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 ....
ISBN ....
SE-100 44 Stockholm
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.

© 2020 Dong Liu, unless otherwise noted.

Tryck: Universitetsservice US AB

To my beloved

Abstract

Probabilistic graphical models provide a natural framework for the representation of complex systems and offer straightforward abstraction for the interactions within the systems. Reasoning with probabilistic graphical models allows us to answer inference queries with uncertainty following the framework of probabilistic theory. General inference queries include the computation of marginal probabilities or conditional probabilities of states of a system, or evaluation of the partition function of the underlining distribution of a Markov random field (undirected graphical model). Critically, the success of graphical models in practice largely relies on efficient approximate inference methods that offer fast and accurate reasoning results. Closely related to the inference tasks in graphical models, another fundamental problem is to decide the parameters of a candidate graphical model by extracting information of empirical observations, i.e., parameter learning of a graphical model. The two essential topics (i.e., inference and learning) interact with and facilitate each other. For instance, learning of a graphical model usually uses an inference method as a subroutine while the learned graphical model is then employed for inference tasks in the presence of new evidence.

In this dissertation, we develop new algorithms and models for generic inference in Markov random fields. We firstly present an alternative view of belief propagation in terms of a divergence minimization, which is in contrast to the view of free energy minimization. The alternative view brings the development of a variant algorithm of belief propagation that turns out to generalizes the standard belief propagation algorithm. Insights on the convergence behaviors of the developed algorithm in the binary state space are provided apart from the intuition in development. As a step beyond approximate inference with message passing, we develop a region-based energy network model that performs generic inference via region-based free energy minimization, which turns inference in Markov random fields into an optimization problem. This model incorporates both our essential understanding of inference and computational efficiency of modern neural network models.

The further part of the dissertation focus on the parameter learning of probabilistic graphical models. This part starts with the discussion on parameter learning of undirected graphical models and explains the role of an (approximate) inference method in this routine. As directed graphical models, new finite mixture models incorporating normalizing flows in neural network implementations are presented

for more expressive and flexible modeling. The learning of developed generic models is handled within expectation maximization due to the presence of hidden (or latent) variables. The expressive modeling method and learning are further extended into dynamic systems within a reduced dynamic Bayesian network, i.e., a hidden Markov model. The dissertation closes with a chapter on the likelihood-free learning of a class of directed graphical models, where (directed) generative models induce implicit probabilistic distributions and are learned by the optimal transport distance.

Sammanfattning

Acknowledgements

Contents

Chapter 1

Introduction

1.1 Motivations

Most tasks conducted by a person or an automated system require 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 an 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 *confidently* the diagnose is made.

Instead of randomly guessing, reasoning aims at answering queries with preserving uncertainty by making the best of the available information. Two fundamental problems are inevitable before rational reasoning can be conducted.

- How should we specify the relationship between a conclusion and the available information? In the coronavirus example, the corresponding question to answer is how the doctor should relate the coronavirus infection to 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 for answering queries.

- With the modeled representation, how should a conclusion be drawn? This process of reaching an 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. Then, how likely is it that the candidate is infected with the coronavirus?

In the modeling and inference process, it is not likely that very early 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 on the representation of a problem, and come back to revise them later when more information or evidence is available. This is aligned with our process of learning 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 about the correctness of the assumptions of the model, one can consider a set of models under 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 instantiation of the configuration generates a model representation. By using available observations or information, the model is adjusted to best match 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 hypotheses) 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, i.e., modeling, inference, and learning, our goal is to carry out reasoning while being aware of how confident a conclusion or answer is. These problems can be treated nicely with probabilistic models via Bayesian logic. A probabilistic model is built on the fundamental calculus of probability theory that provides a convenient framework for addressing *uncertainty*, which is desired in reasoning. In addition, probabilistic models offer rich space for modeling problems, where inference can be carried out either exactly or approximately. **More importantly, the modeling or model learning part is not necessarily coupled with inference algorithms**. This proper separation provides the freedom that a certain family of general inference algorithms can be applied to a broad class of probabilistic models. It offers the freedom of trying different model representations of a class without the need for replacing the inference algorithm.

Example 1.1. *Consider the coronavirus infection problem. Using a probabilistic model, we are able to model the problem in a rigid way. Additionally, we can formalize a query in a probabilistic model framework. Assume each symptom among fever, cough, and breathing difficulty can take a 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 to be infected by the coronavirus if symptoms of both fever and breathing difficulty are observed but no cough symptoms.

We have argued that probabilistic theory offers a rigid foundation to model and study the problems, which can be used to answer queries. Unfortunately,

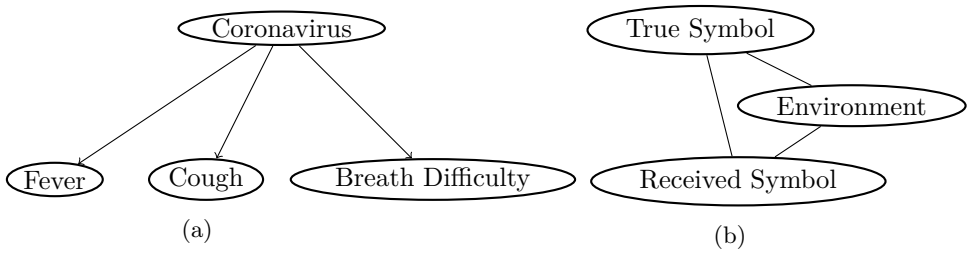


Figure 1.1: Different perspectives on probabilistic graphic models. 1.1a A directed graphical model (Bayesian network) for Example 1.1. 1.1b A toy undirected graphical model (Markov random field).

it soon becomes intractable when dozens or hundreds of relevant attributes are jointly considered. This can be exemplified by giving finer levels of each symptom in the 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 these symptoms, a recent travel itinerary may also be relevant. Additionally, season flu could also similarly bring up some symptoms listed above.

A 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 shown in Figure 1.1a that represents the problem of Example 1.1 into a directed graphical model (also called Bayesian network or generative model), the nodes (or vertexes) correspond to the variables that represent 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 an observable variable is depending 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 contrast to directed graphical models, there are scenarios for which the interaction between related random variables is not directional and an undirected edge is used, which leads to the undirected graphical model representation (or Markov random field, see Section 2.1). 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 jointly 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, e.g., in a sensor network where the environment triggers the

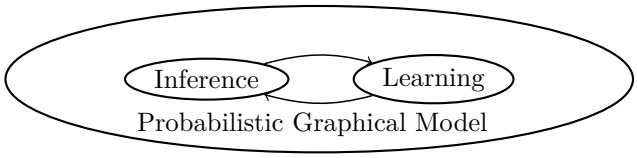


Figure 1.2: Two key aspects in practical graphical models.

sensor to send a message at the same time it affects the communication channel.

A probabilistic graphical model offers a 'scientific language' to do reasoning with uncertainty within the framework of probabilistic theory. It is usually a natural representation of a complex system or problem and offers straightforward abstraction. A probabilistic graphical model bridges the graphical representation of the complex system and the statistic dependency of attributes of the system. The advantage of its representation power is one of the reasons that lead to its popularity in different disciplines.

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

1.2 Scope and the Dissertation Outline

We have given the intuition and motivation of probabilistic graphical models in the last section, and pointed out the interaction between inference and learning in this framework. In this section, we would categorize the main topics within the framework firstly. Then, we would outline the dissertation content.

Inference in probabilistic graphical models 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 variables.
- Computing the marginal distribution over a particular subset of random variables.

- Computing the conditional distribution of 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 dissertation is mainly related to the first three cases mentioned above.

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 the case that the above inference problem can be solved exactly. Thus inference methods can be divided into

- Exact inference,
- 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 perspective of methodology, this dissertation is mainly related with approximate inference under the variational method category.*

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

- Structure learning,
- Parameter learning.

The first case refers to determining 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 dissertation. The term *learning* in the dissertation means the estimation of the parameters of the underlying distribution of a graphical model. This

problem is mainly discussed in Part III, where we would address learning in both undirected and directed graphical models.

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

- Maximum likelihood estimation
- Maximum conditional likelihood
- Bayesian estimation
- Pseudolikelihood or maximum-margin
- Maximum entropy

in general. *Maximum likelihood* is one of the most fundamental principles in model learning. As a variant of maximum likelihood, maximum conditional likelihood is a useful objective for particular applications. Although the full Bayesian learning is usually too complex in practices, using Bayesian priors to regulate the learning of probabilistic graphical models can avoid some issues encountered by pure maximum likelihood learning. Our work is related to the first three cases in Part III. At times, likelihood based learning of certain classes of graphical models is too expensive, which may motivate the choice of pseudolikelihood or maximum-margin that formulate learning objective as a function of (conditional) marginals. Maximum entropy has a close relation to the representation of graphical models in exponential family form. But it is out of the scope of this dissertation. Nevertheless, at the end of Part III, an alternative learning metric to maximum likelihood is introduced.

1.2.1 Publications

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

1. **Dong Liu**, Ragnar Thobaben, and Lars K. Rasmussen. Region-based energy neural network for approximate inference. Preprint, 2020.
   Code: https://github.com/FirstHandScientist/renn
2. **Dong Liu**, Minh Thành Vu, Li Zuxing, and Lars K. Rasmussen. α belief propagation for approximate Bayesian inference. Preprint, 2020.
   Code: https://github.com/FirstHandScientist/alpha-bp
3. Anubhab Ghosh, Antoine Honoré, **Dong Liu**, Gustav Eje Henter, and Saikat Chatterjee. Robust Classification using Hidden Markov Models and Mixtures of Normalizing Flows. Prepint, 2020.
4. Zuxing Li, Gyorgy Dan, and **Dong Liu**. A Game Theoretic Analysis of LQG Control under Adversarial Attack. Preprint, 2020.
5. Andrea Scotti, Nima N. Moghadam, **Dong Liu**, Karl Gafvert and Jinliang Huang. Graph Neural Networks for Massive MIMO Detection and Higher-Order QAM. Preprint, 2020.

6. **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

7. **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

8. 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

9. **Dong Liu**, Nima N. Moghadam, Lars K. Rasmussen, Jinliang Huang, and Saikat 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

10. **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

11. **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.

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

13. **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 dissertation is based on the author's works which are partially published in 1, 2, 6, 7, 8,9, 10 listed above.

1.2.2 Outline of the Dissertation

A general outline of the rest of the dissertation is as follows. Part I introduces the background knowledge and notations in support of the successive chapters. In what follows, the main ideas and techniques are present with two parts. Part II consisting of Chapter 3 and 4 focuses on inference techniques. Part III consisting of Chapter 5, 6, 7, and 8, covers mainly the learning topics. Chapter 9 discusses some future directions along this line of research and closes the dissertation. We summarize the content more specifically as follows.

The background part (Part I) first reviews the basics of probabilistic graphical models e.g., directed and undirected graphical models (also known as Bayesian network and Markov random fields), that are the fundamental frameworks for the content in this dissertation. Then, the inference tasks and basic message-passing algorithms (e.g., mean field and belief propagation) are reviewed. To give more intuition on approximate inference, the connection to variational free energy is discussed. The background part ends with the discussion on the essential learning principles of probabilistic graphical models.

Part II focuses on the development of inference in Markov random fields. Firstly, an alternative view of belief propagation is developed in contract to free energy minimization in Chapter 3. A variant belief propagation algorithm is developed based on the alternative understanding. The convergence conditions of the proposed algorithm are studied and offered in the binary-state case of each variable node in pairwise Markov random fields. Secondly, in Chapter 4, this part then takes the path of addressing inference approximately by optimizing the region-based free energy which generalizes the Bethe free energy corresponding to the belief propagation algorithm. The developed method incorporates deep neural networks in solving the optimization problem of region-based free energy.

Part III moves the focus to learning of probabilistic graphical models. This part begins with the explanation of the role of inference in addressing learning problems of probabilistic graphical models in Chapter 5 and shows numerically the impact of different inference algorithms on the learning of Markov random fields. The rest content of this part is mainly on model learning with partial observations, i.e., there is the presence of hidden variables, which is addressed in expectation maximization framework. Chapter 6 explores the more expressive probabilistic graphical models, i.e., finite mixtures of models with normalizing flows that are likelihood-tractable, and learning of these models. Chapter 7 extends the idea of graphical models with flows of Chapter 6 into a special case of dynamic Bayesian network, i.e., the hidden Markov model. The model learning is addressed with expectation maximization and applications of this model is demonstrated. With the idea that more general nonlinear mapping can induce more expressive probabilistic distributions, Chapter 8 studies the learning of likelihood-free generative models with a different metric, i.e., optimal transport. The learning of this kind of model is developed in the way of both direct optimal transport distance minimization and generative adversarial training.

Chapter 9 concludes the dissertation with closing remarks.

Part I

General Background

Chapter 2

Background

In this chapter, we review some background knowledge that is going to be used in this dissertation. 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 dependencies 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 local message interactions of a graphical model in exact or approximate ways, which are usually infeasible to solve directly.

More formally, a graphical model is a graphical representation of a collection of random variables (along with their domains) where their statistical dependencies are 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 necessarily 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 distribution 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 is also known as a Bayesian network or generative model in the literature [14, Chapter 8]. We might use the names alternatively. The non-negative functions in graphical models encode 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 is represented as the product of these conditional probability functions,

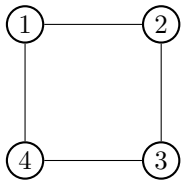
$$p(\boldsymbol{x}) = \prod_{n=1}^{N} p(x_n|\mathcal{P}(x_n)), \tag{2.1}$$

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 $\{p(x_n|\mathcal{P}(x_n))\}$, are normalized and proper distributions. The directed graph of a directed graphical model or a Bayesian network is required to be an *acyclic* graph. That is to say, if starting from a node 1 (associated with x_1) and following a directed path $x_1 \to \cdots \to x_n$ along the directed edges of the graph, there is no path such that $n = 1$. 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 interpret 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. However, there are practical scenarios where interaction between two variables can not be naturally described by impact with directionality, which may bring the difficulty of deciding the direction of an edge between them. Also, there are cases where a Bayesian network can not encode all independence constraints of a distribution, which could be due to that certain structures are not appropriated by Bayesian networks (see [92, seciton 3.4.2]). Therefore, an alternative representation method can be an undirected graphical model, i.e., a Markov random field (MRF). Under certain conditions, a Bayesian network can be perfectly represented by a Markov random field without loss of independence information by moralizing edges [92, Section 4.5] [14, Section 8.3.4]. 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 MRFs as follows.

Example 2.1. *As shown in Figure 2.1, the MRF encodes the dependencies of four random variables x_1, x_2, x_3, and x_4, where node i is associated with variable x_i and each has a binary domain, i.e., $\mathcal{X}_i = \{0, 1\}$ for $i = 1, 2, 3, 4$. Four potential factors of the MRF together define the joint distribution*

$$p(\boldsymbol{x}) = \frac{1}{Z}\varphi_{1,2}(x_1, x_2)\varphi_{2,3}(x_2, x_3)\varphi_{1,4}(x_1, x_4)\varphi_{3,4}(x_3, x_4)$$



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

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

...

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

where $Z = \sum_{x_1,x_2,x_3,x_4} \varphi_{1,2}(x_1, x_2)\varphi_{2,3}(x_2, x_3)\varphi_{1,4}(x_1, x_4)\varphi_{3,4}(x_3, x_4)$ *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*. Obviously, the partition function normalizes the potential factors such that $p(\boldsymbol{x}; \boldsymbol{\theta})$ is a proper probability.

Remark 2.1 (On set $\mathcal{F}$ of potential factors)**.** *In our definition in* (2.2)*, the set of potential factors, i.e.,* $\mathcal{F}$*, is not required to be the set of all maximal cliques of the graph* $\mathcal{G}$*. A clique of the graph* $\mathcal{G}$ *is a subset of nodes in* $\mathcal{G}$*, within which any two nodes are connected by an edge of* $\mathcal{G}$*. A maximal clique is a clique such that it is not possible to include any other nodes from* $\mathcal{G}$ *into the clique without it ceasing to be a clique. Note there is literature that defines MRF over maximal cliques [14, Section 8.3.2]. Actually, we can always reformulate* (2.2) *into factorization over maximal cliques by redefining a maximal clique as the product of potential factors (in* $\mathcal{F}$*) that are subsets of the maximal clique. Nevertheless, the definition in* (2.2) *without maximal clique requirement allows more flexibility of factorization and finer specification of potential factors. These properties may also benefit computations.*

Remark 2.2 (Comparison of directed and undirected graphical models)**.** *We can compare directed and undirected graphical models with regard to the following aspects.*

- *Representation: The structure and the parameterization in directed graphical models provide a natural representation for many types of real-world domains. For domains where interactions are naturally directional, directional graphical models can best manifest the abstraction and encode the dependencies. On the contrary, for domains where interactions are symmetrical without certain directionality, MRFs would be natural choices since forcing a directionality in these cases may cause an incomplete representation of dependencies. Furthermore, MRFs can be either cyclic or acyclic, which allows intuitive graphical representation with weaker constraint and gives the flexibility of graph structures.*

- *Local nonnegative functions: The local functions are conditional probability functions in directed graphical models. In undirected graphical models, the local functions are potential factors that allow more flexible forms.*

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

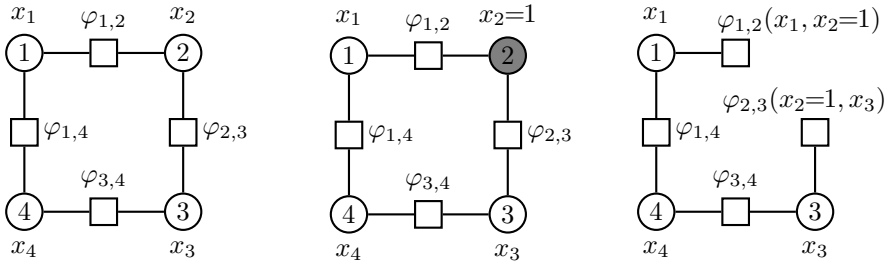
- *Normalization: Since each local function is a conditional probability function in directed graphical models, the partition function for normalization is not needed. An MRF, in general, comes with a partition function, since potential factors are not necessarily normalized.*

Remark 2.3 (Graphical models with both directed and undirected edges)**.** *In the above discussions, we have presented both directed and undirected graphical models. In literature, a single graphical model may also encode both directed and undirected dependencies, i.e., there are both directed and undirected edges in its graph. Perhaps the most well-known models in this track are the conditional random fields (CRFs). See [163] for more introduction.*

2.1.1 An Alternative Representation

The previous graphical representations do not explicitly include the conditional probability functions or potential factors in their graphs. An alternative representation to a graphical model (undirected or directed) is *factor graph* [94], which is a bipartite graph topology. In a factor graph, a potential factor or a conditional probability funciton is explicitly represented as a factor node, as the counterpart of the variable node associated with a random variable. We focus on the correspondence between a MRF and its factor graph here.

Definition 2.1. *A factor graph $\mathcal{G}_F$, is a bipartite graph that represents the factorization of* (2.2)*. A factor graph has two types of nodes: i) a variable node for*



(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 in (a). Conditioning of the MRF in (b). The reduced MRF in (c).

each variable x_i; ii) a factor node for each potential function φ_a. An edge is placed between a variable node i and factor node a if and only if x_i is argument of φ_a. We 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 2.2. *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.*

2.1.2 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 $\boldsymbol{x}_U$. When an evidence is observed, its likelihood can be computed as

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

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 the likelihood of an (partial) evidence can be computed by partition function and sub-partition functions. As shall be seen in Part III, this likelihood function, or its appropriations would be used for model learning.

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 proportional 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 2.1 is shown in Figure 2.2b, where node 2 is clamped to evidence $x_2 = 1$.

In addition to the above intuitions, conditioning can also be understood as a process of reducing the graph of an MRF. When an 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 regard to $\mathcal{V}_U$ are modified accordingly [92, 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 an MRF applies to the MRF with nodes clamped as well. An 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 the MRF framework is capable of handling evidence and conditioning as well. We would come back to this topic in Part III.

2.2 Divergence

Before we get into further discussions about inference and learning, we first introduce the concept of *divergence* measures since principles of both learning and inference are closed related to divergence measures. A divergence measure plays 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 2.2. *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* [96,97], KL divergence for short, which is one of the most widely used divergence measures in machine learning, statistics and information theory.

Definition 2.3. *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. A symmetric divergence that generalizes the KL divergence would be introduced in Section 3.1. Furthermore, an extra metric that is used to compare distributions, i.e., optimal transport, would be introduced in Section 8.1.

2.3 Inference Tasks

Given a probability distribution $p(\boldsymbol{x})$ as the underlying distribution of a graphical model, inference in general can be divided into four kinds of tasks, as explained in Chapter 1.2. Our work in this dissertation is 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 a 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 closely related with the inference of the partition function, i.e.,

- 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 introduce the methods for the above tasks at a high-level.

2.4 Exact Inference

In this section, we briefly treat the exact inference of graphical models. Please see [14, 92, 186, 187] for detailed discussions. In order to compute a marginal probability, we need to sum or integrate the joint probability distribution over some variables. This operation can be performed as a sequence of operations by a predefined ordering of the variables. Since the underline distribution of a graphic model (directed or undirected) is a product of factors, the sequence or order can be chosen in a way that facilitates the sum or integral operations. Thus the computation of the marginal probability is up to choose a specific ordering of remaining variables to eliminate (sum or integrate out) repeatedly. This brings the very fundamental exact inference method, *variable elimination*.

When we want to ask multiple queries (marginal probabilities) on a graphical model, we repetitively call the variable elimination method for each query until all

queries are answered. The *sum-product* algorithm is more efficient for this case. It is a *message passing* algorithm that operates on graphical models. More importantly, the intermediate results, the messages, are shared in all computations. Thus one call of this message passing algorithm returns answers to multiple queries. The sum-product algorithm, also known as *belief propagation*, has an iterative message update rule. A message update step from a factor a to its neighboring variable node i is

$$m_{a\to i}(x_i) \propto \sum_{\boldsymbol{x}_a \setminus x_i} \varphi_a(\boldsymbol{x}_a) \prod_{j\in a\setminus i} \prod_{a'\in \mathrm{ne}_j\setminus a} m_{a'\to j}(x_j), \tag{2.8}$$

where $a', a \in \mathcal{F}$ and $\mathrm{ne}_i = \{a | i \in a, a \in \mathcal{F}\}$ denotes the neighboring factors of the variable node i. The left-hand-side of $\propto$ is the updated message and the right-hand-side of $\propto$ is a function of factors and old messages in (2.8). The marginal for x_i is simply $p(x_i) \propto \prod_{a\in \mathrm{ne}_i} m_{a\to i}(x_i)$ after all messages have been updated accordingly. For a tree-structured graph, the message passing rule in (2.8) with pre-defined ordering of sequential updates gives exact marginals.

However, when a graphical model is not tree-structured, the above methods can not give exact inference. If exact inference is still desired here, one may consider converting the graph of the graphical model into a clique tree, an acyclic graph whose nodes are maximal cliques, and apply the *junction tree* algorithm.

It is known that the computational complexity of the junction tree method grows exponentially in the size of the maximal clique. Therefore, for certain classes of graphs, exact inference is feasible by using the methods mentioned above. On the other hand, there are many graphical models where exact inference is too expensive in computation or simply prohibitive. We need to turn to approximate inference to cope with such models.

2.5 Variational Inference

In solving inference tasks with approximation, 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* [131] is a quantity used to find such

a approximation. The variational free energy is defined by

$$\begin{aligned} F_V(b) &= \mathrm{KL}(b(\boldsymbol{x})||p(\boldsymbol{x};\boldsymbol{\theta})) - \log Z(\boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \frac{b(\boldsymbol{x})}{p(\boldsymbol{x};\boldsymbol{\theta})} - \log Z(\boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \frac{b(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x};\boldsymbol{\theta})}, \end{aligned} \tag{2.9}$$

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 2.4 (Interpretation to variational free energy)**.** *Note from 2.9, the minimization with regard to 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 2.9 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 2.5 (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 2.6, this type of divergence measure is what we need in model learning.*

2.5.1 Variational Free Energy and Mean Field

Mean field method, similar to the sum-product algorithm in Section 2.4, is an iterative message passing approach. The message update rule of mean field algorithm is described as follow.

$$\log b_i(x_i) \propto \sum_{a\in \mathrm{ne}_i} \sum_{\boldsymbol{x}_a \backslash x_i} \log \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}_a) \prod_{j\in a\backslash i} b_j(x_j). \tag{2.10}$$

The algorithm does iterative update of the belief on each node of the graphical model. An update step for node i use its neightbors' old beliefs $\{b_j(x_j), j \in a\backslash i, a \in \mathrm{ne}_i\}$ and factors to assign new belief $b_i(x_i)$. After the iterative algorithm converges, the approximation to marginal $p(x_i)$ is $b_i(x_i)$.

The intuition of mean field algorithm can be given by the variational free energy defined in Section 2.5. In fact, mean field approach uses the fully-factorized distribution,

$$b_{MF}(\boldsymbol{x}) = \prod_{i=1}^{N} b_i(x_i), \tag{2.11}$$

to approximate the joint distraction $p(\boldsymbol{x};\boldsymbol{\theta})$. The update rule (2.10) can be obtained by minimizing the 'difference' between the fully-factorized distribution and $p(\boldsymbol{x};\boldsymbol{\theta})$, i.e., the variational free energy (2.9). Specifically, substituting (2.11) into the variational free energy (2.9) gives

$$F_{MF}(b) = -\sum_{a\in\mathcal{F}}\sum_{\boldsymbol{x}_a} \log \varphi_a(\boldsymbol{x}_a;\boldsymbol{\theta}) \prod_{i\in a} b_i(x_i) + \sum_{i\in\mathcal{V}}\sum_{x_i} b_i(x_i)\log b_i(x_i). \tag{2.12}$$

Minimizing $F_{MF}(b)$ over $\{b_i\}$ would give us the mean field update rule (2.10).

2.5.2 Bethe Free Energy and (Loopy) Belief Propagation

We described the sum-product algorithm (belief propagation) for exact inference in tree-structured graphs in Section 2.4. For graphical models with the presence of cycles or loops, we can still apply the sum-product algorithm as if there was no loop. This is known as the loopy belief propagation (loopy BP). Of course, the inference results are no longer exact anymore. But loopy BP still works well in many practical cases.

Now we give the intuition of loopy BP from the perspective of free energy minimization, which comes with the name of Bethe approximation in literature. 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 $\{b_i(x_i)\}$ [187]. In this case, the Bethe free energy is given by

$$F_{Bethe}(b) = \sum_{a}\sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a)\log\frac{b_a(\boldsymbol{x}_a)}{\varphi_a(\boldsymbol{x}_a)} - \sum_{i=1}^{N}(|\mathrm{ne}_i|-1)\sum_{x_i} b_i(x_i)\log b_i(x_i), \tag{2.13}$$

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

$$\begin{aligned}
\min_{\{b_a(\boldsymbol{x}_a)\},\{b_i(x_i)\}} \quad & F_{Bethe}(b) \\
\text{s.t.} \quad & \sum_{\boldsymbol{x}_a\setminus x_i} b_a(\boldsymbol{x}_a) = b_i(x_i), \\
& \sum_{\boldsymbol{x}_a} b_a(\boldsymbol{x}_a) = \sum_{x_i} b_i(x_i) = 1, \\
& 0 \leqslant b_i(x_i) \leqslant 1, \\
& i \in \mathcal{V}, a \in \mathcal{F},
\end{aligned} \tag{2.14}$$

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 2.1, gives the (loopy) BP message passing rule that has the same formula as in (2.8).

These variational approaches would be further discussed and compared in Chapter 3 and 4 of Part II.

2.6 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.)* according to $p^*(\boldsymbol{x})$. The task is then to use the information from the dataset to learn a distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ 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 $p(\boldsymbol{x};\boldsymbol{\theta})$ can be formulated as minimizing the KL divergence

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

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 depend 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.

Note although we can use the expected log-likelihood for the model learning task and for even model comparison (comparing a trained model with another one), we lose 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

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

The 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 (see Chapter 5 for more analyses) since

- it is often that exact computation of $p(\boldsymbol{x};\boldsymbol{\theta})$ is not possible; or
- there are some elements of $\boldsymbol{x}$ which are not observable (hidden or latent variables).

For the first case, the typical treatment is to approximate the exact log-likelihood. This is done via approximation by employing inference methods or making simplified assumptions on dependency structure of the graphical model of $p(\boldsymbol{x})$. Then, optimization is carried out with regard to the approximated log-likelihood. These methods include surrogate likelihood [113, 175], pseudo-likelihood [99, 141], piecewise likelihood [108, 164], saddle-point approximation [159, 182].

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 is not limited to) classic Gaussian mixture models (GMMs) and hidden Markov models (HMMs). There are latent variables because:

- Use of abstract variables 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, see e.g., Example 1.1. 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 fail from time to time.

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 anymore 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 widely used methods in learning with latent variable is *expectation maximization (EM)* [32]. In the 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. The detailed treatment can be found in Part III

In cases where the posterior of $\boldsymbol{x}_U$ is not feasible to compute, the variational EM [174, Section 6.2.2] or Monte Carlo EM (need sampling technique) [128] can be applied. There are also neural network based methods with Monte Carlo estimator to cope with the latent variable problems, see e.g., [55, 88, 95, 99].

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

In an alternative track, 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 [54]. A generator in a deep generative model induces a distribution that can either be an explicit or implicit distribution. The former problem dates back to [31] and receives more attention in recent years with latest work such as variational autoencoder [88], normalizing flows [36, 86] and neural ordinary difference equations [20]. The training of explicit models is based on the maximum likelihood principle or variational likelihood bound. Chapter 6 and 7 would give detailed treatment with modeling and learning with normalizing flows. Chapter 9 will further discuss the implicit deep generative models and their learning.

The latter case brings an implicit distribution, where the maximum likelihood principle is not applicable anymore. In this case, a state-of-art method is the generative adversarial model that employs a discriminator to play the role of divergence measure [7, 55, 154], 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 [27, 156] in this track, which has also been applied for training of Boltzmann machine [123] auto-encoders [167] and generative adversarial networks [8]. We would discuss this further in Chapter 9.

Part II

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 [14, 94]. In Section 2.5, we introduced the classic belief propagation as the minimization of free energy, instead of an iterative message-passing routine. Interesting to note, the message-passing rule of BP was developed as early as 1986 [135] and had been popularly used in different fields before the free energy optimization intuition was developed in literature [187].

BP can solve inference problems in linear-time exactly when graphs are loop-free or tree-structured [94]. The message-passing routine of BP can be boiled down into variable elimination in tree-structured graphs[1] and message scheduling, which corresponds to determining the variable elimination order. The message scheduling can be omitted and equivalent exact inference results can be obtained, when an alternative belief update algorithm is applied which is also known as Lauritzen-Spiegelhalter algorithm [92, Section 10.3]. BP and its variants are widely applied in large computation systems due to their 'magic' of reducing the exponential number of operations for inference with enumeration into linear complexity. This is possible because:

- An underline distribution of a graphical model is usually factorized, and each sub-expression (a factor) depends only on a small number of variables.

- The intermediate results are computed once and cached as messages, which are reused in coming computations.

[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).

Inevitably, many real-world signals are naturally modeled by graph representations with loops. Surprisingly, although lost its optimality guarantees in loopy graphs, loopy BP is still a practical method and gives reasonable good inference results by running it as if there ware no loops. 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 into BP in general graphs, we take a variational approach to develop an interpretable variant of BP, which is termed as α-BP. The intuition of α-BP starts with a surrogate distribution $q(\boldsymbol{x})$, which is 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 complete (fully-connected) graphs while still maintaining the 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 a 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 a 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 tests [22]. Around a decade later, Alfréd Rényi proposed his version of divergence as well [148]. In the 80s of last century, Amari extended Chernoff's version of α-divergence [5], which is now widely used in studies of 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 [192], the definition of α-divergence (Amari's version, with correction term to accommodate unnormalized measure) is as follows,

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

where α is the parameter of this divergence. Different from the KL divergence definition in Section 2.2, p and q here are not necessarily to be normalized measures.

