Cumulative distribution networks: Inference, estimation and applications of graphical models for cumulative distribution functions

by

Jim C. Huang

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Cumulative distribution networks:

Inference, estimation and applications of graphical models for cumulative distribution functions

Jim C. Huang

Doctor of Philosophy

Graduate Department of Electrical and Computer Engineering

University of Toronto

2009

This thesis presents a class of graphical models for directly representing the joint cumulative distribution function (CDF) of many random variables, called *cumulative distribution networks* (CDNs). Unlike graphical models for probability density and mass functions, in a CDN, the marginal probabilities for any subset of variables are obtained by computing limits of functions in the model. We will show that the conditional independence properties in a CDN are distinct from the conditional independence properties of directed, undirected and factor graph models, but include the conditional independence properties of bidirected graphical models. As a result, CDNs are a parameterization for bidirected models that allows us to represent complex statistical dependence relationships between observable variables. We will provide a method for constructing a factor graph model with additional latent variables for which graph separation of variables in the corresponding CDN implies conditional independence of the separated variables in both the CDN and in the factor graph with the latent variables marginalized out. This will then allow us to construct multivariate extreme value distributions for which both a CDN and a corresponding factor graph representation exist.

In order to perform inference in such graphs, we describe the 'derivative-sum-product' (DSP) message-passing algorithm where messages correspond to derivatives of the joint

cumulative distribution function. We will then apply CDNs to the problem of learning to rank, or estimating parametric models for ranking, where CDNs provide a natural means with which to model multivariate probabilities over ordinal variables such as pairwise preferences. We will show that many previous probability models for rank data, such as the Bradley-Terry and Plackett-Luce models, can be viewed as particular types of CDN. Applications of CDNs will be described for the problems of ranking players in multiplayer team-based games, document retrieval and discovering regulatory sequences in computational biology using the above methods for inference and estimation of CDNs.

Acknowledgements

I'd first like to acknowledge my advisor Brendan Frey for his guidance, great wisdom and insights throughout the years. As an advisor, Brendan has provided an excellent environment for doing research in which I have had the freedom to explore unique research directions and develop as a researcher. I'd like to thank Zoubin Ghahramani, Frank Kschischang, Quaid Morris and Radford Neal as members of my doctoral thesis committee for providing valuable feedback that significantly improved the writing of this thesis. I'd like to thank Ilya Sutskever for his proofreading services, helpful comments on drafts of the thesis and for many exciting conversations around the globe at various times. I'd like to also acknowledge Tom Minka, Ricardo Silva and Yee Whye Teh for insights that have contributed to the writing of this thesis. My thanks go out to Nebojsa Jojic, Anitha Kannan and John Winn for hosting me during my internships at Microsoft Research, for providing challenging environments in which to do research and for being exemplary mentors from whom I have learned much.

I'd like to thank Tomas Babak and Desiree Tillo for discussions and collaborations in bioinformatics and without whom I would have been woefully unaware of the intriguing complexities of molecular biology and medicine. I'd like to thank Danilo Silva for highly insightful discussions on mathematics, music and philosophy and the interplay between the three. Thanks go out to present and past members of Brendan Frey's Probabilistic and Statistical Inference (PSI) group and the University of Toronto Machine Learning group. Finally, I'd like to thank the members of my former band and my brother Kenneth, to whom this thesis is dedicated.

Contents

List of Tables

List of Figures

Notation

Symbol	Definition
	Basic notation
$\mathbb{P}$	Probability measure
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^+$	The set of non-negative real numbers
$\mathbb{R}^N$	Euclidean vector space of N-dimensional vectors
$\mathcal{X}$	Countable discrete state space
$\lvert \cdot \rvert$	Cardinality of a set
$\delta(\cdot)$	Dirac delta function
$F(\mathbf{x})$	Cumulative distribution function
$P(\mathbf{x})$	Probability density/mass function
$F(\mathbf{x}\lvert M)$	Conditional cumulative distribution function given event M
$P(\mathbf{x}\lvert M)$	Conditional probability density function given event M
$Gaussian(\cdot; \mathbf{m}, \mathbf{\Sigma})$	Multivariate Gaussian probability density with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{\Sigma}$
$\Phi(\cdot; \mathbf{m}, \mathbf{\Sigma})$	Multivariate Gaussian cumulative distribution function with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{\Sigma}$

Symbol	Definition
	Graphical models
V	Discrete set of variable nodes
E	Discrete set of edges
S	Discrete set of factor/function nodes
G	Graph consisting of nodes in V and edge set E
$\mathcal{G}$	Bipartite graph consisting of nodes in (V, S) and edge set E
$c, \mathcal{C}$	Clique, set of cliques
α, β	Indices for variable nodes in V
(α, β)	Directed edge from node $\alpha \in V$ to node $\beta \in V$
A, B, C, D, U	Discrete subsets of variable nodes in V
X_α	Random variable corresponding to node $\alpha \in V$
$\mathbf{X}_A$	Vector of random variables $X_\alpha \; \forall \; \alpha \in A$
x_α	Observed value for random variable X_α
$\mathbf{x}_A$	Vector of observed values for $X_\alpha \; \forall \; \alpha \in A$
$\omega(\mathbf{x}_A), \mathbf{X}_A \leq \mathbf{x}_A$	Event $\cap_{\alpha \in A}\{X_\alpha \leq x_\alpha\}$
$\mathcal{Z}$	Normalizing constant
ψ_c	Clique potential function
s	Function/factor nodes in S
$\mathcal{N}(\alpha), \mathcal{N}(A)$	Neighboring function/factor nodes for variable node α, variable node set A in bipartite graph $\mathcal{G}$
$\mathcal{N}(s)$	Neighboring variable nodes for function/factor node s in bipartite graph $\mathcal{G}$
$f_s(\mathbf{x}_s), \phi_s(\mathbf{x}_s)$	Specification for function/factors in a bipartite graph $\mathcal{G}$, with $\mathbf{x}_s \equiv \mathbf{x}_{\mathcal{N}(s)}$

Symbol	Definition
τ_s^α	Subtree of bipartite graph $\mathcal{G}$ rooted at variable node $\alpha \in V$ and containing function/factor node $s \in S$
τ_α^s	Subtree of bipartite graph $\mathcal{G}$ rooted at function/factor node $s \in S$ containing variable node $\alpha \in V$
$T_s(\tau_\alpha^s), T_\alpha(\tau_s^\alpha)$	Products of function/factors contained in the subtrees τ_α^s and τ_s^α
$\mu_{s\to\alpha}(x_\alpha), \mu_{\alpha\to s}(x_\alpha)$	Sum-product messages from factor node s to variable node α and from variable node α to factor node s
$\mu_{s\to\alpha}(\mathbf{x}), \lambda_{s\to\alpha}(\mathbf{x})$	Derivative-sum-product messages from function node s to variable node α
$\mu_{\alpha\to s}(\mathbf{x}), \lambda_{\alpha\to s}(\mathbf{x})$	Derivative-sum-product messages from variable node α to function node s
$A \perp\!\!\!\perp B$	Marginal independence of variable sets $\mathbf{X}_A$ and $\mathbf{X}_B$
$A \perp\!\!\!\perp B \vert C$	Conditional independence of variable sets $\mathbf{X}_A$ and $\mathbf{X}_B$ given $\mathbf{X}_C$
$A \perp\!\!\!\perp B \vert \omega(\mathbf{x}_C)$	Conditional independence of variable sets $\mathbf{X}_A$,$\mathbf{X}_B$ given event $\omega(\mathbf{x}_C)$
$\partial_{\mathbf{x}_A}\left[\cdot\right]$	Mixed partial derivative/finite difference with respect to function arguments $x_\alpha \; \forall \; \alpha \in A$
$\int_{-\infty}^{\mathbf{x}_A} \cdot \, d\mathbf{u}_A$	Integral with respect to function arguments u_α, over the range $-\infty \leq u_\alpha \leq x_\alpha \; \forall \; \alpha \in A$
$\int_{\mathbf{x}_A} \cdot \, d\mathbf{u}_A$	Integral over the range $-\infty < x_\alpha < \infty, \; \forall \; \alpha \in A$,
$\lim_{\mathbf{x}_A \to \mathbf{z}_A}$	Limit as $x_\alpha \to z_\alpha \; \forall \; \alpha \in A$
$g(-\infty, \mathbf{y}), g(\infty, \mathbf{y})$	$\lim_{\mathbf{x}\to-\infty} g(\mathbf{x}, \mathbf{y})$, $\lim_{\mathbf{x}\to\infty} g(\mathbf{x}, \mathbf{y})$ respectively
$\mathbf{x}_A \leq \mathbf{y}_A$	Vector inequality consisting of elementwise inequalities $x_\alpha \leq y_\alpha \; \forall \; \alpha \in A$

Symbol	Definition
	Learning to rank
$X_\alpha \preceq X_\beta$	Stochastic ordering relationship between random variable X_β and X_α
Γ_t	Game t
$\mathcal{P}_t, \mathcal{T}_t$	Set of players and partition of players into teams for game t
$\mathcal{D}$	Set of observations
D_n	Observation in $\mathcal{D}$
G_n	Order graph for observation D_n
$\mathcal{E}$	Set of edges in an order graph
$\mathcal{V}$	Set of nodes in an order graph
α, β	Nodes in $\mathcal{V}$
$\alpha \prec \beta$	Preference of β over α
e, e'	Indices for edges in $\mathcal{E}$, with $e = (\alpha, \beta)$
$\pi_{\alpha\beta}, \pi_e$	Preference variable for edge $e = (\alpha, \beta)$
L	Dimension of feature vector space
$\mathbf{x}$	Feature vector in $\mathbb{R}^L$
$\rho(\mathbf{x}; \mathbf{a})$	Ranking function with parameters $\mathbf{a}$
r_e	Difference $\rho(\mathbf{x}_\beta; \boldsymbol{\theta}) - \rho(\mathbf{x}_\alpha; \boldsymbol{\theta})$ for edge $e = (\alpha, \beta),\ \alpha, \beta \in V$
$\mathcal{L}$	Loss functional
$\nabla_{\boldsymbol{\theta}}$	Gradient operator with respect to parameter vector $\boldsymbol{\theta}$
$\mathcal{A}$	Alphabet for sequences
s_α, L_α	Sequence of length L_α for node $\alpha \in \mathcal{V}$, so that $s_\alpha \in \mathcal{A}^{L_\alpha}$
$\mathbf{x}_\alpha$	Feature vector for node $\alpha \in \mathcal{V}$
$\mathbf{M}$	Position-specific scoring matrix (PSSM)

Chapter 1

Introduction

Probabilistic graphical models are widely used for representing statistical dependence relationships between random variables in problems such as error-correction coding [41], information retrieval [7, 9, 44, 70] and computational systems biology [25, 26, 27, 28, 29]. A graphical model provides a pictorial means of specifying a joint probability density function (PDF) of many continuous random variables, the joint probability mass function (PMF) of many discrete random variables, or a distribution defined over a mixture of continuous and discrete variables. Each variable in the model corresponds to a node in a graph and edges between nodes in the graph convey statistical dependence relationships between the variables in the model. The graphical formalism allows one to ascertain the conditional independence relationships between random variables in a model by inspecting the corresponding graph, where the separation of nodes in the graph implies a particular conditional independence relationship between the corresponding variables. For example, in computational biology, genes that participate in certain biological processes are likely to share a common set of regulators that control their patterns of expression. The constraints dictating which genes are regulated by which regulators can be expressed in the form of a graph with edges linking regulator nodes to regulated gene nodes.

A consequence of representing independence constraints between subsets of variables

using a graph is that the joint probability factors into a product of functions defined over subsets of neighboring nodes in the graph. This allows us to decompose a large multivariate distribution into a product of simpler functions, so that the task of inference and estimation of such models can also be simplified and efficient algorithms for performing these tasks can be implemented. For any given application, a graphical model also allows us to simplify the computations required for parameter estimation and inference for variables in the model. As graphical models have been applied to increasingly complex problems with larger numbers of variables to be modeled, the computational complexity required for estimation and inference has also dramatically increased. For many real-world problems, one must often introduce latent variables into a model in order to explain complex statistical dependence relationships between observable variables and then marginalize out these latent variables to obtain a model over the observable variables. There are possibly an infinite number of latent variable models associated with any given model defined over observable variables, so designing latent variables for any given application can often present difficulties in terms of model identifiability [10], which causes problems if one wishes to interpret the parameters in a graphical model. Finally, performing inference and estimation under such models often requires one to either approximate intractable marginalization operations [50] or to sample from the model using Markov Chain Monte Carlo (MCMC) methods [53]. These issues may hamper the applicability of graphical models for many real-world problems in the presence of latent variables, since it may be computationally expensive or challenging to perform exact inference and estimation, in addition to selecting amongst several possible latent variable models for any given problem.

Another limitation of graphical models is that the joint PDF/PMF itself might not be appropriate for certain applications. For example, in learning to rank, the *cumulative distribution function* (CDF) arises naturally as a probability measure over inequality events of the type $\{X \leq x\}$. The joint CDF lends itself to problems that are easily

described in terms of inequality events in which statistical dependence relationships also exist among events. Examples of this type of problem include web search and document retrieval [7, 9, 35, 70], predicting ratings for movies [57] or predicting multiplayer game outcomes with a team structure [23]. In contrast to the canonical problems of classification or regression, in learning to rank we are required to learn some mapping from inputs to inter-dependent output variables so that we may wish to model both stochastic orderings of variable states and statistical dependence relationships between variables.

As a means to address the above issues, in this thesis we present a class of graphical models called *cumulative distribution networks* (CDN) in which we represent the joint CDF of a set of variables. In Chapter 3 we will present the basic properties of CDNs and show that the rules for ascertaining conditional independence relationships amongst variables in a CDN are distinct from the rules in directed, undirected and factor graphs [56, 42, 41]. In contrast to these graphical models, marginalization in CDNs involves tractable operations such as computing limits and the derivatives of local functions in the graph. In addition, the global normalization constraint can be enforced locally for each function in the CDN, unlike the case of Markov random fields. We will show that the conditional independence properties in a CDN include the conditional independence properties for bidirected graphs [13, 58, 59], so that CDNs are a parameterization for such models. We will provide a method for constructing a factor graph corresponding to a CDN in which we introduce additional latent variables into the factor graph. Under such a construction, the joint probability modeled by the CDN satisfies the same conditional independence relationships amongst CDN variables as those of its corresponding factor graph with the latent variables implicitly marginalized out.

In Chapter 4, we will discuss the problem of performing inference under CDNs. The principal challenge is to compute the derivatives of the joint CDF. We will describe a message-passing algorithm for inference in CDNs called the *derivative-sum-product algorithm*.

In Chapter 5, we will take advantage of the graphical representation for the CDF in order to apply CDNs to the problem of learning to rank. We will devise a model and estimation method for multiplayer games where the CDN provides us with a means for modelling the team-based structure of such games in addition to ordering constraints between model variables. We will then develop a general framework for structured ranking learning in which we are given many observations of partial orderings over many objects to be ranked and we wish to learn a model to rank these objects whilst accounting for a statistical dependence relationships between preferences. We will then present results on applications of structured ranking learning to the problems of document retrieval and regulatory sequence discovery in computational biology [25, 26, 27, 28, 29]. Chapter 6 will present future research directions for estimation, inference and applications of CDNs.

Chapter 2

Background

2.1 Cumulative distribution functions

In this section, we will review the properties of CDFs for real-valued random variables. Here we will omit some proofs for conciseness and refer the reader to [55] for additional properties of CDFs and additional proofs.

