Non-Uniform Ising and Heisenberg Models and Spin-Pair Correlation Function. *Progress of Theoretical Physics*, 85(2):243–255, 02 1991. ISSN 0033-068X.

[69] Ronald C. Neath. On convergence properties of the monte carlo em algorithm, 2012.

[70] N. Noorshams and M. J. Wainwright. Stochastic belief propagation: A low-complexity alternative to the sum-product algorithm. *IEEE Transactions on Information Theory*, 59(4):1981–2000, April 2013.

[71] M. Opper and D. Saad. *Advanced Mean Field Methods: Theory and Practice*. Neural information processing series. MIT Press, 2001. ISBN 9780262150545.

[72] Manfred Opper. *A Bayesian Approach to Online Learning.* Cambridge University Press, USA, 1999.

[73] Manfred Opper and Ole Winther. Gaussian processes for classification: Mean-field algorithms. *Neural Comput.*, 12(11):2655–2684, November 2000. ISSN 0899-7667. URL `http://dx.doi.org/10.1162/089976600300014881`.

[74] J Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, 1986. ISSN 0004-3702.

[75] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, pages 133–136. AAAI Press, 1982.

[76] Marco Pretti. A message-passing algorithm with damping. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11):P11008–P11008, nov 2005.

[77] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *International Conference on Machine Learning*, pages 5241–5250, 2019.

[78] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561, Berkeley, Calif., 1961. University of California Press. URL `https://projecteuclid.org/euclid.bsmsp/1200512181`.

[79] T. G. Roosta, M. J. Wainwright, and S. S. Sastry. Convergence analysis of reweighted sum-product algorithms. *IEEE Transactions on Signal Processing*, 56(9):4293–4305, Sep. 2008. ISSN 1053-587X.

[80] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, *et al.* Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[81] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving GANs using optimal transport. In *International Conference on Learning Representations*, 2018.

[82] F. Santambrogio. *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling.* Progress in Nonlinear Differential Equations and Their Applications. Springer International Publishing, 2015. ISBN 9783319208282. URL `https://books.google.se/books?id=UOHHCgAAQBAJ`.

[83] Vivek Srikumar, Gourab Kundu, and Dan Roth. On amortizing inference cost for structured prediction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1114–1124, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[84] Charles A. Sutton and Andrew McCallum. Piecewise training for undirected models. *CoRR*, abs/1207.1409, 2012. URL `http://arxiv.org/abs/1207.1409`.

[85] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein Auto-Encoders. *ArXiv e-prints*, November 2017.

[86] I. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf. AdaGAN: Boosting Generative Models. *ArXiv e-prints*, January 2017.

[87] Dustin Tran, Rajesh Ranganath, and David Blei. Hierarchical implicit models and likelihood-free variational inference. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5523–5533. Curran Associates, Inc., 2017.

[88] Dustin Tran, Rajesh Ranganath, and David Blei. Hierarchical implicit models and likelihood-free variational inference. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5523–5533. Curran Associates, Inc., 2017.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[90] M. J. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. now, 2008. URL `https://ieeexplore.ieee.org/document/8187302`.

[91] Martin J. Wainwright. Estimating the "wrong" graphical model: Benefits in the computation-limited setting. *J. Mach. Learn. Res.*, 7:1829–1859, 2006. URL `http://jmlr.org/papers/v7/wainwright06a.html`.

[92] Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Comput.*, 12(1):1–41, January 2000. ISSN 0899-7667. URL `https://doi.org/10.1162/089976600300015880`.

[93] Max Welling. On the choice of regions for generalized belief propagation. *CoRR*, abs/1207.4158, 2012.

[94] Max Welling, Tom Minka, and Yee Whye Teh. Structured region graphs: Morphing ep into gbp. In *UAI*, January 2005.

[95] Max Welling and Yee Whye Teh. Belief optimization for binary networks: A stable alternative to loopy belief propagation. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI'01, pages 554–561, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

[96] Wim Wiegerinck and Tom Heskes. Fractional belief propagation. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 438–445, Cambridge, MA, USA, 2002. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2968618.2968673`.

[97] Sam Wiseman and Yoon Kim. Amortized bethe free energy minimization for learning mrfs. In *Advances in Neural Information Processing Systems 32*, pages 15520–15531. Curran Associates, Inc., 2019.

[98] Eric P. Xing. Learning in fully observed markov networks. Technical report, 2014. URL `https://www.cs.cmu.edu/~epxing/Class/10708-14/scribe_notes/scribe_note_lecture8.pdf`.

[99] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *ArXiv e-prints*, June 2018.

[100] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, July 2005. ISSN 1557-9654.

[101] Jonathan Yedidia, William Freeman, and Yair Weiss. *Understanding belief propagation and its generalizations*, volume 8, pages 239–269. 01 2003. ISBN 1558608117.

[102] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 668–674, Cambridge, MA, USA, 2000. MIT Press.

[103] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard S. Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. *CoRR*, abs/1803.07710, 2018.

[104] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-Attention Generative Adversarial Networks. *ArXiv e-prints*, May 2018.

[105] R. Zhang, C. A. Bouman, J. Thibault, and K. D. Sauer. Gaussian mixture markov random field for image denoising and reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 1089–1092, Dec 2013.

[106] Huaiyu Zhu and Richard Rohwer. Information geometric measurements of generalisation. Technical report, 1995.