In Section 2.2, the KL divergence was defined over two normalized measures. Here we extend that definition to a generalized case where p and q are not necessarily normalized, as an extended divergence from the normalized case. As shall be see, KL divergence is closely related with α-divergence. KL divergence for general measures is defined as

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

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

Remark 3.1. *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 the basic properties of divergence measures, both α-divergence and KL divergence are zero when $p = q$, and they are non-negative.

Denote the KL-projection by

$$\mathrm{proj}[p] = \operatorname*{argmin}_{q\in\mathcal{Q}} \mathrm{KL}(p\|q), \tag{3.3}$$

where $\mathcal{Q}$ is a family of distributions that q belongs to. According to the stationary point equivalence Theorem in [119], $\mathrm{proj}[p^\alpha q^{1-\alpha}]$ 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

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

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

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 $\mathcal{Q}$

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

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

Remark 3.2. *We take $\boldsymbol{x}$ to be discrete by default in this section. The sum operation should be replaced by integral for continuous variable $\boldsymbol{x}$.*

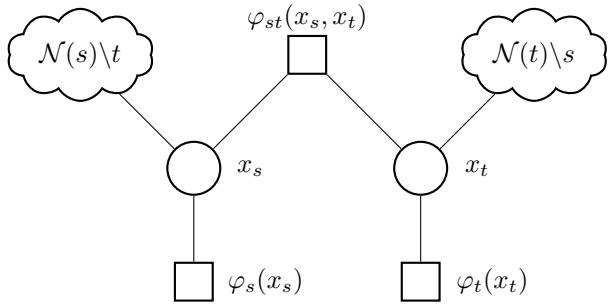


Figure 3.1: Factor graph illustration of $p(\boldsymbol{x})$ in (3.5).

3.2 α Belief Propagation Algorithm

3.2.1 Pairwise MRF

We consider a probability distribution over random vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$, where $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{X}} = \prod_{i=1}^{N} \mathcal{X}_i$. For explanation simplicity, we define that each domain is instantiated as the same discrete finite set $\mathcal{X}$, i.e., $\boldsymbol{\mathcal{X}} = \prod_{i=1}^{N} \mathcal{X}$ and $x_i \in \mathcal{X}$. Note α-BP is applicable for the different domain setting $\boldsymbol{\mathcal{X}} = \prod_{i=1}^{N} \mathcal{X}_i$. Let us denote the undirected 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 s and t 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 as 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 $\varphi_s : \mathcal{X} \to (0, \infty)$ and $\varphi_{st} : \mathcal{X} \times \mathcal{X} \to (0, \infty)$ are factor potentials. Here the normalization is omitted. Note the union of the node set and the 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.

3.2.2 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. $m_{ts} : \mathcal{X} \to (0, \infty)$, 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 in assumed density filtering algorithm [51, 132] and expectation propagation [117, 119]. Without loss of generality, we begin to refine the factor $\tilde{\varphi}_{ts}(x_t, x_s)$ via α-divergence with its α-parameter assignment as α_{ts}. Define $q^{\setminus(t,s)}(\boldsymbol{x})$ as the product of all other factors excluding $\tilde{\varphi}_{ts}(x_t, x_s)$

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

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

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

we consider the following tractable problem

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

which searches for new factor $\tilde{\varphi}_{ts}^{\text{new}}(x_t, x_s)$ such q can approximate p better. In (3.11), $\mathcal{D}_{\alpha_{ts}}(\cdot)$ denotes the α-divergence with the 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

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

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

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

Since KL-projection onto a fully factorized distribution reduces to matching the marginals [92, Proposition 8.3], substituting (3.13) into (3.12), we obtain

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

Solving (3.14) gives the message passing rule as

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

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

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

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 the 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

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

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, $\text{KL}(p\|q)$ 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

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

which is exactly the messages of standard BP [14]. 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 (2.13).

Note although the 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 are actually using information projections from $p(\boldsymbol{x})$ to a fully-factorized space via KL divergence as explained in Section 2.5. In addition, α-BP is different from standard BP with damping technique. The latter case uses a message update rule that differs from (3.18) slightly in the way of assigning updated messages.

Additionally, α-BP differs from the tree-reweighted belief propagation [174] in the way of message update rule and also how algorithm is derived. The tree-reweighted BP shares some similarity with α-BP in the formula of the message-passing rule, namely, the pairwise log-potential functions are scaled by a weight and reweighted old messages appear in the 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 (2.13) 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

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

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 a way that is different from tree-reweighted BP.

From the practical perspective, α-BP as a meta algorithm can be used with other methods in a hybrid way. Inspired by [53] and assembling methods [74], we can

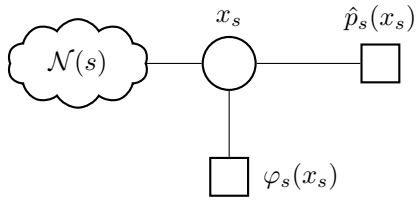


Figure 3.2: Modified graphical model with prior factor.

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.

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 3.1. *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*

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

Definition 3.1 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 $\mathcal{X}$ with $\mathcal{X} = \{-1, 1\}$, to gain insights into the convergence behavior of α-BP. 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

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

Example 3.1. *The Ising model, from statistical physics [73, 130], is such a model whose potentials fulfill the symmetric property in* (3.23)*. In the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of*

an Ising model, the random variable x_s takes the spin value $\{-1, +1\}$. The values of $\boldsymbol{x}$ associated with the nodes of $\mathcal{G}$ might represent the orientations of magnets in a field or the states of particles of gas in the context of statistical physics. Two elements x_t and x_s of $\boldsymbol{x}$ are allowed to interact directly only if there is an edge between them directly, i.e., $(t, s) \in \mathcal{E}$. The potential factors of an Ising model is in exponential family. This leads the joint distribution

$$p(\boldsymbol{x}; \boldsymbol{\theta}) \propto \exp\left\{\sum_{s\in\mathcal{V}} b_s x_s + \sum_{(t,s)\in\mathcal{E}} J_{ts} x_t x_s\right\},$$

the factors of which fulfill the symmetric property in (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 the 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 3.1. *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 $\mathcal{X}$, 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 denote 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 [152], which are given in the following 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

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

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

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

where $\boldsymbol{M}(\boldsymbol{\alpha}, \boldsymbol{\theta})$ is defined in (3.24), and $\leqslant$ in (3.35) denotes the element-wise inequality. Also note that in (3.35), the $|\cdot|$ operator on a vector is a bit abused and denotes the element-wise absolute operation on the vector. From (3.35), we could further have

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

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

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

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

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 $\boldsymbol{z}$, and we have

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

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

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

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 $\boldsymbol{z}_0^* = \boldsymbol{z}_1^*$ for $\lambda^*(\boldsymbol{M}) < 1$ and completes the uniqueness of the fixed point. □

Remark 3.3. *From Theorem 3.1 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 $\boldsymbol{\theta}$.*

Remark 3.4. *The condition in Theorem 3.1 tells if α-BP converges to a fixed point. However, it does not guarantee if the converged fixed point is a global optimum. Error analysis of BP and its variant is a challenging task. We discussed that the exact inference of BP in tree-structured graphs in Section 2.4. The global optimum of the loopy BP algorithm can be guaranteed in rare cares. For instance, a ferromagnetic Ising model in which neighbors prefer to be in aligned states can get guaranteed global optimum with loopy BP [91]. A successful application of (loopy) BP with bounded errors lies in coding theory, e.g., running a BP decoder on a graph whose structure (degrees of nodes) fulfills certain constraints can have its error bounded [150] [151, Section 7.2].*

Given the fact that α-BP would converge if the condition in Theorem 3.1 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 3.1. *α-BP converges 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 satisfied, 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

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

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 3.1. □

Remark 3.5 (Interpretation to the Results)**.** *Both Theorem 3.1 and Corollary 3.1 offer sufficient conditions for convergence of α-BP to a unique fixed point. The obtained results allow us to check if applying α-BP in a certain configuration guarantees convergence without running the algorithm practically to try out, which saves computation resources when problems are at scales. To test the fulfillment of the conditions, the first step it to convert a problem to be solved into its pairwise MRF representation $\mathcal{G}(\mathcal{V},\mathcal{E})$. Secondly, construct the matrix $\boldsymbol{M}$ according to* (3.24)*. Note the undirected edge set $\mathcal{E}$ of $\mathcal{G}$ is used to construct a directed edge set $\vec{\mathcal{E}}$ by assigning*

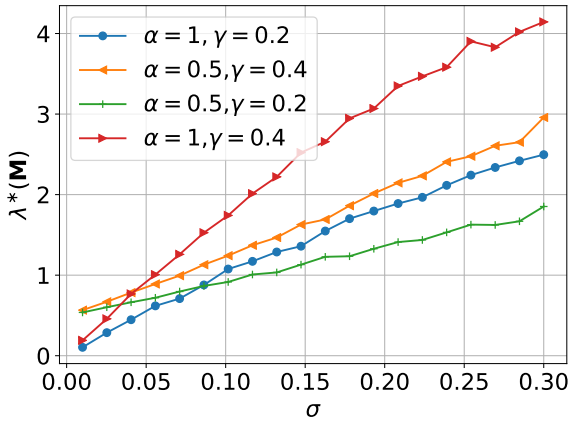


Figure 3.3: The largest singular value of $\boldsymbol{M}$ defined in (3.24) versus variance of potential parameter $\boldsymbol{\theta}$. A value of each curve is the mean of 100 graph realizations.

$(s \to t)$ and $(t \to s)$ with the same potential factor φ_{st} of $\mathcal{G}$. Thirdly, the condition fulfillment can be check by computing the largest singular value $\lambda^*(\boldsymbol{M})$ and testing $\lambda^*(\boldsymbol{M}) < 1$ as stated in Theorem 3.1. Fulfilling this condition is to say that α-BP is guaranteed to converge in the current setting.

The size of the constructed matrix $\boldsymbol{M}$ is $|\vec{\mathcal{E}}| \times |\vec{\mathcal{E}}|$. $\boldsymbol{M}$ could be pretty large if a graph $\mathcal{G}$ is large or dense, which might cast the computation issue of the largest singular value of this matrix. In this case, Corollary 3.1 offers the conditions to avoid the computation of the largest singular value. To be specific, after constructing the matrix (3.24)*, Corollary 3.1 allows column-wise tests of $\boldsymbol{M}$, i.e., the column value of matrix $\boldsymbol{M}$ returning the largest quantity according to* (3.43) *should be smaller than* 1 *to guarantee the convergence of α-BP. Note for a column $(u \to v)$, the elements of $\boldsymbol{M}$ involved in the testing computation are the factors corresponding to edges connected to node v in $\mathcal{G}$. Alternatively, row-wise test can be carried out according to* (3.44) *with similar complexity.*

3.5 Experiments

In this section, we first give simulation results for the convergence condition of α-BP explained in Theorem 3.1. Then the application of α-BP to a MIMO detection problem is demonstrated.

3.5.1 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 the Erdos-Rényi (ER) model [42]. In generating a graph by ER 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

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

with $\varphi_{ts}(x_t, x_s) = e^{-2J_{t,s}x_t x_s}$ and $\varphi_s(x_s) = e^{-J_{s,s}x_s^2 - b_s x_s}$. $\boldsymbol{J}$ 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).

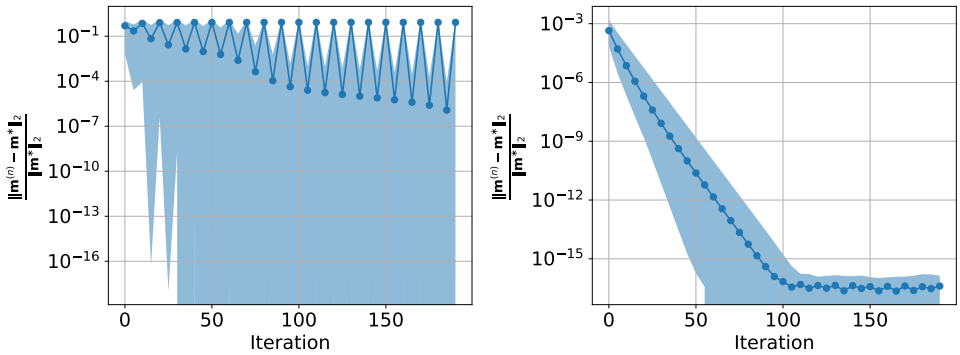
For this set of experiments, we set $\mathcal{X} = \{-1, 1\}$ and $N = 16$. To specify (3.49), the non-zero entries of $\boldsymbol{J}$ are sampled from a Gaussian distribution $\mathsf{N}(0, \sigma^2)$, i.e. $J_{ts} \sim \mathsf{N}(0, \sigma^2)$ if $J_{ts} \neq 0$. For entries of $\boldsymbol{b}$, we 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 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 the potential factors of the graph edges makes it more difficult to fulfill the convergence condition in Theorem 3.1. 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 the choice of α value in α-BP also makes a difference, and its effect depends on the graph itself. Additionally, α-BP with assignment $\alpha = 0.5$ leads to flatter curves compared to standard BP ($\alpha = 1$) and guarantees convergence for a larger range of σ, which corresponding a larger range of graphs. How to tune α value to fulfill the condition of Theorem 3.1 depends not only on 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 seen that this setting does not fulfill the condition in Theorem 3.1. 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.

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.

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



(a) (b)

Figure 3.4: Numerical illustration of convergence, with normalized error $\|\boldsymbol{m}^{(n)} - \boldsymbol{m}^*\|_2 / \|\boldsymbol{m}^*\|_2$ versus the number of iterations. Number of nodes $N = 16$. (a) Parameter setting: $\gamma = 0.4$, $\alpha = 1$ (equivalent to standard BP), $\sigma = 0.5$. (b) 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.

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 3.1. 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 3.1. The result in Figure 3.4b is consistent with our analysis on the convergence of α-BP.

3.5.2 Complete Graph Case: Application to MIMO Detection

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

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

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

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

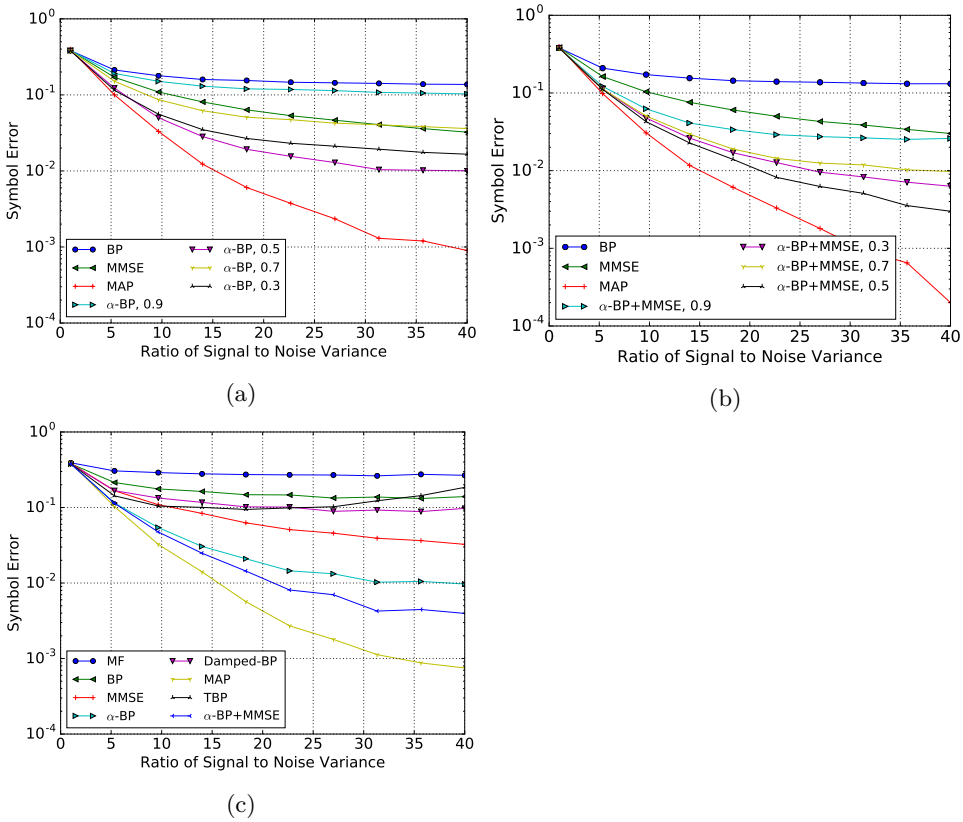


Figure 3.5: Numerical results of α-BP: symbol error of MIMO detection.

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

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

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

We test the application of α-BP to the MIMO signal detection numerically. We run the α-BP, without the prior trick (Section 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 $\operatorname{argmin}_{x_i\in\mathcal{X}} |x_i - \hat{\mu}_i|$.

Figure 3.5a shows that BP even underperforms MMSE but α-BP can outperform MMSE by assigning smaller value of α. Note that MMSE has a higher computation complexity since it requires the matrix inverse computation whose complexity is proportional to N^3. 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 Section 3.3. Figure 3.5b exemplifies this effects by using prior belief from MMSE, $\hat{p}_i(x_i) \propto \exp\{-(x_i - \hat{\mu}_i)^2/(2\hat{\mathbf{\Sigma}}_{i,i})\}$, 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 [139] (with legend 'Damped-BP'), and Tree-reweighted belief propagation [174] (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 studied belief propagation in the way of α-divergence minimization and proposed an alternative belief propagation method. 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 regard to both the high-level optimization objectives and the practical iterative update rules. All methods share a common interpretation, *inference an optimization*, i.e., inference methods could be recovered from optimization problems (free energy minimization or a divergence minimization). Although they started from different optimization cost functions, the derived iterative update rules share some similarities. The study enriches our insight into deterministic approximate inference approaches.

Given the wide application of the family of belief propagation methods, when the implementation of them can have guaranteed convergence is a fundamental question, which is of practical interest in general. This issue was addressed in the binary support case in this chapter via the method of contraction condition. The derived sufficient conditions for α-BP gives us sufficient conditions about whether we can expect it to converge. The sufficient conditions allow us to check if α-BP is guaranteed to converge without actually running the algorithm for a given problem.

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 is applicable to wide range of settings. The wide applications include but are not limited to imaging processing [191], multi-input-multi-output (MIMO) signal detection in digital communication [29, 75], inference on structured lattice [44], and machine learning [107, 124, 189]. The empirical success of belief propagation (BP) rules came much earlier than its theoretical examination, which dates back to 80s in last century [135].

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 [92, Section 10] and generalized belief propagation (GBP) [186]. 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 method provides an 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 the structure of the converted graph makes a significant difference in the 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 to gain by using junction tree compared to enumerating configurations in very dense graphs. For the latter, constructing a region graph itself is a challenging task and still needs 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 [188], variants of BP have been derived to improve BP in a general graph. Fractional BP in [181] applies a correction coefficient to each factor and obtain a message passing rule similarly as minimization of Bethe free energy. Generalized BP in [188] propagates belief between different regions of a graph, and damping BP in [139] updates beliefs by combining old and new beliefs. [174] relaxes a Bethe free energy into an upper

bound of the partition function and the tree-reweighed BP is obtained. Technique such as damping is also explored to seek convergence of BP and its variants [139]. Another track falls to the variational method framework, introduced by Opper and Winther [133] and Minka [117, 118], 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 [119] and shows a unifying view of message passing algorithms. The following work, stochastic EP by [105], explores EP's variant method for applications to large datasets.

Due to the fundamental role of BP for probabilistic inference and related applications, research of seeking insight into BP performance and study on its convergence have been constantly carried out. [177] presents the convergence condition of BP in graphs containing a single loop. Work in [66] analyzes the Bethe free energy and offers sufficient conditions on the uniqueness of BP fixed point. Closely related to the content of this chapter, [126] studies the sufficient conditions for BP convergence to a unique fixed point (α-BP generalizes BP). [129] proposes a stochastic BP algorithm for high-dimensional discrete spaces and gives the convergence conditions of it. [91] 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, [40, 114] studies the convergence condition of Gaussian BP inference over distributed linear Gaussian models. [152] 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 the previous chapter, we presented an iterative message passing algorithm and discussed the connection with mean field, belief propagation, and tree-reweighted belief propagation algorithms. These methods fellow different message passing rules (fixed-point iterations) that have been heuristically developed. The heuristic efforts usually include the message-passing rules, message update schedules, manual tuning and trials for convergence to stable solutions. Sometimes, it needs to switch to an alternative approximate inference method if the initial selection can not meet practical requirements, e.g., belief propagation can be exact in tree-structured graphs but generally performs worse than mean field method in densely-connected graphs. However, on our way towards automated inference methods, we intend to reduce the heuristic efforts and manual trials without degenerating their performance. In this chapter, we discuss one promising way to achieve this target. The principle idea is to do inference by solving an optimization problem, i.e., to treat inference as optimization.

In fact, we have touched this topic in Section 2.5, where we interpreted what message passing updates of mean field and belief propagation are actually doing. It turns out that the message passing rules are fixed-point iterations as solutions to 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 an example, since the message passing rule can be obtained from the minimization of Bethe free energy cost, we may just as well solve the optimization problem by other optimization techniques such as gradient descent. However, an early attempt on this track showed that it might suffer from stability issues for peaky potential functions compared to iterative message passing method [180].

Another limitation lies in the Bethe approximation itself. When representing the Bethe approximation in a factor graph, a factor node associated with a potential

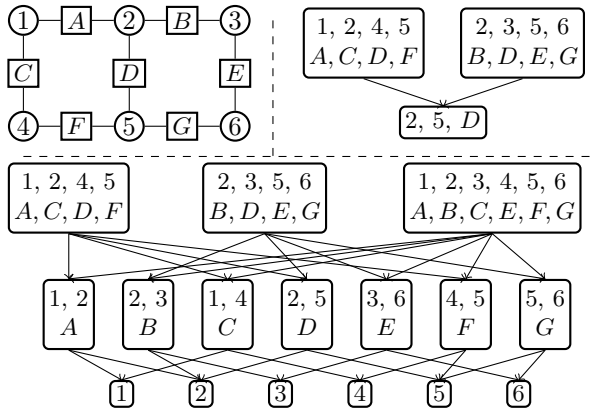


Figure 4.1: Illustration of a factor graph for 2-by-3 grid (top left, variable nodes are indexed by number and factor nodes by letters), and two alternative regions graphs (two levels for the top right one and three levels for bottom one) constructed from the factor graph.

function is related to 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 neighboring factor of the variable node. The interactions between multiple variables cannot 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 a graphical representation to overcome the above-mentioned limitation. Along with that, we would bring the concept of inference as optimization into the representation with clusters.

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 define 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 shown in the top-left example in Figure 4.1) in Section 2.5. We then can compute the marginal distributions of (4.1) from a collection of updated

messages as discussed in Chapter 3. The fundamental limitation of methods in this family is that a message over a single variable is communicated instead of messages over multiple variables (as pointed out above). This issue can be relieved by using a hyper-graph or cluster graph where a node is associated with multiple variables. A region graph is such a kind of graph, which was proposed by [178, 186] to assist 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 the definition of a region graph, we need to define the *region* first, which is on the basis of a factor graph $\mathcal{G}_{\mathcal{F}}(\mathcal{V} \cup \mathcal{F}, \mathcal{E}_{\mathcal{F}})$ (see Definition 2.1).

Definition 4.1. *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.*

A region in a region graph acts as a node, and there are directed edges between regions which are defined according to specific rules. Formally, a region graph is defined as follows:

Definition 4.2. *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 the set of regions at level l, and $R_i^{[l]} \in \mathcal{R}_l$ is the i-th region node at level l. This means $\mathcal{R}_0$ is a 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 does not matter at which level it locates. Lastly, since a region may be associated with both variable and factor nodes, we denote the scope of R by $\boldsymbol{x}_R = \{x_i | i \in R\}$.

Example 4.1. *Take the top-right region graph in Figure 4.1 as an example. There are total two layers in the region graph, i.e., $\mathcal{R}_0$ and $\mathcal{R}_1$. We have $\mathcal{R}_0 = \left\{R_1^{[0]}, R_2^{[0]}\right\}$ where $R_1^{[0]} = \{1, 2, 4, 5, A, C, D, F\}$ and $R_2^{[0]} = \{2, 3, 5, 6, B, D, E, G\}$. As for $\mathcal{R}_1$, there is only one region in it and it is $R_1^{[1]} = \{2, 5, D\}$. There are two directed edges, i.e., $\left(R_1^{[0]}, R_1^{[1]}\right)$ and $\left(R_2^{[0]}, R_1^{[1]}\right)$.*

GBP operates on a 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., along the directed edge $(P, R) \in \mathcal{E}$. Let us define the factors in region R as $A_R = \{a | a \in R\}$. $\mathcal{P}(R)$ 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

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

where $\mathcal{N}(P,R)$ is the set of all connected pairs of regions (I,J) such that J is in $\hat{\mathcal{D}}(P)$ but not $\hat{\mathcal{D}}(R)$ while I is not in $\hat{\mathcal{D}}(P)$, $\mathcal{H}(P,R)$ is the set of all connected pairs of regions (I,J) such that J is in $\hat{\mathcal{D}}(R)$ while I is in $\hat{\mathcal{D}}(P)$ but not $\hat{\mathcal{D}}(R)$. They are formally defined as

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

By comparing the GBP with previously discussed mean field, loopy BP and α-BP in Chapter 2 and 3, it can be seen that the message update of GBP is more complex. The ordering or sequence of GBP message updates in a region graph also affects its performance (e.g., see [186]). After the message passing phase, the belief for each region R, i.e., the approximation to marginal $p(\boldsymbol{x}_R)$, is computed as

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

Example 4.2. *We demonstrate an example of GPB message update using the three-level region graph in Figure 4.1. Following our notation, we have* $\mathcal{R}_0 = \left\{R_1^{[0]}, R_2^{[0]}, R_3^{[0]}\right\}$ *where* $R_1^{[0]} = \{1,2,4,5,A,C,D,F\}$, $R_2^{[0]} = \{2,3,5,6,B,D,E,G\}$, *and* $R_3^{[0]} = \{1,2,3,4,5,6,A,B,C,D,F,G\}$. *For* $\mathcal{R}_1$, *we have* $\mathcal{R}_1 = \left\{R_1^{[1]},\cdots,R_7^{[1]}\right\}$ *with* $R_1^{[1]} = \{1,2,A\}$ *and* $R_7^{[1]} = \{5,6,G\}$. *The same notation applies to the bottom level, e.g.,* $R_2^{[2]} = 2$, *etc. Considering the message from region* $R_1^{[1]} = \{2,5,D\}$ *to* $R_5^{[2]} = \{5\}$, *i.e.,* $m_{R_1^{[1]}\to R_5^{[2]}}$, *we have*

$$\begin{aligned}\hat{\mathcal{D}}(R_4^{[1]}) &= \{\{2\},\{5\},\{2,5,D\}\} = \left\{R_2^{[2]}, R_5^{[2]}, R_4^{[1]}\right\}\\ \hat{\mathcal{D}}(R_5^{[2]}) &= \{\{5\}\} = \left\{R_5^{[2]}\right\}\end{aligned}$$

The corresponding sets $\mathcal{N}$ *and* $\mathcal{H}$ *defined in* (4.3) *are*

$$\begin{aligned}\mathcal{N}(R_1^{[1]}, R_5^{[2]}) &= \left\{\left(R_1^{[0]}, R_4^{[1]}\right), \left(R_2^{[0]}, R_4^{[1]}\right), \left(R_1^{[1]}, R_2^{[2]}\right), \left(R_2^{[1]}, R_2^{[2]}\right)\right\},\\ \mathcal{H}(R_1^{[1]}, R_5^{[2]}) &= \emptyset.\end{aligned}$$

Then the message from region $R_1^{[1]}$ *to* $R_5^{[2]}$ *follows*

$$m_{R_1^{[1]}\to R_5^{[2]}} \propto \sum_{x_2} \varphi_D m_{R_1^{[0]}\to R_4^{[1]}} m_{R_2^{[0]}\to R_4^{[1]}} m_{R_1^{[1]}\to R_2^{[2]}} m_{R_2^{[1]}\to R_2^{[2]}}.$$

The other messages can be updated similarly by following the rule in (4.2). *When the message passing phase is done, the beliefs can be computed according to* (4.4). *We exemplify the computation for* $b_{R_4^{[1]}}(x_2, x_5)$ *here. Since*

$$\begin{aligned}
\mathcal{P}(R_4^{[1]}) &= \{\{1,2,4,5,A,C,D,F\},\{2,3,5,6,B,D,E,G\}\} = \left\{R_1^{[0]}, R_2^{[0]}\right\},\\
\mathcal{D}(R_4^{[1]}) &= \{\{2\},\{5\}\} = \left\{R_2^{[2]}, R_5^{[2]}\right\},\\
\mathcal{P}(R_2^{[2]}) &= \{\{1,2,A\},\{2,3,B\},\{2,5,D\}\} = \left\{R_1^{[1]}, R_2^{[1]}, R_4^{[1]}\right\},\\
\mathcal{P}(R_5^{[2]}) &= \{\{2,5,D\},\{4,5,F\},\{5,6,G\}\} = \left\{R_4^{[1]}, R_6^{[1]}, R_7^{[1]}\right\},
\end{aligned}$$

we can formulate

$$b_{R_4^{[1]}}(x_2, x_5) \propto \varphi_D m_{R_1^{[0]}\to R_4^{[1]}} m_{R_2^{[0]}\to R_4^{[1]}} m_{R_1^{[1]}\to R_2^{[2]}} m_{R_2^{[1]}\to R_2^{[2]}} m_{R_6^{[1]}\to R_5^{[2]}} m_{R_7^{[1]}\to R_5^{[2]}}.$$

4.2 Region-based Free Energy

Recall that BP corresponds to the minimization of Bethe free energy (Section 2.5). It is interesting to note that GBP also corresponds to a free energy defined over region graphs, i.e., the *region-based free energy*. Each region R in a region graph has its own region energy, defined as follows.

Definition 4.3. *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 over a region graph $\mathcal{G}_R$ is defined over the region energies and also the region beliefs. Formally, the region-based free energy is defined as follows.

Definition 4.4. *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. *The integer counting numbers are to balance the contribution of each region to the overall region-based free energy (see Section 4.3.2 for detailed discussions).*

Region-based free energy is a more generalized free energy than the variational free energy discussed in Section 2.5. For instance, Bethe free energy, corresponding to loopy BP, can be viewed as a special case of region-based free energy.

4.2.1 Recover Bethe Free Energy from Region-based Free Energy

If we define two types of regions (large regions and small regions) directly from a factor graph $\mathcal{G}_F(\mathcal{V} \cup \mathcal{F}, \mathcal{E}_F)$, which is a two-level region graph with large regions locating at the root level (level 0) and small regions locating at level 1,

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

The edges of the original factor graph $\mathcal{G}_F$ are preserved in the constructed region graph with adding directions from a factor node to its neighboring variable nodes. 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

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

Putting (4.8) and (4.9) together recovers the Bethe free energy (2.13).

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 $p(\boldsymbol{x}_R)$. A region belief $b_R(\boldsymbol{x}_R)$ is an approximation to $p(\boldsymbol{x}_R)$. Instead of propagating messages as GBP, or directly optimizing with regard to variables $b_R(\boldsymbol{x}_R)$ of interest, we use the reparameterization technique that is also used by recent works [4, 88, 159, 182], to model the values of interest to be the output of a neural network. We then optimize with regard to the parameters of the neural network. This technique is called amortizing.

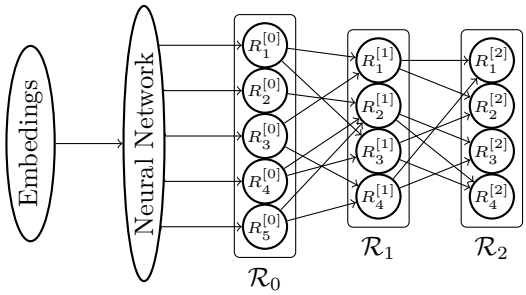


Figure 4.2: Illustration of a RENN with three levels of regions ($\mathcal{R}_0$, $\mathcal{R}_1$, $\mathcal{R}_2$).

Note that in RENN, the neural network only needs to directly model the 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 number of neural network parameters compared to directly modeling the beliefs of all regions.