2.1.1 Joint cumulative distribution functions

Definition 2.1.1. Let $\mathbf{X}$ be a set of random variables, denoted individually as X_α. The joint *cumulative distribution function* $F(\mathbf{x})$ is defined as the function $F : \mathbb{R}^{|\mathbf{X}|} \mapsto [0, 1]$ such that

$$F(\mathbf{x}) = \mathbb{P}\left[\cap_{X_\alpha \in \mathbf{X}} \left\{X_\alpha \leq x_\alpha\right\}\right] \equiv \mathbb{P}\Big[\mathbf{X} \leq \mathbf{x}\Big] \tag{2.1}$$

for some probability $\mathbb{P}$, where $\mathbf{X} \leq \mathbf{x}$ denotes the elementwise inequality between the vector of random variables $\mathbf{X}$ and vector $\mathbf{x}$. Thus the CDF is a probability measure defined over the intersection of events $\{X_\alpha \leq x_\alpha\}$. Alternately, the CDF can be defined in terms of the joint probability density function (PDF) or probability mass function

(PMF) $P(\mathbf{x})$ via

$$F(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} P(\mathbf{u})\, d\mathbf{u}, \tag{2.2}$$

where $\int_{-\infty}^{\mathbf{x}} \cdot\, d\mathbf{u}$ denotes the multivariate integral with respect to vector $\mathbf{u}$ and $P(\mathbf{x})$, if it exists, satisfies $P(\mathbf{x}) \geq 0\ \forall\ \mathbf{x} \in \mathbb{R}^{|\mathbf{X}|}$, $\int_{\mathbf{x}} P(\mathbf{x})\, d\mathbf{x} = 1$ and $P(\mathbf{x}) = \partial_{\mathbf{x}}\Big[F(\mathbf{x})\Big]$ where $\partial_{\mathbf{x}}\Big[\cdot\Big]$ denotes the higher-order mixed derivative operator $\partial_{x_1,\cdots,x_K}\Big[\cdot\Big] \equiv \dfrac{\partial^K}{\partial x_1 \cdots \partial x_K}$ for $\mathbf{x} = [x_1\ \cdots\ x_K] \in \mathbb{R}^K$. □

Our analysis here encompasses both discrete and continuous random variables. In the discrete case where $\mathbf{x} \in \mathcal{X}$ for some discrete set $\mathcal{X} \subset \mathbb{R}^{|\mathbf{X}|}$, we can define $P(\mathbf{x}) = \mathbb{P}[\mathbf{X} = \mathbf{x}]$ as a sum of shifted multivariate Dirac delta functions $\delta(\mathbf{x})$ with the property that

$$\int_{\mathbf{x}} \delta(\mathbf{x} - \mathbf{x}^*) g(\mathbf{x})\, d\mathbf{x} = g(\mathbf{x}^*) \tag{2.3}$$

for any function $g : \mathbb{R}^{|\mathbf{X}|} \mapsto \mathbb{R}$ and $\mathbf{x}^* \in \mathbb{R}^{|\mathbf{X}|}$. Thus in the discrete case, we can write $P(\mathbf{x})$ as

$$P(\mathbf{x}) = \mathbb{P}[\mathbf{X} = \mathbf{x}] = \sum_{\mathbf{u} \in \mathcal{X}} p_{\mathbf{u}} \delta(\mathbf{x} - \mathbf{u}). \tag{2.4}$$

Theorem 2.1.1. A function $F : \mathbb{R}^{|\mathbf{X}|} \mapsto [0, 1]$ is a CDF for some probability $\mathbb{P}$ if and only if F satisfies the following conditions:

1. The CDF $F(\mathbf{x})$ converges to unity as all of its arguments tend to ∞, or

$$F(\infty) \equiv \lim_{\mathbf{x} \to \infty} F(\mathbf{x}) = 1. \tag{2.5}$$

2. The CDF $F(\mathbf{x})$ converges to 0 as any of its arguments tends to $-\infty$, or

$$F(-\infty, \mathbf{x} \setminus x_\alpha) \equiv \lim_{x_\alpha \to -\infty} F(x_\alpha, \mathbf{x} \setminus x_\alpha) = 0 \quad \forall X_\alpha \in \mathbf{X}. \tag{2.6}$$

3. The CDF $F(\mathbf{x})$ is monotonically non-decreasing, so that

$$F(\mathbf{x}) \leq F(\mathbf{y})\ \forall\ \mathbf{x} \leq \mathbf{y},\ \mathbf{x}, \mathbf{y} \in \mathbb{R}^{|\mathbf{X}|}. \tag{2.7}$$

where $\mathbf{x} \leq \mathbf{y}$ denotes elementwise inequality of all the elements in vectors $\mathbf{x}, \mathbf{y}$.

4. The CDF $F(\mathbf{x})$ is right-continuous, so that

$$\lim_{\boldsymbol{\epsilon} \to \mathbf{0}^+} F(\mathbf{x} + \boldsymbol{\epsilon}) \equiv F(\mathbf{x}) \; \forall \; \mathbf{x} \in \mathbb{R}^{|\mathbf{X}|}. \tag{2.8}$$

□

The above conditions can be shown to follow from the Kolmogorov axioms of probability: for a detailed derivation, we refer the reader to [55].

Proposition 2.1.2. Let $F(\mathbf{x}_A, x_\beta)$ be the joint CDF for variables $\{\mathbf{X}_A, X_\beta\}$. The joint probability of the event $\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{a < X_\beta \leq b\}$ for any random variable $X_\beta \notin \mathbf{X}_A$ and any subset of variables $\mathbf{X}_A \subset \mathbf{X}$ is given in terms of $F(\mathbf{x}_A, x_\beta)$ as

$$\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{a < X_\beta \leq b\}\Big] = F(\mathbf{x}_A, b) - F(\mathbf{x}_A, a). \tag{2.9}$$

□

Proposition 2.1.3. Let $F(\mathbf{x}_A, \mathbf{x}_B)$ be the joint CDF for variables $\mathbf{X}$ where $\mathbf{X}_A, \mathbf{X}_B$ for a partition of the set of variables $\mathbf{X}$. The joint probability of the event $\{\mathbf{X}_A \leq \mathbf{x}_A\}$ is then given in terms of $F(\mathbf{x}_A, \mathbf{x}_B)$ as

$$F(\mathbf{x}_A) \equiv \mathbb{P}\Big[\mathbf{X}_A \leq \mathbf{x}_A\Big] = \lim_{\mathbf{x}_B \to \infty} F(\mathbf{x}_A, \mathbf{x}_B). \tag{2.10}$$

□

The above proposition follows directly from the definition of a CDF in which

$$\lim_{\mathbf{x}_B \to \infty} \cap_{\alpha \in A \cup B} \{X_\alpha \leq x_\alpha\} = \cap_{\alpha \in A} \{X_\alpha \leq x_\alpha\}. \tag{2.11}$$

Thus, marginal CDFs of the form $F(\mathbf{x}_A)$ can be computed from the joint CDF by computing limits. Furthermore, the CDF is closed under marginalization, so that $F(\mathbf{x}_A, \infty)$ satisfies the properties of a CDF.

2.1.2 Conditional cumulative distribution functions

In this thesis we will be making use of the concept of a conditional CDF for some subset of variables $\mathbf{X}_A$ conditioned on event M. We formally define the conditional CDF below.

Definition 2.1.2. Let M be an event with $\mathbb{P}[M] > 0$. The conditional CDF $F(\mathbf{x}_A \mid M)$ conditioned on event M is defined as

$$F(\mathbf{x}_A \mid M) \equiv \mathbb{P}\Big[\mathbf{X}_A \leq \mathbf{x}_A \mid M\Big] = \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap M\Big]}{\mathbb{P}\Big[M\Big]}. \tag{2.12}$$

□

We will now find the above conditional CDF for different types of events M. In what follows, we will use the notation $\partial_{\mathbf{x}_A}\Big[\cdot\Big]$ to denote a mixed derivative with respect to variables $\mathbf{x}_A$.

Lemma 2.1.4. Let $F(\mathbf{x}_C)$ be a marginal CDF obtained from the joint CDF $F(\mathbf{x})$ as given by Proposition 2.1.3 for some $\mathbf{X}_C \subseteq \mathbf{X}$. Consider some variable set $\mathbf{X}_A \subseteq \mathbf{X}$. Let $M = \omega(\mathbf{x}_C) \equiv \{\mathbf{X}_C \leq \mathbf{x}_C\}$ for $\mathbf{X}_C \subset \mathbf{X}$. If $F(\mathbf{x}_C) > 0$, then $F(\mathbf{x}_A|\omega(\mathbf{x}_C)) \equiv F(\mathbf{x}_A|\mathbf{X}_C \leq \mathbf{x}_C) = \dfrac{F(\mathbf{x}_A, \mathbf{x}_C)}{F(\mathbf{x}_C)}$. □

Lemma 2.1.5. Consider some variable set $\mathbf{X}_A \subseteq \mathbf{X}$. Let $M = \{x_\beta < X_\beta \leq x_\beta + \epsilon\}$ with $\epsilon > 0$ for some scalar random variable $X_\beta \notin \mathbf{X}_A$. If $F(x_\beta)$ and $F(\mathbf{x}_A, x_\beta)$ are differentiable with respect to x_β so that $\partial_{x_\beta}\Big[F(x_\beta)\Big]$ and $\partial_{x_\beta}\Big[F(\mathbf{x}_A, x_\beta)\Big]$ exist with $\partial_{x_\beta}\Big[F(x_\beta)\Big] > 0$, then the conditional CDF $F(\mathbf{x}_A|x_\beta) \equiv \lim_{\epsilon \to 0^+} F(\mathbf{x}_A|x_\beta < X_\beta < x_\beta + \epsilon) = \lim_{\epsilon \to 0^+} \dfrac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{x_\beta < X_\beta \leq x_\beta + \epsilon\}\Big]}{\mathbb{P}\Big[x_\beta < X_\beta \leq x_\beta + \epsilon\Big]}$ is given by

$$F(\mathbf{x}_A|x_\beta) = \frac{\partial_{x_\beta}\Big[F(\mathbf{x}_A, x_\beta)\Big]}{\partial_{x_\beta}\Big[F(x_\beta)\Big]} \propto \partial_{x_\beta}\Big[F(\mathbf{x}_A, x_\beta)\Big]. \tag{2.13}$$

Proof. We can write

$$F(\mathbf{x}_A|x_\beta < X_\beta \leq x_\beta + \epsilon) = \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{x_\beta < X_\beta \leq x_\beta + \epsilon\}\Big]}{\mathbb{P}\Big[x_\beta < X_\beta \leq x_\beta + \epsilon\Big]}$$
$$= \frac{\frac{1}{\epsilon}\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{x_\beta < X_\beta \leq x_\beta + \epsilon\}\Big]}{\frac{1}{\epsilon}\mathbb{P}\Big[x_\beta < X_\beta \leq x_\beta + \epsilon\Big]} = \frac{\frac{F(\mathbf{x}_A, x_\beta+\epsilon) - F(\mathbf{x}_A, x_\beta)}{\epsilon}}{\frac{F(x_\beta+\epsilon)-F(x_\beta)}{\epsilon}}. \tag{2.14}$$

Taking limits, and given differentiability of both $F(x_\beta)$ and $F(\mathbf{x}_A, x_\beta)$ with respect to x_β, the conditional CDF $F(\mathbf{x}_A|x_\beta)$ is given by

$$F(\mathbf{x}_A|x_\beta) \equiv \frac{\lim_{\epsilon\to 0^+} \frac{F(\mathbf{x}_A, x_\beta+\epsilon) - F(\mathbf{x}_A, x_\beta)}{\epsilon}}{\lim_{\epsilon\to 0^+} \frac{F(x_\beta+\epsilon)-F(x_\beta)}{\epsilon}} = \frac{\partial_{x_\beta}\Big[F(\mathbf{x}_A, x_\beta)\Big]}{\partial_{x_\beta}\Big[F(x_\beta)\Big]} \propto \partial_{x_\beta}\Big[F(\mathbf{x}_A, x_\beta)\Big], \tag{2.15}$$

where the proportionality constant does not depend on $\mathbf{x}_A$. □

Lemma 2.1.6. Let $M = \{\mathbf{x}_C < \mathbf{X}_C \leq \mathbf{x}_C + \boldsymbol{\epsilon}\} \equiv \cap_{\gamma\in C}\{x_\gamma < X_\gamma \leq x_\gamma + \epsilon\}$ with $\epsilon > 0$ for $\mathbf{X}_C \subset \mathbf{X}$ and $\boldsymbol{\epsilon} = [\epsilon \ \cdots \ \epsilon]^T \in \mathbb{R}^{|\mathbf{X}_C|}$. Consider the set of random variables $\mathbf{X}_A \subset \mathbf{X}$ with $\mathbf{X}_C \cap \mathbf{X}_A = \emptyset$. If both $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]$ and $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$ exist for all $\mathbf{x}_C$ with $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big] > 0$, then the conditional CDF $F(\mathbf{x}_A|\mathbf{x}_C) \equiv \lim_{\boldsymbol{\epsilon}\to\mathbf{0}^+} F(\mathbf{x}_A|\mathbf{x}_C < \mathbf{X}_C \leq \mathbf{x}_C + \boldsymbol{\epsilon}) = \lim_{\boldsymbol{\epsilon}\to\mathbf{0}^+} \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{\mathbf{x}_C < \mathbf{X}_C \leq \mathbf{x}_C + \boldsymbol{\epsilon}\}\Big]}{\mathbb{P}\Big[\mathbf{x}_C < \mathbf{X}_C \leq \mathbf{x}_C + \boldsymbol{\epsilon}\Big]}$ is given by

$$F(\mathbf{x}_A|\mathbf{x}_C) = \frac{\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]}{\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]} \propto \partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big], \tag{2.16}$$

where $\partial_{\mathbf{x}_C}\Big[\cdot\Big]$ is a mixed derivative operator with respect to $\{x_\gamma, \gamma \in C\}$.

Proof. We can proceed by induction on variable set $\mathbf{X}_C$ with the base case given by Lemma 2.1.4. Let $\mathbf{X}_C = \mathbf{X}_{C'} \cup X_\beta$ with $X_\beta \notin \mathbf{X}_{C'} \cup \mathbf{X}_A$. Let $M' \equiv M'(\boldsymbol{\xi}) = \{\mathbf{x}_{C'} \leq \mathbf{X}_{C'} \leq \mathbf{x}_{C'} + \boldsymbol{\xi}\} = \cap_{\gamma\in C'}\{x_\gamma < X_\gamma \leq x_\gamma + \xi\}$ and $M \equiv M(\boldsymbol{\xi}, \epsilon) = M' \cap \{x_\beta < X_\beta \leq x_\beta + \epsilon\}$

with $\boldsymbol{\epsilon} = [\boldsymbol{\xi}^T \ \epsilon]^{\mathrm{T}}$. Suppose that $\partial_{\mathbf{x}_{C'}}\Big[F(\mathbf{x}_{C'})\Big] > 0$ and we have computed

$$F(\mathbf{x}_A, x_\beta|\mathbf{x}_{C'}) \equiv \lim_{\boldsymbol{\xi}\to 0^+} F\Big(\mathbf{x}_A, x_\beta|M'(\boldsymbol{\xi})\Big) = \frac{\partial_{\mathbf{x}_{C'}}\Big[F(\mathbf{x}_A, x_\beta, \mathbf{x}_{C'})\Big]}{\partial_{\mathbf{x}_{C'}}\Big[F(\mathbf{x}_{C'})\Big]} \tag{2.17}$$

and

$$F(x_\beta|\mathbf{x}_{C'}) \equiv \lim_{\boldsymbol{\xi}\to 0^+} F\Big(x_\beta|M'(\boldsymbol{\xi})\Big) = \frac{\partial_{\mathbf{x}_{C'}}\Big[F(x_\beta, \mathbf{x}_{C'})\Big]}{\partial_{\mathbf{x}_{C'}}\Big[F(\mathbf{x}_{C'})\Big]}. \tag{2.18}$$

Then we can write

$$F(\mathbf{x}_A \mid M) = \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{x_\beta < X_\beta \leq x_\beta + \epsilon\} \mid M'\Big]}{\mathbb{P}\Big[x_\beta < X_\beta \leq x_\beta + \epsilon \mid M'\Big]} = \frac{\frac{F(\mathbf{x}_A, x_\beta+\epsilon|M') - F(\mathbf{x}_A, x_\beta|M')}{\epsilon}}{\frac{F(x_\beta+\epsilon|M') - F(x_\beta|M')}{\epsilon}}. \tag{2.19}$$

Thus, since $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big] > 0$ by hypothesis, we obtain

$$\begin{aligned} F(\mathbf{x}_A|\mathbf{x}_C) &= \lim_{\epsilon\to 0^+, \boldsymbol{\xi}\to 0^+} \frac{\frac{F(\mathbf{x}_A, x_\beta+\epsilon|M') - F(\mathbf{x}_A, x_\beta|M')}{\epsilon}}{\frac{F(x_\beta+\epsilon|M') - F(x_\beta|M')}{\epsilon}} = \frac{\lim_{\epsilon\to 0^+} \frac{F(\mathbf{x}_A, x_\beta+\epsilon|\mathbf{x}_{C'}) - F(\mathbf{x}_A, x_\beta|\mathbf{x}_{C'})}{\epsilon}}{\lim_{\epsilon\to 0^+} \frac{F(x_\beta+\epsilon|\mathbf{x}_{C'}) - F(x_\beta|\mathbf{x}_{C'})}{\epsilon}} \\ &= \frac{\partial_{x_\beta, \mathbf{x}_{C'}}\Big[F(\mathbf{x}_A, x_\beta, \mathbf{x}_{C'})\Big]}{\partial_{x_\beta, \mathbf{x}_{C'}}\Big[F(x_\beta, \mathbf{x}_{C'})\Big]} = \frac{\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]}{\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]}. \end{aligned} \tag{2.20}$$

□

Proposition 2.1.7. Let $F(\mathbf{x}_C)$ and $F(\mathbf{x}_A, \mathbf{x}_C)$ be marginal CDFs obtained from the joint CDF $F(\mathbf{x})$ as given by Proposition 2.1.3. If $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$ and $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]$ exist and are continuous, then computing $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$, $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]$ and $F(\mathbf{x}_A|\mathbf{x}_C) \propto \partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$ is invariant to the order of differentiation. □

The above proposition is a generalization of Schwarz's theorem to mixed derivatives of a multivariate function with respect to $n > 2$ variables.