4.3.1 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 representation mapping $\boldsymbol{f}$ followed by the softmax function in a RENN only needs to directly output the beliefs on root regions $\mathcal{R}_0$. For the rest of the regions $\{R \in \mathcal{R} \backslash \mathcal{R}_0\}$ that are not root regions in the region graph, 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 the 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 with regard to 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 with regard to every variable node is eliminated in the factor graph, i.e., it has converged.

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 the region graph. That is to say, apart from the averaging effect from definition (4.11) itself, a hard-penalty is enforced straightforwardly in our cost.

Different from GBP that minimizes region-based free energy by iterative message-passing, RENN minimizes the region-based free energy by optimizing with regard to 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:

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

where $d(\cdot,\cdot)$ is a distance metric or divergence to measure the mismatch between the beliefs (L_2 distance is used in our experiments), λ is a positive regularization parameter. To summarize, a RENN takes embedding vectors as input and output beliefs directly by minimizing regularized region-based free energy. Embedding vectors are tunable variables (part of paramter $\boldsymbol{w}$), and will be explained in Section 4.4.1, although they are not explicit in the objective function (4.12).

Example 4.3. *We demonstrate a toy RENN in this example. As illustrated in Figure 4.2, a three-level RENN takes embedding vectors as input and outputs the beliefs on $\mathcal{R}_0$ directly. 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 with regard to $\boldsymbol{\omega}$.*

Remark 4.1 (Region-based Free Energy and the Partition Function)**.** *We explained in Section 2.5 that Bethe free energy is an approximation to the negative log-partition function, i.e., $-\log Z(\boldsymbol{\theta})$. This relationship between free energy and partition function similarly exists for RENN. After the energy minimization phase as stated in* (4.12)*, the minimized region-based free energy $F_R(\mathcal{B}^*;\boldsymbol{\theta})$ is an approximation to the negative log-partition $-\log Z(\boldsymbol{\theta})$, where $\mathcal{B}^* = \{b_R(\boldsymbol{x}_R;\boldsymbol{\omega}^*)|R\in\mathcal{R}\}$ with $\boldsymbol{\omega}^*$ as the parameter of RENN after the optimization phase.*

4.3.2 Region Graph Construction for RENN

In this section we 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 as long as the resulted graph fulfills the Definition 4.2. But this does not mean we can rely on an arbitrary region graph to do our inference. Conditions such as *valid* region graph (will be discussed in the following section) and *maxent-normality* [179, 186] have been proposed for region graphs. But these conditions do not provide rules for

how to construct "good" region graphs. Our approach to this issue is to combine the cluster variation method [83, 127] with the *tree-robust* condition [47] (that was originally developed to improve the accuracy of GBP) to obtain practical region graph construction for RENN.

In what follows, we firstly explain the concept of valid region graphs. Then the method of region graph construction for RENN is detailed.

Determining the Counting Numbers

In Definition 4.4, region-based free energy is defined as 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 [186], 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 $\mathcal{G}_F$ falls in the region R of the 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

The cluster variation method was introduced by Kikuchi and other physicists [83, 127], which started with the intuition of approximating free energy by using larger sets of variable nodes instead of the single-node factorization in mean field approximation.

The 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 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 the previous section, 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 a region graph. We follow the path of [47,179] for this issue. Specifically, we use the *tree-robust* condition [47] to build the root regions for our RENN. These root regions are then used to grow the other hierarchical levels of the region graph by the cluster variation method. The tree-robust condition was developed originally for GBP to gain better approximations. GBP has better accuracy on tree-robust graphs than on non-tree-robust graphs. We will show in our experiment in Section 4.4 that RENN outperforms GBP even in tree-robust graphs.

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

Definition 4.5. *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 4.6. *Let T be a spanning tree of graph $\mathcal{G}$. A cycle basis $\mathcal{CB}$ is tree exact with regard to 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 4.6 tells us that if a cycle basis is tree exact with regard to 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 4.7. *A cycle basis $\mathcal{CB}$ is tree-robust if it is tree exact with regard to all spanning trees of $\mathcal{G}$.*

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, i.e., the undirected graph of MRF representation
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 $\text{path}_1 = e$
    Find the shortest path path_2 from s to t via *used* edges
  else
    Find a path from s to a *visited* u that contains edge e, this path is set as path_1.
    Find the shortest path path_2 from s to u via *used* edges
  end if
  Add cycle C consisting of path_1 and path_2 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

We use two theorems from [47] for choosing cycle basis in two specific graph classes, 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 a unique edge.

Theorem 4.1. *In a planar graph $\mathcal{G}$, the cycle basis comprised of the faces of the graph $\mathcal{G}$ is tree-robust.*

Theorem 4.2. *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.*

Tree-robust root regions can also be constructed for general graphs, which can be seen as an extension from Theorem 4.1 and 4.2. For general graph case, it basically is to find a subgraph that is a planar or compote graph, and then extract the corresponding tree-robust basis, after which extra cycles are added in by following Algorithm 1. To be more specific, the Algorithm 1 returns a partially tree-robust basis, since tree-robust condition for a general graph requires inspecting all subsets of cycles in a candidate basis, which is usually prohibitive.

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. Apart from the inference experiments, we also carried out the MRF model learning experiments. Analysis and experiments on MRF learning are discussed in Chapter 5.

4.4.1 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 $\boldsymbol{\theta} = \{J_{ij}, h_i | (i,j) \in \mathcal{E}_F, i,j \in \mathcal{V}\}$. J_{ij} is always sampled from standard normal distribution, i.e. $J_{ij} \sim \mathsf{N}(0,1)$, meanwhile $h_i \sim \mathsf{N}(0,\gamma^2)$ with varying γ in different experiment cases.

In the inference experiments, we are interested in how well beliefs from RENN approximate true marginal distributions of $p(\boldsymbol{x};\boldsymbol{\theta})$. We quantify 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 an approximated value, i.e., the region-based free energy for RENN or GBP while a counterpart free energy form for mean field or loopy BP as explained in Section 2.5.

In all experiments, for each evaluation of RENN, mean field, (loopy) BP [125], damped BP [139] with damping factor 0.5, and GBP [186] 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 [182] targeting the Bethe free energy, is also used as a comparison. To make the comparison with Inference Net fair, RENN and Inference Net use the same neural network structures and hidden dimensions. Each variable x_i is associated with a learnable embedding vector $\boldsymbol{e}_i$. A transform layer [171] consumes $\boldsymbol{e}_i$ and outputs 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.

4.4.2 Inference on Grid Graphs

We first evaluate how well RENN can estimate the marginal distributions, compared with benchmark algorithms/models, with regard to marginal ℓ_1 errors and Pearson correlation ρ, for 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. 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

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	$\mathbf{0.025} \pm 0.020$
	ρ	0.913 ± 0.049	$\mathbf{0.984} \pm 0.021$
	$\log Z$ Error	3.564 ± 2.823	0.384 ± 0.223
1	ℓ_1 Error	0.145 ± 0.028	$\mathbf{0.016} \pm 0.010$
	ρ	0.783 ± 0.091	$\mathbf{0.995} \pm 0.010$
	$\log Z$ Error	0.825 ± 0.841	0.364 ± 0.201

the relative strength of standalone node log-potentials to pairwise log-potentials. The results are reported as 'mean $\pm$ standard deviation' in tables.

The marginal approximation can be reflected by ℓ_1 error and correlation coefficient ρ. These two metrics also reflect the belief consistency since both univariate and pairwise marginals are used in evaluation computation. 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 sometimes 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 [186], 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 competitive in different considered cases.

Note a region graph in this set of experiments uses all faces of a 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, the performance of GBP drops slightly after including the infinite face. But the number of nodes in the region built from the infinite face would scale with the perimeter of a grid graph. Since RENN already has reasonable good accuracy outperforming benchmark methods as shown in Table 4.1 and 4.2, we suggest dropping the infinite face in constructing region graphs from grids.

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

4.4.3 Inference on Challenging Complete Graphs

In this section, we evaluate RENN in comparison with benchmark methods on more challenging graphs, i.e. complete graphs in which every two nodes are connected by a unique edge. Due to the high complexity, we carry out the inference experiments on complete graphs of size $n = 9$ and $n = 16$ but with a richer setting of γ, to be able to 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 with regard to ℓ_1 error, i.e. 0.109 versus 0.130, but falls behind RENN with regard to 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 an advantage 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 errors 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 a much better estimation of $\log Z$ in complete 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. Different 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 perform 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 variables. This is in contrast to the 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 needs to handle region graph construction. In the construction of region graphs, we select the tree-robust condition that is based on a cycle basis for root regions. The presence of loops in graphs casts known difficulties for classical

message-passing methods. The construction principle of RENN gives it the chance to relieve the difficulty due to 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 the definition of a child region's belief from the marginalization of its parental regions' beliefs; ii) the penalty cost of inconsistency in the objective function.

In the coming chapter, i.e., Chapter 5, we will 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 [94, 136] 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 [187]. The attempts in improving the approximate inference with BP and gaining better insight has been made, where the representative works are fractional BP [181], tree-reweighted BP [174], generalized BP (GBP, , also known as parent-to-child algorithm) [186, 188], 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 region 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 a 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 popular in recent years, where they have success in areas of deep graphical generative models for structured data modeling [78, 103, 141]. 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 needs sampling which brings in the trade-off between training speed and estimation variance. These issues also lie in the one of most successful model, variational autoencoders [88, 115, 167], and other variational methods [38, 65, 95, 170].

Apart from the directed graphical models, there is also a track of work on using neural networks to model the message passing functions. [4] models the intractable message update functions by Gaussian distributions with parameters as the output of a neural network, and then follows the typical message passing rules to do itera-

tive message updates of standard BP. [64, 77] 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 [52, 189] use a graph network update messages and a separate network to map messages into targeted results. Training of these models needs true marginal distributions which might be an issue for cases where it is too expensive or prohibitive to collect the true marginals.

Closely related to work in this chapter, Bethe free energy is directed minimized in [180,182,184] instead of iterative message passing methods, which can be treated as special cases of RENN. [180] uses a gradient descent method to alternative minimize single variable marginals. [182] parameterizes marginals directly by a neural network and minimizes Bethe free energy.

Part III

Learning

Chapter 5

Learning with Inference

In Part I and II, we have discussed different approaches to perform inference. But these discussions were based on one assumption: *a probabilistic graphical model is known or given*. From this chapter onward, we mainly focus on topics in learning probabilistic graphical models. As introduced in Chapter 2, learning of a graphical model includes structure learning and parameter learning. We restrict our discussion to parameter learning in graphical models. That is to say, 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 2.6, the most essential learning principle is *maximum likelihood* that is derived from the minimization of KL-divergence. Since we do not have access to the true distribution $p^*(\boldsymbol{x})$, practical maximum likelihood learning is via tuning the parameter $\boldsymbol{\theta}$ of a model $p(\boldsymbol{x};\boldsymbol{\theta})$ using information from samples of $p^*(\boldsymbol{x})$, i.e. $\mathcal{D} = \left\{\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^M\right\}$.

We begin with the explanation of why inference is required in learning, and then, discuss MRF learning via RENN. Numerical comparisons with learned MRFs via classic message-passing methods are presented and discussed afterwards. We also include learning with hidden variables, an important topic in graphical model learning, in this chapter.

5.1 Why does learning of an MRF require inference?

Let us continue the discussion on learning in Section 2.6. 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 with regard to $\boldsymbol{\theta}_a$ gives

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

In (5.2), a computation trick is used, i.e., $\frac{\partial f(x)}{\partial x} = f(x)\frac{\partial \log f(x)}{\partial x}$ holds for a differentiable function $f(x)$. Applying all sample from dataset $\mathcal{D}$ to (5.2) gives

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

We similarly have the gradient of $\mathcal{L}$ as

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

In general, there is no closed-form solution to the maximization problem of the 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 with regard to the gradient of potential function φ_a, while on the right-hand-side of (5.5) the expectation is computed by the 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 5.1 (Interpretation to learning with inference)**.** *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 with regard to the sufficient statistics. The corresponding equation of* (5.5) *in this case is reduced into moment matching.*

For log-linear potential function φ_a, $\forall\ a \in \mathcal{F}$, the first term of $l(\boldsymbol{x};\boldsymbol{\theta})$ is linear with regard to $\boldsymbol{\theta}$ and the second term, i.e. $-\log Z(\boldsymbol{\theta})$, is concave. Thus the log-likelihood is concave and there is a unique globally optimal log-likelihood value. But this does not necessarily mean that there is always a unique global optimum $\boldsymbol{\theta}^*$, since the parameterization of an MRF can be redundant. That is to say, in the case of over-parameterization of an MRF, there could exit multiple global optima after maximum likelihood learning.

Apart from the classical techniques such as iterative proportional fitting [174, Section 6.1] [183], 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 an example: we need to use one of the inference approaches introduced in Part I and II to approximate the marginals $\{p(\boldsymbol{x}_a;\boldsymbol{\theta}), a \in \mathcal{F}\}$ and then do the update of the parameters

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

Remark 5.2 (Learning with a prior)**.** *When an MRF scales up, the number of parameters, i.e. elements of $\boldsymbol{\theta}$, could be large. If training data is limited, it may lead to overfitting. To address this issue, a typical trick is to use prior distributions to regulate the parameters. This principle comes from Bayesian learning where priors that stand for domain knowledge and usually are subjective or biased, and likelihoods from observations form the objective function for learning together. Since the formulated joint likelihood of random variable $\boldsymbol{x}$ and parameter $\boldsymbol{\theta}$ is proportional to the posterior of $\boldsymbol{\theta}$ given the evidence, maximization of the joint likelihood is also termed as maximum a posterior. Learning with priors offers benefits at least in two ways: i). When the size of training data is small, the biased prior from domain knowledge has a stronger impact on the choice of the model parameter $\boldsymbol{\theta}$ than the likelihood from observations. ii). As the number of training samples increases, the prior's impact gets weaker and the likelihood of observations dominates. Therefore, employing a prior can help to avoid overfitting in the case of insufficient training data, while still allowing objective information from observations to affect the choice of model parameters. The latter dominates as there are richer training samples. For a concrete example, let us apply a zero-mean Gaussian prior distribution to $\boldsymbol{\theta}$. Then, the log-likelihood of dataset $\mathcal{D}$ becomes*

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{\boldsymbol{x}\in\mathcal{D}} [\log \tilde{p}(\boldsymbol{x};\boldsymbol{\theta}) - \log Z(\boldsymbol{\theta})] - \frac{\|\boldsymbol{\theta}\|^2}{2\sigma^2}, \tag{5.7}$$

where σ^2 is the variance of the Gaussian prior, which is free to determine. The rest of the analysis follows. Alternatively, Laplacian prior is also commonly used, which brings a ℓ_1-norm regularization over $\boldsymbol{\theta}$ and encourages sparsity.

5.2 MRF Learning with Inference of RENN

In Section 4.3.1, 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 MRF $p(\boldsymbol{x};\boldsymbol{\theta})$ with inference by RENN here.

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

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

As discussed in Section 5.1, inference is needed in this MRF learning problem due to the intractable partition function, whose gradient with regard to the model parameter is an expectation with the support of marginal distributions. RENN, similar to other inference methods, is able to infer both the marginal distributions (via beliefs) and partition functions (via region-based free energy) more efficiently and accurately, as explained in Chapter 4. Thus there are two alternative ways to employ the inference results of RENN for MRF learning:

- use belief $\{b_a(\boldsymbol{x}_a)\}$ obtained from a RENN in place of marginal distributions $\{p(\boldsymbol{x}_a;\boldsymbol{\theta})\}$ to compute gradient of the likelihood $\mathcal{L}$ in (5.4). Then employ a gradient-based optimization algorithm to optimize the model parameter $\boldsymbol{\theta}$;
- use the region-based free energy of a RENN in place of the intractable negative log-partition function $-\log Z(\boldsymbol{\theta})$. Make use of autodiff methods in morden framework such as PyTorch or Tensorflow for $\boldsymbol{\theta}$. Then do alternating optimization routine with regard to the model parameter $\boldsymbol{\theta}$ and RENN parameters under the same objective function.

The first alternative is straightforward by following the discussion in Section 5.1. We give a further explanation to the second alternative in this section.

The region-based free energy $F_R(\mathcal{B};\boldsymbol{\theta})$ in (4.6) of Definition 4.4 is the exactly 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 MRF learning and RENN 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 does not need iterative message propagation. Additionally, RENN can be implemented with modern toolboxes and enjoys the GPU computation capacity. Consequently, learning MRF with RENN can be much faster, which will be shown in the following section.

5.3 Numerical Comparisons in MRF Learning

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

We perform 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 $\boldsymbol{\theta}'$,

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**

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: seconds) for two training cases in Table 5.1 and 5.2.

	Grid Graph $n = 225$	Complete Graph $n = 16$
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

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 MRF with the obtained training dataset by using RENN (Section 5.2). The trained MRF by RENN is evaluated with the testing dataset with regard to the 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. In addition, 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 the model parameter $\boldsymbol{\theta}$ is updated in the optimization steps.

In the case 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 Further Discussion on Learning

Parameter learning becomes more challenging when the variable vector $\boldsymbol{x}$ is only *partially 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 hidden or 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 cannot maximize the complete evidence log-likelihood as we did in Section 5.1. Instead, we marginalize out the latent variable $\boldsymbol{x}_U$ 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 $Z(\boldsymbol{x}_O; \boldsymbol{\theta}) = \sum_{\boldsymbol{x}_U} \tilde{p}(\boldsymbol{x}; \boldsymbol{\theta})$. In general, there are two categories of methods in dealing with learning in partial observed scenarios:

- Directly optimize $l(\boldsymbol{x}_O; \boldsymbol{\theta})$ with regard to $\boldsymbol{\theta}$. 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, can be used to do the inference on (conditional) marginal distributions.

- Optimize a lower bound of $l(\boldsymbol{x}_O; \boldsymbol{\theta})$. Methods such as (variational) EM, variational Bayes and variational auto-encoder (VAE) [89] belongs to this category.

The first track is straightforward since it is equivalent to estimating 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 step of model parameter learning sequentially. Note different from the 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 $\boldsymbol{x}_O$.

Alternatively, we may work with a lower bound of $l(\boldsymbol{x}_O; \boldsymbol{\theta})$ instead. In this approach, the most classical framework is EM, which can be well explained by our target of maximizing $l(\boldsymbol{x}_O; \boldsymbol{\theta})$. Recap the variational free energy F_V in (2.9) in

Section 2.5. The intuition can be obtained by substituting the posterior $q(\boldsymbol{x}_U|\boldsymbol{x}_O)$ into the variational free energy and observing that

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

where $\boldsymbol{x}_O$ is the clamped value of observation. Since KL divergence is non-negative,

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

Substituting (5.12) into (5.10), we have $l(\boldsymbol{x}_O;\boldsymbol{\theta})$ bounded from bellow

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

which is exactly the lower bound (2.19). This gives 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 5.3. *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 5.4. *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

Equipping Expectation Maximization with Normalizing Flows

When learning with partially observed variables in graphical models, EM is one of the most widely used frameworks in practice. In directed graphical models, immediate relief is that we do not need to deal with the partition function $Z(\boldsymbol{\theta})$ anymore. However, this does not mean that directed graphical models with partially 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 to track uncertainty of the 'unobserved' part.

As discussed in Section 2.6, the presence of latent or hidden variables is either due to the data availability or abstraction of data generation. 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. the hypothesis space, based on the maximum likelihood criteria. If the hypothesis space is very limited, the ability to represent our true p^* is limited, leading to an 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.

A common approach for enriching the hypothesis space of candidate models is using mixture models by assuming p^* as a multi-modal distribution. A Gaussian mixture model is a representative model in this family. The EM framework has been successfully and widely applied to the learning of Gaussian mixture models, in which case closed-form update rules are available. The advantage comes with the

limitation to linear dependencies and the restriction on available free parameters that can be tuned, i.e. a limitation of flexibility.

To explore further, we discuss more flexible models beyond Gaussian in this chapter. We start by inducing normalizing flows that are going to be used as probabilistic models in place of Gaussian models. A normalizing flow can consist of as many non-linear invertible transformations as needed, and thus is highly flexible. Besides, the ability to compute likelihoods remains, and it allows for efficient sampling in normalizing flows. Then, mixture models that employ normalizing flows are proposed, which are highly flexible. Due to the abstraction that we made on the true p^*, i.e. the multi-modal assumption, and the introduction of normalizing flows, there are hidden variables in the proposed models. Thus, model learning is addressed by adapting the EM framework. Since closed-form iterative update solutions are not available anymore, we turn to gradient search based optimization to design practical algorithms for model learning in the EM framework.

6.1 Normalizing Flow

Normalizing flow is a practical approach to improve the flexibility of probabilistic models 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 is 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 $\tilde{\boldsymbol{x}} \in \mathbb{R}^N$. Then we have

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

where $\boldsymbol{z} \in \mathbb{R}^N$ is the initial random variable with know density function $p(\boldsymbol{z})$. To explicitly denote the chain, we have

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

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

$$\boldsymbol{z} = \boldsymbol{h}_0 \underset{\tilde{\boldsymbol{f}}^{[1]}}{\overset{\tilde{\boldsymbol{g}}^{[1]}}{\longleftrightarrow}} \boldsymbol{h}_1 \underset{\tilde{\boldsymbol{f}}^{[2]}}{\overset{\tilde{\boldsymbol{g}}^{[2]}}{\longleftrightarrow}} \cdots\cdots\cdots \underset{\tilde{\boldsymbol{f}}^{[L]}}{\overset{\tilde{\boldsymbol{g}}^{[L]}}{\longleftrightarrow}} \tilde{\boldsymbol{x}} = \boldsymbol{h}_L$$

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 of the flow is invertible, the full mapping is invertible. Then, the probability density function relation between $p(\boldsymbol{z})$ and $p(\tilde{\boldsymbol{x}})$ follows the change of variable formula

$$p(\tilde{\boldsymbol{x}}) = p(\boldsymbol{z})\big|\det(\boldsymbol{J})\big|, \text{ with } \boldsymbol{z} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}), \tag{6.3}$$

where $\det(\boldsymbol{J})$ denotes the determinant of Jacobian $\boldsymbol{J}$

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial \tilde{f}_1}{\partial \tilde{x}_1} & \cdots & \frac{\partial \tilde{f}_1}{\partial \tilde{x}_N} \\ \vdots & \cdots & \vdots \\ \frac{\partial \tilde{f}_N}{\partial \tilde{x}_1} & \cdots & \frac{\partial \tilde{f}_N}{\partial \tilde{x}_N} \end{bmatrix}. \tag{6.4}$$

The challenge lies in the determinant computation of the Jacobian $\boldsymbol{J}$, or equivalently, that of each transformation $\tilde{\boldsymbol{g}}^{[l]}$. [37] introduced the model NICE to transform only half of the variables at a chain step, which results in triangular Jacobian of the transformation. If each transformation has a triangular Jacobian, determinant computation becomes efficient in (6.3). This flow architecture divides the feature $\boldsymbol{h}_l$ at the l'th layer into two subparts as $\boldsymbol{h}_l = [\boldsymbol{h}_{l,a}^\intercal, \boldsymbol{h}_{l,b}^\intercal]^\intercal$ where $(\cdot)^\intercal$ denotes transpose operation. We have the following forward and inverse relations between $(l-1)$'th and l'th layers:

$$\begin{aligned} \boldsymbol{h}_{l-1} &= \begin{bmatrix} \boldsymbol{h}_{l-1,a} \\ \boldsymbol{h}_{l-1,b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_{l,a} \\ \boldsymbol{m}_a(\boldsymbol{h}_{l,a}) \odot \boldsymbol{h}_{l,b} + \boldsymbol{m}_b(\boldsymbol{h}_{l,a}) \end{bmatrix}, \\ \boldsymbol{h}_l &= \begin{bmatrix} \boldsymbol{h}_{l,a} \\ \boldsymbol{h}_{l,b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_{l-1,a} \\ (\boldsymbol{h}_{l-1,b} - \boldsymbol{m}_b(\boldsymbol{h}_{l-1,a})) \oslash \boldsymbol{m}_a(\boldsymbol{h}_{l-1,a}) \end{bmatrix}, \end{aligned} \tag{6.5}$$

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). Then, for the flow model, the determinant of the Jacobian matrix is

$$\det(\boldsymbol{J})|_{\boldsymbol{z}=\tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}})} = \prod_{l=1}^{L} \det(\boldsymbol{J}_l)|_{\boldsymbol{h}_l}, \text{ with } \boldsymbol{h}_L = \tilde{\boldsymbol{x}}, \tag{6.6}$$

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 determinant of the Jacobian matrix as

$$\begin{aligned} \det(\boldsymbol{J}_l)|_{\boldsymbol{h}_l} &= \det \begin{bmatrix} \frac{\partial \boldsymbol{h}_{l-1}}{\partial \boldsymbol{h}_l} \end{bmatrix} \\ &= \det \begin{bmatrix} \boldsymbol{I}_a & \boldsymbol{0} \\ \frac{\partial \boldsymbol{h}_{l-1,b}}{\partial \boldsymbol{h}_{l,a}} & \operatorname{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a})) \end{bmatrix} \\ &= \det\left(\operatorname{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a}))\right), \end{aligned} \tag{6.7}$$

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

$$\begin{aligned} p(\tilde{\boldsymbol{x}}) &= p(\boldsymbol{z})\big|\det(\boldsymbol{J})\big| \\ &= p(\boldsymbol{z}) \prod_{l=1}^{L} |\det\left(\operatorname{diag}(\boldsymbol{m}_a(\boldsymbol{h}_{l,a}))\right)|, \end{aligned} \tag{6.8}$$

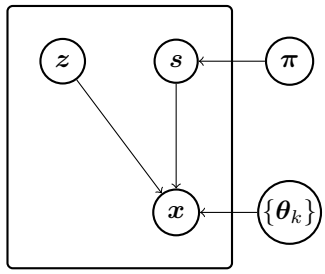


Figure 6.1: Graphical Model of GenMM.

with $\boldsymbol{z} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}})$.

(6.5) describes a *coupling* mapping between the 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 overall mapping $\tilde{\boldsymbol{g}}$. Therefore, techniques such as alternating the positions of the identity mapping [36] or using 1×1 convolution operations before each coupling mapping [86] are used to treat this issue. Sometimes, it may not be necessary to carry all features thorough all L chain steps. For instance, [36] [86] split some hidden layer signal $\boldsymbol{h}$ and model a part of it directly as standard Gaussian to reduce computation and memory burden.

Remark 6.1. *There are alternative ways of formulating the transformation of variables. A popular one is the Inverse autoregreeesive flow (IAF) proposed in [149]. In the IAF model, a chain step of transformation is formulated as*

$$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + \boldsymbol{u} \cdot m(\boldsymbol{w}^\intercal \boldsymbol{h}_{l-1} + b)$$

where m is non-linear mapping, $\boldsymbol{w}, \boldsymbol{u} \in \mathbb{R}^N$ and $b \in \mathbb{R}$. IAF does no use partial identity mapping as NICE. Its Jacobian is not triangular, and thus, determinant computation is more expensive. Moreover, whether it is invertible depends on the mapping m. The follow-up work by [90] created an element-wise transformation in a customized way to improve IAF such that its Jacobian and inverse are easier to compute.

In contrast to the above finite-chain flow where L is a discrete finite integer, the method developed in [20] introduced a continuous-transformation of variables by solving an ordinary differential equation (ODE). Comparing to the discrete finite chain of transformation in normalizing flow, ODE can be viewed as a continuous chain of variable transformation. The inverse mapping and Jacobian determinant are not required in ODE method. But ODE needs to solve differentiation equations which is also challenging.

6.2 Generator Mixture Model and EM Training

In this section, we introduce a generative model that is a mixture of K flow models. All the K flow models have a common input latent variable $\boldsymbol{z} \in \mathbb{R}^N$. Here,

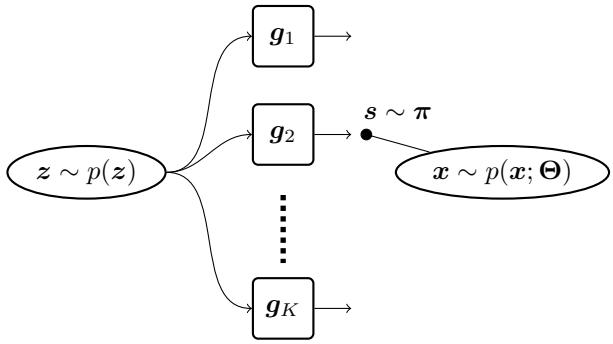


Figure 6.2: Signal diagram of GenMM.

a flow $\boldsymbol{g}_k(\boldsymbol{z}) : \mathbb{R}^N \to \mathbb{R}^N$ is an instance of $\tilde{\boldsymbol{g}}$. $\boldsymbol{g}_k$ acts as the k-th generator and depends on a set of parameters $\boldsymbol{\theta}_k$ as $\boldsymbol{g}_k(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k)$. We denote the induced probability density function by $\boldsymbol{g}_k$ as $p(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{g}_k(\boldsymbol{z})} := p_k(\boldsymbol{x})$, i.e. $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z}))$. For simplicity, we assume that all K generators have the same chain structure. Furthermore, the distribution of $\boldsymbol{z}$ is fixed as Gaussian $\mathsf{N}(\boldsymbol{0}, \boldsymbol{I})$. The induced probability density function of $\boldsymbol{x} \in \mathbb{R}^N$ of the proposed mixture model with K mixture components is given as:

$$\begin{aligned} p(\boldsymbol{x}; \boldsymbol{\Theta}) &= \textstyle\sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x}) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}_k(\boldsymbol{z})) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k)). \end{aligned} \tag{6.9}$$

We use $\boldsymbol{\Theta}$ to denote the set of all parameters $\{\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$, where $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$ 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 (6.9) is called a generator mixture model (GenMM). The graphical model of GenMM is illustrated in Figure 6.1. The plate in this figure is common notation in template models (see [92, Section 6]), denoting that we can instantiate multiple tuple $(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{s})$ instances having dependency as illustrated. Furthermore, the signal diagram of GenMM is illustrated in Figure 6.2. The GenMM can be considered as a high-complexity model because each mixture component $p_k(\boldsymbol{x})$ has its own parameter set $\boldsymbol{\theta}_k$.

Assuming that the data in $\mathcal{D}$ is sampled i.i.d. from p^*, the maximum likelihood estimation problem is

$$\hat{\boldsymbol{\Theta}} = \operatorname*{argmax}_{\boldsymbol{\Theta}} \log \prod_{\boldsymbol{x}^i \in \mathcal{D}} p(\boldsymbol{x}^i; \boldsymbol{\Theta}), \tag{6.10}$$

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 $p(s_k = 1) = \pi_k$. The variable $\boldsymbol{s}$ is the hidden variable in EM. We will use γ_k to denote the posterior probability $p(s_k = 1|\boldsymbol{x})$ calculated as

$$\gamma_k = p(s_k = 1|\boldsymbol{x}; \boldsymbol{\Theta}) = \frac{\pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))}{\sum_{l=1}^K \pi_l p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_l))}. \tag{6.11}$$

Assume that a value $\boldsymbol{\Theta}^{\text{old}}$ of the parameter set $\boldsymbol{\Theta}$ is given, the iterative steps in EM algorithm update $\boldsymbol{\Theta}$ as follows.

1. E-step: Evaluation of γ_k^i is

$$\gamma_k^i(\boldsymbol{\Theta}^{\text{old}}) = \frac{\pi_k^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z}^i; \boldsymbol{\theta}_k^{\text{old}}))}{\sum_{l=1}^K \pi_l^{\text{old}} p(\boldsymbol{g}(\boldsymbol{z}^i; \boldsymbol{\theta}_l^{\text{old}}))}. \tag{6.12}$$

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

$$\boldsymbol{\Theta}^{\text{new}} = \underset{\boldsymbol{\Theta}}{\text{argmax}}\ \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}), \tag{6.13}$$

where the expected likelihood is

$$\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}) = \sum_i \sum_k \gamma_k^i(\boldsymbol{\Theta}^{\text{old}}) \log \pi_k p_k(\boldsymbol{x}^i). \tag{6.14}$$

For the GenMM in (6.9), realizing EM requires 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 $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z})) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$. This is fulfilled by flow realization of generator $\boldsymbol{g}_k$. Since the chain-step transformations can be any invertible complex mapping as discussed in Section 6.1, they can be implemented by neural networks. The neural network implementation allows high flexibility of GenMM in representing complex-structured multi-modal data.

6.2.1 EM Learning of GenMM

The mixture model GenMM is illustrated in Figure 6.2, where K generators with a certain prior distribution share the same latent distribution $p(\boldsymbol{z})$. With a neural network based flow as the generator $\boldsymbol{g}_k$ for the k'th mixture component in GenMM, the probability density function $p_k(\boldsymbol{x})$ for any $\boldsymbol{x}$ can be computed exactly. Recall that $p_k(\boldsymbol{x}) = p(\boldsymbol{g}_k(\boldsymbol{z})) = p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}_k))$ and let $\boldsymbol{f}_k$ be the inverse of $\boldsymbol{g}_k$. Then, the posterior probability can be computed further from (6.12) as

$$\gamma_k(\boldsymbol{\Theta}^{\text{old}}) = \frac{\pi_k^{\text{old}} p(\boldsymbol{f}_k(\boldsymbol{x}))\left|\det\left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|}{\sum_{j=1}^K \pi_j^{\text{old}} p(\boldsymbol{f}_j(\boldsymbol{x}))\left|\det\left(\frac{\partial \boldsymbol{f}_j(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|}, \tag{6.15}$$

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_{EM};
Set a learning rate η.
Build two models with parameter sets:
$\boldsymbol{\Theta}^{\text{old}} = \{\boldsymbol{\pi}^{\text{old}}, \boldsymbol{\theta}_1^{\text{old}}, \ldots, \boldsymbol{\theta}_K^{\text{old}}\}$,
$\boldsymbol{\Theta} = \{\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$.
Initialize the generator prior distribution $\pi_k = 1/K$;
Initialize $\boldsymbol{\theta}_k$ of $\boldsymbol{g}_k$, for all $k = 1, \ldots, K$ randomly.
$\boldsymbol{\Theta}^{\text{old}} \leftarrow \boldsymbol{\Theta}$.
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 the dataset $\mathcal{D}$
    Compute $\gamma_k^i(\boldsymbol{\Theta}^{\text{old}})$ as in (6.15), for all $\boldsymbol{x}^i$ and $k = 1, \ldots, K$
    Compute $\mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}\right)$ as in (6.16)
    $\partial g_k \leftarrow \nabla_{\boldsymbol{\theta}_k} \frac{1}{n^b} \mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}\right), \forall \boldsymbol{\theta}_k \in \boldsymbol{\Theta}$
    $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k + \eta \cdot \partial g_k, \forall \boldsymbol{\theta}_k \in \boldsymbol{\Theta}$
  end for
  if $(t \mod t_{\text{EM}}) = 0$ **then**
    $\pi_k \leftarrow \mathbb{E}_{\mathcal{D}}\left[\gamma_k\right]$
    $\boldsymbol{\Theta}^{\text{old}} \leftarrow \boldsymbol{\Theta}$.
  end if
end for

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

$$\begin{aligned}
&\mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}\right) \\
=&\sum_{i=1}^{|\mathcal{D}|} \sum_{k=1}^{K} \gamma_k^{(i)}(\boldsymbol{\Theta}^{\text{old}}) \left[\log\ \pi_k + \log\ p(\boldsymbol{f}_k(\boldsymbol{x}^{(i)})) + \log \left| \det \left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right) \right| \right]. \quad (6.16)
\end{aligned}$$

We usually deal with a large dataset for model learning, i.e. $|\mathcal{D}|$ can be large. In that case we implement the EM algorithm in batch fashion. Recall that $\boldsymbol{\Theta} = \{\boldsymbol{\pi}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$ and hence the M-step optimization problem $\text{argmax}_{\boldsymbol{\Theta}}\, \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}})$ is addressed in two steps:

(a) optimizing with regard to $\{\boldsymbol{\theta}_k\}_{k=1}^K$,

(b) optimizing with regard to $\boldsymbol{\pi}$.

Finding a closed-form solution for the problem $\text{argmax}\, \{\boldsymbol{\theta}_k\}_{k=1}^K \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}})$ is not feasible in general. Instead, we use the batch-size gradient decent to optimize with regard to $\{\boldsymbol{\theta}_k\}_{k=1}^K$. Further, optimization in the batch fashion leads to a

practical problem as follows. Since $\boldsymbol{\theta}_k$ is the parameter set of the 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 $\boldsymbol{\Theta}^{\text{old}}$ that is needed to compute the posteriors $\gamma_k(\boldsymbol{\Theta}^{\text{old}})$ and to update $\boldsymbol{\pi}$. Thus, in learning GenMM, we maintain two such models with parameter sets $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}^{\text{old}}$. At the beginning of an EM step, $\boldsymbol{\Theta} = \boldsymbol{\Theta}^{\text{old}}$. While we optimize $\{\boldsymbol{\theta}_k\}_{k=1}^K$ of $\boldsymbol{\Theta}$ with batch-size gradient decent, we use the model with old parameter set $\boldsymbol{\Theta}^{\text{old}}$ 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: $\boldsymbol{\Theta}^{\text{old}} \leftarrow \boldsymbol{\Theta}$.

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

$$\boldsymbol{\pi}^{\text{new}} = \underset{\boldsymbol{\pi}}{\operatorname{argmax}}\ \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}),\ \text{s.t.} \sum_{k=1}^{K} \pi_k = 1. \tag{6.17}$$

The update of prior follows the solution

$$\pi_k^{\text{new}} = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \gamma_k^i(\boldsymbol{\Theta}^{\text{old}}). \tag{6.18}$$

The detail to get the solution is derived in the Section 6.2.2. For a given dataset with empirical distribution $P_d(\boldsymbol{x})$, γ_k is evaluated with batch data in order to calculate the cost $\mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}\right)$ 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., $\pi_k \leftarrow \mathbb{E}_{\mathcal{D}}\left[\gamma_k\right]$.

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.

6.2.2 Detailed Derivation of the Update for π

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

$$\mathcal{F}(\boldsymbol{\Theta}) = \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}) + \lambda \left(1 - \sum_{k=1}^{K} \pi_k\right), \tag{6.19}$$

where λ is the Lagrange multiplier. Then

$$\begin{aligned}\boldsymbol{\pi}^{\text{new}} =& \underset{\boldsymbol{\pi}}{\text{argmax}}\ \mathcal{F}(\boldsymbol{\Theta}) \\ =& \underset{\boldsymbol{\pi}}{\text{argmax}} \sum_{i=1}^{|\mathcal{D}|}\sum_{k=1}^{K}\gamma_k^i(\boldsymbol{\Theta}^{\text{old}})\Bigg[\log\ \pi_k + \log\ p(\boldsymbol{f}_k(\boldsymbol{x}^i)) \\ &+ \log\ \left|\det\left(\frac{\partial \boldsymbol{f}_k(\boldsymbol{x}^i)}{\partial \boldsymbol{x}^i}\right)\right|\Bigg] + \lambda\left(1-\sum_{k=1}^{K}\pi_k\right) \\ =& \underset{\boldsymbol{\pi}}{\text{argmax}} \sum_{i=1}^{\mathcal{D}}\sum_{k=1}^{K}\gamma_k^i(\boldsymbol{\Theta}^{\text{old}})\log\ \pi_k + \lambda\left(1-\sum_{k=1}^{K}\pi_k\right), \end{aligned} \tag{6.20}$$

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

$$\frac{\partial \mathcal{F}}{\partial \pi_k} = 0, k = 1, 2, \cdots, K, \tag{6.21}$$

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

6.2.3 On the Convergence of GenMM

Now we analysis the convergence behaviour of GenMM. Under the conditions that we present as follows, GenMM converges.

Proposition 6.1. *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 (6.3) is a continuous function of $\boldsymbol{\theta}_k$. Due to (6.3) and the continuity of Gaussian density $p(\boldsymbol{z})$, the probability density function $p_k(\boldsymbol{x})$ is a continuous function of $\boldsymbol{\theta}_k$. Therefore, $p(\boldsymbol{x})$ given in (6.9) is a continuous function of $\boldsymbol{\Theta}$. Denote the likelihood in (6.10) as $\mathcal{L}(\boldsymbol{\Theta}) = \log\prod_i p(\boldsymbol{x}^{(i)};\boldsymbol{\Theta})$. The maximum value of $\mathcal{L}(\boldsymbol{\Theta})$ is bounded due to continuity of $p(\boldsymbol{x})$ with regard to $\boldsymbol{\Theta}$. Define $\hat{F}(\boldsymbol{\Theta}) = \mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{\Theta}^{\text{old}}\right) - \sum_{i=1}^{|\mathcal{D}|}\sum_{k=1}^{K}\gamma_k^i(\boldsymbol{\Theta}^{\text{old}})\log\gamma_k^{(i)}(\boldsymbol{\Theta}^{\text{old}})$ (note $\hat{F}$ is the empirical version of lower bound in (5.13)). It is well known that $\hat{F}(\boldsymbol{\Theta})$ is a lower bound on the likelihood function $\mathcal{L}(\boldsymbol{\Theta})$, i.e. $\mathcal{L}(\boldsymbol{\Theta}) \geqslant \hat{F}(\boldsymbol{\Theta})$.

Note that the essence of EM algorithm is that the likelihood function value is elevated by increasing the value of its lower bound $\hat{F}(\boldsymbol{\Theta})$. Since the maximum value of the log-likelihood $\mathcal{L}(\boldsymbol{\Theta})$ is finite, $\hat{F}(\boldsymbol{\Theta})$ can not grow unbounded.

□

6.3 A low-complexity model

There are K flow generators implemented as 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 technique is applied as the use of shared covariance matrices in a tied Gaussian mixture model, in linear discriminant analysis [12,14,85], and the use of a common subspace in non-negative matrix factorization [60]. Based on the idea of sharing parameters, we propose a low-complexity model which we refer to as a latent mixture model as follows.

6.3.1 Latent mixture model

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

$$p(\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k p_k(\boldsymbol{z}), \tag{6.22}$$

where $p_k(\boldsymbol{z})$ is probability density function of a Gaussian distribution $\mathsf{N}(\boldsymbol{z}; \boldsymbol{\mu}_k, \boldsymbol{C}_k)$ 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 $\boldsymbol{g}(\boldsymbol{z}) : \mathbb{R}^N \to \mathbb{R}^N$ 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 6.3. Similarly, we use $\underline{\boldsymbol{\Theta}}$ to denote the set of all parameters $\{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{C}_1, \ldots, \boldsymbol{C}_K, \boldsymbol{\theta}\}$. 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

$$\begin{aligned} p(\boldsymbol{x}; \underline{\boldsymbol{\Theta}}) &= \textstyle\sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x}) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}) | s_k = 1) \\ &= \textstyle\sum_{k=1}^{K} \pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_k, \boldsymbol{C}_k). \end{aligned} \tag{6.23}$$

The LatMM is illustrated in Figure 6.3 where the neural network $\boldsymbol{g}$ is shared. Learning of LatMM requires solving the maximum likelihood estimation problem

$$\hat{\underline{\boldsymbol{\Theta}}} = \underset{\underline{\boldsymbol{\Theta}}}{\operatorname{argmax}} \ \log \prod_i p(\boldsymbol{x}^{(i)}; \underline{\boldsymbol{\Theta}}), \tag{6.24}$$

which we address using EM. We have

$$\underline{\gamma}_k = p(s_k = 1 | \boldsymbol{x}; \underline{\boldsymbol{\Theta}}) = \frac{\pi_k p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_k, \boldsymbol{C}_k)}{\sum_{l=1}^{K} \ \pi_l p(\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\theta}); \boldsymbol{\mu}_l, \boldsymbol{C}_l)}. \tag{6.25}$$

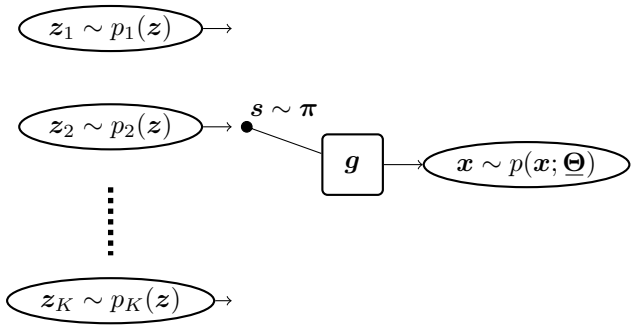


Figure 6.3: Diagram of LatMM.

Similar to the case of GenMM, realization of the corresponding EM algorithm associated with LatMM in Figure 6.23 also requires 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. Again $\boldsymbol{g}$ is instantiated from flow $\tilde{\boldsymbol{g}}$ in Section 6.1. Then, the problem is how to learn the parameters of LatMM.

6.3.2 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, $\boldsymbol{C}_k = \text{diag}(\boldsymbol{\sigma}_k^2)$. 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 $\boldsymbol{z}_k = \boldsymbol{\mu}_k + \boldsymbol{\sigma}_k \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} \sim \mathsf{N}(\boldsymbol{0}, \boldsymbol{I})$. According to Section 6.3.1, the posterior and objective function in M-step of LatMM can be respectively computed as

$$\underline{\gamma}_k(\underline{\boldsymbol{\Theta}}^{\text{old}}) = \left. \frac{\pi_k^{\text{old}} p_k(\boldsymbol{z})}{\sum_{j=1}^K \pi_j^{\text{old}} p_j(\boldsymbol{z})} \right|_{\boldsymbol{z}=\boldsymbol{f}(\boldsymbol{x})}, \tag{6.26}$$

$$\begin{aligned} &\underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Theta}}, \underline{\boldsymbol{\Theta}}^{\text{old}}\right) \\ &= \sum_{i=1}^n \log \left| \det \left(\frac{\partial \boldsymbol{f}(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right) \right| + \sum_{k=1}^K \underline{\gamma}_k^{(i)}(\underline{\boldsymbol{\Theta}}^{\text{old}}) \left[\log \pi_k + \log p_k \left(\boldsymbol{f}(\boldsymbol{x}^{(i)}); \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2 \right) \right], \end{aligned} \tag{6.27}$$

where $\boldsymbol{f}$ is the inverse of $\boldsymbol{g}$. Similar to Section 6.2.1, update of prior $\boldsymbol{\pi}$ follows $\pi_k \leftarrow \mathbb{E}_{\mathcal{D}}[\underline{\gamma}_k(\boldsymbol{x})]$. 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

Algorithm 3 EM for learning LatMM

Input: Empirical distribution $P_d(\boldsymbol{x})$ of dataset;
Latent mixture distribution:
$\sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{z}; \boldsymbol{\mu}_k, \operatorname{diag}(\boldsymbol{\sigma}_k^2)\right)$
Set a total number of epochs T of training, prior $\boldsymbol{\pi}$ update gap t_π, EM update gap t_{EM}, a learning rate η; Set hyperparameter a , b for prior of $\boldsymbol{\sigma}_k^{-1}, \forall k$.
Build two models with parameter sets:
$\underline{\boldsymbol{\Theta}}^{\mathrm{old}} = \{\boldsymbol{\pi}^{\mathrm{old}}, \boldsymbol{\mu}_1^{\mathrm{old}}, \ldots, \boldsymbol{\mu}_K^{\mathrm{old}}, \boldsymbol{\sigma}_1^{\mathrm{old}}, \ldots, \boldsymbol{\sigma}_K^{\mathrm{old}}, \boldsymbol{\theta}^{\mathrm{old}}\}$,
$\underline{\boldsymbol{\Theta}} = \{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_K, \boldsymbol{\theta}\}$.
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.
$\underline{\boldsymbol{\Theta}}^{\mathrm{old}} \leftarrow \underline{\boldsymbol{\Theta}}$
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 $\underline{\gamma}_k(\underline{\boldsymbol{\Theta}}^{\mathrm{old}})$ by (6.26), $\forall \boldsymbol{x}^{(i)}$ and $k = 1, 2, \cdots, K$
    Compute $\underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Theta}}, \underline{\boldsymbol{\Theta}}^{\mathrm{old}}\right)$ by (6.27)
    $\partial\boldsymbol{\theta}, \partial\boldsymbol{\mu}_k, \partial\boldsymbol{\sigma}_k \leftarrow \nabla_{\boldsymbol{\theta}, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k} \frac{1}{n^b} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Theta}}, \underline{\boldsymbol{\Theta}}^{\mathrm{old}}\right) + \frac{1}{K} \log \prod_{k=1}^{K} \Gamma(\boldsymbol{\sigma}_k^{-1}; a, b)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \cdot \partial\boldsymbol{\theta}$
    $\boldsymbol{\mu}_k \leftarrow \boldsymbol{\mu}_k + \eta \cdot \partial\boldsymbol{\mu}_k, \forall k$
    $\boldsymbol{\sigma}_k \leftarrow \boldsymbol{\sigma}_k + \eta \cdot \partial\boldsymbol{\sigma}_k, \forall k$
  end for
  if $(t \mod t_{\mathrm{EM}}) = 0$ **then**
    $\pi_k \leftarrow \mathbb{E}_{P_d}\left[\underline{\gamma}_k(\underline{\boldsymbol{\Theta}}^{\mathrm{old}})\right]$
    $\underline{\boldsymbol{\Theta}}^{\mathrm{old}} \leftarrow \underline{\boldsymbol{\Theta}}$
  end if
end for

of the mixture source overfits and collapses onto a data sample, the likelihood can be large and the parameter learning can be problematic. This problem is known as the singularity problem of Gaussian mixture [14]. We avoid this problem by using the following alternatives:

- Assume that for each $\forall k = 1, 2, \cdots, K$, there is a parameter prior distribution for $\boldsymbol{C}_k = \operatorname{diag}(\boldsymbol{\sigma}_k^2)$. 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 with regard to $\boldsymbol{\Theta}$ is reformulated as

$$\max_{\underline{\boldsymbol{\Theta}}} \frac{1}{|\mathcal{D}|} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Theta}}, \underline{\boldsymbol{\Theta}}^{\mathrm{old}}\right) + \frac{1}{K} \log \prod_{k=1}^{K} \Gamma(\boldsymbol{\sigma}_k^{-1}; a, b). \tag{6.28}$$

- Alternatively, we use an l_2 regularization on $\boldsymbol{\sigma}_k$, which formulates the opti-

mization step as

$$\max_{\underline{\boldsymbol{\Theta}}} \frac{1}{|\mathcal{D}|} \underline{\mathcal{Q}}\left(\underline{\boldsymbol{\Theta}}, \underline{\boldsymbol{\Theta}}^{\text{old}}\right) - \lambda \sum_{k=1}^{K} \frac{(1-\boldsymbol{\sigma}_k)^2}{K}, \tag{6.29}$$

where λ is the regulation parameter.

6.3.3 On complexity of models and new variant models

We have proposed two models, GenMM and LatMM. GenMM has a high complexity whereas LatMM has 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 [161, 162].

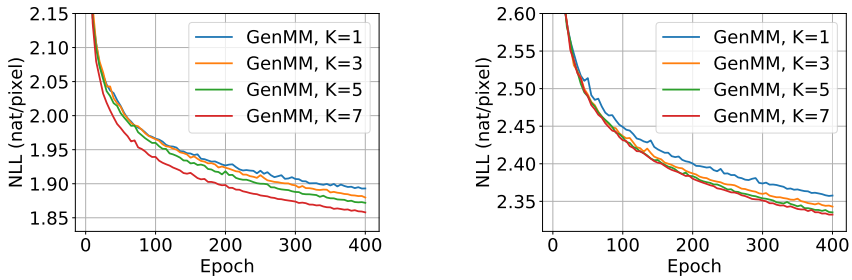
Therefore several new models can be derived using our proposed models, GenMM and LatMM. In spite of the scope and potential, the development of an analytical methodology to derive new model architectures turns out to be challenging. Traditionally the development is trial-and-error driven. Development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

6.4 Experimental Results

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

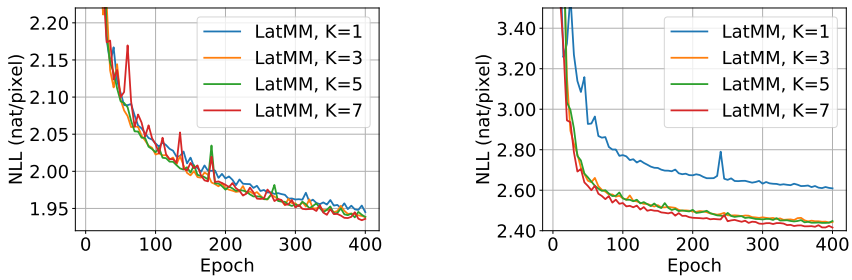
6.4.1 Experimental setup

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 [86] that is developed based on RealNVP [36] and NICE [37]. As introduced in Section 6.1, the operation in (6.5) is a coupling layer. Since only a part of the input is mapped nonlinearly after a coupling layer and the rest part remains the same, permutation [36]



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

Figure 6.4: 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 6.5: 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.

or 1×1 convolution operation [86] is used to alternate the part of the signal that goes through identity mapping. In Glow structure, a basic *flow step* is the concatenation of three layers: Actnorm (element-wise affine mapping) $\rightarrow$ 1×1 Convolution (for permutation purpose) $\rightarrow$ Coupling layer. A *flow block* consists of: a squeeze layer, several flow steps, a split layer. A squeeze layer reshapes the signal. A split layer allows the flow model to split some elements of hidden layers out and model them directly as standard Gaussian, which relieves the computation burden. In our experiments, there are also split layers that make dimension of $\boldsymbol{z}$ one fourth of dimension $\boldsymbol{x}$, and split signals in hidden layers are modeled by 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 $b = 1$.

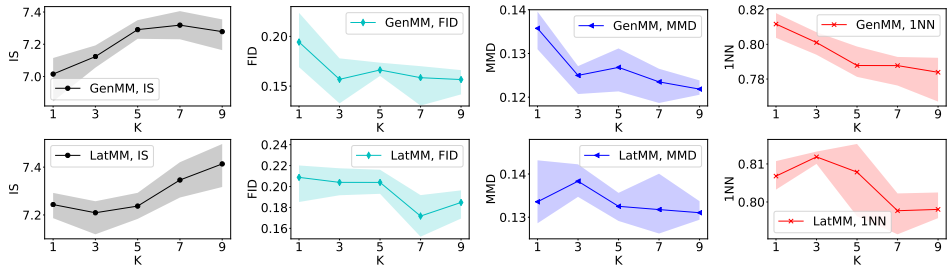


Figure 6.6: IS, FID, MMD and 1NN of GenMM and LatMM for MNIST dataset. GenMM and LatMM are trained on 60000 images of MNIST. The results are evaluated on 2000 samples per simulation point (1000 samples generated by GenMM or LatMM for corresponding K, 1000 samples from MNIST). 5 experiments are carried 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.

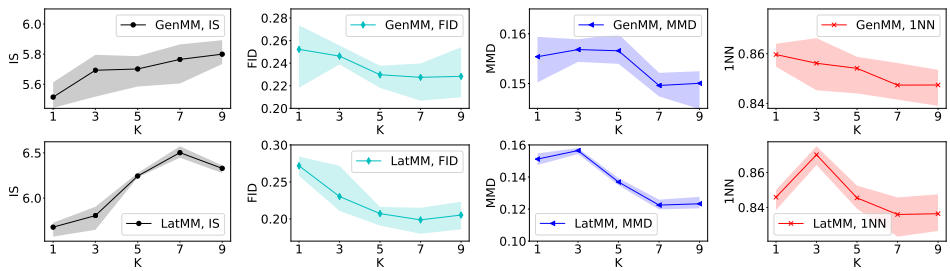


Figure 6.7: IS, FID, MMD and 1NN of GenMM and LatMM for Fashion-MNIST dataset. GenMM and LatMM are trained on 60000 images of Fashion-MNIST. The results are evaluated on 2000 samples per simulation point (1000 samples generated by GenMM or LatMM for corresponding K, 1000 samples from Fashion-MNIST). 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.

6.4.2 Evaluation of Proposed Models

In order to see if the proposed algorithms of GenMM and LatMM help to improve the 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 the assessment of models. The metrics are Inception Score (IS) [11, 154, 185], Frechet Inception Distance (FID) [67], Maximum Mean Discrepancy (MMD) [185] and two-sample test based 1-Nearest Neighbor (1NN) score [112]. 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

Table 6.1: The lowest NLL value of GenMM for curves in Figure 6.4 (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

testing by assuming these two distributions 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 the 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 [185] 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 6.4 and Figure 6.5, respectively. Subsets of MNIST and Fashion-MNIST are used to train our mixture models in order to assess their performance with regard to NLL when a different number of mixture components K is used. All the curves in Figure 6.4 and Figure 6.5 show that NLL decreases as the 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 the optimization at the M-step. In each figure of Figure 6.4 and Figure 6.5, a NLL curve corresponding to a larger total number of mixture components, K, reaches a smaller NLL value after training for the same number of epochs. The results are consistent since as K increases, both GenMM and LatMM have smaller NLL. These results are consistent with our common sense that mixture models fit real data better. The lowest NLL values of curves in Figure 6.4 in training GenMM models are reported in Figure 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 6.6 for MNIST dataset and Figure 6.7 for Fashion-MNIST dataset. Let us first address the results in Figure 6.6. It can be observed that IS increases with the 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 the increase of K. Their trends also saturate with increase in K. The trends obey common practice that performance improves with increasing model complexity, and then deteriorates if the model complexity continues to increase. As in Figure 6.6, similar trends are also observed in Figure 6.7. In some cases, performance for $K = 3$ is poorer than $K = 1$. We assume that the random initialization of parameters in

(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 6.8: Generated samples by GenMM and LatMM for MNIST and Fashion-MNIST datasets.

the mixture models 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.

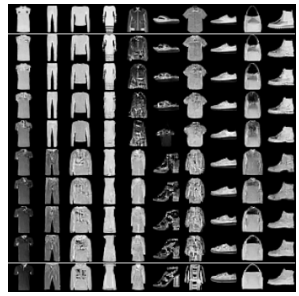
6.4.3 Sample Generation and Interpolation

Next we show generated samples from the proposed models trained with MNIST and Fashion-MNIST in Figure 6.8. In the figure, we show generated samples from GenMM and LatMM for MNIST and Fashion-MNIST datasets. We use different values 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 well in practice.

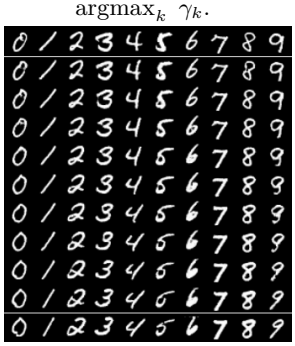
In the second experiment, we explore the power of invertibility for interpolation in the latent domain, i.e., the domain of $\boldsymbol{z}$. We assume the latent domain encodes abstraction of the images. We use samples from MNIST and Fashion-MNIST datasets for this 'interpolation' experiment. In Figure 6.9, we have six subfigures. For each subfigure, the first row and the last row contain the real (true) data samples from MNIST and Fashion-MNIST dataset. In each column, we find latent variables 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 variables as $\alpha\boldsymbol{z}_1 + (1-\alpha)\boldsymbol{z}_2$, where $0 < \alpha < 1$. The latent code, i.e. the latent variable, 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 6.9, we observe the change visually from the first row to last row, i.e. how the first row slowly changes to the last row. We use GenMM for Figure 6.9a, 6.9b, 6.9c, and LatMM for Figure 6.9d, 6.9e, 6.9f. 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 $\boldsymbol{g}_{k^*}^{-1}(\boldsymbol{x}) = \boldsymbol{f}_{k^*}(\boldsymbol{x})$,

(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.

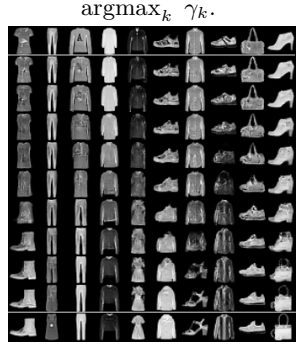


(c) Interpolation by GenMM, K=9. Identity of $\boldsymbol{g}_k$ is chosen by $\text{argmax}_k\ \gamma_k$.



(d) Interpolation by LatMM, K=9.

(e) Interpolation by LatMM, K=9.

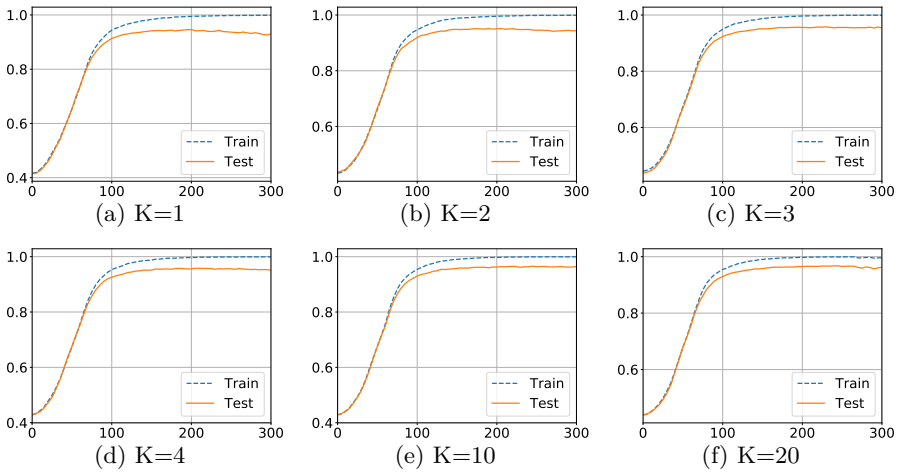


(f) Interpolation by LatMM, K=9.

Figure 6.9: 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.

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 $\alpha \geqslant 0.5$.

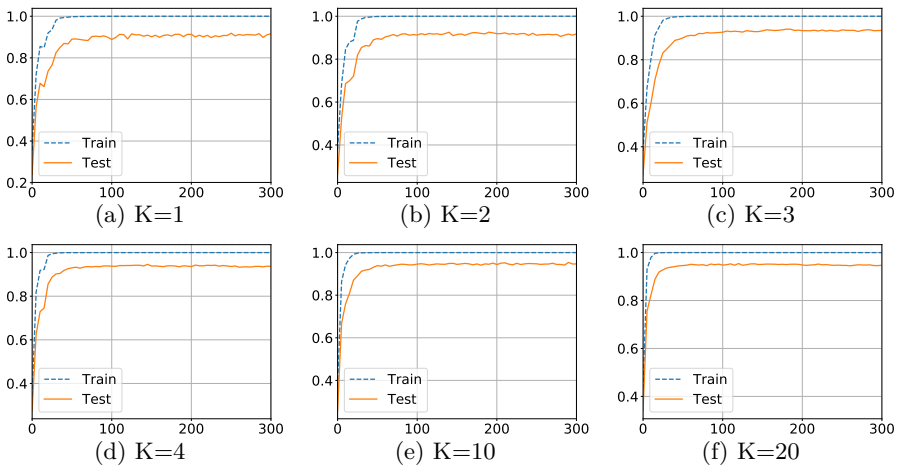
The second experiment on interpolation shows an interesting result for modeling multi-modal data. The distribution of ten digits together in MNIST dataset is expected to be multi-modal. The aspect of multi-modal distribution is addressed using the experimental result shown in Figure 6.9b. We use similar experimental steps as that in Figure 6.9a 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 6.9b, we observe the presence



(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 Letter.