Lemma 2.1.8. Let $F(\mathbf{x}_C)$ and $F(\mathbf{x}_A, \mathbf{x}_C)$ be marginal CDFs obtained from the joint CDF $F(\mathbf{x})$ as given by Proposition 2.1.3. If $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]$ and $\partial_{\mathbf{x}_A,\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$ exist for all $\mathbf{x}_C$ and $\mathbf{x}_A$ then $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big] \geq 0 \; \forall \; \mathbf{x}_C$ and $\mathbf{x}_A$.

Proof. If both $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_C)\Big]$ and $\partial_{\mathbf{x}_A,\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big]$ exist then they must be the PDFs $P(\mathbf{x}_C)$ and $P(\mathbf{x}_A, \mathbf{x}_C)$ respectively. We can then write

$$F(\mathbf{x}_A, \mathbf{x}_C) = \int_{-\infty}^{\mathbf{x}_C} \int_{-\infty}^{\mathbf{x}_A} P(\mathbf{u}_A, \mathbf{u}_C) \; d\mathbf{u}_A \cdot d\mathbf{u}_C \tag{2.21}$$

so that

$$\begin{aligned}
\partial_{\mathbf{x}_C}\Big[F(\mathbf{x}_A, \mathbf{x}_C)\Big] &= \partial_{\mathbf{x}_C}\left[\int_{-\infty}^{\mathbf{x}_C} \int_{-\infty}^{\mathbf{x}_A} P(\mathbf{u}_A, \mathbf{u}_C) \; d\mathbf{u}_A \cdot d\mathbf{u}_C\right] \\
&= \partial_{\mathbf{x}_C}\left[\int_{-\infty}^{\mathbf{x}_C} \int_{-\infty}^{\mathbf{x}_A} P(\mathbf{u}_A|\mathbf{u}_C)P(\mathbf{u}_C) \; d\mathbf{u}_A \cdot d\mathbf{u}_C\right] \\
&= P(\mathbf{x}_C) \int_{-\infty}^{\mathbf{x}_A} P(\mathbf{u}_A|\mathbf{x}_C) \; d\mathbf{u}_A \\
&= P(\mathbf{x}_C)F(\mathbf{x}_A|\mathbf{x}_C),
\end{aligned} \tag{2.22}$$

where we have made use of the Fundamental Theorem of Calculus. The fact that $P(\mathbf{x}_C)$ and $F(\mathbf{x}_A|\mathbf{x}_C)$ are both non-negative completes the proof. □

Thus the above lemma demonstrates that for a joint CDF, mixed derivatives with respect to any and all subsets of variables are always non-negative.

2.2 Stochastic orderings of random variables

In many applications, it is useful to define the stochastic ordering of random variables. Informally, a stochastic ordering relationship $X \preceq Y$ holds between two random variables X, Y if samples of Y tend to be larger than samples of X. We can formalize this notion in terms of constraints on the marginal CDFs $F_X(x)$ and $F_Y(y)$. We will define below the concept of *first-order* stochastic orderings among random variables, as this is the primary

definition for a stochastic ordering that we will make use of in this thesis. We refer the reader to [43, 62] for additional definitions of stochastic orderings.

Definition 2.2.1. Consider two scalar random variables X and Y with marginal CDFs $F_X(x)$ and $F_Y(y)$. Then X and Y are said to satisfy the first-order stochastic ordering constraint $X \preceq Y$ if $F_X(t) \geq F_Y(t)$ for all $t \in \mathbb{R}$. □

The above definition of stochastic ordering is stronger than the constraint $\mathbb{E}[X] \leq \mathbb{E}[Y]$ which is often used and one can show that $X \preceq Y$ implies the former constraint. Note that the converse is not true: $\mathbb{E}[X] \leq \mathbb{E}[Y]$ does not necessarily imply $X \preceq Y$. For example, consider two Gaussian random variables X and Y for which $\mathbb{E}[X] \leq \mathbb{E}[Y]$ but $Var[X] \gg Var[Y]$. An illustration of a pairwise first-order constraint $X \preceq Y$ is shown in Figure 2.1, where $F_X(t) \geq F_Y(t)$ for all $t \in \mathbb{R}$. The definition of a stochastic ordering can also be extended to disjoint sets of variables $\mathbf{X}_A$ and $\mathbf{X}_B$.

Definition 2.2.2. Let $\mathbf{X}_A$ and $\mathbf{X}_B$ be disjoint sets of variables so that $\mathbf{X}_A = \{X_{\alpha_1}, \cdots, X_{\alpha_K}\}$ and $\mathbf{X}_B = \{X_{\beta_1}, \cdots, X_{\beta_K}\}$ for some strictly positive integer K. Let $F_{\mathbf{X}_A}(\mathbf{t})$ and $F_{\mathbf{X}_B}(\mathbf{t})$ be the CDFs of $\mathbf{X}_A$ and $\mathbf{X}_B$. Then $\mathbf{X}_A, \mathbf{X}_B$ are said to satisfy the stochastic ordering relationship $\mathbf{X}_A \preceq \mathbf{X}_B$ if

$$F_{\mathbf{X}_A}(\mathbf{t}) \geq F_{\mathbf{X}_B}(\mathbf{t}) \tag{2.23}$$

for all $\mathbf{t} \in \mathbb{R}^K$. □

Note that in general, it does not follow that if $\mathbf{X}_A \preceq \mathbf{X}_B$, then A is not independent of B, or $A \not\perp\!\!\!\perp B$. Furthermore, the reverse statement is generally not true. A simple counter-example in the latter case is two independent Gaussian random variables X and Y with identical variances and $\mathbb{E}[X] \leq \mathbb{E}[Y]$.

In addition to the ability to specify stochastic orderings of variables via CDFs, we can also construct multivariate CDFs from a set of univariate CDFs. We describe the corresponding concept of a *copula* function that allows us to accomplish this in the next section.

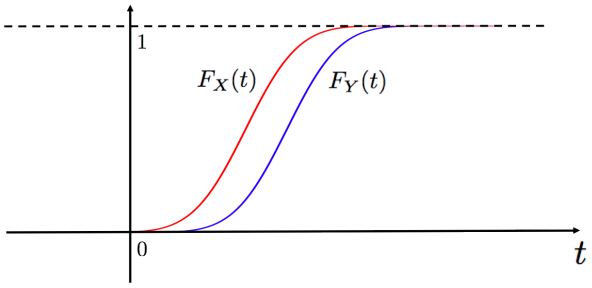


Figure 2.1: Two random variables X, Y satisfy the stochastic ordering $X \preceq Y$ if and only if $F_X(t) \geq F_Y(t)$ for all $t \in \mathbb{R}$.

2.3 Copulas

Copula functions are often used to model statistical dependence relationships between random variables as a function of the marginal distributions of these variables. Copulas allow one to specify a joint cumulative distribution function over variables $X_1, \cdots, X_K$ in terms of the marginal CDFs $F_1(x_1), \cdots, F_K(x_K)$ whilst allowing for statistical dependence relationships between the variables. We will formally define the copula below and present some of the corresponding properties.

Definition 2.3.1. A copula $\zeta : [0,1]^K \mapsto [0,1]$ is a joint cumulative distribution function whose marginal PDFs are uniform. The copula function $\zeta(\mathbf{u}) = \zeta(u_1, \cdots, u_K)$ must satisfy

$$\begin{aligned}
&\lim_{u_k \to 0} \zeta(\mathbf{u}) = 0 \; \forall \; k = 1, \cdots, K \\
&\lim_{\mathbf{u} \setminus u_k \to \mathbf{1}} \zeta(\mathbf{u}) = u_k \; \forall \; k = 1, \cdots, K \\
&\lim_{\mathbf{u} \to \mathbf{1}} \zeta(\mathbf{u}) = 1.
\end{aligned} \tag{2.24}$$

□

The existence of such copula functions for modelling joint CDFs $F(\mathbf{x})$ is established in Sklar's theorem [54], which we present below.

Theorem 2.3.1 (Sklar's theorem)**.** Let $F(\mathbf{x}) \equiv F(x_1, \cdots, x_K)$ be a joint CDF over the

variables $\mathbf{X} \equiv \{X_1, \cdots, X_K\}$ with marginal CDFs $F_1(x_1), \cdots, F_K(x_K)$. Then there exists a copula function ζ such that

$$F(\mathbf{x}) = \zeta\Big(F_1(x_1), \cdots, F_K(x_K)\Big) \; \forall \; x_k \in \mathbb{R}, k = 1, \cdots, K. \tag{2.25}$$

Conversely, if ζ is a copula function and $F_1(x_1), \cdots, F_K(x_K)$ are marginal CDFs over the variables $X_1, \cdots, X_K$, then $F(\mathbf{x})$ is the joint CDF over variables $\mathbf{X}$ with marginals CDFs given by $F_1(x_1), \cdots, F_K(x_K)$. Furthermore, if the marginal CDFs $F_1(x_1), \cdots, F_K(x_K)$ are continuous, then ζ is unique. □

Thus, Sklar's theorem allows one to model any joint CDF in terms of its marginals through the use of a copula function, so that copulas allow one to model a variety of multivariate CDFs. Note that the definition of a copula does not by itself imply any particular set of conditional independence relationships among variables. In Chapter 3, we will make use of the ability of copulas to model CDFs in tandem with the graphical modelling framework that is the focus of this thesis to construct multivariate cumulative distributions defined over graphs in which we can introduce additional conditional independence relationships amongst variables that are not implied by the definition of the copula. To this end, we will now establish some terminology and concepts for graphical models that will prove useful in later chapters.

2.4 Graph terminology

In this section, we will define some terms that will be frequently used throughout the thesis in the context of graphs.

Definition 2.4.1. A *graph* is a pair $G = (V, E)$ where V is a finite set of nodes and $E \subseteq V \times V$ is a set of edges consisting of ordered pairs (α, β) of distinct nodes $\alpha, \beta \in V$. □

The above definition of a graph does not allow for loops of the form (α, α) nor does it allow for multiple edges between two nodes α, β. That is, the graphs to be discussed throughout the thesis are assumed to be *simple graphs*.

Having defined a graph, we can distinguish between three types of graphs $G = (V, E)$. If for any distinct $\alpha, \beta \in V$, $(\alpha, \beta) \in E \Leftrightarrow (\beta, \alpha) \in E$, then G is said to be an *undirected graph*, whereas if $(\alpha, \beta) \in E \Rightarrow (\beta, \alpha) \notin E$, then G is said to be a *directed graph*. We will occasionally denote undirected edges (α, β) as $\alpha - \beta$ and directed edges as $\alpha \rightarrow \beta, \alpha \leftarrow \beta$. Furthermore, we will also refer to *bidirected graphs* as graphs that are defined identically to undirected graphs for the purposes of this thesis, but in which edges in the graph are exclusively bi-directed and denoted as $\alpha \leftrightarrow \beta$. Note that a bidirected edge $\alpha \leftrightarrow \beta$ in a bidirected graph does not correspond to both $\alpha \rightarrow \beta$ and $\alpha \leftarrow \beta$. As we will demonstrate shortly, although bidirected graphs are defined identically to undirected graphs, they lead to different semantics in the context of probability models defined on graphs, so that we distinguish between undirected edges and bidirected edges.

With the above notation in mind, we will now proceed to define various terms for denoting subsets of nodes and/or edges in a graph.

Definition 2.4.2. Let $G = (V, E)$ be a graph. Then the set of *neighbors* $\mathcal{N}(\alpha)$ of a node $\alpha \in V$ is given by

$$\mathcal{N}(\alpha) = \{\beta \in V, \beta \neq \alpha : (\alpha, \beta) \in E \text{ or } (\beta, \alpha) \in E\}. \tag{2.26}$$

□

Definition 2.4.3. Let $G = (V, E)$ be a directed graph. The *parents* of node $\alpha \in V$ is defined with respect to G as

$$\mathsf{pa}(\alpha) = \{\beta \in V, \beta \neq \alpha : (\beta, \alpha) \in E\}. \tag{2.27}$$

□

Definition 2.4.4. Let $G = (V, E)$ be a directed graph. The *children* of node $\alpha \in V$ is defined with respect to G as

$$\mathsf{ch}(\alpha) = \{\beta \in V, \beta \neq \alpha : (\alpha, \beta) \in E\}. \tag{2.28}$$

□

Definition 2.4.5. Let $G = (V, E)$ be a graph. An *undirected path* of length n between two nodes $\alpha, \beta \in V$ is defined as a sequence of distinct vertices $\gamma_0, \cdots, \gamma_n$ with $\gamma_0 = \alpha$ and $\gamma_n = \beta$ such that the for $i = 1, \cdots, n$, either $(\gamma_{i-1}, \gamma_i) \in E$ or $(\gamma_i, \gamma_{i-1}) \in E$. □

Definition 2.4.6. A *directed path* of length n from node α to node β is defined as a sequence of distinct vertices $\gamma_0, \cdots, \gamma_n$ with $\gamma_0 = \alpha$ and $\gamma_n = \beta$ such that for $i = 1, \cdots, n$, $(\gamma_{i-1}, \gamma_i) \in E$ and $(\gamma_i, \gamma_{i-1}) \notin E$. □

The above definition for a path specifies that the sequence $\gamma_0, \cdots, \gamma_n$ does not contain repeated nodes, so that the paths defined above are in fact called simple paths. In the sequel we will only make use of the definition for simple paths, unless otherwise specified. In the case in which the nodes α and β are the same, the path is then called a *cycle*, which we now define.

Definition 2.4.7. Let $G = (V, E)$ be a graph. For $\alpha \in V$, an *undirected cycle* of length n is defined as an undirected path $\gamma_0, \cdots, \gamma_n$ with $\gamma_0 = \alpha$ and $\gamma_n = \alpha$. □

Definition 2.4.8. A *directed cycle* of length n is defined as a directed path $\gamma_0, \cdots, \gamma_n$ with $\gamma_0 = \alpha$ and $\gamma_n = \alpha$. □

In the context of directed graphs, a directed graph that does not contain any directed cycles is said to be a *directed acyclic graph* (DAG).

Definition 2.4.9. Let $G = (V, E)$ be a directed acyclic graph. The *descendants* of node $\alpha \in V$ is defined with respect to G as the set of all nodes $\beta \in V \setminus \alpha$ for which there is a directed path in G leading from α to β. □

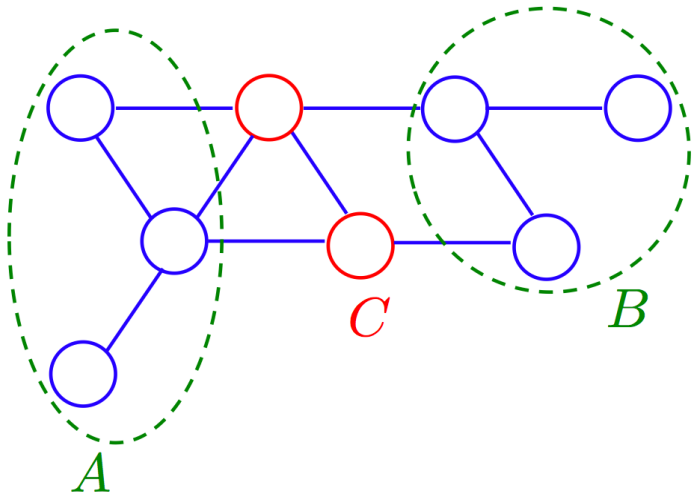


Figure 2.2: Example of a set C (consisting of nodes in red) separating node sets A and B in an undirected graph.