(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 Norb

of digits two and eight, while we expect that the column should be comprised of only images of digit zero. A natural question is why interpolation leads to the generating 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 6.9a is that a sample is produced by a randomly selected generator $\boldsymbol{g}_k(\boldsymbol{z})$ from K possible choices. We compute the interpolated latent code using the same procedure as that in Figure 6.9a, but use the generator where its identity k is randomly sampled from the prior $\boldsymbol{\pi}$ directly. The generated images in this interpolation experiment reveal a clue that each generator models a subset

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 [165]
Satimage	0.8900	0.8975	0.9045	0.9085	0.9105	**0.9160**	0.9090 [76]
Norb	0.9184	0.9257	0.9406	0.9459	0.9538	**0.9542**	0.8920 [153]

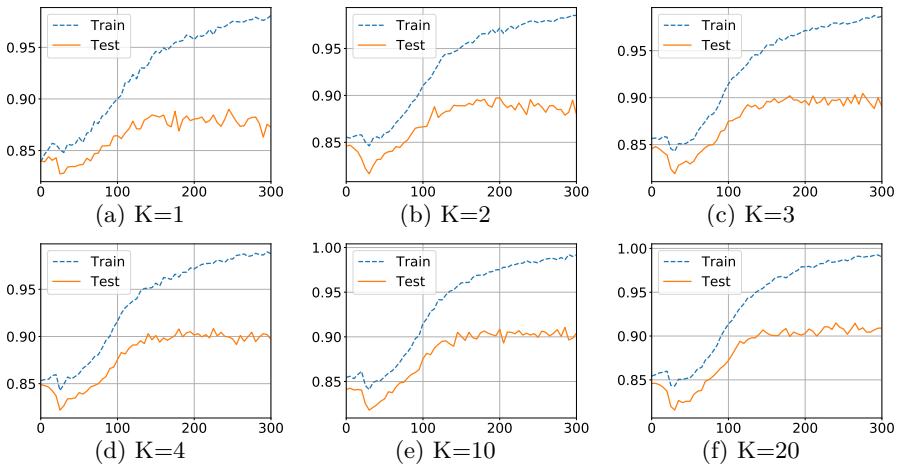


Figure 6.12: Train and Test Accuracy Curves versus Epochs on Dataset Satimage.

of the whole training dataset. We can qualitatively argue that the use of multiple generators helps for modeling the multi-modal distribution.

6.4.4 Application to Classification Task

In this section, we apply our proposed mixture models to classification tasks using the maximum likelihood criterion. We compare classification performance with 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. $p(\boldsymbol{x}; \boldsymbol{\Theta}_y)$ is trained with the y-th class's data. After we have all $p(\boldsymbol{x}; \boldsymbol{\Theta}_y)$, $\forall y = 1, 2, \cdots, Y$ trained, a new sample $\boldsymbol{x}$ is predicted by $\text{argmax}_y\, p(\boldsymbol{x}; \boldsymbol{\Theta}_y)$.

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 epochs 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 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 tasks, 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 6.10, 6.12 and 6.11 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 values of K are shown. In these sets of figures, all accuracy curves climb and flatten 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, the 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.

6.5 Summary

In this chapter, we considered the learning problem in the presence of incomplete data (or unobserved variables) in directed graphs. We introduced normalizing flows that were implemented as neural networks to model the conditional distributions, in order to increase modeling flexibility. Building a model more expressively helps reducing the inherent bias in modeling. The proposed models allow both exact likelihood computation and also efficient sampling.

Learning methods of the proposed models were proposed. Due to the presence of neural network based flows in conditional distribution modeling, expectation maximization was tailored to adapt to their learning and the maximization step was approximately solved with back-propagation technique. The approximation was also partially due to the fact that batch-based optimization had to been used to accommodate large datasets. From the perspective of the numerical evaluations, the proposed models are valid. The numerical results show that the normal statistical behavior of modeling performance versus model complexity remains valid. More expressive models benefit the applications to data representation (data generating) and classification.

The considered models in this chapter assume that an observation instance with its unobserved variable is independent of other observation instances (with their

corresponding hidden variables). This assumption could bring poor models when the true signal is temporal and the independence does not hold. In the next chapter, we would consider how to model and learn a model where this independence is gone.

6.6 Related Work

Using mixture models to get gain in modeling flexibility is a common technique, which is especially useful when the target dataset is multi-modal. This gain comes with unobserved variables. The unobserved variables usually are related to the abstraction of data, which helps us interpret the data. But they also pose challenges in model learning. In the literature, expectation maximization [33] is a standard approach for learning parameters of an explicit mixture model where the likelihood is tractable. The Gaussian mixture model is the most classic one in this framework [14]. Apart from the explicit likelihood, when sufficient statistics are explicitly present in a model, stochastic EM [17] is applicable to accommodate large datasets. Instead of computing expectation step with regard to every datum, stochastic EM carries out expectation step with regard to a randomly selected datum to reduce the complexity of vanilla EM. The benefit of reduced complexity comes with the drawback of increased variance in parameter learning. This issue triggered the stochastic EM with variance reduction [19] and fast incremental EM [80] methods.

When likelihood is not tractable (e.g., implicit probabilistic models), different objectives need to be designed to guide the modeling learning. Generative adversarial networks (GANs) have gained increasing attention in this track [56,154,155,190]. GANs are efficient for generating samples and successful in several applications [100, 154]. In 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 an implicit distribution. The implicit distraction targets at approximating the underlying distribution of the given dataset, via the guidance of a discriminator. To capture diversity in data (multi-modal), the usage of multiple generators has been considered. For instance, multiple Wasserstein GANs [8] are used in [82] with appropriate mutual information based regularization to encourage the diversity of samples generated by different GANs. A mixture GAN approach is proposed in [71] using multiple generators and multi-classification solution to encourage diversity of samples. Multi-agent diverse GAN [50] 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 empirical or appropriate methods have to be used to cope with the unobserved random variable. For instance, extra neural networks are used to model the posterior of the unobserved variables [39, 41].

Another track of mixture modeling is based on the ensembling method that combines weaker learners together to boost the overall performance [58, 168]. 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 for the miss-captured part. However, measuring the difference between the current-step mixture model and the underlying distribution of the dataset quantitatively is a nontrivial task, especially when the likelihood is not tractable. In addition, since incremental building components are used in the mixture modeling, parallel training of model components is not allowed.

Chapter 7

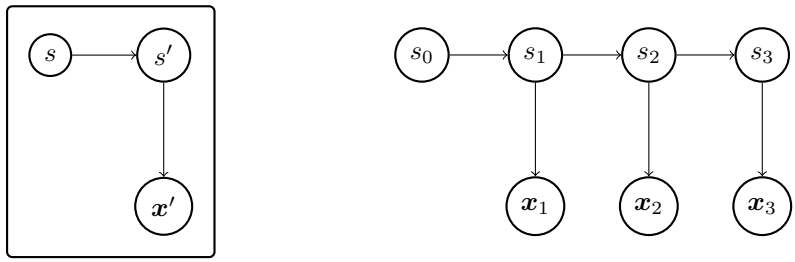
Powering Hidden Markov Model by Normalizing Flows

We have been mainly discussing the topics of modeling and learning when an observation instance is independent of others so far. In another word, the assumption of independent and identically distributed observation instances has been used. In this chapter, we extend our discussion to the topic of modeling and learning for sequential or temporal signals, such as speech signal, trajectories of a robot's movement, DNA sequence, etc.

In modeling a dynamic system that generates sequential signals, we are usually interested in reasoning about the system state that evolves over time. For simplification, the timeline over which a dynamic system generates a sequential signal is discretized, i.e. time is sliced. Thus for each time instant t, we can take a measurement of the dynamic system, which corresponds to an observed variable $\boldsymbol{x}_t$. Due to the limitation of our measure accuracy or the abstraction in modeling itself, the state of the system at this time instant is not directly available, which corresponds to an unobserved or hidden variable s_t.

The straightforward issue is the complexity of the graphical representation of dependencies for a dynamic system, which in turn affects model learning and inference. As the dynamic system evolves, the dependencies (correspondence to edges in the graphical model) can be arbitrarily complex since the system states on any two time instants can be dependent if no constraint is enforced. The complexity of the resulting graphical model can be too high for practical usages. Therefore, constraints are required to allow reasonable model learning and state estimations. The most widely used constraint is probably the Markov assumption, i.e. $(\boldsymbol{x}_{t+1}, s_{t+1})$ is independent of $(\boldsymbol{x}_{1:t-1}, s_{1:t-1})$ if $(\boldsymbol{x}_t, s_t)$ is given. This assumption indeed reduces the complexity of graphical structures in modeling a dynamic system, since we do not need to draw any edges between variables with time interval larger than one in the graphical representation.

The other issue is the parameterization of the model of the dynamic system. If



(a) The template model of HMM.

(b) The unrolled HMM instance to time $t = 3$.

Figure 7.1: From HMM template to instance.

we directly model $\{\boldsymbol{x}_t, s_t, t = 1, 2, \cdots, T\}$ jointly, the parameterization would grow exponentially as the system evolves over time. A good solution to this issue is to introduce the *template* concept into the graphical model. In the assumption, the variable $\boldsymbol{x}_t$ or s_t becomes an instance of a *template variable*. More importantly, each dependency between two sequential time instants becomes an instance of a *template factor*. When the dynamic system evolves from time t to $t + 1$, we only need to instantiate from the template variables and template factors without adding new parameters to the model. Template based dynamic Bayesian networks belong to such kind of models.

We add one more assumption that variable $\boldsymbol{x}_t$ is independent of the rest of variables $\{\boldsymbol{x}_{t'}, s_{t'}, t' \neq t\}$ if the state s_t is given, and reduce a dynamic Bayesian network into the classic hidden Markov model (HMM). Similar to Chapter 6, we bring the normalizing flows into the *template factors* modeling to increase the flexibility of HMM in modeling dynamic systems in this chapter.

7.1 Hidden Markov Model

We first illustrate the concepts of Markov assumption and *template* with an example before giving the definition of HMM. As shown in Figure 7.1a, the plate denotes the template within which the generic variables and their dependencies (template variables and factors) are represented. As a dynamic system evolves with time, the hidden variable s and observed variable $\boldsymbol{x}$ can be instantiated for each time slice t, along with their dependencies. Figure 7.1b is an instantiated example up to $t = 3$ from the generic template. The Markov assumption is embedded in the HMM instances from the template since Markov independence is fulfilled no matter how long the system evolves. More importantly, the conditional probabilities $p(s_{t+1}|s_t),\ \forall\ t$, instantiated from template factors (defined with the directed edge $s \rightarrow s'$ in Figure 7.1a), share the same parameterization. Similarly, the probabilities $p(\boldsymbol{x}_t|s_t),\ \forall\ t$ share parameterization with the template factor defined with edge $s' \rightarrow \boldsymbol{x}'$. This definition circumvents the exponential growth of parameterization.

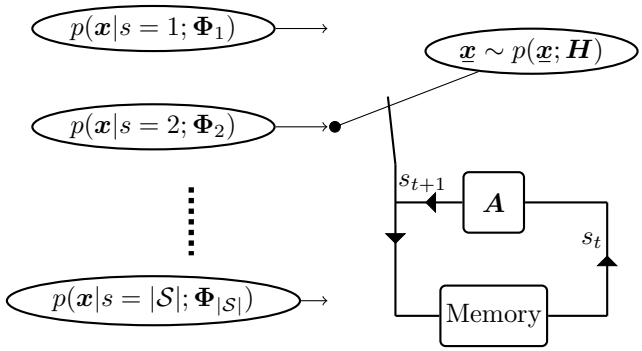


Figure 7.2: HMM model: a generative illustration.

With the intuition from Figure 7.1, we now define the HMM with its parameterization. A HMM $\boldsymbol{H}$ defined in a hypothesis space $\mathcal{H}$, i.e. $\boldsymbol{H} \in \mathcal{H}$, is capable to model a time-span signal $\underline{\boldsymbol{x}} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T]^\intercal$, where $\boldsymbol{x}_t \in \mathbb{R}^N$ is the N-dimensional signal at time t, $[\cdot]^\intercal$ denotes transpose, and T denotes the time length[1]. We define the hypothesis set of HMM as $\mathcal{H} := \{\boldsymbol{H} | \boldsymbol{H} = \{\mathcal{S}, \boldsymbol{q}, \boldsymbol{A}, p(\boldsymbol{x}|s;\boldsymbol{\Phi}_s)\}\}$, where

- $\mathcal{S}$ is the set of hidden states of $\boldsymbol{H}$.
- $\boldsymbol{q} = \left[q_1, q_2, \cdots, q_{|\mathcal{S}|}\right]^\intercal$ is the initial state distribution of $\boldsymbol{H}$ with $|\mathcal{S}|$ as cardinality of $\mathcal{S}$. For $i \in \mathcal{S}$, $q_i = p(s_1 = i; \boldsymbol{H})$. We use s_t to denote the state s at time t.
- $\boldsymbol{A}$ matrix of size $|\mathcal{S}| \times |\mathcal{S}|$ is the transition matrix of states in $\boldsymbol{H}$. That is, $\forall i, j \in \mathcal{S}$, $\boldsymbol{A}_{i,j} = p(s_{t+1} = j | s_t = i; \boldsymbol{H})$.
- For a given hidden state s, the density function of the observable signal is $p(\boldsymbol{x}|s;\boldsymbol{\Phi}_s)$, where $\boldsymbol{\Phi}_s$ is the parameter set that defines this probabilistic model. Denote $\boldsymbol{\Phi} = \{\boldsymbol{\Phi}_s | s \in \mathcal{S}\}$.

A HMM for signal representation is illustrated in Figure 7.2. The model assumption is that different time instances of the signal $\underline{\boldsymbol{x}}$ are generated by different signal sources where each signal source is associated with a hidden state of the HMM. In the framework of HMM, at each time instance t, signal $\boldsymbol{x}_t$ is assumed to be generated by a distribution with density function $p(\boldsymbol{x}_t|s_t;\boldsymbol{\Phi}_{s_t})$, and s_t is decided by the hidden Markov process. Putting these together gives us the probabilistic model $p(\underline{\boldsymbol{x}};\boldsymbol{H})$.

Remark 7.1. *The graphical model of an HMM is very similar to that of a CRF, especially linear-chain CRF. The straightforward difference between an HMM and a linear-chain CRF is the graphical model representation. An HMM is a pure directed probabilistic graphical model while there are both directed and undirected edges in a*

[1]The length for sequential data varies.

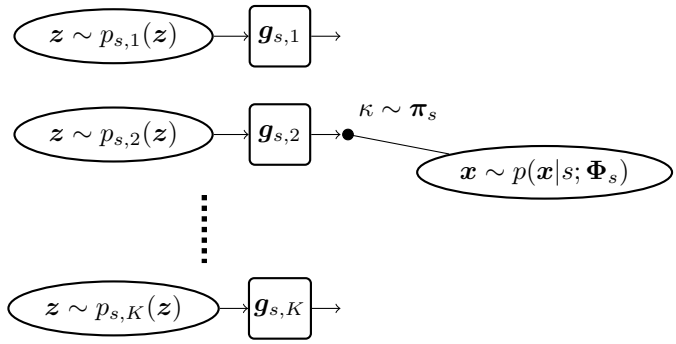


Figure 7.3: Template factor of GenHMM: conditional probabilistic model of observed variable $\boldsymbol{x}$ given hidden state s.

CRF. In a nutshell, omitting observed variables and their variables, the remaining variables and their edges form an MRF in a CRF. This difference also affects the model training. An HMM is a generative model and models $p(\underline{\boldsymbol{x}}, \underline{s})$ jointly. Its training target is to maximize the joint probability. In contrast, a CRF models a conditional distribution $p(\underline{s}|\underline{\boldsymbol{x}})$ and is learned via discriminative training. See [163] for a more detailed discussion on the difference between them.

7.2 Generator-mixed HMM

In this section, we introduce a generator-mixed hidden Markov model (GenHMM). The highlight of GenHMM is that its state probability distributions are induced by normalizing flows for flexible dependency modeling between hidden states and observed signals. The learning of GenHMM is discussed here at a high level while the detailed solutions would be presented in Section 7.3.

7.2.1 Generators of GenHMM

In this section, we introduce normalizing flows to each state probabilistic models of our GenHMM, which models the conditional probability of observation given a hidden state. Recall that $\boldsymbol{x} \in \mathbb{R}^N$. The subscript is omitted when it does not cause ambiguity. The probabilistic model of GenHMM for each hidden state is a mixture of K flow generators that are implemented by neural networks, where K is a positive integer. The probabilistic model of a state $s \in \mathcal{S}$ is then given by

$$p(\boldsymbol{x}|s;\boldsymbol{\Phi}_s) = \sum_{\kappa=1}^{K} \pi_{s,\kappa} p(\boldsymbol{x}|s,\kappa;\boldsymbol{\theta}_{s,\kappa}), \tag{7.1}$$

where κ is a random variable following a categorical distribution, with probability $\pi_{s,\kappa} = p(\kappa|s;\boldsymbol{H})$. Naturally $\sum_{\kappa=1}^{K} \pi_{s,\kappa} = 1$. Denote $\boldsymbol{\pi}_s = [\pi_{s,1}, \pi_{s,2}, \cdots, \pi_{s,K}]^\intercal$. In (7.1), $p(\boldsymbol{x}|s,\kappa;\boldsymbol{\theta}_{s,\kappa})$ is further defined as the induced distribution by a generator

$\boldsymbol{g}_{s,\kappa} : \mathbb{R}^N \to \mathbb{R}^N$, such that $\boldsymbol{x} = \boldsymbol{g}_{s,\kappa}(\boldsymbol{z})$, where $\boldsymbol{z}$ is a latent variable following a distribution with density function $p_{s,\kappa}(\boldsymbol{z})$. Generator $\boldsymbol{g}_{s,\kappa}$ is parameterized by $\boldsymbol{\theta}_{s,\kappa}$. Let us denote the collection of the parameter sets of generators for state s as $\boldsymbol{\theta}_s = \{\boldsymbol{\theta}_{s,\kappa} | \kappa = 1, 2, \cdots, K\}$. For a flow generator $\boldsymbol{g}_{s,\kappa}$, we have

$$p(\boldsymbol{x}|s, \kappa; \boldsymbol{\theta}_{s,\kappa}) = p_{s,\kappa}(\boldsymbol{z}) \left| \det \left(\frac{\partial \boldsymbol{g}_{s,\kappa}(\boldsymbol{z})}{\partial \boldsymbol{z}} \right) \right|^{-1}, \text{ with } \boldsymbol{x} = \boldsymbol{g}_{s,\kappa}(\boldsymbol{z}), \tag{7.2}$$

which is discussed in detail in Section 6.1.

The signal generating process of the probability distribution for a state s of GenHMM is shown in Figure 7.3, in which the generator identity is up to the random variable κ. This serves as the template factor in instantiating the chain of a HMM. The normalizing flows with neural network realizations of their coupling mapping (6.5) offer a richer probability density function space and higher model expressivity. This naturally enriches the feasible space $\mathcal{H}$. Putting these together with the typical Markov assumption, the template model of GenHMM is illustrated in Figure 7.4. As a dynamic system evolves, we can instantiate from the template model in Figure 7.4 to create the dynamic chain of graphical model representation. Similar to a typical HMM, the parameterization of GenHMM can be fully explained by its template model and does not grow exponentially when instantiating over the evolving system.

7.2.2 Learning with the EM framework

Assume the sequential signal $\underline{\boldsymbol{x}}$ follows the true distribution $p^*(\underline{\boldsymbol{x}})$, which is unknown. We would like to use GenHMM to model this distribution. Alternatively, we are addressing

$$\min_{\boldsymbol{H} \in \mathcal{H}} \mathrm{KL}(p^*(\underline{\boldsymbol{x}}) \| p(\underline{\boldsymbol{x}}; \boldsymbol{H})). \tag{7.3}$$

For practical consideration, we only have access to the samples of $p^*(\underline{\boldsymbol{x}})$, i.e. a dataset consisting of samples drawn from this distribution. For the given dataset, we denote its empirical distribution by $\hat{p}(\underline{\boldsymbol{x}}) = \frac{1}{R} \sum_{r=1}^{R} \delta_{\underline{\boldsymbol{x}}^r}(\underline{\boldsymbol{x}})$, where R denotes the total number of sequential samples and superscipt $(\cdot)^r$ denotes the index of r-th sequential signal. Similar to the analysis in Section 2.6, the KL divergence minimization problem can be reduced to a likelihood maximization problem

$$\operatorname*{argmax}_{\boldsymbol{H} \in \mathcal{H}} \frac{1}{R} \sum_{r=1}^{R} \log p(\underline{\boldsymbol{x}}^r; \boldsymbol{H}). \tag{7.4}$$

For the likelihood maximization, the first problem that we need to address is how to deal with the hidden sequential variables of model $\boldsymbol{H}$, namely $\underline{\boldsymbol{s}} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \cdots, \boldsymbol{s}_T]^\intercal$ and $\underline{\boldsymbol{\kappa}} = [\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2, \cdots, \boldsymbol{\kappa}_T]^\intercal$. For a sequentially observable variable $\underline{\boldsymbol{x}}$, $\underline{\boldsymbol{s}}$ is the hidden state sequence corresponding to $\underline{\boldsymbol{x}}$, and $\underline{\boldsymbol{\kappa}}$ is the hidden variable sequence representing the generator identity sequence that actually generates $\underline{\boldsymbol{x}}$.

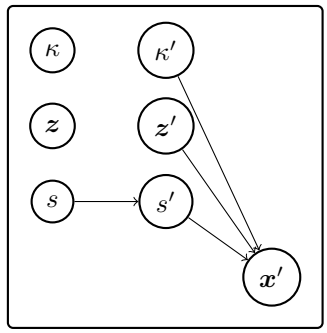


Figure 7.4: The template model of GenHMM.

Note that for a sequentially observable $\underline{\boldsymbol{x}}$, there is also a sequential $\underline{\boldsymbol{z}}$ brought by flows. But the mapping between $\boldsymbol{x}$ and $\boldsymbol{z}$ is deterministic for fixed $p_{s,\kappa}(\boldsymbol{z})$ when the corresponding $\boldsymbol{s}$ and κ are given, which is defined by $\boldsymbol{g}_{s,\kappa}$.

Since directly maximizing likelihood is not an option for our problem in (7.4), we address this problem with the EM framework, similar to the way we dealt with hidden variable in Section 6.2. This divides our problem into two iterative steps: i) using the joint posterior of hidden variable sequences $\underline{\boldsymbol{s}}$ and $\underline{\boldsymbol{\kappa}}$ to obtain an "expected likelihood" of the observable variable sequence $\underline{\boldsymbol{x}}$, i.e. the E-step; ii) maximizing the expected likelihood with regard to the model $\boldsymbol{H}$, i.e. the M-step. Assume model $\boldsymbol{H}$ is at a configuration of $\boldsymbol{H}^{\text{old}}$, we formulate these two steps as follows.

- E-step: the expected likelihood function

$$\mathcal{Q}(\boldsymbol{H};\boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\, p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H})\right], \tag{7.5}$$

  where $\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}[\cdot]$ denotes the expectation operator by distribution $\hat{p}(\underline{\boldsymbol{x}})$ and $p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})$.

- M-step: the maximization step

$$\max_{\boldsymbol{H}}\ \mathcal{Q}(\boldsymbol{H};\boldsymbol{H}^{\text{old}}). \tag{7.6}$$

The problem (7.6) can be reformulated as

$$\max_{\boldsymbol{H}}\ \mathcal{Q}(\boldsymbol{H};\boldsymbol{H}^{\text{old}}) = \max_{\boldsymbol{q}}\ \mathcal{Q}(\boldsymbol{q};\boldsymbol{H}^{\text{old}}) + \max_{\boldsymbol{A}}\ \mathcal{Q}(\boldsymbol{A};\boldsymbol{H}^{\text{old}}) + \max_{\boldsymbol{\Phi}}\ \mathcal{Q}(\boldsymbol{\Phi};\boldsymbol{H}^{\text{old}}), \tag{7.7}$$

where the decomposed optimization problems are

$$\mathcal{Q}(\boldsymbol{q};\boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\, p(s_1;\boldsymbol{H})\right], \tag{7.8}$$

$$\mathcal{Q}(\boldsymbol{A};\boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\sum_{t=1}^{T-1}\log\, p(s_{t+1}|s_t;\boldsymbol{H})\right], \tag{7.9}$$

$$\mathcal{Q}(\boldsymbol{\Phi};\boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\, p(\underline{\boldsymbol{x}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{s}};\boldsymbol{H})\right]. \tag{7.10}$$

We can see that the solution of $\boldsymbol{H}$ depends on the posterior probability $p(\underline{\boldsymbol{s}}|\underline{\boldsymbol{x}};\boldsymbol{H})$.

Remark 7.2 (On posterior)**.** *Although the principle here in dealing with the hidden variables is similar to that in Section 6.2, new issues arise for dynamic systems. Due to the presence of incomplete observations (hidden variables), we need to 'complete' the missing information by evaluating the posteriors of hidden variables for each partial observation $\boldsymbol{x}_t$. In GenHMM, for each $\underline{\boldsymbol{x}}$, there are missing hidden sequences ($\underline{\boldsymbol{s}}$, $\underline{\boldsymbol{\kappa}}$). Although the evaluation of the posterior according to Bayes theorem is straightforward, the computational complexity of $p(\underline{\boldsymbol{s}}|\underline{\boldsymbol{x}};\boldsymbol{H})$ grows exponentially with the length of $\underline{\boldsymbol{s}}$. Therefore, we employ forward-backward algorithm [14] to do the posterior computation efficiently. As we will detail in the next section, what is needed to formulate the problem, are actually the $p(s|\underline{\boldsymbol{x}};\boldsymbol{H})$ and $p(s,\kappa|\underline{\boldsymbol{x}};\boldsymbol{H})$. For the joint posterior $p(s,\kappa|\underline{\boldsymbol{x}};\boldsymbol{H})$, it can be computed by the Bayes rule when posterior of hidden state is available.*

Remark 7.3 (Connection to other methods or models)**.** *The forward-background algorithm mentioned in Remark 7.2 does exact inference to posteriors, i.e. the exact computation of posterior probabilities. At times, the posterior inference is addressed by approximate inference. For instance, it might be due to the modeling of a dynamic system that does not allow exact inference. [140] applies expectation propagation [117] to non-linear dynamic models with continuous hidden states.*

The considered hidden states are discrete in GenHMM. For linear dynamic models with continuous hidden states, the best-known approach is the Kalman filter [79] and smoothing [146]. To generalize the problem into non-linear dynamic models, which is practical consideration since many real-world problems are indeed non-linear, known classical methods such as extended Kalman filter [110] and unscented Kalman filter [176] are applicable. Further exploration in recent literature also considered more flexible models such as neural networks in dynamic models. For instance, [46] combined the inference process of Kalman filter and graph neural network in pursuit of more flexible non-linear dynamic models, and [84] proposed hierarchical hidden state space to model and infer hidden structures with multi-layer perceptrons.

With such a solution framework ready, we detail the practical learning algorithm for GenHMM with normalizing flows embedded in next section.

7.3 Practical Solution to GenHMM

In this section, we detail the solution for realizing and learning GenHMM. The convergence of GenHMM is also discussed in this section.

7.3.1 Realizing $g_{s,\kappa}$ by a Flow Model

Each generator $\boldsymbol{g}_{s,\kappa}$ is realized as a feed-forward neural netowrk. We define generator $\boldsymbol{g}_{s,\kappa}$ as a L-layer flow model and formulate its mapping by layer-wise concatenation

$$\boldsymbol{g}_{s,\kappa} = \boldsymbol{g}_{s,\kappa}^{[L]} \circ \boldsymbol{g}_{s,\kappa}^{[L-1]} \circ \cdots \circ \boldsymbol{g}_{s,\kappa}^{[1]}, \tag{7.11}$$

where superscript $[l]$ denotes the layer index and $\circ$ denotes mapping concatenation. As detailed in Section 6.1, generator $\boldsymbol{g}_{s,\kappa}$ is invertible and denote its inverse mapping as $\boldsymbol{f}_{s,\kappa} = \boldsymbol{g}_{s,\kappa}^{-1}$. Then (7.2) can be rewritten as

$$\log p(\boldsymbol{x}|s,\kappa;\boldsymbol{\theta}_{s,\kappa}) = \log p_{s,\kappa}(\boldsymbol{f}_{s,\kappa}(\boldsymbol{x})) + \log \left| \det \left(\frac{\partial \boldsymbol{f}_{s,\kappa}(\boldsymbol{x})}{\partial \boldsymbol{z}} \right) \right|. \tag{7.12}$$

By decomposing the flow model into layer-wise mapping, the part of Jacobian matrix determinant becomes

$$\det(\nabla \boldsymbol{f}_{s,\kappa}) = \textstyle\prod_{l=1}^{L} \det(\nabla \boldsymbol{f}_{s,\kappa}^{[l]}), \tag{7.13}$$

where $\nabla \boldsymbol{f}_{s,\kappa}^{[l]}$ is the Jacobian of the mapping from the l-th layer to the $(l-1)$-th layer, i.e., the inverse transformation. Then (7.12) can be further rewritten as

$$\log p(\boldsymbol{x}|s,\kappa;\boldsymbol{\theta}_{s,\kappa}) = \log p_{s,\kappa}(\boldsymbol{f}_{s,\kappa}(\boldsymbol{x})) + \sum_{l=1}^{L} \log \left| \det \left(\nabla \boldsymbol{f}_{s,\kappa}^{[l]} \right) \right|. \tag{7.14}$$

7.3.2 Learning of GenHMM

In this section, we address the problem of learning GenHMM.

Learning of Generators and Their Weights

The learning of mixture of generators is actually to solve the problem in (7.10), which can be further divided into two subproblems: i) generator learning; ii) mixture weights of generators learning. Let us define notations: $\boldsymbol{\Pi} = \{\boldsymbol{\pi}_s | s \in \mathcal{S}\}$, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_s | s \in \mathcal{S}\}$. Then the problem in (7.10) becomes

$$\max_{\boldsymbol{\Phi}} \mathcal{Q}(\boldsymbol{\Phi}; \boldsymbol{H}^{\text{old}}) = \max_{\boldsymbol{\Pi}} \mathcal{Q}(\boldsymbol{\Pi}; \boldsymbol{H}^{\text{old}}) + \max_{\boldsymbol{\Theta}} \mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}}), \tag{7.15}$$

where

$$\mathcal{Q}(\boldsymbol{\Pi}; \boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}), p(\underline{\boldsymbol{s}}, \underline{\boldsymbol{\kappa}} | \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}})} \left[\log\, p(\underline{\boldsymbol{\kappa}} | \underline{\boldsymbol{s}}; \boldsymbol{H}) \right], \tag{7.16}$$

$$\mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}}) = \mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}), p(\underline{\boldsymbol{s}}, \underline{\boldsymbol{\kappa}} | \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}})} \left[\log\, p(\underline{\boldsymbol{x}} | \underline{\boldsymbol{s}}, \underline{\boldsymbol{\kappa}}; \boldsymbol{H}) \right]. \tag{7.17}$$

We firstly address the generator learning problem, i.e. $\max_{\boldsymbol{\Theta}} \mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}})$. This is boiled down to maximize the cost function of neural networks that can be formulated as

$$\begin{aligned}
&\mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}}) \\
=&\frac{1}{R}\sum_{r=1}^{R}\sum_{\underline{\boldsymbol{s}}^r}\sum_{\underline{\boldsymbol{\kappa}}^r} p(\underline{\boldsymbol{s}}^r, \underline{\boldsymbol{\kappa}}^r|\underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \sum_{t=1}^{T^r} \log\, p(\boldsymbol{x}_t^r|s_t^r, \kappa_t^r; \boldsymbol{H}) \\
=&\frac{1}{R}\sum_{r=1}^{R}\sum_{t=1}^{T^r}\sum_{s_t^r=1}^{|\mathcal{S}|}\sum_{\kappa_t^r=1}^{K} p(s_t^r|\underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) p(\kappa_t^r|s_t^r, \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \log\, p(\boldsymbol{x}_t^r|s_t^r, \kappa_t^r; \boldsymbol{H}), \quad (7.18)
\end{aligned}$$

where T^r is the length of the r-th sequential data. In (7.18), the state posterior $p(s_t|\underline{\boldsymbol{x}}, \boldsymbol{H}^{\text{old}})$ is computed by forward-backward algorithm. The posterior of κ is

$$\begin{aligned}
p(\kappa|s, \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}}) &= \frac{p(\kappa, \underline{\boldsymbol{x}}|s; \boldsymbol{H}^{\text{old}})}{p(\underline{\boldsymbol{x}}|s, \boldsymbol{H}^{\text{old}})} \\
&= \frac{\pi_{s,\kappa}^{\text{old}} p(\boldsymbol{x}|s, \kappa, \boldsymbol{H}^{\text{old}})}{\sum_{\kappa=1}^{K} \pi_{s,\kappa}^{\text{old}} p(\boldsymbol{x}|s, \kappa, \boldsymbol{H}^{\text{old}})}, \quad (7.19)
\end{aligned}$$

where the last equation is due to the fact that $\boldsymbol{x}_t$ among sequence $\underline{\boldsymbol{x}}$ is conditional independent of $\boldsymbol{x}_{t'}$ given s_t and κ_t for $t \neq t'$.