Definition 2.4.10. Let $G = (V, E)$ be an undirected graph. A *clique* is a subset $C \subseteq V$ such that $\alpha \in C, \beta \in C \Rightarrow (\alpha, \beta) \in E$. □

In other words, the set of nodes in a clique is fully connected. Furthermore, a *maximal clique* is a clique such that it is not possible to include any other nodes from the graph in the set without it ceasing to be a clique.

Definition 2.4.11. Given a graph $G = (V, E)$ and a nonempty subset of nodes $C \subseteq V$, C is said to be a *connected set* if $\alpha \in C, \beta \in C$ implies that there is an undirected path between α and β, where we allow for $C = \alpha$. □

Definition 2.4.12. Given a graph $G = (V, E)$ and disjoint subsets of nodes $A, B, C \subseteq V$, C is said to *separate* A from B with respect to G if all undirected paths from any node $\alpha \in A$ to any node $\beta \in B$ contain a node $\gamma \in C$. □

The above notion of graph separation is illustrated in Figure 2.2 for disjoint node sets A, B, C in an undirected graph.

Bipartite graphs

The definitions presented above can also be modified for *bipartite graphs* in which the set of nodes consists of two disjoint sets of nodes such that every edge in the graph connects a node in one set to a node in the other set. We will formally define a bipartite graph below.

Definition 2.4.13. A bipartite graph $\mathcal{G} = ((V, S), E) \equiv (V, S, E)$ is a triplet of sets V, S, E where V, S are two disjoint sets of nodes and $E \subseteq V \times S \cup S \times V$ is a set of edges that correspond to ordered pairs (α, s) or (s, α) for $\alpha \in V$ and $s \in S$. □

In the sequel, a bipartite graph will be assumed to be undirected so that for any $\alpha \in V, s \in S$, $(\alpha, s) \in E \Leftrightarrow (s, \alpha) \in E$ and so any edge in a bipartite graph can be written $s - \alpha$, although this is not a strict requirement of the above definition (see [16]). Since bipartite graphs contains two sets of nodes, we will introduce some additional notation. Let $\mathcal{N}(\alpha)$ and $\mathcal{N}(s)$ denote the sets

$$\mathcal{N}(\alpha) = \{s \in S : (\alpha, s) \in E\}$$
$$\mathcal{N}(s) = \{\alpha \in V : (\alpha, s) \in E\}.$$

Furthermore, let $\mathcal{N}(A) = \cup_{\alpha \in A} \mathcal{N}(\alpha)$. An example of the neighboring set $\mathcal{N}(A)$ for a node set A is given in Figure 2.3.

In a bipartite graph, the definition of separation is identical to that presented above, so that given a bipartite graph $\mathcal{G} = (V, S, E)$ and disjoint subsets of nodes $A, B, C \subseteq V$, C separates A from B with respect to $\mathcal{G}$ if and only if all paths from any node $\alpha \in A$ to any node $\beta \in B$ include a node $\gamma \in C$.

2.5 Probabilistic graphical models

When modelling probabilities over many random variables for real-world applications, it is often advantageous to augment the analysis using graphs. These provide a simple way

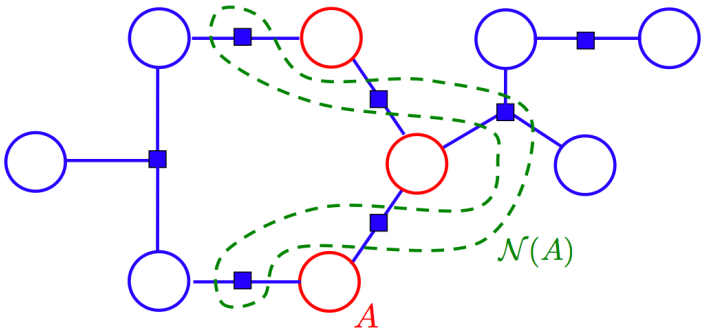


Figure 2.3: Example of the set of neighboring nodes $s \in S$ $\mathcal{N}(A)$ in a bipartite graph for $A \subseteq V$. The set A consists of nodes in red and neighboring nodes $s \in S$ enclosed by the dotted curve are in the set $\mathcal{N}(A)$.

to visualize the set of conditional independence relationships amongst many variables and can be used to design models defined over large numbers of variables. A graph/graphical model, is said to represent/provide a representation for a joint probability over a set of random variables if and only if the variables over which the probability is defined satisfy conditional independence relationships that are implied by separation of nodes in the graph, where nodes in the graph correspond to variables in the model. Equivalently, a probability is said to be defined over a graph if and only if the variables over which the probability is defined satisfy conditional independence relationships that are implied by separation of nodes in the graph. The graph separation of nodes also implies that the joint probability factors into a product of functions defined over variables for which the corresponding nodes are neighbors in the graph.

As a result of allowing one to model conditional independence relationships amongst variables, graphs also allow one to simplify the computations required for inference in the model, which requires computing conditional probabilities of the form $P(x|y)$. Examples of such problems of inference would be "Given a sequence of binary 0/1 symbols, what is the probability that the next symbol will be a 1?", or "Given a person's salary, what is the probability of his/her voting Republican?". Such graphical representations also

allow for insights into independence relationships amongst model variables that can be obtained by inspection of the graph. A *probabilistic graphical model* accounts for the set of independence relationships between variables in a corresponding probability distribution. Each node $\alpha \in V$ in the graph corresponds to a random variable X_α. The graph allows us to visualize independence constraints between subsets of variables in the model, so that for any particular graph separation criterion, variables that are separated in the graph with respect to the given criterion also satisfy a particular marginal or conditional independence constraint. A consequence of this is that the joint probability over all of the variables can typically be decomposed into a product of functions, each depending only on a subset of the variables in the model. In such cases, a graph is said to model (or represent) a probability if the probability factors into a product of functions such that the variables in the model satisfy independence constraints dictated by separation of nodes in the graph.

In the following, we review three classes of graphical model that are commonly used in practice:

1. *Directed graphical models*, or *Bayesian networks*, which consist of a directed acyclic graph and a set of conditional probabilities defined for each variable in the model given its parents in the directed graph.

2. *Undirected graphical models*, or *Markov random fields*, which consist of an undirected graph and a set of potential functions defined over cliques of variables in the model.

3. *Factor graph models*, or simply *factor graphs*, which consist of bipartite graphs of the form $\mathcal{G} = (V, S, E)$ in which factors in the model for the joint probability correspond to factor nodes $s \in S$ in the graph and random variables in the model correspond to nodes $\alpha \in V$.

In addition to these three classes of graphical model, there are also *bidirected graphical*

models, which consist of probabilities defined on bidirected graphs. An example of each of the above types of graphical model for five variables X_1, X_2, X_3, X_4, X_5 is shown in Figures 2.4(a), 2.4(b) and 2.4(c).

Before we proceed to describe each of the above types of graphical model, we first note that the models describe joint probability density functions (PDFs) in the case of continuous variables, joint probability mass functions (PMFs) in the case of discrete variables, or hybrid PDFs/PMFs in the case of a mixture of continuous and discrete random variables. Here we will denote all of these as $P(\mathbf{x})$ and we will by default refer to $P(\mathbf{x})$ as the probability density function, or simply probability. Also, throughout this manuscript we will refer to a node α in a graph and the corresponding random variable X_α interchangeably (similarly for sets of nodes A and the associated random variables $\mathbf{X}_A$).

2.5.1 Directed graphical models

Definition 2.5.1. A *directed graphical model*, or *Bayesian network*, consists of both a directed acyclic graph (DAG) $G = (V, E)$ and a set of conditional probability functions so that the joint probability $P(\mathbf{x})$ is given by

$$P(\mathbf{x}) = \prod_{\alpha \in V} P(x_\alpha | \mathbf{x}_{\mathsf{pa}(\alpha)}), \tag{2.29}$$

where $\mathbf{x}_{\mathsf{pa}(\alpha)}$ denotes the configuration of the parent nodes for α. In the case where $\mathsf{pa}(\alpha) = \emptyset$, we simply have $P(x_\alpha | \mathbf{x}_{\mathsf{pa}(\alpha)}) = P(x_\alpha)$. □

A main feature of the directed graphical model is that the graph allows one to incorporate knowledge about causal relationships between variables, so that edges between a node α and its parents allow us to model the causal influence of the parent variables on variable X_α through the conditional distribution $P(x_\alpha | \mathbf{x}_{\mathsf{pa}(\alpha)})$. The directionality of edges in Bayesian networks equivalently allows one to ascertain conditional independence relationships by inspecting the graph, so that statistical independence relationships may

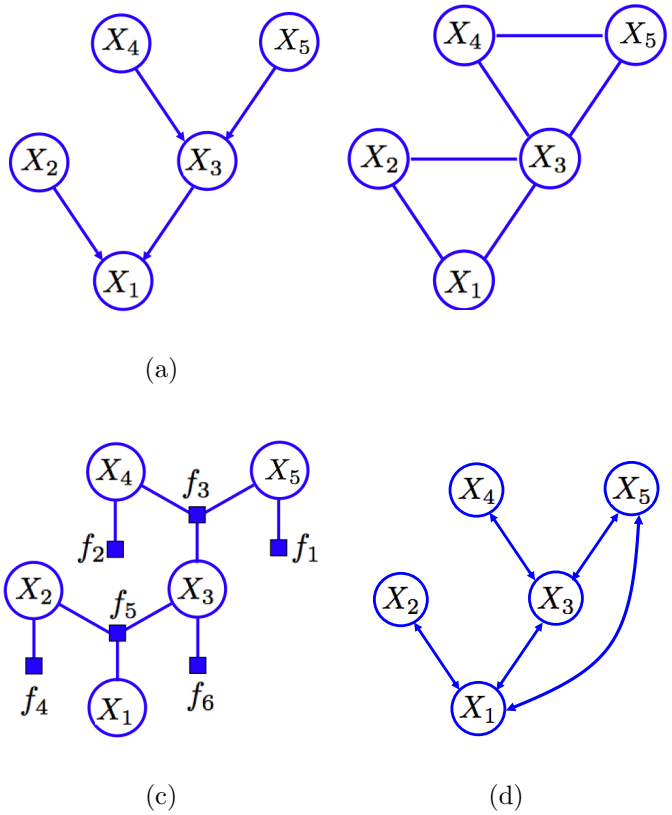


Figure 2.4: a) A Bayesian network over five variables X_1, X_2, X_3, X_4, X_5; b) A Markov random field; b) A factor graph; d) A bidirected graphical model.

exist only between certain subsets of variables in the model. Thus the Bayesian network allows us to model a more complex joint density as a product of local conditional probabilities that each depend on only some subset of the variables. To illustrate such a graphical model, consider the following example.

Example 2.5.1. For the Bayesian network shown in Figure 2.4(a), the joint probability over the variables X_1, X_2, X_3, X_4, X_5 is given by

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1|x_2, x_3)P(x_3|x_4, x_5)P(x_2)P(x_4)P(x_5). \tag{2.30}$$

□

2.5.2 Undirected graphical models

Definition 2.5.2. An *undirected graphical model*, or *Markov random field*, consists of an undirected graph $G = (V, E)$ and a set of clique potential functions $\psi_c(\mathbf{x}_c) \geq 0$ for c in the set of maximal cliques $\mathcal{C}$ of the graph G so that the joint probability $P(\mathbf{x})$ is given by

$$P(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c), \tag{2.31}$$

where the *normalization constant* $\mathcal{Z} = \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c)$ ensures that $P(\mathbf{x})$ sums to unity. When variables are continuous, we replace the above summation operator by an integration operator. □

Note that the potential functions ψ_c are not required to directly correspond to marginal or conditional probabilities. This is in contrast to Bayesian networks, where each function in the expression for the joint probability corresponds to the conditional probability of a variable given its parent variables. However, in special cases such as when the Markov random field is constructed by starting with a Bayesian network, the potential functions may indeed have such an interpretation. One consequence of the generality of the potential functions $\psi_c(\mathbf{x}_c)$ is that their product will in general not sum or integrate to unity, so that we require an explicit normalization factor given by $\mathcal{Z}^{-1}$. To illustrate a simple Markov random field over five variables X_1, X_2, X_3, X_4, X_5, consider the following example.

Example 2.5.2. The Markov random field shown in Figure 2.4(b) models the joint probability

$$P(x_1, x_2, x_3, x_4, x_5) = \frac{1}{\mathcal{Z}} \psi_1(x_1, x_2, x_3) \psi_2(x_3, x_4, x_5), \tag{2.32}$$

where $\mathcal{Z} = \sum_{x_1, x_2, x_3, x_4, x_5} \psi_1(x_1, x_2, x_3) \psi_2(x_3, x_4, x_5)$. □

It is worth noting that if the potentials ψ_c are allowed to be negative, then the joint probability $P(\mathbf{x})$ does not necessarily factor into a product form as given in (2.31) (see [42, 51] for an example).

2.5.3 Factor graph models

Definition 2.5.3. A *factor graph model*, or simply *factor graph*, consists of a bipartite graph $\mathcal{G} = (V, S, E)$, where V denotes a set of nodes corresponding to variables and S denotes nodes corresponding to factors in the model for the joint probability, with edges in E connecting factor nodes in S to variable nodes in V. The factor graph also includes a specification of the *factors*, or functions $f_s(\mathbf{x}_s)$ for each factor node $s \in S$, where $\mathbf{x}_s \equiv \mathbf{x}_{\mathcal{N}(s)}$ and $f_s : \mathbb{R}^{|\mathcal{N}(s)|} \mapsto \mathbb{R}^+$. Given graph $\mathcal{G}$ and factors $f_s(\mathbf{x}_s)$, the joint probability $P(\mathbf{x})$ is then given as the product

$$P(\mathbf{x}) = \prod_{s \in S} f_s(\mathbf{x}_s). \tag{2.33}$$

□

In a factor graph, there is a node $\alpha \in V$ for every variable in the joint probability and additional nodes $s \in S$ (depicted by squares) for each factor $f_s(\mathbf{x}_s)$ in the model for the joint probability $P(\mathbf{x})$.

Example 2.5.3. The factor graph shown in Figure 2.4(c) corresponds to the joint probability

$$P(x_1, x_2, x_3, x_4, x_5) = f_1(x_5) f_2(x_4) f_3(x_3, x_4, x_5) f_4(x_2) f_5(x_1, x_2, x_3) f_6(x_3). \tag{2.34}$$

□

Factor graphs make the factorization of the joint probability over variables explicit. In a factor graph, one can have multiple functions defined over the same sets of variables and functions need not be defined over maximal cliques in the factor graph, whereas in

a Markov random field one must combine these functions together into a single potential defined over a clique of variable nodes in the graph. Similarly, the factors in a factor graph can provide an explicit factorization for a joint probability that would not be explicit in an Bayesian network.

To convert between graphical models, if we are given a probability model that is expressed in terms of an undirected graph, then we can readily convert it to a factor graph by creating variable nodes in the factor graph corresponding to the nodes in the original undirected graph and additional factor nodes for each of the maximal cliques in the undirected graph, so that the factors $f_s(\mathbf{x}_s)$ are set to the clique potentials ψ_c of the corresponding Markov random field. Note that there may be several different factor graphs that correspond to the same Markov random field in terms of the set of conditional independence properties being modelled. From (2.31), we see that Markov random fields are a special case of factor graphs in which the factors correspond to potential functions over the maximal cliques, where the normalizing constant $\mathcal{Z}^{-1}$ can be viewed as a factor defined over the empty set of variables.

We can also show that Bayesian networks, whose corresponding factorization is defined by (2.29), represent special cases of factor graphs in which the factors $f_s(\mathbf{x}_s)$ correspond to conditional probabilities. To convert a Bayesian network to a factor graph, we first create variable nodes in the factor graph corresponding to the nodes of the directed graph, factor nodes corresponding to the conditional probabilities in the model and edges connecting factor nodes to their arguments. In addition, edges in the factor graph may be directed [16] in order to indicate conditional probabilities in the model. Again, there can be multiple factor graphs that can correspond to the same Bayesian network in terms of the set of conditional independence properties being modelled.

2.5.4 Bidirected graphical models

In addition to Bayesian networks, Markov random fields and factor graphs, there are also bidirected graphical models [13, 58, 59, 69] in which edges connecting nodes in the graph are bidirected of the form $\alpha \leftrightarrow \beta$. Unlike the semantics of directed/undirected edges in Bayesian networks and Markov random fields, bidirected edges allow one to enforce marginal independence constraints between variables in the model, so that the lack of an edge $\alpha \leftrightarrow \beta$ between two nodes $\alpha, \beta \in V$ in a bidirected graph imply different conditional independence relationships compared to the absence of an edge in an undirected/directed graphical model. An example of such a graph for a model with five variables X_1, X_2, X_3, X_4, X_5 is shown in Figure 2.4(d). As in the case of Bayesian networks, Markov random fields and factor graph models for probability densities, the joint probability density modeled by a bidirected graphical model factors into a product of local probabilities defined over connected subsets of variable nodes in the graph [13, 59]. Examples of bidirected graphical models are covariance graphs [37], binary models for marginal independence [13] and latent variable mixture models [63] that are constructed as Bayesian networks with latent variables. In a bidirected graphical model, the conditional independence properties are distinct from those of Bayesian networks, Markov random fields and factor graphs, as we will demonstrate below.

2.6 Conditional independence in graphical models

Conditional independence properties play an important role in using probabilistic models by simplifying both the probability model and the computations needed to perform inference and learning under that model. If we are given the joint probability over a set of variables, then in principle we could test whether any potential conditional independence relationship holds by repeated application of the sum and product rules of probability. More precisely, we could assess whether for any disjoint subsets of variables $\mathbf{X}_A$, $\mathbf{X}_B$ and

$\mathbf{X}_C$ we would have $P(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_C) = P(\mathbf{x}_A|\mathbf{x}_C)P(\mathbf{x}_B|\mathbf{x}_C)$ so that $\mathbf{X}_A$ is independent of $\mathbf{X}_B$ given $\mathbf{X}_C$. In practice, such an approach would be very time consuming for an arbitrary joint probability as this would require us to marginalize over many variables in the model in order to compute the quantities in the above expression. One important feature of graphical models is that conditional independence relationships satisfied by the joint probability can be read directly from the graph without having to perform any analytical manipulations. It is worth emphasizing here that one can only ascertain conditional independence relationships due to graph separation in a graphical model: one cannot ascertain conditional *dependence* between variables due to lack of graph separation. For example, consider the simple graphical models $\alpha \rightarrow \beta$, $\alpha - \beta$ and $\alpha - \blacksquare - \beta$. Although clearly α and β are not separated in all three graphs, we can set $P(x_\beta|x_\alpha) = g(x_\beta)$ in the first model, $f(x_\alpha, x_\beta) = \psi(x_\alpha, x_\beta) = g(x_\beta)h(x_\alpha)$ in the second and third models so that X_α is marginally independent of X_β in all three models. Thus with the above example is mind, we will now describe these properties for each of the above types of graphical model: a more complete derivation of these and relevant theorems can be found in [16, 42, 56].

2.6.1 Conditional independence axioms

Suppose we wish to ascertain whether a particular conditional independence statement is implied by a given graph, so that for any disjoint variable sets $\mathbf{X}_A, \mathbf{X}_B$ and $\mathbf{X}_C$, any joint probability defined over the graph satisfies $P(\mathbf{x}_A, \mathbf{x}_B \mid \mathbf{x}_C) = P(\mathbf{x}_A \mid \mathbf{x}_C)P(\mathbf{x}_B \mid \mathbf{x}_C)$ and so $\mathbf{X}_A$ is conditionally independent of $\mathbf{X}_B$ given $\mathbf{X}_C$. We will use the notation of [12] to denote the above conditional independence relationship as $A \perp\!\!\!\perp B \mid C$ for the context of graphical models in which variable sets $\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C$ correspond to node sets A, B, C in the graphical model. The ternary relation $A \perp\!\!\!\perp B \mid C$ then satisfies the following axioms:

- If $A \perp\!\!\!\perp B \mid C$ then $B \perp\!\!\!\perp A \mid C$

- If $A \perp\!\!\!\perp B \mid C$ and $U \subset B$, then $A \perp\!\!\!\perp U \mid C$
- If $A \perp\!\!\!\perp B \mid C$ and $U \subset B$, then $A \perp\!\!\!\perp B \mid (C \cup U)$
- If $A \perp\!\!\!\perp B \mid C$ and $A \perp\!\!\!\perp U \mid (C \cup B)$, then $A \perp\!\!\!\perp (B \cup U) \mid C$

The above axioms correspond to the *semi-graphoid* axioms [42, 47, 56], which encapsulate the formal properties of the notion of mutual irrelevance or separation of sets. With the above axioms in mind, we will now present the resulting conditional independence relationships which can be obtained by inspecting the graph for various graphical models.

2.6.2 Conditional independence in directed graphical models

Consider disjoint sets of nodes $A, B, C \subseteq V$ in a directed acyclic graph $G = (V, E)$. In the case of Bayesian networks, A and B are conditionally independent given C if all undirected paths between nodes in A and B are *blocked* by C [56].

Definition 2.6.1. Let $A, B, C \subseteq V$ be disjoint subsets of nodes in the graph $G = (V, E)$. An undirected path from node $\alpha \in A$ to node $\beta \in B$ in a directed acyclic graph is said to be *blocked* by node set C if the path contains a node γ such that either:

- $\gamma \in C$ is on the path and arrows in the path do not meet head-to-head at γ, or
- $\gamma \in V \setminus C$ is on the path, γ has no descendants in C and arrows on the path meet head-to-head at γ.

□

Node sets A and B are then said to be *d-separated* by C with respect to G if all paths from any node $\alpha \in A$ to any node $\beta \in B$ are blocked by node set C [56]. An example for each of the above rules for blocked paths in a Bayesian network are shown in Figures 2.5(a), 2.5(b) and 2.5(c).

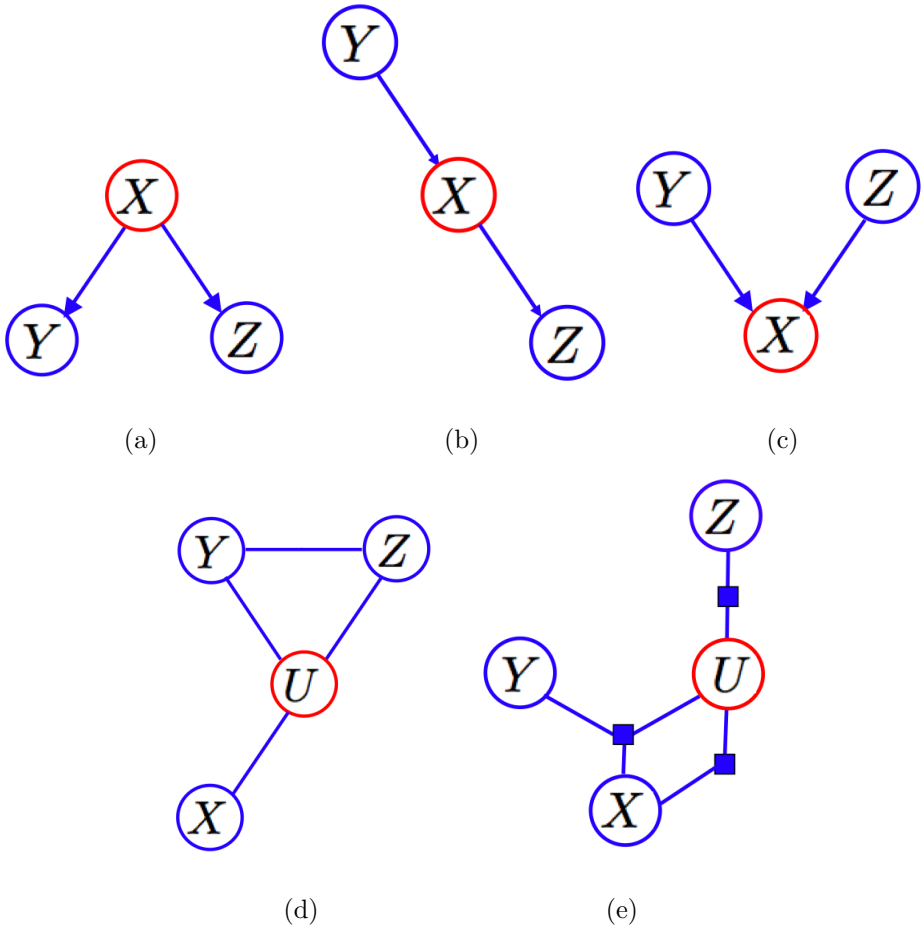


Figure 2.5: a),b) Y and Z are d-separated by X and so $Y \perp\!\!\!\perp Z|X$; c) Y and Z are not d-separated by X; d) Y and Z are separated from X by U and so $Y, Z \perp\!\!\!\perp X|U$; e) X and Z are separated by U and so $X \perp\!\!\!\perp Z|U$.

2.6.3 Conditional independence in undirected graphical models

The conditional independence relationships satisfied by Markov random fields are more straightforward than those of Bayesian networks due to the absence of directed edges, although this implies that it is more difficult to represent marginal independence relationships between variables in a Markov random field. For a Markov random field, the corresponding probability $P(\mathbf{x})$ obeys the *Markov property*, which we present below.

Proposition 2.6.1 (Markov property)**.** Let $G = (V, E)$ be an undirected graph and let $A, B, C \subseteq V$ be disjoint sets of nodes in G. If C separates A from B relative to graph

G, then $A \perp\!\!\!\perp B|C$ holds for any joint probability defined over G. □

It can be shown that the Markov property and the factorization of the joint probability into clique potentials are equivalent. This is formalized by the Hammersley-Clifford theorem, which we present below.

Theorem 2.6.2 (Hammersley and Clifford)**.** Given an undirected graph $G = (V, E)$, a probability distribution $P(\mathbf{x})$ satisfies the Markov property if and only if $P(\mathbf{x})$ can be modeled as $P(\mathbf{x}) = \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c)$ where $\mathcal{C}$ is the set of maximal cliques in the graph G and $\psi_c(\mathbf{x}_c) \geq 0$ are potential functions defined over the random variables corresponding to the nodes in clique c. □

The Hammersley-Clifford theorem establishes the equivalence between the above Markov property and the factorization of the joint probability $P(\mathbf{x})$ into clique potentials $\psi_c(\mathbf{x}_c)$ under the assumption of non-negativity of the potentials ψ_c [42, 51]. A detailed proof of the Hammersley-Clifford theorem can be found in [42]. An example of the Markov property is shown for a simple undirected graph in Figure 2.5(d).

2.6.4 Conditional independence in factor graph models

We have shown that both Bayesian networks and Markov random fields can be converted to factor graphs. Although for any given joint probability, one can have multiple corresponding factor graphs, one can nevertheless inspect the factor graph for separation of sets of nodes to assess conditional independence between two set of variables. More precisely, let $A, B, C \subseteq V$ be disjoint subsets of variable nodes in the factor graph $\mathcal{G}$. Then the separation of A from B by C with respect to $\mathcal{G}$ implies $A \perp\!\!\!\perp B|C$ in the factor graph. An example of such separation is given in Figure 2.5(e).

It can also be shown that all of the conditional independence properties of Bayesian networks can also be represented using directed factor graphs [16] by including directed edges and allowing factors to correspond to conditional distributions of variables given

their parents. In general, all conditional independence relationships that are implied by graph separation in Bayesian networks and Markov random fields can be modeled using directed factor graphs. However, there also exist models for which the set of conditional independence relationships can be modeled by a factor graph but not by Bayesian networks or Markov random fields [16].

2.6.5 Conditional independence in bidirected graphical models

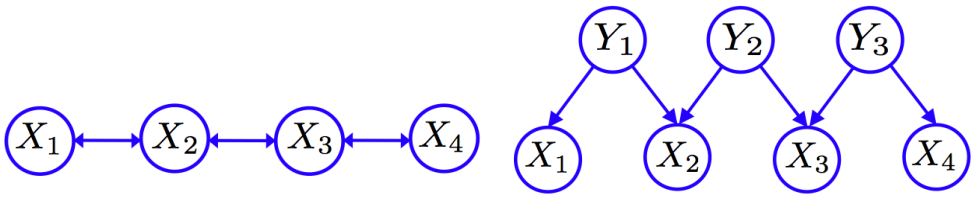


Figure 2.6: a) A bidirected graphical model in which graph separation of nodes implies the marginal independence relationships $X_1 \perp\!\!\!\perp X_3, X_2 \perp\!\!\!\perp X_4, X_1 \perp\!\!\!\perp X_4$; b) A possible Bayesian network with latent variables Y_1, Y_2, Y_3 in which graph separation implies the same marginal independence relationships $X_1 \perp\!\!\!\perp X_3, X_2 \perp\!\!\!\perp X_4, X_1 \perp\!\!\!\perp X_4$.

In a bidirected graphical model in which the joint probability density is defined over a bidirected graph $G = (V, E)$, the absence of an edge (α, β) in E implies the marginal independence relationship $\alpha \perp\!\!\!\perp \beta$. The conditional independence properties of bidirected graphical models are then distinct from the conditional independence properties of Bayesian networks, Markov random fields and factor graphs. The independence property for bidirected graphical models corresponds to the dual Markov property of [37], which we present below.

Theorem 2.6.3 (Dual Markov property)**.** Let $A, B, C \subseteq V$ be three disjoint node sets. If $V \setminus (A \cup B \cup C)$ separates A from B with respect to the bidirected graph $G = (V, E)$. Then $A \perp\!\!\!\perp B|C$. □

The above set of conditional independence relationships in a bidirected graphical model can also be obtained from a Bayesian network defined over the variable nodes in V, with additional latent variables introduced such that for any bidirected edge $\alpha \leftrightarrow \beta$ in the bidirected graphical model, we introduce a latent variable node γ and directed edges $\alpha \leftarrow \gamma \rightarrow \beta$ in the Bayesian network (Figures 2.6(a),2.6(b)). The dual Markov property is then satisfied by any probability obtained from marginalizing out the latent variables in the directed graphical model. Bidirected graphical models can be viewed as a particular case of *mixed graphical models*, or *ancestral graphical models* [58], which consist of probabilities defined over graphs $G = (V, E)$ containing a mixture of undirected edges $\alpha - \beta$, directed edges $\alpha \rightarrow \beta$ and bidirected edges $\alpha \leftrightarrow \beta$. For any disjoint subsets of variable nodes $A, B, C \subseteq V$ in a mixed graph, the conditional independence relationship $A \perp\!\!\!\perp B|C$ holds if A and B are *m-separated* by C with respect to G. The notion of m-separation is a generalization of the notion of d-separation for mixed graphs in which we account for the presence of different types of edges in the mixed graph [58]. Thus, bidirected graphs correspond to mixed graphs that contain only bidirected edges and for which the conditional independence property corresponds to the independence property for a mixed graph containing only bidirected edges. It is worth contrasting the conditional independence properties of bidirected graphical models with those of Markov random fields: in a Markov random field, the absence of an edge between two nodes $\alpha, \beta \in V$ implies that $\alpha \perp\!\!\!\perp \beta|V \setminus (\alpha \cup \beta)$ due to the Markov property, but the marginal independence relationship $\alpha \perp\!\!\!\perp \beta$ is not implied by the absence of the edge $\alpha - \beta$. In contrast, in a bidirected graph the absence of an edge $\alpha \leftrightarrow \beta$ implies that $\alpha \perp\!\!\!\perp \beta$, but the conditional independence relationship $\alpha \perp\!\!\!\perp \beta|V \setminus (\alpha \cup \beta)$ is not implied. In Chapter 3 we will re-visit such graphs in more detail, as they are fundamentally connected to the class of graphical models that are the focus of this thesis.

2.7 The sum-product algorithm

In a graphical model, we often wish to compute quantities of the form $P(\mathbf{x}_A)$ or $P(\mathbf{x}_A \mid \mathbf{x}_B)$, so that we must solve the problem of marginalizing over variables in the model by computing quantities of the form $\sum_{\mathbf{x}_{V \setminus A}} P(\mathbf{x})$. We will now discuss the problem of marginalization in graphical models, where we wish to compute the marginal probabilities for variables in the graph of the form $P(\mathbf{x}_A)$ for some subset of variable nodes A in the graph. Here we will review the *sum-product* algorithm [5, 18, 41] for performing marginalization over variables in the factor graph model.

To begin, let $\mathcal{G} = (V, S, E)$ be a tree-structured bipartite graph and suppose we wish to compute the marginal $P(x_\alpha)$ for some variable X_α. We note that we can root the graph at some node α and we can write the joint probability as

$$P(\mathbf{x}) = \prod_{s \in S} f_s(\mathbf{x}_s) = \prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big), \tag{2.35}$$

where $\mathbf{x}_{\tau_s^\alpha}$ denotes the vector of configurations for all variables in the subtree τ_s^α rooted at variable node α and containing only one factor node neighboring α, namely s (Figure 2.7), and $T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)$ corresponds to the product of all factors located in the subtree τ_s^α.