By substituting (7.14) into (7.18), we have cost function for neural networks as

$$\begin{aligned}
&\mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}}) \\
=&\frac{1}{R}\sum_{r=1}^{R}\sum_{t=1}^{T^r}\sum_{s_t^r=1}^{|\mathcal{S}|}\sum_{\kappa_t^r=1}^{K} p(s_t^r|\underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) p(\kappa_t^r|s_t^r, \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \\
&\left[\log\, p_{s_t^r,\kappa_t^r}(\boldsymbol{f}_{s_t^r,\kappa_t^r}(\boldsymbol{x}_t^r)) + \sum_{l=1}^{L} \log\, |\det(\nabla \boldsymbol{f}_{s,\kappa}^{[l]})|\right]. \quad (7.20)
\end{aligned}$$

The generators of GenHMM simply use standard Gaussian distribution for latent variables $\boldsymbol{z} \sim p_{s,\kappa}(\boldsymbol{z})$. Since training dataset can be too large to do whole-dataset iterations, batch-size stochastic gradient descent can be used to maximize $\mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}})$ with regard to parameters of generators.

In what follows we address the problem $\max_{\boldsymbol{\Pi}} \mathcal{Q}(\boldsymbol{\Pi}; \boldsymbol{H}^{\text{old}})$ in our generative model learning. The conditional distribution of hidden variable κ, $\pi_{s,\kappa} = p(\kappa|s; \boldsymbol{H})$, is obtained by solving the following problem

$$\begin{aligned}
\pi_{s,\kappa} = \underset{\pi_{s,\kappa}}{\text{argmax}}\ &\mathcal{Q}(\boldsymbol{\Pi}; \boldsymbol{H}^{\text{old}}) \quad (7.21) \\
s.t.\ &\sum_{\kappa=1}^{K} \pi_{s,\kappa} = 1, \forall s = 1, 2, \cdots, |\mathcal{S}|.
\end{aligned}$$

To solve problem (7.21), we formulate its Lagrange function as

$$\mathcal{F} = \mathcal{Q}(\boldsymbol{\Pi}; \boldsymbol{H}^{\text{old}}) + \sum_{s=1}^{|\mathcal{S}|} \lambda_s \left(1 - \sum_{\kappa=1}^{K} \pi_{s,\kappa}\right). \tag{7.22}$$

Solving $\frac{\partial \mathcal{F}}{\partial \pi_{s,\kappa}} = 0$ gives

$$\pi_{s,\kappa} = \frac{1}{\lambda_s} \sum_{r=1}^{R} \sum_{t=1}^{T^r} p(s_t^r = s, \kappa_t^r = \kappa | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}). \tag{7.23}$$

With condition $\sum_{\kappa=1}^{K} \pi_{s,\kappa} = 1, \forall s = 1, 2, \cdots, |\mathcal{S}|$, we have

$$\lambda_s = \sum_{\kappa=1}^{K} \sum_{r=1}^{R} \sum_{t=1}^{T^r} p(s_t^r = s, \kappa_t^r = \kappa | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}). \tag{7.24}$$

Then the solution to (7.21) is

$$\pi_{s,\kappa} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^r} p(s_t^r = s, \kappa_t^r = \kappa | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}})}{\sum_{k=1}^{K} \sum_{r=1}^{R} \sum_{t=1}^{T^r} p(s_t^r = s, \kappa_t^r = k | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}})}, \tag{7.25}$$

where

$$p(s, \kappa | \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}}) = p(s | \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}}) p(\kappa | s, \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}}). \tag{7.26}$$

Here $p(s | \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}})$ can be computed by forward-backward algorithm, while $p(\kappa | s, \underline{\boldsymbol{x}}; \boldsymbol{H}^{\text{old}})$ is given by (7.19).

With the generator learning obtained, it remains to solve the initial distribution update and transition matrix update of HMM in GenHMM, i.e. the problem (7.8) and (7.9). These two problems are basically two constrained optimization problems. The solutions to them are available in literature [14]. But to keep the learning algorithm for GenHMM complete, we give the update rules for $\boldsymbol{q}$ and $\boldsymbol{A}$ as follows.

Initial Probability Update

The problem in (7.8) can be reformulated as

$$\begin{aligned}
&\mathcal{Q}(\boldsymbol{q}; \boldsymbol{H}^{\text{old}}) \\
=&\frac{1}{R} \sum_{r=1}^{R} \sum_{\underline{\boldsymbol{s}}^r} p(\underline{\boldsymbol{s}}^r | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \log p(s_1^r; \boldsymbol{H}) \\
=&\frac{1}{R} \sum_{r=1}^{R} \sum_{s_1^r=1}^{|\mathcal{S}|} \sum_{s_2^r=1}^{|\mathcal{S}|} \cdots \sum_{s_{T^r}^r}^{|\mathcal{S}|} p(s_1^r, s_2^r, \cdots, s_{T^r}^r | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \log p(s_1^r) \\
=&\frac{1}{R} \sum_{r=1}^{R} \sum_{s_1^r=1}^{|\mathcal{S}|} p(s_1^r | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \log p(s_1^r; \boldsymbol{H}).
\end{aligned} \tag{7.27}$$

$p(s_1^r; \boldsymbol{H})$ is the probability of initial state of GenHMM for r-th sequential sample. Actually $q_i = p(s_1 = i; \boldsymbol{H})$, $i = 1, 2, \cdots, |\mathcal{S}|$. Solution to the problem

$$\boldsymbol{q}=\underset{\boldsymbol{q}}{\operatorname{argmax}}\ \mathcal{Q}(\boldsymbol{q}; \boldsymbol{H}^{\text{old}}),\ \text{s.t.} \sum_{i=1}^{|\mathcal{S}|} q_i = 1, q_i \geqslant 0, \forall i. \tag{7.28}$$

is

$$q_i = \frac{1}{R} \sum_{r=1}^{R} p(s_1^r = i | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}), \forall\ i = 1, 2, \cdots, |\mathcal{S}|. \tag{7.29}$$

Transition Probability Update

The problem (7.9) can be reformulated as

$$\begin{aligned}
&\mathcal{Q}(\boldsymbol{A}; \boldsymbol{H}^{\text{old}}) \\
=&\sum_{r=1}^{R} \sum_{\underline{\boldsymbol{s}}^r} p(\underline{\boldsymbol{s}}^r | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \sum_{t=1}^{T^r-1} \log\, p(s_{t+1}^r | s_t^r; \boldsymbol{H}) \\
=&\sum_{r=1}^{R} \sum_{t=1}^{T^r-1} \sum_{s_t^r=1}^{|\mathcal{S}|} \sum_{s_{t+1}^r=1}^{|\mathcal{S}|} p(s_t^r, s_{t+1}^r | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}) \log\, p(s_{t+1}^r | s_t^r; \boldsymbol{H}).
\end{aligned} \tag{7.30}$$

Since $\boldsymbol{A}_{i,j} = p(s_{t+1}^r = j | s_t^r = i; \boldsymbol{H})$ is the element of transition matrix $\boldsymbol{A}$, the solution to the problem

$$\begin{aligned}
\boldsymbol{A} =&\underset{\boldsymbol{A}}{\operatorname{argmax}}\ \mathcal{Q}(\boldsymbol{A}; \boldsymbol{H}^{\text{old}}) \\
&\text{s.t.}\ \ \boldsymbol{A} \cdot \boldsymbol{1} = \boldsymbol{1}, \boldsymbol{A}_{i,j} \geqslant 0\ \forall i, j,
\end{aligned} \tag{7.31}$$

is

$$\boldsymbol{A}_{i,j} = \frac{\bar{\xi}_{i,j}}{\sum_{k=1}^{|\mathcal{S}|} \bar{\xi}_{i,k}}, \tag{7.32}$$

where

$$\bar{\xi}_{i,j} = \sum_{r=1}^{R} \sum_{t=1}^{T^r-1} p(s_t^r = i, s_{t+1}^r = j | \underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}}). \tag{7.33}$$

7.3.3 On Convergence of GenHMM

In pursuit of representing a dataset by GenHMM, we are interested if the learning solution discussed in Section 7.3.2 would converge. The properties on GenHMM's convergence are analyzed as follows.

Proposition 7.1. *Assume that parameter* $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_{s,\kappa} | s \in \mathcal{S}, \kappa = 1, 2, \cdots, K\}$ *is in a compact set,* $\boldsymbol{f}_{s,\kappa}$ *and* $\nabla \boldsymbol{f}_{s,\kappa}$ *are continuous with regard to* $\boldsymbol{\theta}_{s,\kappa}$ *in GenHMM. Then GenHMM converges.*

Proof. We begin with the comparison of log-likelihood evaluated under $\boldsymbol{H}^{\text{new}}$ and $\boldsymbol{H}^{\text{old}}$. The log-likelihood of dataset given by $\hat{p}(\underline{\boldsymbol{x}})$ can be reformulated as

$$\begin{aligned}
&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[\log p(\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{new}})\right]\\
=&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\frac{p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H}^{\text{new}})}{p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\right]\\
&+\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[KL(p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\|p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{new}}))\right],
\end{aligned}$$

where the first term on the right hand side of the above inequality can be further written as

$$\begin{aligned}
&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\frac{p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H}^{\text{new}})}{p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\right]\\
=&\mathcal{Q}(\boldsymbol{H}^{\text{new}};\boldsymbol{H}^{\text{old}})+\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\right].
\end{aligned}$$

According to Section 7.3.2, the optimization problems give

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{q}^{\text{new}};\boldsymbol{H}^{\text{old}})&\geqslant\mathcal{Q}(\boldsymbol{q}^{\text{old}};\boldsymbol{H}^{\text{old}}),\\
\mathcal{Q}(\boldsymbol{A}^{\text{new}};\boldsymbol{H}^{\text{old}})&\geqslant\mathcal{Q}(\boldsymbol{A}^{\text{old}};\boldsymbol{H}^{\text{old}}),\\
\mathcal{Q}(\boldsymbol{\Pi}^{\text{new}};\boldsymbol{H}^{\text{old}})&\geqslant\mathcal{Q}(\boldsymbol{\Pi}^{\text{old}};\boldsymbol{H}^{\text{old}}),\\
\mathcal{Q}(\boldsymbol{\Theta}^{\text{new}};\boldsymbol{H}^{\text{old}})&\geqslant\mathcal{Q}(\boldsymbol{\Theta}^{\text{old}};\boldsymbol{H}^{\text{old}}).
\end{aligned}$$

Since

$$\begin{aligned}
&\mathcal{Q}(\boldsymbol{H}^{\text{new}};\boldsymbol{H}^{\text{old}})\\
=&\mathcal{Q}(\boldsymbol{q}^{\text{new}};\boldsymbol{H}^{\text{old}})+\mathcal{Q}(\boldsymbol{A}^{\text{new}};\boldsymbol{H}^{\text{old}})+\mathcal{Q}(\boldsymbol{\Pi}^{\text{new}};\boldsymbol{H}^{\text{old}})+\mathcal{Q}(\boldsymbol{\Theta}^{\text{new}};\boldsymbol{H}^{\text{old}}),
\end{aligned}$$

it gives

$$\mathcal{Q}(\boldsymbol{H}^{\text{new}};\boldsymbol{H}^{\text{old}})\geqslant\mathcal{Q}(\boldsymbol{H}^{\text{old}};\boldsymbol{H}^{\text{old}}).$$

With the above inequality, and the fact that $\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\right]$ is independent of $\boldsymbol{H}^{\text{new}}$, we have the inequality

$$\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\frac{p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H}^{\text{new}})}{p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\right]\geqslant\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\frac{p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H}^{\text{old}})}{p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\right].$$

Due to $KL(p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\|p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}}))=0$, we have

$$\begin{aligned}
&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[\log p(\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{new}})\right]\\
\geqslant&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}}),p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\left[\log\frac{p(\underline{\boldsymbol{x}},\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}};\boldsymbol{H}^{\text{old}})}{p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})}\right]\\
&+\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[KL(p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\|p(\underline{\boldsymbol{s}},\underline{\boldsymbol{\kappa}}|\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}}))\right]\\
=&\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[\log p(\underline{\boldsymbol{x}};\boldsymbol{H}^{\text{old}})\right].
\end{aligned}$$

Since $\boldsymbol{f}_{s,\kappa}$ and $\nabla\boldsymbol{f}_{s,\kappa}$ are continuous with regard to $\boldsymbol{\theta}_{s,\kappa}$ in GenHMM, $\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[\log p(\underline{\boldsymbol{x}};\boldsymbol{H})\right]$ is bounded. The above inequality shows $\mathbb{E}_{\hat{p}(\underline{\boldsymbol{x}})}\left[\log p(\underline{\boldsymbol{x}};\boldsymbol{H})\right]$ is non-decreasing in learning of GenHMM. Therefore, GenHMM will converge.

□

7.3.4 Algorithm of GenHMM

Algorithm 4 Learning of GenHMM

1: **Input:** Empirical distribution $\hat{p}(\boldsymbol{x})$ of dataset
2: Initializing $\boldsymbol{H}^{\text{old}}, \boldsymbol{H} \in \mathcal{H}$ gives:
$\boldsymbol{H}^{\text{old}} = \{\mathcal{S}, \boldsymbol{q}^{\text{old}}, A^{\text{old}}, p(\boldsymbol{x}|s; \boldsymbol{\Phi}_s^{\text{old}})\}$,
$\boldsymbol{H} = \{\mathcal{S}, \boldsymbol{q}, A, p(\boldsymbol{x}|s; \boldsymbol{\Phi}_s)\}$,
in which generators $\{\boldsymbol{g}_{s,\kappa}|s \in \mathcal{S}, \kappa = 1, 2, \cdots, K\}$ are all initialized randomly.
3: $\boldsymbol{H}^{\text{old}} \leftarrow \boldsymbol{H}$
4: Set learning rate η, neural network optimization batches N per EM step
5: **for** $\boldsymbol{H}$ not converge **do**
6: **for** epoch $n < N$ **do**
7: Sample a batch of data $\{\underline{\boldsymbol{x}}^r\}_{r=1}^{R_b}$ from dataset $\hat{p}(\underline{\boldsymbol{x}})$ with batch size R_b
8: Compute posterior $p(s_t^r, \kappa_t^r|\underline{\boldsymbol{x}}^r; \boldsymbol{H}^{\text{old}})$
9: Formulate loss $\mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{H}^{\text{old}}\right)$ in (7.20)
10: $\partial\boldsymbol{\Theta} \leftarrow \nabla_{\boldsymbol{\Theta}}\mathcal{Q}\left(\boldsymbol{\Theta}, \boldsymbol{H}^{\text{old}}\right)$
11: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \eta \cdot \partial\boldsymbol{\Theta}$
12: **end for**
13: $\boldsymbol{q} \leftarrow \underset{\boldsymbol{q}}{\text{argmax}}\ \mathcal{Q}(\boldsymbol{q}; \boldsymbol{H}^{\text{old}})$ by (7.29)
14: $\boldsymbol{A} \leftarrow \underset{\boldsymbol{A}}{\text{argmax}}\ \mathcal{Q}(\boldsymbol{A}; \boldsymbol{H}^{\text{old}})$ by (7.33)
15: $\boldsymbol{\Pi} \leftarrow \underset{\boldsymbol{\Pi}}{\text{argmax}}\ \mathcal{Q}(\boldsymbol{\Phi}; \boldsymbol{H}^{\text{old}})$ by (7.25)
16: $\boldsymbol{H}^{\text{old}} \leftarrow \boldsymbol{H}$
17: **end for**

To summarize the learning solution in Section 7.3.2, we wrap our algorithm into pseudocode as shown in Algorithm 4. We use the Adam [87] optimizer for optimization with regard to the parameters of generators in GenHMM. As shown from line 6 to 10 in Algorithm 4, the batch-size stochastic gradient decent can be naturally embedded into the learning algorithm of GenHMM.

As described by the pseudocode in Algorithm 4, the learning of GenHMM is divided into optimizations with regard to generators' parameters $\boldsymbol{\Theta}$, initial probability $\boldsymbol{q}$ of hidden state, transition matrix $\boldsymbol{A}$, and generator mixture weights $\boldsymbol{\Pi}$. Different from the optimization with regard to $\boldsymbol{q}$, $\boldsymbol{A}$ and $\boldsymbol{\Pi}$, which have optimal solutions, generator learning usually cannot give optimal solution to problem $\max_{\boldsymbol{\Theta}} \mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H}^{\text{old}})$. In fact, given that no optimal $\boldsymbol{\Theta}$ is obtained, learning of GenHMM can still converge as long as quantity $\mathcal{Q}(\boldsymbol{\Theta}; \boldsymbol{H})$ are improving in iterations in Algorithm 4, since the inequalities in Proposition 7.1 still hold. Therefore optimal $\boldsymbol{\Theta}$ in each iteration is not required for convergence of GenHMM as long as the loss in (7.20) is getting improved.

Remark 7.4. *The above discussed maximum likelihood training is also known as generative training, which models the signal generating process with the defined generative model. An alternative training principle is discriminative training where a conditional distribution is modeled directly and optimized as the objective. Since a generative model characterizes a joint distribution, it is usually possible to train a generative model via discriminative training, with mildly alternating the objective function. On the contrast, it is usually not likely to train a discriminative model via generative training, since the direct conditional probability modeling of the discriminative model omits modeling distribution of the observable signal that is highly structured. These two training principles are possible to be used together at different training phases. For instance, apply the generative training firstly, and then a discriminative training phase for the generative-trained model which acts as a parameter fine-tuning phase [69]. See detailed insightful discussion about these two principles by [98].*

7.4 Application to Speech Recognition

To show the validity of our model, we implement our model in PyTorch and test it with speech sequential data. We first discuss the experimental setups and then show the experimental results.

7.4.1 Experimental Setup

The dataset used for sequential data modeling and classification is TIMIT where the speech signal is sampled at 16kHz. The TIMIT dataset consists of 5300 phoneme-labeled speech utterances which are partitioned into two sets: a train set consists of 4620 utterance, and a test set consists of 1680 utterances. There are totally 61 different types of phones in TIMIT. We performed experiments in two cases: i) full 61-phoneme classification case; ii) 39-phonme classification case, where 61 phonemes are folded onto 39 phonemes as described in [111].

For extraction of feature vectors, we use 25ms frame length and 10ms frame shift to convert soundtracks into standard Mel-frequency cepstral coefficients (MFCCs) features. Experiments using the deltas and delta-deltas of the features are also carried out.

Our experiments are performed for: i) standard classification tasks (Table 7.2, 7.3, 7.4, 7.5), ii) classification under noise perturbation (table 7.6, 7.7). The criterion used to report the results includes accuracy, precision and F1 scores. In all experiments, generators $\{\boldsymbol{g}_{s,\kappa}|s \in \mathcal{S}, \kappa = 1, 2, \cdots, K\}$ of GenHMM are implemented as flow models. Specifically, our generator structure follows that of a RealNVP described in [36]. As discussed, the coupling layer as shown in (6.5) maps a part of its input signal identically. The implementation is such that layer $l + 1$ would alternate the input signal order of layer l such that no signal remains the same after two consecutive coupling layers. We term such a pair of consecutive coupling

Table 7.1: Configuration of generators of GenHMM in Experiments

Latent distribution $p_{s,\kappa}(\boldsymbol{z})$ $s \in \mathcal{S}, \kappa = 1, 2, \cdots, K$	Standard Gaussian
Number of flow blocks	4
Non-linear mapping $\boldsymbol{m}_a$, $\boldsymbol{m}_b$	Multiple layer perception 3 layers and with hidden dimension 24

Table 7.2: Test accuracy table for 39 dimensional features and folded 39 phonemes.

Model	Criterion	K=1	K=3	K=5
GMM-HMM	Accuracy	62.3	68.0	68.7
	Precision	67.9	72.6	73.0
	F1	63.7	69.1	69.7
GenHMM	Accuracy	76.7	**77.7**	77.7
	Precision	76.9	**78.1**	78.0
	F1	76.1	**77.1**	77.0

Table 7.3: Test accuracy table for 39 dimensional features and 61 phonemes.

Model	Criterion	K=1	K=3	K=5
GMM-HMM	Accuracy	53.6	59.6	61.9
	Precision	59.1	63.9	65.7
	F1	54.7	60.5	62.7
GenHMM	Accuracy	69.5	70.6	**70.7**
	Precision	69.2	70.5	**71.0**
	F1	68.6	69.6	**69.6**

layers as a *flow block*. In our experiments, each generator $\boldsymbol{g}_{s,\kappa}$ consists of four *flow blocks*. The density of samples in the latent space is defined as Normal, i.e. $p_{s,\kappa}(\boldsymbol{z})$ is the density function of standard Gaussian. The configuration for each generator is shown as Table 7.1.

For each GenHMM, the number of states is adapted to the training dataset. The exact number of states is decided by computing the average length of MFCC frames per phone in training dataset, and clipping the average length into $\{3, 4, 5\}$. Transition matrix $\boldsymbol{A}$ is initialized as an upper triangular matrix for GenHMM.

7.4.2 Numerical Results

We firstly show the phoneme classification using 39 dimensional MFCC features (MFCC coefficients, deltas, and delta-deltas), to validate one possible usage of our

Table 7.4: Test accuracy table for 13 dimensional features and folded 39 phonemes.

Model	Criterion	K=1	K=3	K=5
GMM-HMM	Accuracy	48.5	51.2	52.4
	Precision	56.2	58.3	59.5
	F1	50.3	53.0	54.2
GenHMM	Accuracy	61.1	**62.1**	62.1
	Precision	61.1	61.9	**62.1**
	F1	59.7	**60.7**	60.2

proposed model. Since generative training is carried out in our experiments, GMM-HMM is trained and tested as a reference model in our experiments. Training and testing of GMM-HMM are in the same condition as GenHMMs are trained and tested. Dataset usage for GenMM and GMM-HMM is the same, and the number of states for GMM-HMM is the same as that for GenHMM in modeling each phoneme. Apart from setting the reference model, we also run the experiment comparisons with different total numbers of mixture components.

Table 7.2 and 7.3 shows the results for this experiments, in which we test both the folded 39-phoneme classification case (the conventional way) in Table 7.2 and the 61-phoneme classification case in Table 7.3. As shown in both 61-phoneme and 39-phoneme cases, GenHMM gets significantly higher accuracy than GMM-HMM for the same number of mixture components. The comparisons with regarding to precision and F1 scores show similar trends and also demonstrate significant improvement of GenHMM's performance. As our expectation, GenHMM has better modeling capacity of sequential data since we bring in the neural network based generators into GenHMM, which should be able to represent complex relationships between states of HMM and sequential data. Apart from the gain of using neural network based generative models, there are also increases of accuracy, precision, and F1 scores as the number of mixture components in GenHMM is increased from $K = 1$ to $K = 5$. The sequential dependency of data is modeled by HMM itself, while each state of HMM can have a better representation using a mixture probabilistic model if data represented by the state is multi-mode. Comparing the results in 39-phoneme and 61-phoneme cases, GenHMM gets higher accuracy for 39-phoneme classification than it does for 61-phoneme classification. The total training dataset size remains the same as 61 phonemes are folded into 39 phonemes. There are less training data available per phonemes and more classes to be recognized in the 61-phoneme case, which makes the task more challenging.

Similar experiments are carried out by using only the MFCC coefficients as feature input (excluding deltas and delta-deltas). The results are shown in Table 7.4 and 7.5. The superior performance of GenHMM remains compared with reference model GMM-HMM, with regarding accuracy, precision, and F1 scores. The gain by using mixture generators is also presented in this set of experiments while the difference between 61-phoneme and 39-phoneme cases is similar to the set of experiments

Table 7.5: Test accuracy table for 13 dimensional features and 61 phonemes.

Model	Criterion	K=1	K=3	K=5
GMM-HMM	Accuracy	37.1	40.6	42.2
	Precision	44.6	47.4	48.8
	F1	38.8	42.1	43.7
GenHMM	Accuracy	50.3	50.8	**52.3**
	Precision	49.3	50.9	**52.1**
	F1	47.8	48.3	**49.3**

Table 7.6: Test accuracy table of perturbation with white noise ($K = 3$, folded 39 phonomes).

Model	Criterion	White Noise SNR			
		15dB	20dB	25dB	30dB
GMM-HMM	Accuracy	36.6	44.2	50.8	57.1
	Precision	59.2	64.2	68.4	70.6
	F1	39.9	47.7	53.9	59.9
GenHMM	Accuracy	52.4	62.0	69.7	**74.3**
	Precision	60.0	65.9	71.7	**74.8**
	F1	52.5	62.0	69.3	**73.5**

in Table 7.2 and 7.3.

Apart from standard classification testing, we also test the robustness of our model to noise perturbations. We train GenHMM with $K = 3$ by clean TIMIT training data in the case of folded 39 phonemes with 39 dimensional features. The testing dataset is perturbed by either the same type of noise with a different signal-

Table 7.7: Test accuracy table of perturbation by different type of noise (SNR=20dB, $K = 3$, folded 39 phonomes).

Model	Criterion	Noise Type			
		White	Pink	Babble	Volvo
GMM-HMM	Accuracy	44.2	48.8	57.7	66.6
	Precision	64.2	66.1	67.0	71.9
	F1	47.7	52.3	59.7	67.8
GenHMM	Accuracy	62.0	65.1	70.0	**75.7**
	Precision	65.9	67.8	70.4	**75.9**
	F1	62.0	64.6	69.0	**75.3**

to-noise ratio (SNR) as shown in Table 7.6, or different type of noises with the same SNR as shown in Table 7.7. The noise data is from NOISEX-92 database. The baseline of these two sets of experiments is the accuracy testing of GenHMM and GMM-HMM on clean testing data in the same experimental condition, where GenHMM has 77.7% and GMM-HMM gets 68.0% as shown in Table 7.2. The similar superior performance of GenHMM with regarding to precision and F1 scores is also shown. It is shown in Table 7.6 that GMM-HMM's performance degenerates more than GenHMM's performance at the same level of noise perturbation, though the accuracy of both models increases along with the increase of SNR. Especially, for SNR=30dB, the accuracy of GenHMM drops only about 3% (from 77.7% to 74.3%), while GMM-HMM encounters more than 10% decrease (from 68.0% to 57.1%) due to the noise perturbation. In Table 7.7, the SNR remains constant and GenHMM is tested with perturbation of different noise types. It is shown that GenHMM still remains higher performance scores at different types of noise perturbations than GMM-HMM. Among these four types of noise, white noise shows the most significant impact on GenHMM while the impact of Volvo noise is negligible.

7.5 Application to Sepsis Detection

In this section, we apply our developed model to the sepsis detection for infants. Sepsis may rapidly develop among newborn babies who are under care in neonatal intensive care units (NICU). The sequential signals in this application are the physiological signals, which are taken as input for sepsis prediction.

Apart from the default maximum likelihood that does generative training of GenHMM, it is also possible to carry out discriminative training as discussed in Remark 7.4. This discriminative training is performed by maximizing the conditional probability of the correct class $y \in \mathcal{Y}$ given its input $\underline{\boldsymbol{x}}$, where $\mathcal{Y}$ is the alphabet of class y. This leads to

$$\max_{\{\boldsymbol{H}_i, i=1,\cdots|\mathcal{Y}|\}} \sum_{(\underline{\boldsymbol{x}},\boldsymbol{y})} \log \frac{p(\underline{\boldsymbol{x}} \mid \boldsymbol{H}_y)p(\boldsymbol{H}_y)}{\sum_{y'} p(\underline{\boldsymbol{x}} \mid \boldsymbol{H}_{y'})p(\boldsymbol{H}_{y'})}. \tag{7.34}$$

It can be seen that the complexity in discriminative training of GenHMM increases as $|\mathcal{Y}|$ is large, since the marginal $p(\underline{\boldsymbol{x}}) = \sum_{y'} p(\underline{\boldsymbol{x}} \mid \boldsymbol{H}_{y'})p(\boldsymbol{H}_{y'})$ has to be evaluated for each sequential sample. But for this sepsis detection, there are two classes, i.e. $\mathcal{Y} = \{1, 2\}$ and thus the computation is affordable. The classes prior probabilities $p(\boldsymbol{H}_i),\ i = 1, 2$ are infered from the training dataset. We use *dGenHMM* to denote that GenHMM is finely tuned by discriminative training with objective (7.34), after default generative training phase.

7.5.1 Patient Dataset

The bedside monitor signals of 48 premature infants who have been under care at a NICU in Karolinska University hospital have been collected. The sequential

signals used are the Respiratory Frequency (RF), the beat to beat interval (RRi), and the blood oxygen saturation level ($Sp0_2$). All signals were sampled at 1Hz and segmented into 20 minutes time frames. Each time frame was then labeled based on information retrieved from the Electronic Health Records (EHR). Similar to heart rate observation system (HeRO) [57,68], a logistic regession method using RRi, we aim at detecting septic events earlier than clinical suspicion of sepsis. In our study we use a threshold of 72h prior to blood sample, to label a time frame as *septic*, according to the practices in [61]. A time frame was retro-actively labeled 1 if it occurred at most 72h prior to clinical suspicion. A time frame was labeled 0 if it occurred during a day when no notes were entered in the infant's EHR. Time frames not labeled either 0 or 1 were discarded.

Our final dataset consists of 22 patients, among which 13 males and 9 females. The birth weight was 1.61 ± 1.10 kg and the gestational age at birth 30.9 ± 6.14. Our dataset consisted of 3501 time frames, among which 1774 with label 0 and 1727 with label 1. All time frames have a constant size of $T = 1200$ samples and are 3-dimensional.

7.5.2 Experimental Setup

To make the experiments more interesting, we add a set of baseline methods in the comparisons. Baseline model HeRO is used, which uses the RRi signal for feature extraction and applies a logistic regression to the extracted features. See [72] about detailed feature extraction. Pulse oximetry predictive score (POPS) [160] uses mean, standard deviation, skewness, kurtosis, and min-max cross correlation between RR-interval and $SpO2$ to compute a risk score. POPS is also a logistic regression method. For the logistic regressions, the optimal regularization parameter was found with cross-validation and grid search in the set $\{10^{-5}, \cdots, 10^{5}\}$.

We performed binary classification by the maximum likelihood with GMM-HMM, GenHMM, and dGenHMM as probabilistic models. The GMM-HMM hyper-parameters were the number of states and the number of Gaussians per state. For GenHMM and dGenHMM, the hyper-parameters were the number of states, the number of flows per state, chain-length in each flow, the size of the networks in a coupling layer of a flow. Given our limited input dimension, the size of the networks in the coupling layers was fixed to 3. The chain-length in the coupling layer of each flow was varied between 4 and 8. We varied the number of states in HMMs in $\{3, 6, 9\}$ and show the results in Table 7.8.

We repeated our experiments 3 times and each time a random 30% of the patients were left out for testing. This lead to 2361 ± 353 time series in the training sets and 1140 ± 353 time series in the testing sets. We used a different set of input features to test our models in different conditions. HMMs are trained on raw time series and on raw time series with first and second order derivatives. For logistic regression models, HeRO is trained with 3-dimensional features, and POPS with 10-dimensional features. The results associated with these two sets of features are presented in Table 7.9. Extreme learning machines (ELM) and support vector

machine (SVM) with a gaussian kernel trained on the raw time series data are also added into the comparison.

7.5.3 Numerical Results

The results for the HMMs are presented in Table 7.8 and the comparison with other benchmarks are presented in Table 7.9.