The marginal probability $P(x_\alpha)$ for the root node α is then given by

$$P(x_\alpha) = \sum_{\mathbf{x}_{V \setminus \alpha}} P(\mathbf{x}) = \sum_{\mathbf{x}_{V \setminus \alpha}} \left[\prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big) \right]. \tag{2.36}$$

We can take advantage of the fact that the graph has a tree structure and hence the node sets τ_s^α and $\tau_{s'}^\alpha$ are disjoint, so that

$$\sum_{\mathbf{x}_{V \setminus \alpha}} \left[\prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big) \right] = \prod_{s \in \mathcal{N}(\alpha)} \sum_{\mathbf{x}_{\tau_s^\alpha \setminus \alpha}} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big) = \prod_{s \in \mathcal{N}(\alpha)} \mu_{s \to \alpha}(x_\alpha). \tag{2.37}$$

We have introduced the set of functions $\mu_{s \to \alpha}(x_\alpha)$ defined by

$$\mu_{s \to \alpha}(x_\alpha) = \sum_{\mathbf{x}_{\tau_s^\alpha \setminus \alpha}} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big). \tag{2.38}$$

We can view the function $\mu_{s \to \alpha}(x_\alpha)$ as a message being passed from factor node $s \in \mathcal{N}(\alpha)$ in the factor graph to a neighboring variable node α.

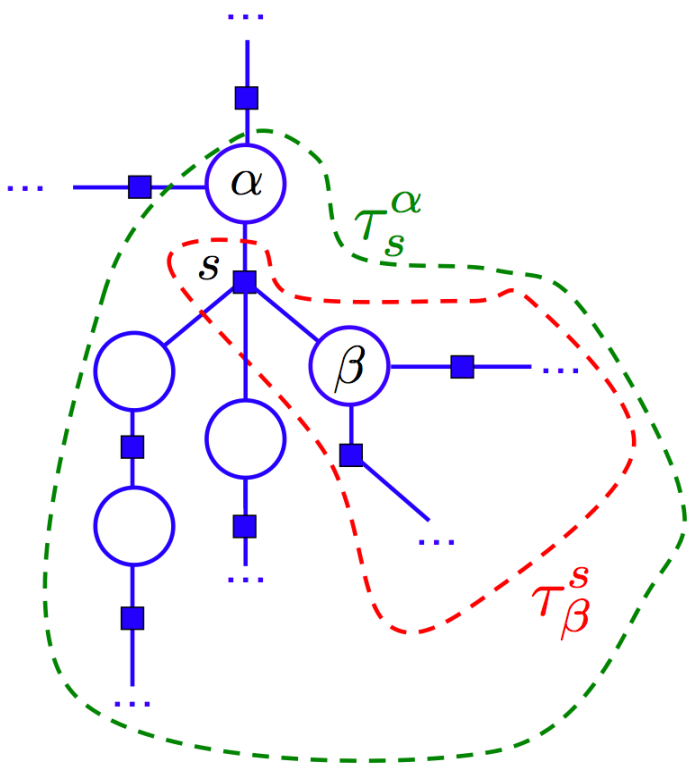


Figure 2.7: Example of the subtrees $\tau_s^\alpha, \tau_\beta^s$ for a tree-structured factor graph $\mathcal{G}$.

We can now write $T_s\left(\mathbf{x}_{\tau_s^\alpha}\right)$ as a product of factors owing to the tree structure of the graph $\mathcal{G}$, so that

$$T_s\left(\mathbf{x}_{\tau_s^\alpha}\right) = f_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right), \tag{2.39}$$

where $\mathbf{x}_{\tau_\beta^s}$ denotes the vector of configurations for all variables in the subtree τ_β^s that is rooted at factor node s and contains only one neighbor of s, namely β (Figure 2.7), and T_β is the product of all factors in the subtree τ_β^s. Substituting Equation (2.39) into Equation (2.38), we obtain

$$\mu_{s\to\alpha}\left(x_\alpha\right) = \sum_{\mathbf{x}_{\tau_s^\alpha\setminus\alpha}} \left[f_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right)\right] \tag{2.40}$$

$$= \sum_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}} \left[f_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \sum_{\mathbf{x}_{\tau_\beta^s\setminus\beta}} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right)\right] \tag{2.41}$$

$$= \sum_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}} \left[f_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \mu_{\beta\to s}(x_\beta)\right], \tag{2.42}$$

where we have defined the message $\mu_{\beta \to s}(x_\beta)$ sent from variable node β to factor node s.

Finally, to compute the messages $\mu_{\beta \to s}(x_\beta)$ from variable nodes to factor nodes, we can write each of the functions $T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right)$ as a product such that

$$T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right) = \prod_{s' \in \mathcal{N}(\beta) \setminus s} T_{s'}\left(\mathbf{x}_{\tau_{s'}^\beta}\right), \tag{2.43}$$

where $T_{s'}$ is defined identically to T_s above but for factor node s'. Substituting this into the expression for $\mu_{\beta \to s}(x_\beta)$ in Equation (2.42) yields

$$\mu_{\beta \to s}(x_\beta) = \sum_{\mathbf{x}_{\tau_\beta^s \setminus \beta}} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right) = \prod_{s' \in \mathcal{N}(\beta) \setminus s} \sum_{\mathbf{x}_{\tau_{s'}^\beta \setminus \beta}} T_{s'}\left(\mathbf{x}_{\tau_{s'}^\beta}\right) \tag{2.44}$$

$$= \prod_{s' \in \mathcal{N}(\beta) \setminus s} \mu_{s' \to \beta}(x_\beta). \tag{2.45}$$

Thus, to compute messages from variable nodes to factor nodes, we simply take the product of all incoming messages except for the message incoming from the destination factor node. Note that variables with only two neighboring factors simply pass messages through, unchanged. Messages can be computed recursively from one another: given an arbitrary root variable node α, we propagate messages up from leaf nodes to the root node. Here, leaf variable nodes α' send the message $\mu_{\alpha' \to s}(x_{\alpha'}) = 1$ while leaf factor nodes $\phi_s(x_{\alpha'})$ send the message $\mu_{s \to \alpha'}(x_{\alpha'}) = \phi_s(x_{\alpha'})$. By then multiplying together the messages incoming into variable node α once all messages have been passed, we can then obtain the marginal distribution $P(x_\alpha)$ by computing

$$P(x_\alpha) = \prod_{s \in \mathcal{N}(\alpha)} \mu_{s \to \alpha}(x_\alpha). \tag{2.46}$$

We can thus efficiently compute the marginal probabilities for any node α in the factor graph by applying the above message-passing algorithm to the factor graph rooted at α. However, there is an even more efficient method to compute all marginal probabilities of the form $P(x_\alpha)$ in a tree-structured graph $\mathcal{G}$. We can choose an arbitrary root node α

and we can then pass messages from leaf nodes to α, and then from α back to the leaf nodes. By passing messages in this fashion, we compute the marginal probability for all variable nodes by multiplying all incoming messages for each node as in Equation (2.46). The sum-product algorithm is summarized in Table 2.1: to illustrate the algorithm, we will apply it to a simply toy example.

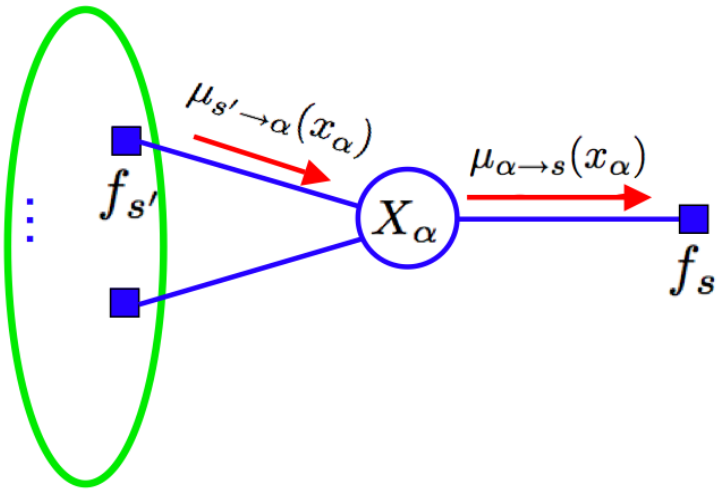


Figure 2.8: Messages in the sum-product algorithm

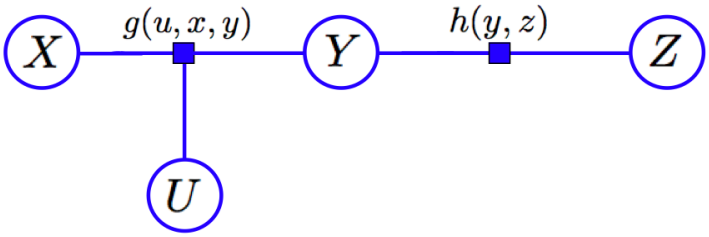


Figure 2.9: A toy example of a factor graph over four variables U, X, Y, Z.

Example 2.7.1. Consider a simple toy example of a factor graph over four random variables U, X, Y, Z. The corresponding factor graph is shown in Figure 2.9 whose joint probability mass function is given by

$$P(u, x, y, z) = g(u, x, y)h(y, z). \tag{2.50}$$

Let Z be the root node so that X and U are leaf nodes. Then the messages from leaves

- For leaf variable nodes α, propagate $\mu_{\alpha\to s}(x_\alpha) = 1$. For leaf factor nodes s, propagate $\mu_{s\to\alpha}(x_\alpha) = f_s(x_\alpha)$.

- (Messages from variables to factors) For each non-leaf variable α and neighboring factors $s \in \mathcal{N}(\alpha)$, propagate

$$\mu_{\alpha\to s}(x_\alpha) = \prod_{s'\in\mathcal{N}(\alpha)\setminus s} \mu_{s'\to\alpha}(x_\alpha) \tag{2.47}$$

- (Messages from factors to variables) For each non-leaf factor node s and neighboring variables $\alpha \in \mathcal{N}(s)$,

$$\mu_{s\to\alpha}(x_\alpha) = \sum_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}} f_s(\mathbf{x}_s) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \mu_{\beta\to s}(x_\beta) \tag{2.48}$$

- Pass messages from root to leaf nodes according to above

- For each node $\alpha \in V$, compute the marginal $P(x_\alpha)$ as

$$P(x_\alpha) \propto \prod_{s\in\mathcal{N}(\alpha)} \mu_{s\to\alpha}(x_\alpha) \tag{2.49}$$

Table 2.1: The sum-product algorithm in a factor graph.

to root are given by

$$\begin{aligned}
\mu_{X\to g}(x) &= 1 \\
\mu_{X\to g}(u) &= 1 \\
\mu_{g\to Y}(y) &= \sum_{u,x} \Big(g(u,x,y)\mu_{X\to g}(x)\mu_{U\to g}(u) \Big) \\
\mu_{Y\to h}(y) &= \mu_{g\to Y}(y) \\
\mu_{h\to Z}(z) &= \sum_{y} \Big(h(y,z)\mu_{Y\to h}(y) \Big).
\end{aligned}$$

Figure 2.10(a) shows the flow of the above messages. Once we have propagated from

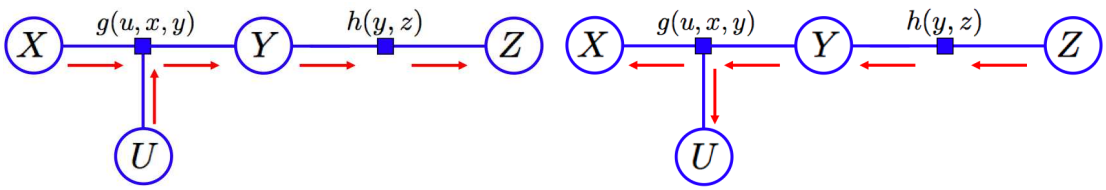
leaves to root, the messages from root to leaf are given by

$$\begin{aligned}
\mu_{Z\to h}(z) &= 1 \\
\mu_{h\to Y}(y) &= \sum_z h(y,z) \\
\mu_{Y\to g}(y) &= \mu_{h\to Y}(z) \\
\mu_{g\to U}(u) &= \sum_{x,y} \Big(g(u,x,y)\mu_{X\to g}(x)\mu_{Y\to g}(y) \Big) \\
\mu_{g\to X}(x) &= \sum_{u,y} \Big(g(u,x,y)\mu_{U\to g}(u)\mu_{Y\to g}(y) \Big).
\end{aligned} \tag{2.51}$$

The flow of messages from root to leaf are shown in Figure 2.10(b). Once we have propagated messages in each direction along each edge, we can evaluate the marginal distributions for each variable. For example, we can verify that the marginal probability $P(z)$ is computed correctly by multiplying all incoming messages to the node for variable Z:

$$\begin{aligned}
P(z) = \mu_{h\to Z}(z) &= \sum_y \Big(h(y,z)\mu_{Y\to h}(y) \Big) \\
&= \sum_y \left(h(y,z) \sum_{u,x} \Big(g(u,x,y)\mu_{X\to g}(x)\mu_{U\to g}(u) \Big) \right) \\
&= \sum_y \left(h(y,z) \sum_{u,x} \Big(g(u,x,y) \Big) \right) \\
&= \sum_{u,x,y} \Big(h(y,z)g(u,x,y) \Big) = \sum_{u,x,y} P(u,x,y,z).
\end{aligned} \tag{2.52}$$

The above example illustrates that we can take advantage of the factor graph model for a joint probability to reduce the number of computations needed to compute the marginals. If the graph is a tree, then the sum-product algorithm is guaranteed to yield the exact marginals [18, 41]. However, if one applies the set of message updates in the sum-product algorithm for graphs with cycles, the algorithm is not guaranteed to converge to the exact marginals. In spite of this, the sum-product algorithm has been applied to many problems involving graphs with cycles where one passes messages over many iterations: this is often referred to as *loopy belief propagation* and has been shown



(a) (b)

Figure 2.10: Flow of messages in the sum-product algorithm applied to Example 2. a) From leaf variable nodes X, U to the root variable node Z; b) From the root node Z to the leaf nodes X, U.

to converge and provide good approximations to marginal probabilities [41, 68] in a wide variety of applications such as error-correction decoding [19], random satisfiability [49] and clustering and facility location [17].

2.7.1 Inference using sum-product

The sum-product algorithm can be also be used for the problem of inference in which we seek to compute probability distributions of the type $P(\mathbf{x}_A|\mathbf{x}_B)$. If we condition on observing variables $\mathbf{X}_B$ in the factor graph, then we need to marginalize over variable nodes in $V \setminus A \cup B$ using the sum-product algorithm. Since

$$P(\mathbf{x}_A|\mathbf{x}_B) \equiv \frac{P(\mathbf{x}_A, \mathbf{x}_B)}{P(\mathbf{x}_B)} = \frac{\sum_{\mathbf{x}_{V\setminus(A\cup B)}} P(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_{V\setminus(A\cup B)})}{\sum_{\mathbf{x}_{V\setminus\{B\}}} P(\mathbf{x}_B, \mathbf{x}_{V\setminus B})}$$
$$\propto \sum_{\mathbf{x}_{V\setminus(A\cup B)}} P(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_{V\setminus(A\cup B)}), \tag{2.53}$$

we can use the sum-product algorithm to compute the above summation (up to a scaling constant) with variables $\mathbf{X}_B$ fixed to $\mathbf{x}_B$. The scaling constant in the above expression for $P(\mathbf{x}_A|\mathbf{x}_B)$ corresponds to the summation $\sum_{\mathbf{x}_{V\setminus\{B\}}} P(\mathbf{x}_B, \mathbf{x}_{V\setminus B})$ and can be obtained from the messages used to compute the sum $\sum_{\mathbf{x}_{V\setminus(A\cup B)}} P(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_{V\setminus(A\cup B)})$. Thus, the sum-product algorithm can be used to compute any conditional distribution of the above form by marginalizing over all unobserved variables in the factor graph save for those variables for which we wish to perform inference. For any variable X_α that is observed

with value x_α^* we can modify messages $\mu_{\alpha \to s}(x_\alpha)$ so as to constrain the variables X_α to values x_α^*. This yields

$$\mu_{\alpha \to s}(x_\alpha) = \delta(x_\alpha - x_\alpha^*) \prod_{s' \neq s} \mu_{s' \to \alpha}(x_\alpha). \tag{2.54}$$

In Chapter 4 we will describe a similar message-passing algorithm called *derivative-sum-product* (DSP) for inference in which the quantities of interest correspond to derivatives of a joint CDF defined over a bipartite graph, so that the marginalization operation in the sum-product algorithm is replaced by differentiation.

Chapter 3

Cumulative distribution networks

As we have seen earlier, graphical models allow us to simplify the computations required for obtaining conditional probabilities of the form $P(\mathbf{x}_A|\mathbf{x}_B)$ or $P(\mathbf{x}_A)$ by allowing us to account for conditional independence constraints in terms of graph separation constraints. Graphical modelling frameworks such as directed, undirected and factor graphs are therefore amenable to efficiently computing a conditional or marginal probability of the above form. However, for many applications it may be desirable to compute other conditional and marginal probabilities such as probabilities defined over events of the type $\{\mathbf{X} \leq \mathbf{x}\}$. Here we will present the cumulative distribution network (CDN), which is a graphical framework for directly modelling the joint cumulative distribution function, or CDF. With the CDN, we can thus expand the set of possible queries so that in addition to formulating queries as conditional/marginal probabilities of the form $P(\mathbf{x}_A)$ and $P(\mathbf{x}_A|\mathbf{x}_B)$, we can also compute probabilities of the form $F(\mathbf{x}_A|\mathbf{X}_B \leq \mathbf{x}_B), F(\mathbf{x}_A|\mathbf{x}_B), P(\mathbf{x}_A|\mathbf{X}_B \leq \mathbf{x}_B)$ and $F(\mathbf{x}_A)$, where $F(\mathbf{x}_U) \equiv \mathbb{P}\Big[\mathbf{X}_U \leq \mathbf{x}_U\Big]$ is a *cumulative distribution function.* Examples of this new type of query could be "Given that the drug dose was less than 1 mg, what is the probability of the patient living at least another year?". A significant advantage with CDNs is that the graphical representation of the joint CDF may naturally allow for queries which would otherwise be difficult to compute under other types of graphical

model.

In this chapter, we will define the CDN and we will show that the conditional independence properties of such graphical models are distinct from the properties for Bayesian networks, Markov random fields and factor graphs. We will then show that the conditional independence properties of CDNs include the properties of bidirected graphical models [13, 59] and so CDNs can provide a class of parameterizations for such probability models. We will later show in Chapter 5 that CDNs provide a tractable means of parameterizing models for learning to rank in which we can construct multivariate CDFs from product of CDFs defined over subsets of variables.

3.1 The cumulative distribution network: A graphical model for cumulative distribution functions

Definition 3.1.1. A *cumulative distribution network* (CDN) is an undirected bipartite graphical model consisting of a bipartite graph $\mathcal{G} = (V, S, E)$, where V denotes variable nodes and S denotes factor nodes, with edges in E connecting factor nodes to variable nodes. The CDN also includes a specification of *functions* $\phi_s(\mathbf{x}_s)$ for each function node $s \in S$, where $\mathbf{x}_s \equiv \mathbf{x}_{\mathcal{N}(s)}$ and each function $\phi_s : \mathbb{R}^{|\mathcal{N}(s)|} \mapsto [0,\ 1]$ satisfies the properties of a CDF (Theorem 2.1.1). The joint CDF over the variables in the CDN is then given by the product over functions ϕ_s, or

$$F(\mathbf{x}) = \prod_{s \in S} \phi_s(\mathbf{x}_s). \tag{3.1}$$

□

Example 3.1.1. For the CDN in Figure 3.1, the joint CDF over three variables X, Y, Z is given by

$$F(x, y, z) = \phi_a(x, y)\phi_b(x, y, z)\phi_c(y, z)\phi_d(z). \tag{3.2}$$

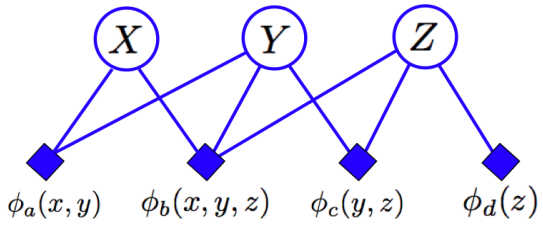


Figure 3.1: A cumulative distribution network (CDN) over three variables and four functions.

□

In the CDN, each function node (depicted as a diamond) corresponds to one of the functions $\phi_s(\mathbf{x}_s)$ in the model for the joint CDF $F(\mathbf{x})$. Thus, one can think of the CDN as a factor graph for modelling the joint CDF instead of the joint PDF. Since the CDN is a graphical model for the joint CDF, the functions in the CDN must be such that their product satisfies the conditions of Theorem 2.1.1 so that $F(\mathbf{x})$ is a CDF for some probability $\mathbb{P}$. The following lemma establishes that it is sufficient that the CDN functions ϕ_s be themselves CDFs in order for F to be a CDF.

Lemma 3.1.1. Suppose that $\cup_{s \in S} \mathcal{N}(s) = V$. If all functions $\phi_s(\mathbf{x}_s)$ satisfy the properties of a CDF, then the product $\prod_{s \in S} \phi_s(\mathbf{x}_s)$ also satisfies the properties of a CDF (Theorem 2.1.1).

Proof. If for all $s \in S$, we have $\lim_{\mathbf{x}_s \to \infty} \phi_s(\mathbf{x}_s) = 1$, then $\lim_{\mathbf{x} \to \infty} \prod_{s \in S} \phi_s(\mathbf{x}_s) = 1$. Furthermore, we have that for any $\alpha \in V$, there exists $s \in \mathcal{N}(\alpha)$ so we have $\lim_{x_\alpha \to -\infty} \phi_s(\mathbf{x}_s) = 0$ and hence $\lim_{x_\alpha \to -\infty} \prod_{s \in S} \phi_s(\mathbf{x}_s) = 0$.

To show that the product of monotonically non-decreasing functions is monotonically non-decreasing, we note that $\mathbf{x}_s \leq \mathbf{y}_s$ for all $s \in S$ if $\mathbf{x} \leq \mathbf{y}$. Thus, since for all $s \in S$ we have $\phi_s(\mathbf{x}_s) \leq \phi_s(\mathbf{y}_s)$ for all $\mathbf{x}_s \leq \mathbf{y}_s$ for all $s \in S$, we can write

$$F(\mathbf{x}) = \prod_{s \in S} \phi_s(\mathbf{x}_s) \leq \prod_{s \in S} \phi_s(\mathbf{y}_s) = F(\mathbf{y}). \tag{3.3}$$

Finally, a product of right-continuous functions is also right-continuous. Thus if all of the functions $\phi_s(\mathbf{x}_s)$ satisfy the properties of a CDF, then the product of such functions

also satisfies the properties of a CDF. □

Although the condition that each of the ϕ_s functions be a CDF is sufficient for the overall product to satisfy the properties of a CDF, we emphasize that it is not a necessary condition. One could construct a function which satisfies the properties of a CDF from a product of functions that are not CDFs. The lemma above ensures, however, that we can construct CDNs by multiplying together CDFs to obtain another CDF.

The above definition and lemma do not assume differentiability of the CDF or of the CDN functions. The following proposition shows that differentiability and non-negativity of the derivatives of functions ϕ_s with respect to all neighboring variables in $\mathcal{N}(s)$ imply both differentiability and monotonicity of the joint CDF $F(\mathbf{x})$.

Proposition 3.1.2. If the mixed derivatives $\partial_{\mathbf{x}_A}\Big[\phi_s(\mathbf{x}_s)\Big]$ satisfy $\partial_{\mathbf{x}_A}\Big[\phi_s(\mathbf{x}_s)\Big] \geq 0$ for all $s \in S$ and $A \subseteq \mathcal{N}(s)$, then $F(\mathbf{x})$ is differentiable, $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] \geq 0$ for all $C \subseteq V$, and $F(\mathbf{x}) \leq F(\mathbf{y})$ for all $\mathbf{x} \leq \mathbf{y}$.

Proof. A product of differentiable functions is differentiable and so $F(\mathbf{x})$ is differentiable. To show that $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] \geq 0 \; \forall \; C \subseteq V$, we can group the functions $\phi_s(\mathbf{x}_s)$ arbitrarily into two functions $g(\mathbf{x})$ and $h(\mathbf{x})$ so that $F(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x})$. The goal here will be to show that if all derivatives $\partial_{\mathbf{x}_A}\Big[g(\mathbf{x})\Big]$ and $\partial_{\mathbf{x}_A}\Big[h(\mathbf{x})\Big]$ are non-negative, then $\partial_{\mathbf{x}_A}\Big[F(\mathbf{x})\Big]$ must also be non-negative. For all $C \subseteq V$, applying the product rule to $F(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x})$ yields

$$\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] = \sum_{A \subseteq C} \partial_{\mathbf{x}_A}\Big[g(\mathbf{x})\Big]\partial_{\mathbf{x}_{C\setminus A}}\Big[h(\mathbf{x})\Big], \tag{3.4}$$

so if $\partial_{\mathbf{x}_A}\Big[g(\mathbf{x})\Big], \partial_{\mathbf{x}_{C\setminus A}}\Big[h(\mathbf{x})\Big] \geq 0$ for all $A \subseteq C$ then $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] \geq 0$. By recursively applying this rule to each of the functions $g(\mathbf{x}), h(\mathbf{x})$ until we obtain sums over terms involving $\partial_{\mathbf{x}_A}\Big[\phi_s(\mathbf{x}_s)\Big] \; \forall \; A \subseteq \mathcal{N}(s)$, we see that if $\partial_{\mathbf{x}_A}\Big[\phi_s(\mathbf{x}_s)\Big] \geq 0$, then $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] \geq 0 \; \forall \; C \subseteq V$.

Now, $\partial_{\mathbf{x}_C}\Big[F(\mathbf{x})\Big] \geq 0$ for all $C \subseteq V$ implies that $\partial_{x_\alpha}\Big[F(\mathbf{x})\Big] \geq 0$ for all $\alpha \in V$. By the Mean Value Theorem for functions of several variables, it then follows that if $\mathbf{x} \leq \mathbf{y}$,

then

$$F(\mathbf{y}) - F(\mathbf{x}) = \sum_{\alpha \in V} \partial_{z_\alpha} \Big[F(\mathbf{z}) \Big] (y_\alpha - x_\alpha) \leq 0, \tag{3.5}$$

and so $F(\mathbf{x})$ is monotonic. □

Proposition 3.1.2 thus shows that we can ensure differentiability and monotonicity of the joint CDF by constraining the derivatives of each of the CDN functions. We note that although it is merely sufficient for the first order derivatives to be non-negative in order for $F(\mathbf{x})$ to be monotonic, the condition that the higher order mixed derivatives of the functions $\phi_s(\mathbf{x}_s)$ be non-negative also implies non-negativity of the first order derivatives. Thus in the sequel, whenever we assume differentiability of CDN functions, we will assume that for all $s \in S$, all mixed derivatives of $\phi_s(\mathbf{x}_s)$ with respect to any and all subsets of argument variables are non-negative. While non-negativity of the mixed derivatives is sufficient for monotonicity of F, it is not a necessary condition. Thus, the sufficient condition allows us to easily design functions whose product yields a function F which satisfies the properties of a CDF, but a result of this is that the CDN may only be able to model a subset of the full set of all possible CDFs.

Having described the above conditions on CDN functions, we will now provide some examples of CDNs constructed from a product of CDFs.

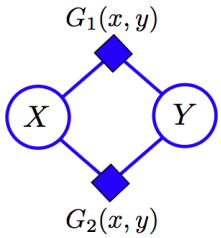
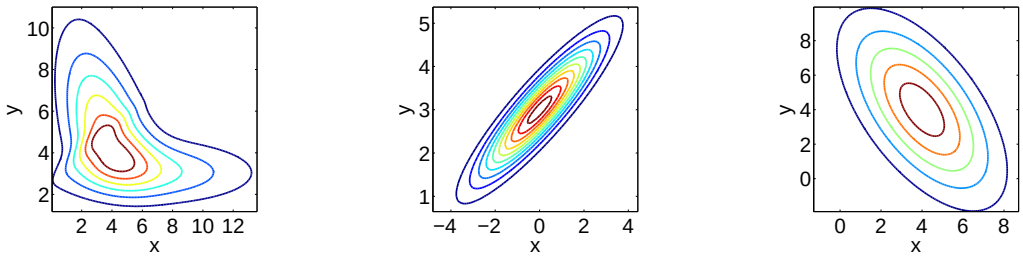


Figure 3.2: A CDN defined over two variables X and Y with functions $G_1(x, y), G_2(x, y)$.

Example 3.1.2 (Product of bivariate Gaussian CDFs)**.** As a simple example of a CDN, consider two random variables X and Y with joint CDF modeled by the CDN in Figure 3.2, so that $F(x, y) = G_1(x, y)G_2(x, y)$ with



(a) (b) (c)

Figure 3.3: a) Joint probability density function $P(x, y)$ corresponding to the distribution function $F(x, y)$ using bivariate Gaussian CDFs as CDN functions; b),c) The PDFs corresponding to $\partial_{x,y}\Big[G_1(x, y)\Big]$ and $\partial_{x,y}\Big[G_2(x, y)\Big]$.

$$
\begin{aligned}
G_1(x, y) &= \Phi\left(\begin{bmatrix} x \\ y \end{bmatrix}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right), \quad \boldsymbol{\mu}_1 = \begin{bmatrix} \mu_{x,1} \\ \mu_{y,1} \end{bmatrix}, \qquad \boldsymbol{\Sigma}_1 = \begin{bmatrix} \sigma^2_{x,1} & \rho_1\sigma_{x,1}\sigma_{y,1} \\ \rho_1\sigma_{x,1}\sigma_{y,1} & \sigma^2_{y,1} \end{bmatrix}, \\
G_2(x, y) &= \Phi\left(\begin{bmatrix} x \\ y \end{bmatrix}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right), \quad \boldsymbol{\mu}_2 = \begin{bmatrix} \mu_{x,2} \\ \mu_{y,2} \end{bmatrix}, \qquad \boldsymbol{\Sigma}_2 = \begin{bmatrix} \sigma^2_{x,2} & \rho_2\sigma_{x,2}\sigma_{y,2} \\ \rho_2\sigma_{x,2}\sigma_{y,2} & \sigma^2_{y,2} \end{bmatrix},
\end{aligned} \tag{3.6}
$$

where $\Phi(\cdot; \mathbf{m}, \mathbf{S})$ is the multivariate Gaussian CDF with mean vector $\mathbf{m}$ and covariance $\mathbf{S}$. Taking derivatives, the density $P(x, y)$ is given by

$$
\begin{aligned}
P(x, y) &= \partial_{x,y}\Big[F(x, y)\Big] = \partial_{x,y}\Big[G_1(x, y)G_2(x, y)\Big] \\
&= G_1(x, y)\partial_{x,y}\Big[G_2(x, y)\Big] + \partial_x\Big[G_1(x, y)\Big]\partial_y\Big[G_2(x, y)\Big] \\
&+ \partial_y\Big[G_1(x, y)\Big]\partial_x\Big[G_2(x, y)\Big] + \partial_{x,y}\Big[G_1(x, y)\Big]G_2(x, y).
\end{aligned} \tag{3.7}
$$

We can compute each of the derivative terms as

$$\partial_{x,y}\Big[G_1(x,y)\Big] = Gaussian\left(\begin{bmatrix}x\\y\end{bmatrix}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right),\ \partial_{x,y}\Big[G_2(x,y)\Big] = Gaussian\left(\begin{bmatrix}x\\y\end{bmatrix}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right) \tag{3.8}$$

$$\begin{aligned}\partial_x\Big[G_1(x,y)\Big] &= \int_{-\infty}^{y} Gaussian\left(\begin{bmatrix}x\\t\end{bmatrix}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) dt \\ &= Gaussian(x; \mu_{x,1}, \sigma^2_{x,1}) \int_{-\infty}^{y} Gaussian(t; \mu_{y|x,1}, \sigma^2_{y|x,1})\, dt \\ &= Gaussian(x; \mu_{x,1}, \sigma^2_{x,1}) \Phi(y; \mu_{y|x,1}, \sigma^2_{y|x,1}) \qquad (3.9)\\ \partial_y\Big[G_1(x,y)\Big] &= \int_{-\infty}^{x} Gaussian\left(\begin{bmatrix}s\\y\end{bmatrix}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) ds \\ &= Gaussian(y; \mu_{y,1}, \sigma^2_{y,1}) \int_{-\infty}^{x} Gaussian(s; \mu_{x|y,1}, \sigma^2_{x|y,1})\, ds \\ &= Gaussian(y; \mu_{y,1}, \sigma^2_{y,1}) \Phi(x; \mu_{x|y,1}, \sigma^2_{x|y,1}) \qquad (3.10)\end{aligned}$$