Table 7.8: Test accuracy of HMMs

Number of states	n=3	n=6	n=9
Raw sequential input			
GMM-HMM	0.68 ± 0.03	**0.68** ± 0.03	**0.69** ± 0.03
GenHMM	0.67 ± 0.04	0.61 ± 0.08	0.63 ± 0.08
dGenHMM	**0.70** ± 0.10	0.67 ± 0.06	0.65 ± 0.04
Raw sequential input + 1st and 2ed order derivatives			
GMM-HMM	**0.75** ± 0.05	**0.74** ± 0.08	**0.74** ± 0.05
GenHMM	0.69 ± 0.07	0.66 ± 0.06	0.59 ± 0.08
dGenHMM	0.71 ± 0.04	0.72 ± 0.10	0.67 ± 0.04

Table 7.9: Comparison of HMM with other models

Model	Performance	
Using feature extraction		
Logistic Regression	POPS	0.54±0.01
	HeRO	0.57±0.04
Using raw sequential input		
SVM		0.60 ± 0.04
ELM		0.60 ± 0.03
dGenHMM		0.70 ± 0.10

The HeRO reaches 57% of correct classifications. It outperforms the POPS algorithms which reach only 54%, which is a method with the lowest performance here. As expected, the logistic classifiers are outperformed by Gaussian kernel SVM, ELM, and our HMMs. SVM and ELM both reach an accuracy of 60% and comparable standard deviations of 4% and 3%. This is lower than dGenHMM which reaches 70% accuracy and outperforms both GenHMM 67%, and GMM-HMM 68%. These results are contrasted by the large standard deviation of dGenHMM 10%, which is larger than both GenHMM 4% and GMM-HMM 3%. When the number of states increases to $n = 6$ and $n = 9$, GMM-HMM reaches 68% and 69% which outperforms GenHMM, and dGenHMM. When the raw sequential input is augmented with 1st and 2nd order derivatives, GMM-HMM reaches its highest performance

at 75% accuracy. GenHMM and dGenHMM also reach their highest performance with 69% at $n = 3$ and 72% at $n = 6$.

Improved performance of GenHMM using discriminative training (dGenHMM) was significant for both 1st and 2nd order derivatives as argument to inputs. This is encouraging, given that our current discriminative training consists of only one epoch, i.e. one-epoch fine-tuning. The performance of GMM-HMM is also significantly increased when adding 1st and 2nd order derivative, reaching higher than dGenHMM. This phenomenon is due to two aspects: i) model complexity; ii) dataset size. GMM-HMM is a simpler model compared to GenHMM. Besides, the number of samples in the dataset is limited, i.e. only 3501 time frames. These two major factors together explain why the performance of GMM-HMM increases while that of GenHMM decreases as n increases from 3 to 9 (an increase of each corresponding model's complexity). Also, discriminative training, dGenHMM, does improve performance. The insufficient training data makes its final accuracy fall behind the much simpler model GMM-HMM.

7.6 Summary

In this chapter, we discussed the representation and learning of a temporal model. With the simplification of Markov assumption and the concept of template, the HMMs, as simplified dynamic Bayesian networks, have a good balance between representation power and computation efficiency. Thus, they have been widely used in disciplines in practice. The modeling expressivity is further enriched by the introduction of normalizing flows. The developed GenHMM is a generic model and allows large freedom of customization, which can be employed for practical applications such as the demonstrated speech recognition and sepsis detection. Due to the presence of hidden variables, the learning of GenHMM is carried out within expectation maximization framework. Adaption is made to accommodate the neural network implementation of normalizing flows and large datasets in parameter learning.

In increasing the representation power of modeling, GenHMM relies on enriching the flexibility of conditional probability models. Alternative options lie with the template model design. In fact, template is the core concept for more general object-relational modeling of probabilistic graphical models. Generally speaking, any object-relation can be used to define a skeleton that functions as a template. Then ground probabilistic graphical models can be formed by assembling the instances from skeletons or templates. The results models can be rich and representative for general object-relation modeling. The sequential instances concatenation shown in the chapter is one way of the general modeling process. It is suggested to bear the learning complexity in mind when making use of flexible representation of template models. It may be the case that the best representation brings prohibitive inference or learning complexity. Therefore, the modeling accuracy and inference/learning complexity should be jointly considered when designing models

for practical applications.

7.7 Literature Work

Probabilistic models of temporal processes are challenging topics in pattern recognition and machine learning, which can be generalized to a wide range of problems of sequential dependency beyond temporal signals. The various applications cover but not limited to reinforcement learning [35, 101], natural language modeling [63, 81], biological sequence analysis such as proteins [9] and DNA [147], etc. The study on temporal process modeling goes back many years. Hidden Markov models (HMMs) were discussed as early as in 1980s by [142], and further by [143]. The generalization of HMMs into probabilistic graphical models, dynamic Bayesian networks, was proposed around the same period in [30].

The classic way is to use a Gaussian mixture model (GMM) per state of HMM [13], where GMMs are used to connect states of HMM to sequential data input. GMM based HMM (GMM-HMM) has become a standard model for sequential data modeling, and been employed widely for practical applications, especially in speech recognition [18, 45].

To represent data in nonlinear manifolds, research attempting at training HMM with neural networks has been carried out to boost the modeling capacity of HMM. A successful work of this track has brought deep neural network (DNN) that is defined by restrictive Boltzmann machines (RBMs) [70] into HMM based models [69, 104, 116]. RBM based HMM is trained with a hierarchical scheme consisting of multiple steps of unsupervised learning, formatting of a classification network, and then supervised learning. The hierarchical procedure comes from the empirical expertise in this domain. The hierarchical learning scheme of RBM/DNN based HMM consists of: i) RBMs are trained one after the other in an unsupervised fashion and are stacked together as one deep neural network model, ii) then a final softmax layer is added to the stack of RBMs to represent the probability of an HMM state given a data input, iii) a discriminative training is performed for the final tuning of the model at the final stage.

With the prevalence of deep learning, further works on neural network based temporal models are present in literature. A representative track is the hybrid method of temporal neural network models and HMM. In [16, 93, 109], a long short-term memory (LSTM) model/recurrent neural network (RNN) is combined with HMM as hybrid. A hierarchical training is carried out by: i) training an HMM first, ii) then doing modified training of LSTM using trained HMM. This hierarchical training procedure is motivated by the intuition of using LSTM or RNN to fill in the gap where HMM can not learn. Another successful exploration is motivated by hierarchical HMM [43], a variant of HMM that tries to model complex multi-scale structures of hidden variables in sequential signal modeling. Work hierarchical multiscale RNN [24] and variational temporal abstraction [84] use neural networks to infer the multi-scale structures of hidden variables, where applications of language

models and navigations are demonstrated successfully.

Chapter 8

An Implicit Probabilistic Generative Model

In previous chapters, we discussed learning in both undirected (Chapter 5) and directed (Chapter 6 and 7) graphical models. The model learning of previous discussions has been mainly under the principle of maximum likelihood, which includes the cases of MRF learning with approximate inference and the likelihood variational lower bound such as (5.13). With maximum likelihood learning, if exact likelihood is not available, its variational or approximate value is used as the objective function.

In this chapter, we introduce a way of learning a generative model that does not use maximum likelihood. Different from distributions induced by normalizing flows in previous chapters, we do not require the induced distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ by a generator in the generative model to be tractable here. In another word, the generator-induced distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ is *implicit* and likelihood-intractable. This relaxation indeed gives us larger freedom of defining the generators since we do not track the Jacobian computation anymore, and thus enlarges the hypothesis space of the model distribution $p(\boldsymbol{x};\boldsymbol{\theta})$. But it also brings the question of how to learn such a model without the tractable density function, since maximum likelihood and its associated KL divergence require evaluation of likelihood according to explicit probability density (mass) functions. Perhaps, a question before the *how* is the *why*, i.e., why do we need such kind of probabilistic models?

The motivation for studying these generative models lies in the applications where sampling from implicit distributions is more important than likelihood tractability. In high-dimensional spaces, deep generative models that are defined with neural network based generators and induce implicit distributions, demonstrate their advantage in the efficiency of modeling complex distributions and sampling, in comparison with explicit statistic models. This allows the representation and manipulation of implicitly high-dimensional distributions via the generative models. For instance, deep generative models have been used in image super-

resolution, image-to-image translation, speech synthesis, etc. Besides, implicit generative models can be naturally incorporated in reinforcement learning where typically an agent tries to learn to finish some tasks via interacting with a dynamic environment. For instance, in model-based methods, using a generative model to represent the environment and generating states for the agent to learn with, offers the way of training agents in simulations; A policy, essentially a conditional distribution of actions given an environment state, may be modeled by an implicit generative model since we mainly need to get the action in the current environment.

In this chapter, we introduce the learning of implicit probabilistic generative models by employing *optimal transport* from transportation theory that does not need the tractability of likelihood. As long as we can sample for a generative model efficiently, learning of the generative model can be formed as the minimization of an optimal transport distance between the induced implicit distribution and an empirical distribution.

8.1 Optimal Transport

Optimal transport (OT) is a geometric tool for comparison of probability distributions, which has a rich history. Its work traces back as early as 18th century in Monge's work [122]. OT has been widely used in different fields, such as economic problems, logistics, production planning, etc. Along with its development in history, OT has been given different names, such as earth mover distance, Kantovich distance, and Wasserstein distance. Abstractly, OT distance measures the minimum transportation cost of from the mass of a distribution to another distribution [173]. Due to its close relation to the comparison of probability distributions and optimization, OT and its variants have also been popularly used in machine learning and gained fast development in both theory and applications [25, 27, 28, 157].

The high-level idea of this chapter is to use the OT distance to guide us to find the distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ of a generative model to approximate the true distribution $p^*(\boldsymbol{x})$. We begin with the introduction to OT itself and make connections to some popular generative models.

We denote the space $(\boldsymbol{\mathcal{X}}, \|\cdot\|_2)$ be our working space, where $\boldsymbol{\mathcal{X}} = \prod_{x=1}^{N} \mathcal{X}_i$ as defined in Chapter 2. $\|\cdot\|_2$ is the Euclidean distance. Assume that $\boldsymbol{\mathcal{X}}_1$, $\boldsymbol{\mathcal{X}}_2$ are finite sample subsets of $\boldsymbol{\mathcal{X}}$. Let $p^*(\boldsymbol{x})$ be a distribution on $\boldsymbol{\mathcal{X}}_1$ and $p(\boldsymbol{x};\boldsymbol{\theta})$ be a distribution on $\boldsymbol{\mathcal{X}}_2$. With mild abuse of notation, we denote the OT distance between $p^*(\boldsymbol{x})$ and $p(\boldsymbol{x};\boldsymbol{\theta})$ by $T(p^*, p)$. According to Kantovich's transportation problem [173], we have $T(p^*, p)$ given by

$$T(p^*, p) = \min_{\pi \in \Pi(p^*, p)} \langle \pi, \boldsymbol{M} \rangle, \tag{8.1}$$

where $\langle \cdot, \cdot \rangle$ stands for the inner product of two matrices, and $\Pi(p^*, p)$ is a set of joint distribution π on the sample sets $\boldsymbol{\mathcal{X}}_1 \times \boldsymbol{\mathcal{X}}_2$ such that π has marginal distributions $p^*(\boldsymbol{x})$ and $p(\boldsymbol{x};\boldsymbol{\theta})$. The cost matrix $\boldsymbol{M}$ has elements $M_{i,j} = d(\boldsymbol{x}_1^i, \boldsymbol{x}_2^j)$ with $d(\cdot, \cdot)$

as a distance (Euclidean distance is used in our implementation), where $\boldsymbol{x}_1^i \in \boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{x}_2^j \in \boldsymbol{\mathcal{X}}_2$ are ith and jth samples from $p^*(\boldsymbol{x})$ and $p(\boldsymbol{x};\boldsymbol{\theta})$, respectively.

Remark 8.1 (Connection to the vanilla generative adversarial network)**.** *In the model learning of previous chapters, the information on how the model should adjust (change its parameter) is given by the likelihood value (and its gradients), stemming from KL divergence. In this chapter, we introduce the OT distance to offer such information in modeling learning. There are other ways to achieve this goal. A popular one is the generative adversarial network (GAN) [54, 56], which casts the generative model learning into an adversarial game between the generator $\boldsymbol{g}$ and a discriminator f. The discriminator f gets input from either output of $\boldsymbol{g}$ or from p^*. The generator $\boldsymbol{g}$ and discriminator f play against each other regarding a min-max objective*

$$\min_{\boldsymbol{g}} \max_{f} \mathbb{E}_{p^*(\boldsymbol{x})} \left[\log(f(\boldsymbol{x}))\right] + \mathbb{E}_{p(\boldsymbol{z})} \left[\log(1 - f(\boldsymbol{g}(\boldsymbol{z})))\right], \tag{8.2}$$

where $p(\boldsymbol{z})$ is a fixed distribution and easy to sample from. The model distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ is an implicit distribution and is induced by $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{z})$. The probability model learning relies on if min-max game finds its equilibrium where distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ is the same as $p^*(\boldsymbol{x})$ and d is optimal. In optimal case of the discriminator, objective of (8.2) *is equivalent to minimize the Jensen-Shannon divergence between $p(\boldsymbol{x};\boldsymbol{\theta})$ and $p^*(\boldsymbol{x})$.*

Remark 8.2 (Connection to Wassertein GAN)**.** *It is interesting to note that the OT minimization can be connected to GANs via its duality form. According to Kantorovich-Rubinstein duality [137, Section 2.4] [172],*

$$T(p^*, p) = \sup_{\|f\|_L \leqslant 1} \mathbb{E}_{p^*(\boldsymbol{x})} \left[f(\boldsymbol{x})\right] - \mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})} \left[f(\boldsymbol{x})\right], \tag{8.3}$$

$\|f\|_L$ is a constraint for f such that f belong to the 1-Lipschitz function family. That is to say, f in (8.3) *can be any function from the 1-Lipschitz function family (which is different from the binary discriminator in* (8.2)*) that maximizes the objective in* (8.3)*. Then minimization of the OT distance is formulated as a min-max problem [8], termed as Wasserstein GAN (WGAN),*

$$\min_{\boldsymbol{g}} \max_{\|f\|_L \leqslant 1} \mathbb{E}_{\boldsymbol{x}\sim p^*(\boldsymbol{x})} \left[f(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \left[f(\boldsymbol{g}(\boldsymbol{z}))\right], \tag{8.4}$$

where $p(\boldsymbol{x};\boldsymbol{\theta}) = p(\boldsymbol{g}(\boldsymbol{z});\boldsymbol{\theta})$ with $\boldsymbol{\theta}$ parameterizing $\boldsymbol{g}$ and $\boldsymbol{z} \sim p(\boldsymbol{z})$. The Lipschitz constraint enforcing is not trivial in general, especially for the deep generative models where $\boldsymbol{g}$ is implemented by neural networks. Nevertheless, the vanilla GAN in (8.2) *is similar to the* (8.4) *from minimization of OT distance.*

Remark 8.3 (Connection to auto-encoders)**.** *It is known that the variational lower bound of the likelihood function as in* (5.13) *is used to train variational auto-encoders, where an encoder and a decoder adjust their parameters to maximize the variational lower bound in the maximum likelihood principle. Since OT offers an*

alternative tool for modeling learning, it is also employed to train auto-encoders in likelihood-free settings, which shares similarities with the variational auto-encoders. See [6, 15, 49, 134] for detailed discussions.

8.2 EOT based Generative Models

The optimal transport problem in (8.1) is highly intractable. Therefore, approximation has to been used for efficient solutions. In Remark 8.2, Kantorovich-Rubinstein duality offers one way to solve the problem in its dual form if the Lipschitz constraint can be properly enforced. An alternative way can be using entropy regulation to seek for *soft* solution. In this section, we introduce the entropy-regularized optimal transport (EOT) cost and then propose generative models accordingly.

8.2.1 Entropy-regularized OT

OT calculates the minimum cost of transporting distribution $p^*(\boldsymbol{x})$ to $p(\boldsymbol{x};\boldsymbol{\theta})$. We use $W(p^*, p)$ to denote entropy-regularized OT (EOT) cost as follows:

$$W(p^*, p) = \min_{\pi \in \Pi(p^*, p)} \langle \pi, \boldsymbol{M} \rangle - \lambda H(\pi), \tag{8.5}$$

where $H(\pi)$ is the entropy of joint distribution π, i.e. $H(\pi) = \sum_{i,j} -\pi_{i,j} \log(\pi_{i,j})$ and $\lambda \in \mathbb{R}_+$ is the regularization parameter. Here $\mathbb{R}_+$ denotes positive scalars. The entropy regularization in (8.5) translates to a requirement that the joint distribution π has a high entropy. The duality of EOT cost in (8.5) is

$$W(p^*, p) = \max_{\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^N} \boldsymbol{\alpha}^\intercal p^* + \boldsymbol{\beta}^\intercal p - \sum_{i,j} \lambda e^{\frac{(\alpha_i + \beta_j - M_{i,j})}{\lambda}}, \tag{8.6}$$

where $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are dual variables, $(\cdot)^\intercal$ means transpose, α_i is the ith element of $\boldsymbol{\alpha}$, and β_i is the jth element of $\boldsymbol{\beta}$

One benefit of having the dual variables in this form is that the optimal dual vector $\boldsymbol{\beta}^*$ of (8.6) is a subgradient of $W(p^*, p)$ with respect to p. Thus, if we can efficiently obtain the optimal dual vector $\boldsymbol{\beta}^*$, we can learning our model distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ via the subgradient $\boldsymbol{\beta}^*$.

There is a computationally efficient algorithm called Sinkhorn algorithm [27,28] to solve (8.6), which alternatively scales the rows and columns of matrix $e^{-\frac{\boldsymbol{M}}{\lambda}}$. This alternative computation gives a pair of vectors $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}_+^N \times \mathbb{R}_+^N$ that defines the optimal primary and dual variavles [28, Proposition 2]

$$\begin{aligned} \pi^* &= \operatorname{diag}(\boldsymbol{u})\, e^{\frac{-\boldsymbol{M}}{\lambda}} \operatorname{diag}(\boldsymbol{v}), \\ \boldsymbol{\beta}^* &= \frac{\log(\boldsymbol{u}^\intercal) \mathbb{1}_N}{N\lambda} \mathbb{1}_N - \frac{\log(\boldsymbol{u})}{\lambda}. \end{aligned} \tag{8.7}$$

where $\operatorname{diag}(\boldsymbol{u})$ is a matrix with diagonal entries from vector $\boldsymbol{u}$ and $\mathbb{1}_N$ is a column vector with ones.

8.2.2 Two Generative Models Derived from EOT

In this section, we propose two generative models. We first develop an EOT based generative model handling signals/data directly. This model is termed as EOT generative model (EOTGM). In our second model, we use a representation mapping where EOT cost is used to optimize the generative model and representation mapping jointly. The second model is named as EOT based GAN (EOTGAN).

EOTGM

Although the true probability distribution $p^*(\boldsymbol{x})$ is not available but its empirical distribution is available, i.e. the dataset. We carry on using the notation $p^*(\boldsymbol{x})$ as the target distribution. $p(\boldsymbol{x};\boldsymbol{\theta})$ is the probability distribution of our model induced by a generator $\boldsymbol{g}: \boldsymbol{\mathcal{Z}} \to \boldsymbol{\mathcal{X}}$, where $\boldsymbol{\mathcal{Z}}$ is the support set of latent distribution $p(\boldsymbol{z})$. The generator $\boldsymbol{g}$ is implemented by a neural network and maps latent signal $\boldsymbol{z} \in \boldsymbol{\mathcal{Z}}$ to signal in $\boldsymbol{\mathcal{X}}$, i.e., $\boldsymbol{g}(\boldsymbol{z}) \in \boldsymbol{\mathcal{X}}$. The latent distribution is assumed to be known. The mapped signal $\boldsymbol{g}(\boldsymbol{z}) \sim p(\boldsymbol{x};\boldsymbol{\theta})$ since $\boldsymbol{g}$ induces the model distribution. Actually, the parameter of $p(\boldsymbol{x};\boldsymbol{\theta})$ is the parameter of the generator $\boldsymbol{g}$. Applying EOT cost to learn $p(\boldsymbol{x};\boldsymbol{\theta})$ is equivalent to minimizing $W(p^*, p)$ with regard to the generator g

$$\underset{\boldsymbol{g}:\boldsymbol{\mathcal{Z}}\to\boldsymbol{\mathcal{X}}}{\operatorname{argmin}} W(p^*, p) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} W(p^*, p). \tag{8.8}$$

Since $\boldsymbol{\beta}^*$ in (8.7) is subgradient of $W(p^*, p)$ with regard to $p(\boldsymbol{x};\boldsymbol{\theta})$, we are able to optimize the generator $\boldsymbol{g}$ such that the induced distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ approximates $p^*(\boldsymbol{x})$, using gradient chain rule gives

$$\nabla_{\boldsymbol{\theta}} W(p^*, p) = (\nabla_{\boldsymbol{\theta}} p)^T \boldsymbol{\beta}^*. \tag{8.9}$$

Alternatively the optimization problem (8.8) can be addressed by solving $\operatorname{argmin}_g \langle \pi^*, \boldsymbol{M} \rangle$ iteratively using auto-gradient functions in PyTorch [1] or TensorFlow [2], where π^* is the solution to the primary problem (8.5) given by (8.7). We propose Algorithm 5 to learn distribution p^* via minimizing the EOT loss with regard to parameter $\boldsymbol{\theta}$ of generator function $\boldsymbol{g}$.

EOTGAN

In this section, we consider representation learning (feature learning) with which the usage of EOT is more meaningful than that directly in signal space. It is well-known that Euclidean distance is not well suited to compare two multimedia signals. For example, Euclidean distance between an image and its rotated version can be large, but they are visually same. In Algorithm 5 we construct cost matrix $\boldsymbol{M}$ in EOT using Euclidean distance between real signals and generated signals. Our new proposal is to transform signal through a representation mapping $\boldsymbol{f}: \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{M}} \subset \mathbb{R}^m$ and we compare features in the representation space via EOT. We assume that Euclidean distance between features in the representation space is

Algorithm 5 EOT based Generative Model (EOTGM)

1: l: the update rate at each iteration. N: the batch size. $\boldsymbol{\theta}_0$: the initial parameter for $\boldsymbol{g}$.
2: **while** $\boldsymbol{\theta}$ has not converged **do**
3: Sample a batch from a real dataset, $\left\{\boldsymbol{x}^i\right\}_{i=1}^N \sim p^*$.
4: Sample $\left\{\boldsymbol{z}^i\right\}_{i=1}^N \sim p(\boldsymbol{z})$, a batch of latent samples.
5: Passing $\left\{\boldsymbol{z}^i\right\}_{i=1}^N$ through the generator $\boldsymbol{g}$.
6: Calculate the cost matrix M.
7: π^*, $\boldsymbol{\beta}^* \leftarrow$ primary and dual solutions of $W(\left\{\boldsymbol{x}^i\right\}_{i=1}^N, \left\{\boldsymbol{g}(\boldsymbol{z}^i)\right\}_{i=1}^N)$ according (8.7).
8: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - l\left(\nabla_{\boldsymbol{\theta}} p\right)^{\intercal} \boldsymbol{\beta}^*$. (Or back propagate using loss $\langle \pi^*, \boldsymbol{M} \rangle$)
9: **end while**

more semantically meaningful. An element of the cost matrix $\boldsymbol{M}_{\boldsymbol{f}}$ in representation domain (feature domain) is

$$d_{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})\|_2. \tag{8.10}$$

Our new objective is joint learning of generator $\boldsymbol{g}$ and representation $\boldsymbol{f}$. A natural question is how to construct $\boldsymbol{f}$ function? Inspired by the triplet loss in [158] aiming at larger distance between distinct classes than in-class distance, we may consider two virtual classes labeled by $p^*(\boldsymbol{x})$ and $p(\boldsymbol{x}; \boldsymbol{\theta})$. This means that the representation function $\boldsymbol{f}$ should have the algebraic property: $d_{\boldsymbol{f}}(\boldsymbol{x}_1, \tilde{\boldsymbol{x}}_1) + \gamma \leqslant d_{\boldsymbol{f}}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ for $\gamma > 0$, where $\boldsymbol{x}_1, \tilde{\boldsymbol{x}}_1 \in \boldsymbol{\mathcal{X}}_1$ are two samples from distribution p^* and $\boldsymbol{x}_2$ is a generated signal from distribution $p(\boldsymbol{x}; \boldsymbol{\theta})$, i.e., an output of $\boldsymbol{g}$. Meanwhile, $\boldsymbol{g}$ tries to mitigate this distinction.

Following the above idea, let us denote the mapped distribution via $\boldsymbol{f}$ by $p^*_{\boldsymbol{f}}$ and $p_{\boldsymbol{f}}$, respectively. Let $\boldsymbol{M}_{\boldsymbol{f}}$ be the cost matrix in representation domain and its elements $M_{\boldsymbol{f},i,j} = d_{\boldsymbol{f}}(\boldsymbol{x}_1^i, \boldsymbol{x}_2^j)$, $\boldsymbol{x}^i \sim p^*(\boldsymbol{x}), \boldsymbol{x}^j \sim p(\boldsymbol{x}; \boldsymbol{\theta})$. Then we learn $\boldsymbol{f}$ and $\boldsymbol{g}$ using alternative optimization, as follows.

- Learning of representation f is minimizing EOT cost

$$W(p^*_{\boldsymbol{f}}, p^*_{\boldsymbol{f}}) = \min_{\widetilde{\pi} \in \Pi(p^*_{\boldsymbol{f}}, p^*_{\boldsymbol{f}})} \langle \widetilde{\pi}, \widetilde{\boldsymbol{M}}_{\boldsymbol{f}} \rangle - \lambda H(\widetilde{\pi}), \tag{8.11}$$

  where $\widetilde{M}_{\boldsymbol{f},i,j} = d_{\boldsymbol{f}}(\boldsymbol{x}^i, \tilde{\boldsymbol{x}}^j)$, $\boldsymbol{x}^i, \tilde{\boldsymbol{x}}^j \sim p^*(\boldsymbol{x})$, and maximizing EOT cost

$$W(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) = \min_{\pi \in \Pi(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}})} \langle \pi, \boldsymbol{M}_{\boldsymbol{f}} \rangle - \lambda H(\pi). \tag{8.12}$$

- Learning of generator $\boldsymbol{g}$ is minimizing EOT cost

$$W(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}). \tag{8.13}$$

Algorithm 6 EOT based GAN (EOTGAN)

1: l: the update rate at each iteration. N: the batch size. $\boldsymbol{\theta}_0$, $\boldsymbol{\omega}_0$: the initial parameters for $\boldsymbol{g}$ and $\boldsymbol{f}$.
2: **while** $\boldsymbol{\theta}$ has not converged **do**
3: Sample two batches of data $\left\{\boldsymbol{x}^i\right\}_{i=1}^N, \left\{\tilde{\boldsymbol{x}}^i\right\}_{i=1}^N$, and latent samples $\left\{\boldsymbol{z}^i\right\}_{i=1}^N$, $\boldsymbol{x}^i, \tilde{\boldsymbol{x}}^i \sim p^*(\boldsymbol{x}), \boldsymbol{z} \sim p(\boldsymbol{z})$.
4: Passing $\left\{\boldsymbol{z}^i\right\}_{i=1}^N$ through $\boldsymbol{g}$.
5: $\widetilde{\pi}^* \leftarrow$ solving $W_{\boldsymbol{f}}\left(\left\{\boldsymbol{f}(\boldsymbol{x}^i)\right\}_{i=1}^N, \left\{\boldsymbol{f}(\tilde{\boldsymbol{x}}^i)\right\}_{i=1}^N\right)$
6: $\pi^* \leftarrow$ sovling $W_{\boldsymbol{f}}\left(\left\{\boldsymbol{f}(\boldsymbol{x}^i)\right\}_{i=1}^N, \left\{\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{z}^i))\right\}_{i=1}^N\right)$
7: $\partial \boldsymbol{f} \leftarrow \nabla_{\boldsymbol{\omega}} \max\left(0, \langle\widetilde{\pi}^*, \widetilde{\boldsymbol{M}}\rangle - \langle\pi^*, \boldsymbol{M}\rangle + \gamma\right)$
8: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - l \cdot \partial \boldsymbol{f}$
9: Sample $\left\{\boldsymbol{z}^i\right\}_{i=1}^N$ and get $\left\{\boldsymbol{g}(\boldsymbol{z}^i)\right\}_{i=1}^N$ via $\boldsymbol{g}$.
10: $\pi^* \leftarrow$ sovling $W_{\boldsymbol{f}}\left(\left\{\boldsymbol{f}(\boldsymbol{x}^i)\right\}_{i=1}^N, \left\{\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{z}^i))\right\}_{i=1}^N\right)$
11: $\partial \boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}}\langle\pi^*, \boldsymbol{M}\rangle$
12: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - l \cdot \partial \boldsymbol{g}$
13: **end while**

Both $W(p^*_{\boldsymbol{f}}, p^*_{\boldsymbol{f}})$ and $W(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}})$ have lower bounds, but no upper bounds. We combine the step 1 in above using a hinge loss and define the following costs.

$$\begin{aligned} \mathcal{L}_{\boldsymbol{f}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) &:= \max\left(0, W(p^*_{\boldsymbol{f}}, p^*_{\boldsymbol{f}}) - W(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) + \gamma\right), \\ \mathcal{L}_{\boldsymbol{g}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) &:= W(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}), \end{aligned} \tag{8.14}$$

where $\gamma > 0$. Hinge loss helps to balance the adversarial training of the $\boldsymbol{f}$ and $\boldsymbol{g}$. Note the our hinge adversarial loss shares similarity only in form to the self-attention GAN [190] and geometric GAN [106] but is motivated differently and defined in different metric. We used neural networks for constructing $\boldsymbol{f}$ and $\boldsymbol{g}$ functions. Let us assume that the parameters of $\boldsymbol{f}$ and $\boldsymbol{g}$ are $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$, respectively. Then the adversarial training between representation $\boldsymbol{f}$ and generator $\boldsymbol{g}$ is the following alternative optimization problem

$$\begin{aligned} &\min_{\boldsymbol{f}} \mathcal{L}_{\boldsymbol{f}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) = \min_{\boldsymbol{\omega}} \mathcal{L}_{\boldsymbol{f}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}), \\ &\min_{\boldsymbol{g}} \mathcal{L}_{\boldsymbol{g}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}) = \min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{g}}(p^*_{\boldsymbol{f}}, p_{\boldsymbol{f}}). \end{aligned} \tag{8.15}$$

The algorithm steps of EOTGAN are shown in Algorithm 6.

Remark 8.4 (Discussion on OT and EOT)**.** *Usage of entropy regularization in EOT avoids the need for Kantorovich-Rubinstein duality of OT, thus is free from Lipschitz constraint. In literature, several methods endeavor to satisfy Lipschitz constraint, for example, projecting neural network parameters into a space fulfilling Lipschitz constraint via weight clipping [8], spectrum normalization [120], or adding gradient penalty into GAN's cost function [59]. Projecting approaches bring the*

problem of neural network capacity underuse and limit its ability to learn complex mapping. Gradient penalty approach takes gradients of each layer's weight parameters of a neural network into GAN's cost, thus computation complexity grows fast as the neural network goes deeper. EOT avoids the above-mentioned problems and also has the benefit of lower computation complexity. With entropy-regularization and Sinkhorn algorithm, the computation complexity scales as $\mathcal{O}(N^2)$ [27]. On the other hand, solving OT cost using interior-point methods has a computational requirement as $\mathcal{O}(N^3 \log N)$.

8.3 Experimental Results

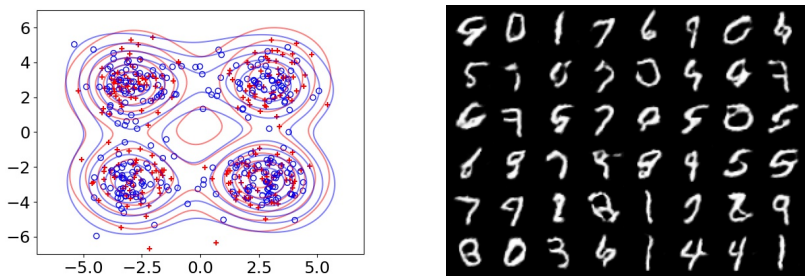
We perform experiments to verify our arguments on loss choice and algorithms. We evaluate our generative models on a toy synthetic dataset of Gaussian-mixture distribution and real image dataset MNIST.

8.3.1 Evaluation Metrics

Similar to Section 6.4.2, the Inception Score (IS) [154] and Frechet Inception Distance (FID) [67] are used as metrics as evaluation of our models. IS is defined as $IS(q) = \exp\big[\mathbb{E}_{\boldsymbol{x}\sim q}\mathrm{KL}\big(p(c|\boldsymbol{x})\|p(c)\big)\big]$, where $\boldsymbol{x} \sim q$ indicates synthetic sample from distribution q under testing, $p(c|\boldsymbol{x})$ is the conditional distribution of class c, and $p(c) = \int_{\boldsymbol{x}} p(c|\boldsymbol{x})q(\boldsymbol{x})\, d\boldsymbol{x}$ is the marginal class distribution computed with q. Large IS score means generated samples contain clear objects. Generative models with high IS can output high diversity of samples. Apart from KL-based metric, an alternative common metric is Frechet Inception Distance (FID) [67]. FID measures the OT distance of two probability distribution by assuming the two distributions are Gaussian, in which case, the closed-form solution is available. Smaller FID means the generated samples are more similar to empirical samples. In short, high IS and low FID are better.

8.3.2 Evauation of EOTGM Using Toy Dataset

We firstly evaluate our proposed EOTGM on a toy dataset sampled from a known probability distribution: two-dimensional four-mixture Gaussian. This mixture Gaussian is our target distribution to learn, i.e., $p^*(\boldsymbol{x})$. The generator $\boldsymbol{g}$ uses a neural network with structure: Input $\rightarrow$ Dense 256 $\rightarrow$ ReLU $\rightarrow$ Dense 256 $\rightarrow$ ReLU $\rightarrow$ Dense 256 $\rightarrow$ ReLU$\rightarrow$Dense 2. The parameter $\boldsymbol{\theta}$ of $\boldsymbol{g}$ here is the set of parameters of this neural network. Latent distribution $p(\boldsymbol{z})$ used here is standard Gaussian: $\mathsf{N}\big(\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)\big)$. The toy dataset is used by Algorithm 5 (EOTGM) to train $\boldsymbol{g}$. In Figure 8.1a, we plot the empirical samples from our toy dataset and the synthetic samples generated by $\boldsymbol{g}$. The corresponding contours are also plotted. It shows that the induced distribution by $\boldsymbol{g}$ approaches the mixture Gaussian distribution well without missing any mode.



(a) (b)

Figure 8.1: Demonstration of sampling. (a) Toy distribution learning (4-mixture Gaussians) using EOTGM. Real samples (red '+') and contour (red curve), versus generated samples (blue 'o') and contour (blue curve) by $\boldsymbol{g}$. (b) Generated samples by EOTGAN for MNIST dataset.

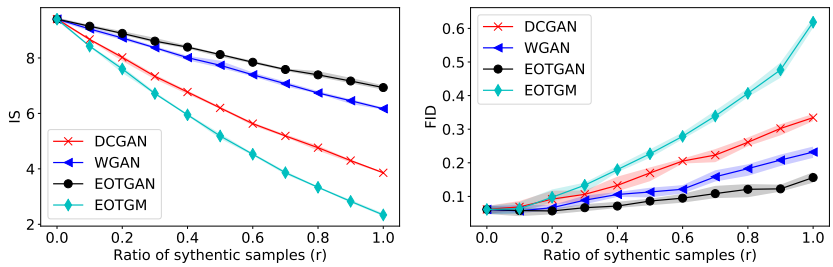
8.3.3 Evaluation of Generative Models Using MNIST

In this section, we evaluate both the generative models using MNIST dataset. The representation mapping $\boldsymbol{f}$ in EOTGAN adapts two converlutional layers appended with fully connected layers[1] similar to [23] [62]. Generator $\boldsymbol{g}$ uses the same setting as that of DCGAN and WGAN. Noise $p(\boldsymbol{z})$ is 100-dimensional Guassian. We report IS and FID scores of EOTGAN in comparison with DCGAN and WGAN. Since EOTGAN is trained with representation mapping $\boldsymbol{f}$ that acts as feature mapping, it is not fair to use this representation mapping $\boldsymbol{f}$ to do the evaluation and make comparisons since it would give EOTGAN advantages. Similar to [185], we train a 34-layer ResNet on MNIST to perform feature extraction for metric measurements of IS and FID. In addition, we put EOTGM (Algorithm 5) in comparison as well.

Data for evaluations is constructed by mixing empirical samples and synthetic samples generated by $\boldsymbol{g}$. We draw the set $\mathcal{S}_{\text{em}}$ of 2000 empirical samples from MNIST dataset. To generate a set $\mathcal{S}_{\text{syn}}$ of synthetic samples we draw $2000r$ samples from the generator network $\boldsymbol{g}$ where $r \in [0, 1]$ while rest $2000(1 - r)$ are sampled directly from MNIST. A demonstration of generated samples from EOTGAN is shown in Figure 8.1b. All the following experiments are on $\mathcal{S}_{\text{em}}$ and $\mathcal{S}_{\text{syn}}$. The way of mixing empirical data and generated data helps us to identify if a metric is intuitively helpful. Among the chosen metrics IS at $r = 0$ serves as an upper bound for the test while the FID at $r = 0$ serves as a lower bound for the corresponding tests.

IS measures how certain a classifier assigns a class label to a given generated sample. The larger IS is, the better the generative model is. We plot IS versus r

[1] 32 Conv2d $5 \times 5 \rightarrow$ PReLU $\rightarrow$ MaxPool $2 \times 2 \rightarrow$ 64 Conv2d $5 \times 5 \rightarrow$ PReLU $\rightarrow$ MaxPool $2 \times 2 \rightarrow$ Dense 256 $\rightarrow$ PReLU $\rightarrow$ Dense 256 $\rightarrow$ PReLU $\rightarrow$ Dense 2



(a) (b)

Figure 8.2: Comparison of IS and FID (on MNIST) versus mixing ratio r. (For each model at a certain mixture ratio, 5 experiments are independently performed. Each solid curve with markers plots the mean of 5 experiments with shaded areas denoting the range of corresponding results.

for different models in Figure 8.2a. IS scores of all four tested models drop with an increasing portion of synthetic samples in $\mathcal{S}_{\text{syn}}$, which is consistent with intuition. IS of EOTGAN drops at the slowest rate among the four models as more synthetic samples, for larger r, are mixed into test data. It shows that EOTGAN outperforms WGAN and DCGAN in this test. EOTGM is found to provide the lowest IS. This may be attributed to the setup that EOT optimization with cost measured by Euclidean distance of signals fails to capture semantic similarity.

In Figure 8.2b the performances of different models are compared using the FID metric. The smaller the FID of a generative model is, the more similar the generated samples are to the empirical samples. EOTGAN is the least affected model among all the four, as the ratio r increases, i.e. the generated samples by EOTGAN are more similar to the empirical ones in the feature space regarding FID. FID of WGAN is larger than that of EOTGAN. As more generated samples are mixed the FIDs of DCGAN and EOTGM grow even faster, which means the samples generated by these two models are less similar to the empirical samples.

8.4 Summary

In this chapter, we took a different path from previous chapters and considered the learning of implement probabilistic generative models. An implicit model offers larger freedom in choosing a generator that induces the model distribution, but brings the challenges of model learning due to the loss of tractability of likelihood. Thus optimal transport was introduced and employed to formulate the cost. Optimal transport distance used in learning the model also has interesting connections with generative adversarial networks and auto-encoder-like models. With an entropy-regularization, the optimal transport cost can be computed more efficiently. Two learning algorithms based on the entropy-regularized optimal transport cost were then discussed and demonstrated.

Different from graphical models discussed in previous chapters, an implicit model has its advantage of flexible generator choice and efficient sampling (or generating sampling), sampling from which is usually a feed-forward pass of the generator once learned. Thus, an implicit model has its application domains where the above-mentioned properties are merited more than explicit form and likelihood tractability of the model's distribution.

8.5 Related Literature

Deep generative models that use the non-linear mappings of neural networks have received a lot of attention as a branch of graphical models. The modeling capacity of deep generative models is impressive in high-dimensional space (e.g., image super-resolution [100], image de-noising [26, 34], image segmentation [166], speech generating [138] etc.), but the learning of these models is not trivial. The learning difficulty lies in modeling challenges within the high-dimensionality or our trading more complex generators (for expressivity) with distribution tractability. Thus, works in pursuit of a better understanding of deep generative models have been carried out, and also different model learning methods or costs have been proposed in relieving the learning difficulty.

One popular track of deep generative models falls into the deep variational models which are usually built with the abstraction of graphical models and neural network based generators. To preserve the efficient model learning and some level of likelihood tractability, deep generators are usually restricted with special choices of parameterization (e.g., reparameterization [88]) or generator structures (e.g., the variational inference by normalizing flows, see [90, 149, 169]). These choices of designs sometimes also make use of extra auxiliary latent variables in order to further increase the flexibility of generators [145]. Interesting works falling in this family includes variational auto-encoders (VAEs) [89], neural variational inference and learning (NVIL) [95, 102, 121], etc. Learning of these models are usually based on the likelihood variational bound (as discussed in Section 5.4) or bounds of this variational bound itself.

If the generator constraints are further relaxed, more freedom is gained in implementing generators with deep neural networks (which also brings model learning challenges). A prominent example is the generative adversarial network (GAN) [56], which comes with an implicit underlining distribution of the model. In the vanilla GAN, a generator produces synthetic samples and a discriminator endeavors to distinguish between real samples and synthetic samples. Generators and discriminators are both implemented using (deep) neural networks, and play an adversary game against each other using a *min-max* optimization to learn parameters of neural networks. More intuitively speaking, for the generator, the game turns out to be minimizing Jensen-Shannon divergence between target distribution (the induced distribution by the generator) when the discriminator is optimal. This interesting game of casting generative model learning into a adversary game triggered more

research in generator and learning method design [10, 50, 71, 82, 120, 144, 144, 190], and also explanation exploration [7, 8, 21, 103, 115, 185].

Apart from the reasons mentioned above on learning difficulty of deep generative models, new issues raise from the deep generative models themselves. Let us take the learning of the vanilla GAN as an example, which has limitations due to the Jensen-Shannon divergence at the optimal region of its discriminator [7]. Firstly, the limitation is that the gradient of the Jensen-Shannon divergence cost with regard to the generator vanishes as discriminator approaches optimal, which stops the generator from further learning. Secondly, the limitation is due to high sensitivity of Jensen-Shannon divergence to slight perturbations, which can be large between a distribution $p(\boldsymbol{x})$ and a distribution $p(\boldsymbol{x} + \boldsymbol{\varepsilon})$ where ε is perturbation. These difficulties were relieved in the followed-up model Wasserstein GAN (WGAN) [8]. Wasserstein distance stems from optimal transport (OT) distance as explained in Section 8.1. The WGAN formulation does not require an explicit discriminator and it does not have the vanishing-gradient issue due to the cost function. Additionally, Wasserstein distance or OT is upper bounded by the standard deviation of perturbation ε, addressing the second limitation.

The story did not end there since the duality of Wasserstein distance brought new challenges as briefed in Remark 8.2. Kantorovich-Rubinstein duality used in WGAN requires a supremum over an infinite set of all Lipschitz functions with Lipschitz constant equal to one. Various sub-optimal techniques were proposed to enforce the Lipschitz property. An example was weight clipping [8] where neural network parameters (weights) are updated first without Lipschitz constraint and then projected to satisfy Lipschitz constraint in each iteration. Different enforcement were used later on in addressing it, e.g., gradient penalty [59] and spectrum normalization [120]. Further work attempted OT in different spaces, e.g., [3, 48].

Part IV

Epilogue

Chapter 9

Closing Remarks

9.1 Summary of the Dissertation

In this dissertation, inference and learning problems in probabilistic graphical models were studied. As introduced in the background in Part I, inference and learning in probabilistic graphical models cover a wide range of topics. This dissertation mainly investigated selected topics in the subject that is broadly applicable to various problems and different disciplines. The dissertation started with the big picture of interesting problems in graphical models, and identified the related topics to which this dissertation related in Part I (Chapter 2).

In Part II (Chapter 3 and 4), we focused on inference in Markov random fields (MRFs), i.e., undirected graphical models. Two meta algorithms (or models) were proposed in addressing the inference in MRFs. We firstly studied a α-divergence minimization problem in an approximate way. The problem in connection with messages over an MRF or its factor graph showed an interesting intuition on belief propagation. The interesting connection gave rise to the α belief propagation (α-BP) algorithm in general graphs. Based on the belief update rule of α-BP, sufficient conditions were developed for the convergence of α-BP to a fixed point for the binary-state space of each variable. The conditions allow us to check if running α-BP for a given problem would be guaranteed to converge without actual implementation and executing the algorithm, which makes a difference in large inference problems. In pursuit of faster and more accurate inference algorithms, Chapter 4 addressed the generic inference issue on a hyper-graph, i.e., a region graph. In a region graph, a node can be associated with multiple random variables and their potential preferences jointly. This change gives room for performance gains of inference, especially when there strongly conflicting potentials that form loops. Instead of developing an iterative message passing algorithm, a neural network was employed to formulate an optimization problem where the region-based free energy of the region graph was to be minimized, leading to the region-based energy neural network (RENN). RENN has the free energy minimization intuition as typical

message passing methods, and also makes use of the efficient optimizers of modern neural network frameworks that offer highly-customizable models. With these advantages, RENN showed competitive performances in inference tasks of MRFs.

Part III moved the focus to learning (parameter learning) of graphical models. In connecting to the inference part in Part II, Part III started with the role that inference plays in general model learning problems in graphic models in Chapter 5, which is also numerically demonstrated by the performance comparison of MRF learning with difference inference algorithms. The rest part extended the model learning discussion into a more general case, i.e., incomplete observations or the presence of hidden variables with a focus on directed graphical models. Learning with hidden variables was firstly treated within expectation maximization framework. The normalizing flows were brought into the directed models in order to gain the model expressivity. The model learning with hidden variables and normalizing flows was firstly addressed for independently-and-identically-distributed samples in Chapter 6, and was then extended into a dynamic model (a hidden Markov model) in Chapter 7 for sequential or temporal signals. The applications of these developed models were demonstrated with image and speech data. The last chapter (Chapter 8) of this part considered a likelihood-free case where generative models were learned via entropy-regulated optimal transport (EOT) distance instead of maximum likelihood. Advantages and disadvantages of EOT in contrast with maximum likelihood were discussed, e.g., more freedom in choosing a generator for the model learning via EOT but the loss of likelihood tractability, etc.

9.2 Open Directions

Given the extensive work devoted to the study of probabilistic graphic models, there are still open problems that remain to be answered. Take the first topic we discussed, i.e., belief propagation methods and its variants including α belief propagation in this dissertation, the issue of convergence to a stable solution of these deterministic approximate inference methods still remains. Problems to which answers would be interesting include but not limited to: i) How fast a belief propagation method can be in converging to a stable solution (if it does converge)? ii) What is the quality of the converged solution? Although we usually enjoy the efficiency of these deterministic approximate inference methods empirically, theoretical insights would be very helpful in algorithm choice and system designs if the guideline is available (answers to the question i) here). Another critical issue is the quality of the returned solution by a selected belief propagation method. The current answer to this question mainly relies on empirical evaluations. One usually has to manually tune the selected algorithm in practice and try out the configurations. Thus, the error analyses of these deterministic approximate inference methods, though challenging, would be valuable in support of this family of methods. There are some works in attempting problem i) and ii) in restricted cases (see, e.g., Section 3.7), i.e., some special cases of graphical models, which are inspiring.

The other important direction is the development of automated inference methods. It would be an interesting study to reduce (or avoid if possible) the manual tuning and adjustment, e.g., message update rules and scheduling methods. On this development track, jointly consider making use of modern hardware (e.g., high-capacity GPUs) in algorithm designs for scaled-up applications is also important since we face more high-dimensional data that comes in large volume nowadays. Work in Chapter 4 made a step in this direction. RENN is able to answer multiple queries with one execution on an MRF. When partial random variables are instantiated as observed evidence, inference on posteriors (condition on evidence) becomes evidence-dependent for RENN and also belief propagation methods, i.e., queries associated with each evidence needs the execution or call of the inference algorithm once (see, e.g., Section 5.4). Modern variational inference such as VAEs (see, e.g., Section 4.6, Section 8.5) circumvents the evidence-dependent issue and presents a very interesting way of modeling by directly learning a posterior distribution of the hidden or latent variables of interests. In this case, answers to certain *predefined* queries (associated with the directly-modeled posterior) can be efficiently computed. Queries falling out of the predefined class have to rely on Monte Carlo estimations or just can not be answered. Therefore, further development in this direction towards automated inference is anticipated.

New issues raise with model learning as well when modern neural networks are adopted into graphical models for higher modeling capacity. Classic statistic models with proper factorization usually allow factor-wise updates from closed-form optimization solutions. This property that factors can be decomposed in updates significantly simplifies the learning of directed graphical models, but is lost when customizable neural network models are incorporated. Meanwhile, the data amount is larger for models to extract useful information to adjust its parameters. The learning becomes more complex with the presence of hidden variables where inference algorithms are required as subroutines (see, e.g., Sectiton 5.4) to generate the information (usually a conditional distribution) missed from the (partial or incomplete) evidence. Therefore, efficient learning algorithms are needed to incorporate the changes, i.e., the missing information or data, loss of decomposing factors in the objective function, and introduction of non-linear functional forms of modern neural network models. This motivates our consideration in three dimensions: i) modeling parameterization or structures (e.g., how to factorize a joint distribution and how to parameterize each factor); ii) the objective function that we aim to optimize (e.g., likelihood, variational bound of likelihood, pseudolikelihood, likelihood-free objectives, etc); iii) subroutine inference (e.g., exact inference, approximate inference). Although these three decisions affect each other, there is sufficient freedom and independence such that each one can be improved separately. Additionally, a learning algorithm design is also dependent on the model's final application scenarios or tasks. For instance, the difference between discriminative and generative learning (see, e.g., Remark 7.4, applications in Sections 7.4 and 7.5).

Bibliography

[1] Pytorch autograd. https://pytorch.org/docs/stable/autograd.html.

[2] Tensorflow automatic differentiation. https://pytorch.org/docs/stable/autograd.html.

[3] Jonas Adler and Sebastian Lunz. Banach wasserstein GAN. *CoRR*, abs/1806.06621, 2018. URL http://arxiv.org/abs/1806.06621.

[4] 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.

[5] 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.

[6] Luca Ambrogioni, Umut Güçlü, Yağmur Güçlütürk, Max Hinne, Marcel AJ van Gerven, and Eric Maris. Wasserstein variational inference. In *Advances in Neural Information Processing Systems*, pages 2473–2482, 2018.

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

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

[9] N.M.R. Ashwin, Leonard Barnabas, Amalraj Ramesh Sundar, Palaniyandi Malathi, Rasappa Viswanathan, Antonio Masi, Ganesh Kumar Agrawal, and Randeep Rakwal. Comparative secretome analysis of colletotrichum falcatum identifies a cerato-platanin protein (epl1) as a potential pathogen-associated molecular pattern (pamp) inducing systemic resistance in sugarcane. *Journal of Proteomics*, 169:2 – 20, 2017. ISSN 1874-3919. URL http://www.sciencedirect.com/science/article/pii/S1874391917301872. 2nd World Congress of the International Plant Proteomics Organization.

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

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

[12] J. R. Bellegarda and D. Nahamoo. Tied mixture continuous parameter modeling for speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(12):2033–2045, Dec 1990. ISSN 0096-3518.

[13] Bing-Hwang Juang, S. Levinson, and M. Sondhi. Maximum likelihood estimation for multivariate mixture observations of markov chains (corresp.). *IEEE Transactions on Information Theory*, 32(2):307–309, March 1986. ISSN 0018-9448.

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

[15] Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the vegan cookbook, 2017.

[16] Jan Buys, Yonatan Bisk, and Yejin Choi. Bridging hmms and rnns through architectural transformations. In *32nd Conference on Neural Information Processing Systems, IRASL workshop*. 2018.

[17] Olivier Cappe and Eric Moulines. On-line expectation-maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, Jun 2009. ISSN 1467-9868. URL `http://dx.doi.org/10.1111/j.1467-9868.2009.00698.x`.

[18] S. Chatterjee and W. B. Kleijn. Auditory model-based design and optimization of feature vectors for automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1813–1825, Aug 2011. ISSN 1558-7916.

[19] Jianfei Chen, Jun Zhu, Yee Whye Teh, and Tong Zhang. Stochastic expectation maximization with variance reduction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7967–7977. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/8021-stochastic-expectation-maximization-with-variance-reduction.pdf`.

[20] 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`.

[21] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2172–2180. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6399-infogan-interpretable-representation-learning-by-information-maximizin` `pdf`.

[22] 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.

[23] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, June 2005.

[24] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016. URL `http://arxiv.org/abs/1609.01704`.

[25] Sebastian Claici, Edward Chien, and Justin Solomon. Stochastic wasserstein barycenters. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 998–1007, 2018.

[26] Antonia Creswell and Anil Anthony Bharath. Denoising adversarial autoencoders. *CoRR*, abs/1703.01220, 2017. URL `http://arxiv.org/abs/1703.01220`.

[27] 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.

[28] Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 685–693,

Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/cuturi14.html.

[29] 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.

[30] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, December 1989. ISSN 0824-7935. URL https://doi.org/10.1111/j.1467-8640.1989.tb00324.x.

[31] 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.

[32] 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.

[33] 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.

[34] Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648, 2016. URL http://arxiv.org/abs/1611.02648.

[35] W. Ding, S. Li, H. Qian, and Y. Chen. Hierarchical reinforcement learning framework towards multi-agent navigation. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 237–242, Dec 2018.

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

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

[38] Justin Domke. Provable smoothness guarantees for black-box variational inference. *CoRR*, abs/1901.08431, 2019. URL http://arxiv.org/abs/1901.08431.

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

[40] 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`.

[41] 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.

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

[43] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Mach. Learn.*, 32(1):41–62, July 1998. ISSN 0885-6125. URL `https://doi.org/10.1023/A:1007469218079`.

[44] 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`.

[45] M. Gales and S. Young. *Application of Hidden Markov Models in Speech Recognition*. now, 2008. ISBN 9781601981202. URL `https://ieeexplore.ieee.org/document/8187420`.

[46] Victor Garcia Satorras, Zeynep Akata, and Max Welling. Combining generative and discriminative models for hybrid inference. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13825–13835. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/9532-combining-generative-and-discriminative-models-for-hybrid-inference.pdf`.

[47] 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.

[48] Mevlana Gemici, Zeynep Akata, and Max Welling. Primal-dual wasserstein gan, 2018.

[49] Aude Genevay, Gabriel PeyrÃ©, and Marco Cuturi. Gan and vae from an optimal transport point of view, 2017.

[50] 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.

[51] 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`.

[52] 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.

[53] 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.

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

[55] 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`.

[56] 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.

[57] M. P. Griffin. Heart Rate Characteristics: Novel Physiomarkers to Predict Neonatal Infection and Death. *PEDIATRICS*, 116(5):1070–1074, November 2005. ISSN 0031-4005, 1098-4275.

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

[59] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, *et al.* Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[60] Sunil Kumar Gupta, Dinh Phung, Brett Adams, and Svetha Venkatesh. Regularized nonnegative shared subspace learning. *Data Mining and Knowledge Discovery*, 26(1):57–97, Jan 2013. ISSN 1573-756X. URL `https://doi.org/10.1007/s10618-011-0244-8`.

[61] Ilan Gur, Gal Markel, Yaron Nave, Igor Vainshtein, Arik Eisenkraft, and Arieh Riskin. A mathematical algorithm for detection of Late-onset Sepsis in very-low birth weight infants: A preliminary diagnostic test evaluation. *Indian Pediatr*, 51(8):647–650, August 2014. ISSN 0019-6061, 0974-7559.

[62] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, June 2006.

[63] Trienani Hariyanti, Saori Aida, and Hiroyuki Kameda. Samawa language part of speech tagging with probabilistic approach: Comparison of unigram, HMM and TnT models. *Journal of Physics: Conference Series*, 1235: 012013, jun 2019. URL `https://doi.org/10.1088%2F1742-6596%2F1235%2F1%2F012013`.

[64] 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.

[65] Jose Hernandez-Lobato, Yingzhen Li, Mark Rowland, Thang Bui, Daniel Hernandez-Lobato, and Richard Turner. Black-box alpha divergence minimization. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1511–1520, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/hernandez-lobatob16.html`.

[66] 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`.

[67] 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.

[68] Jamie Fletcher Hicks and Karen Fairchild. Heart rate observation (HeRO) monitoring was developed for detection of sepsis in preterm infants.[...] The HeRO monitor is now in use in many NICUs in the USA and was approved in 2012 for use in Europe. page 5, 2013.

[69] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of

four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012. ISSN 1053-5888.

[70] Geoffrey E. Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. URL `https://doi.org/10.1007/978-3-642-35289-8_32`.

[71] 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.

[72] A. HonorÃ©, D. Liu, D. Forsberg, K. Coste, E. Herlenius, S. Chatterjee, and M. Skoglund. Hidden markov models for sepsis detection in preterm infants. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1130–1134, 2020.

[73] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift fur Physik*, pages 253–258, 1925.

[74] 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.

[75] 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.

[76] Z. Jiang, Z. Lin, and L. S. Davis. Label consistent k-svd: Learning a discriminative dictionary for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2651–2664, Nov 2013. ISSN 0162-8828.

[77] 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.

[78] 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.

[79] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 1960.

[80] Belhal Karimi, Hoi-To Wai, Eric Moulines, and Marc Lavielle. On the global convergence of (fast) incremental expectation maximization methods. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 2837–2847. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8550-on-the-global-convergence-of-fast-incremental-expectation-maximizatio` `pdf`.

[81] Wahab Khan, Ali Daud, Jamal A Nasir, and Tehmina Amjad. A survey on the state-of-the-art machine learning models in the context of nlp. *Kuwait journal of Science*, 43(4), 2016.

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

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

[84] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. In *Advances in Neural Information Processing Systems*, pages 11566–11575, 2019.

[85] Owen Kimball and Mari Ostendorf. On the use of tied-mixture distributions. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, pages 102–107, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics. ISBN 1-55860-324-7. URL `https://doi.org/10.3115/1075671.1075694`.

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

[87] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[88] 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.

[89] 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`.

[90] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4743–4751. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/`

6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf.

[91] 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.

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

[93] Viktoriya Krakovna and Finale Doshi-Velez. Increasing the Interpretability of Recurrent Neural Networks Using Hidden Markov Models. *arXiv e-prints*, page arXiv:1606.05320, Jun 2016.

[94] 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.

[95] 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.

[96] 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.

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

[98] Julia A. Lasserre, Christopher Bishop, and Tom Minka. Principled hybrids of generative and discriminative models. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 87–94. IEEE Computer Society, June 2006. URL https://www.microsoft.com/en-us/research/publication/principled-hybrids-of-generative-and-discriminative-models/.

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

[100] 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.

[101] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909, 2018. URL `http://arxiv.org/abs/1805.00909`.

[102] Chongxuan Li, Chao Du, Kun Xu, Max Welling, Jun Zhu, and Bo Zhang. To relieve your headache of training an mrf, take advil. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Sylgsn4Fvr`.

[103] 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.

[104] L. Li, Y. Zhao, D. Jiang, Y. Zhang, F. Wang, I. Gonzalez, E. Valentin, and H. Sahli. Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 312–317, Sep. 2013.

[105] 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`.

[106] J. H. Lim and J. C. Ye. Geometric GAN. *ArXiv e-prints*, May 2017.

[107] 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`.

[108] 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.

[109] Larkin Liu, Yu-Chung Lin, and Joshua Reid. Improving the performance of the LSTM and HMM models via hybridization. *CoRR*, abs/1907.04670, 2019. URL `http://arxiv.org/abs/1907.04670`.

[110] L. Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24 (1):36–50, 1979.

[111] Carla Lopes and Fernando Perdigao. Phoneme recognition on the timit database. In Ivo Ipsic, editor, *Speech Technologies*, chapter 14. IntechOpen, Rijeka, 2011. URL https://doi.org/10.5772/17600.

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

[113] 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.

[114] 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.

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

[116] Yajie Miao and Florian Metze. Improving low-resource cd-dnn-hmm using dropout and multilingual dnn training. In *INTERSPEECH*, 2013.

[117] 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.

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

[119] 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/.

[120] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[121] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1791–1799, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/mnih14.html.

[122] G. Monge. *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale, 1781. URL https://books.google.se/books?id=IG7CGwAACAAJ.

[123] 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.

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

[125] 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.

[126] 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`.

[127] 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.

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

[129] 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.

[130] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944. URL `https://link.aps.org/doi/10.1103/PhysRev.65.117`.

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

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

[133] 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`.

[134] Giorgio Patrini, Marcello Carioni, Patrick Forré, Samarth Bhargav, Max Welling, Rianne van den Berg, Tim Genewein, and Frank Nielsen. Sinkhorn autoencoders. *CoRR*, abs/1810.01118, 2018. URL `http://arxiv.org/abs/1810.01118`.

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

[136] 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.

[137] G. Peyré and M. Cuturi. Computational Optimal Transport. *ArXiv e-prints*, March 2018.

[138] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. *CoRR*, abs/1811.00002, 2018. URL `http://arxiv.org/abs/1811.00002`.

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

[140] Yuan Qi. Extending expectation propagation for graphical models. 12 2007.

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

[142] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[143] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[144] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*, November 2015.

[145] Rajesh Ranganath, Dustin Tran, and David M. Blei. Hierarchical variational models, 2015.

[146] H. E. RAUCH, F. TUNG, and C. T. STRIEBEL. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, aug 1965.

[147] S. Ren, V. Sima, and Z. Al-Ars. Fpga acceleration of the pair-hmms forward algorithm for dna sequence analysis. In *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1465–1470, Nov 2015.

[148] 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`.

[149] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2015.

[150] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, 2001.

[151] Tom Richardson and Ruediger Urbanke. *Modern Coding Theory*. Cambridge University Press, USA, 2008. ISBN 0521852293.

[152] 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.

[153] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Florida USA, 16–18 Apr 2009. PMLR.

[154] 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.

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

[156] 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`.

[157] B. Schmitzer. Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *ArXiv e-prints*, October 2016.

[158] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, June 2015.

[159] 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.

[160] B. Sullivan, A. Wallman-Stokes, J. Isler, R. Sahni, J. Moorman, K. Fairchild, and D. Lake. Early Pulse Oximetry Data Improves Prediction of Death and Adverse Outcomes in a Two-Center Cohort of Very Low Birth Weight Infants. *Amer J Perinatol*, 35(13):1331–1338, November 2018. ISSN 0735-1631, 1098-8785.

[161] D. Sundman, S. Chatterjee, and M. Skoglund. Design and analysis of a greedy pursuit for distributed compressed sensing. *IEEE Transactions on Signal Processing*, 64(11):2803–2818, June 2016. ISSN 1053-587X.

[162] Dennis Sundman, Saikat Chatterjee, and Mikael Skoglund. Distributed greedy pursuit algorithms. *Signal Processing*, 105:298 – 315, 2014. ISSN 0165-1684. URL http://www.sciencedirect.com/science/article/pii/S016516841400245X.

[163] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Found. Trends Mach. Learn.*, 4(4):267–373, April 2012. ISSN 1935-8237. URL https://doi.org/10.1561/2200000013.

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

[165] J. Tang, C. Deng, and G. Huang. Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, 27(4):809–821, April 2016. ISSN 2162-237X.

[166] Marvin T. T. Teichmann and Roberto Cipolla. Convolutional crfs for semantic segmentation. *CoRR*, abs/1805.04777, 2018. URL http://arxiv.org/abs/1805.04777.

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

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

[169] Jakub M. Tomczak and Max Welling. Improving variational auto-encoders using householder flow. *CoRR*, abs/1611.09630, 2016. URL http://arxiv.org/abs/1611.09630.

[170] 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.

[171] 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.

[172] C. Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008. ISBN 9783540710509. URL https://books.google.se/books?id=hV8o5R7_5tkC.

[173] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

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

[175] 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.

[176] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000.

[177] 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.

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

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

[180] 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.

[181] 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.

[182] 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.

[183] 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.

[184] Hao Xiong, Yuanzhen Guo, Yibo Yang, and Nicholas Ruozzi. One-shot inference in markov random fields. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 19. AUAI Press, 2019. URL `http://auai.org/uai2019/proceedings/papers/19.pdf`.

[185] 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.

[186] 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.

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

[188] 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.

[189] 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.

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

[191] 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.

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