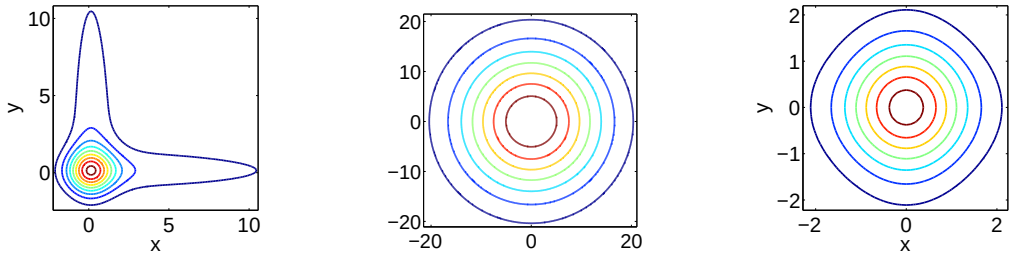
$$\begin{aligned}\partial_x\Big[G_2(x,y)\Big] &= \int_{-\infty}^{y} Gaussian\left(\begin{bmatrix}x\\t\end{bmatrix}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right) dt \\ &= Gaussian(x; \mu_{x,2}, \sigma^2_{x,2}) \int_{-\infty}^{y} Gaussian(t; \mu_{y|x,2}, \sigma^2_{y|x,2})\, dt \\ &= Gaussian(x; \mu_{x,2}, \sigma^2_{x,2}) \Phi(y; \mu_{y|x,2}, \sigma^2_{y|x,2}) \\ \partial_y\Big[G_2(x,y)\Big] &= \int_{-\infty}^{x} Gaussian\left(\begin{bmatrix}s\\y\end{bmatrix}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right) ds \\ &= Gaussian(y; \mu_{y,2}, \sigma^2_{y,2}) \int_{-\infty}^{x} Gaussian(s; \mu_{x|y,2}, \sigma^2_{x|y,2})\, ds \\ &= Gaussian(y; \mu_{y,2}, \sigma^2_{y,2}) \Phi(x; \mu_{x|y,2}, \sigma^2_{x|y,2}), \qquad (3.11)\end{aligned}$$

where

$$\begin{aligned}\mu_{y|x,1} &= \mu_{y,1} + \rho_1 \frac{\sigma_{y,1}}{\sigma_{x,1}}(x - \mu_{x,1}) \quad \mu_{x|y,1} = \mu_{x,1} + \rho_1 \frac{\sigma_{x,1}}{\sigma_{y,1}}(y - \mu_{y,1}) \\ \mu_{y|x,2} &= \mu_{y,2} + \rho_2 \frac{\sigma_{y,2}}{\sigma_{x,2}}(x - \mu_{x,2}) \quad \mu_{x|y,2} = \mu_{x,2} + \rho_2 \frac{\sigma_{x,2}}{\sigma_{y,2}}(y - \mu_{y,2}) \\ \sigma^2_{y|x,1} &= (1 - \rho_1^2)\sigma^2_{y,1} \quad \sigma^2_{x|y,1} = (1 - \rho_1^2)\sigma^2_{x,1} \\ \sigma^2_{y|x,2} &= (1 - \rho_2^2)\sigma^2_{y,2} \quad \sigma^2_{x|y,2} = (1 - \rho_2^2)\sigma^2_{x,2}. \qquad (3.12)\end{aligned}$$

The resulting joint PDF $P(x, y)$ obtained by differentiating the CDF is shown in Figure 3.3(a), where the CDN function parameters are given by $\mu_{x,1} = 0, \mu_{x,2} = 4, \mu_{y,1} = 3, \mu_{y,2} = 4, \sigma_{x,1} = \sqrt{3}, \sigma_{x,2} = \sqrt{5}, \sigma_{y,1} = 1, \sigma_{y,2} = \sqrt{10}, \rho_1 = 0.9, \rho_2 = -0.6$. The PDFs corresponding to $\partial_{x,y}\Big[G_1(x, y)\Big]$ and $\partial_{x,y}\Big[G_2(x, y)\Big]$ are shown in Figures 3.3(b) and 3.3(c). □

The next example provides an illustration of the use of copula functions for constructing multivariate CDFs under the framework of CDNs.



(a) (b) (c)

Figure 3.4: a) Joint probability density function $P(x, y)$ corresponding to the distribution function $F(x, y)$ using bivariate Gumbel copulas as CDN functions, with Student's-t and Gaussian marginal input CDFs; b),c) The PDFs corresponding to $\partial_{x,y}\Big[G_1(x, y)\Big]$ and $\partial_{x,y}\Big[G_2(x, y)\Big]$.

Example 3.1.3 (Product of copulas)**.** We can repeat the above for the case where each CDN function consists of a copula function (see Section 2.3). Copula functions provide a flexible means to construct CDN functions ϕ_s whose product yields a joint CDF under Lemma 3.1.1. Copula functions allow one to construct a multivariate CDF ϕ_s from marginal CDFs $\{F(x_\alpha)\}_{\alpha \in \mathcal{N}(s)}$ so that

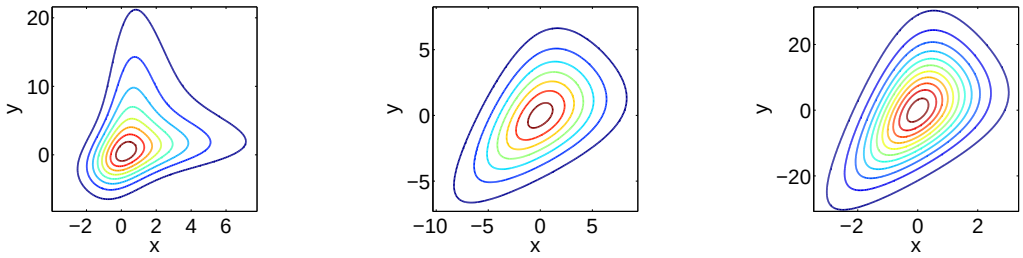
$$\phi_s(\mathbf{x}_s) = \zeta_s\Big(\{F(x_\alpha)\}_{\alpha \in \mathcal{N}(s)}\Big), \tag{3.13}$$

where ζ_s is a copula defined over variables $X_\alpha, \alpha \in \mathcal{N}(s)$. For the CDN shown in Figure

3.2, we can set the CDN functions G_1, G_2 to Gumbel copulas so that

$$G_1(x,y) = \zeta_1(H_{1,x}(x), H_{1,y}(y)) = \exp\left(-\left(-\frac{1}{\theta_1}\Big(\log H_{1,x}(x) + \log H_{1,y}(y)\Big)\right)^{\theta_1}\right),$$
$$G_2(x,y) = \zeta_2(H_{2,x}(x), H_{2,y}(y)) = \exp\left(-\left(-\frac{1}{\theta_2}\Big(\log H_{2,x}(x) + \log H_{2,y}(y)\Big)\right)^{\theta_2}\right), \tag{3.14}$$

with $H_{1,x}, H_{2,x}$ set to univariate Gaussian CDFs with parameters $\mu_{1,x}, \mu_{2,x}, \sigma_{1,x}, \sigma_{2,x}$ and $H_{1,y}, H_{2,y}$ set to univariate Student's-t CDFs with parameters $\sigma_{1,y}, \sigma_{2,y}$. One can then verify that the functions G_1, G_2 satisfy the properties of a copula function (Section 2.3) and so the product of G_1, G_2 yields the CDF $F(x,y)$. An example of the resulting joint probability density $P(x,y)$ obtained by differentiation of $F(x,y)$ for parameters $\mu_{1,x} = \mu_{2,x} = 0, \sigma_{1,x} = \sigma_{2,x} = \sigma_{1,y} = \sigma_{2,y} = 10, \theta_1 = \theta_2 = 1$ is shown in Figure 3.4(a), with the PDFs corresponding to $\partial_{x,y}\Big[G_1(x,y)\Big]$ and $\partial_{x,y}\Big[G_2(x,y)\Big]$ shown in Figures 3.4(b) and 3.4(c). □



(a) (b) (c)

Figure 3.5: a) Joint probability density function $P(x,y)$ corresponding to the distribution function $F(x,y)$ using bivariate sigmoidal functions as CDN functions; b),c) The PDFs corresponding to $\partial_{x,y}\Big[G_1(x,y)\Big]$ and $\partial_{x,y}\Big[G_2(x,y)\Big]$.

Example 3.1.4 (Product of bivariate sigmoids)**.** As another example of a probability density function constructed using a CDN, consider the case in which functions $G_1(x,y)$

and $G_1(x, y)$ in the CDN of Figure 3.2 are set to be multivariate sigmoids of the form

$$G_1(x,y) = \frac{1}{1+\exp(-w_x^1 x)+\exp(-w_y^1 y)}$$
$$G_2(x,y) = \frac{1}{1+\exp(-w_x^2 x)+\exp(-w_y^2 y)}, \tag{3.15}$$

with $w_x^1, w_y^1, w_x^2, w_y^2$ non-negative. An example of the resulting joint probability density $P(x, y)$ obtained by differentiation of $F(x, y) = G_1(x, y)G_2(x, y)$ for parameters $w_x^1 = 1.5, w_y^1 = 0.15, w_x^2 = 0.4, w_y^2 = 0.5$ is shown in Figure 3.5(a), with the PDFs corresponding to $\partial_{x,y}\Big[G_1(x,y)\Big]$ and $\partial_{x,y}\Big[G_2(x,y)\Big]$ shown in Figures 3.5(b) and 3.5(c). □

The above examples demonstrate that one can construct multivariate CDFs by taking a product of CDFs defined over subsets of variables in the graph. Having defined the CDN and presented some basic properties, we will now present the corresponding conditional independence properties that are implied by the definition of a CDN.

3.2 Marginal and conditional independence properties

In this section, we will derive the marginal and conditional independence properties for a CDN. We will see that the conditional independence properties for a CDN are distinct from those of Bayesian networks, Markov random fields or factor graphs. To begin, we consider a toy example of the marginal independence property for a three-variable CDN in Figure 3.6, where variables X and Y are separated by variable Z and so are marginally independent. In a CDN, variables that share no neighbors in the CDN graph are marginally independent: we formalize this now with the following theorem.

Theorem 3.2.1 (Marginal Independence)**.** Let $\mathcal{G} = (V, S, E)$ be a CDN and let $A, B \subseteq V$ be disjoint sets of variables. Then $A \perp\!\!\!\perp B$ if $\mathcal{N}(A) \cap \mathcal{N}(B) = \emptyset$.

Proof. Since $\mathcal{N}(A) \cap \mathcal{N}(B) = \emptyset$, we have

$$F(\mathbf{x}) = \prod_{s \in \mathcal{N}(A)} \phi_s(\mathbf{x}_s) \prod_{s \in \mathcal{N}(B)} \phi_s(\mathbf{x}_s) \prod_{s \notin \mathcal{N}(A) \cup \mathcal{N}(B)} \phi_s(\mathbf{x}_s). \tag{3.16}$$

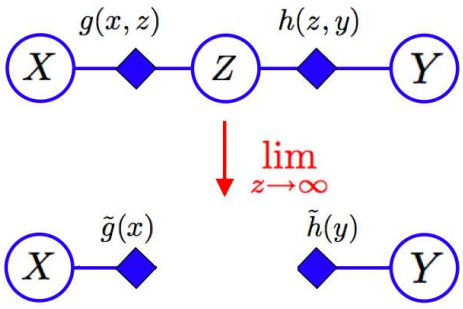


Figure 3.6: Marginal independence property of CDNs: if two variables X and Y share no common function nodes, they are marginally independent.

Marginalizing over all other variables $\mathbf{X}_{V\setminus(A\cup B)}$, we obtain

$$\begin{aligned} F(\mathbf{x}_A, \mathbf{x}_B) &= \lim_{\mathbf{x}_{V\setminus(A\cup B)}\to\infty} F(\mathbf{x}) \\ &= \lim_{\mathbf{x}_{V\setminus(A\cup B)}\to\infty} \prod_{s\in\mathcal{N}(A)} \phi_s(\mathbf{x}_s) \prod_{s\in\mathcal{N}(B)} \phi_s(\mathbf{x}_s) \prod_{s\in S\setminus(\mathcal{N}(A)\cup\mathcal{N}(B))} \phi_s(\mathbf{x}_s) \\ &= \prod_{s\in\mathcal{N}(A)} \lim_{\mathbf{x}_{\mathcal{N}(s)\setminus A}\to\infty} \phi_s(\mathbf{x}_s) \prod_{s\in\mathcal{N}(B)} \lim_{\mathbf{x}_{\mathcal{N}(s)\setminus B}\to\infty} \phi_s(\mathbf{x}_s) \prod_{s\in S\setminus(\mathcal{N}(A)\cup\mathcal{N}(B))} \lim_{\mathbf{x}_{\mathcal{N}(s)}\to\infty} \phi_s(\mathbf{x}_s), \end{aligned} \tag{3.17}$$

where in the last line we have the used the fact that the limit of a product is equal to the product of limits. Let

$$\begin{aligned} g(\mathbf{x}_A) &= \prod_{s\in\mathcal{N}(A)} \lim_{\mathbf{x}_{\mathcal{N}(s)\setminus A}\to\infty} \phi_s(\mathbf{x}_s) \\ h(\mathbf{x}_B) &= \prod_{s\in\mathcal{N}(B)} \lim_{\mathbf{x}_{\mathcal{N}(s)\setminus B}\to\infty} \phi_s(\mathbf{x}_s). \end{aligned} \tag{3.18}$$

Since g, h are products of CDFs, they satisfy the properties of a CDF and so by Lemma 3.1.1, we have $\prod_{s\in S\setminus(\mathcal{N}(A)\cap\mathcal{N}(B))} \lim_{\mathbf{x}_{\mathcal{N}(s)}\to\infty} \phi_s(\mathbf{x}_s) = 1$ and $\lim_{\mathbf{x}_A\to\infty} g(\mathbf{x}_A) = \lim_{\mathbf{x}_B\to\infty} h(\mathbf{x}_B) = 1$. Furthermore, it follows that $F(\mathbf{x}_A) = \lim_{\mathbf{x}_B\to\infty} F(\mathbf{x}_A, \mathbf{x}_B) = g(\mathbf{x}_A)$ and $F(\mathbf{x}_B) = \lim_{\mathbf{x}_A\to\infty} F(\mathbf{x}_A, \mathbf{x}_B) = h(\mathbf{x}_B)$ by marginalizing away the appropriate sets of variables. Thus, we have $F(\mathbf{x}_A, \mathbf{x}_B) = F(\mathbf{x}_A)F(\mathbf{x}_B)$ and so $A \perp\!\!\!\perp B$. □

Note that the converse to the above does not generally hold: if disjoint sets A and B do share functions in S, they can still be marginally independent, as one can easily

construct a bipartite graph in which variable nodes are not separated in the graph but the function nodes connecting A to B correspond to factorized functions so that $A \perp\!\!\!\perp B$.

Having derived the marginal independence property in a CDN, we now consider the conditional independence property of a CDN. To motivate this, we first present a toy example in Figure 3.7 in which we are given CDNs for variables X, Y, Z, W and we condition on variable Z. Here the separation of X and Y by unobserved variable W implies $X \perp\!\!\!\perp Y|Z$, but separation of X and Y by observed variable Z only implies the marginal independence relationship $X \perp\!\!\!\perp Y$. In general, variable sets which are separated in a CDN by unobserved variables will be conditionally independent given all other variables. We formalize this conditional independence property with the following theorem.

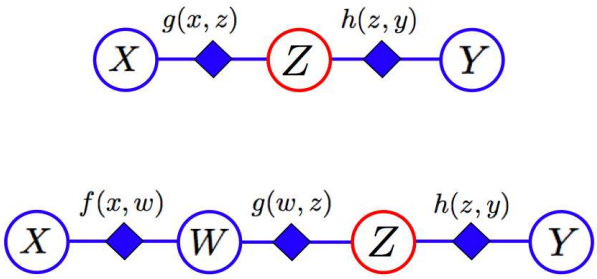


Figure 3.7: Conditional independence in CDNs. Two variables X and Y are marginally independent given the variable Z that separates X from Y with respect to the graph (*top*). When an unobserved variable W separates X from Y, X, Y are conditionally independent given Z (*bottom*).

Theorem 3.2.2 (Conditional independence in CDNs)**.** Let $\mathcal{G} = (V, S, E)$ be a CDN. For all disjoint sets of $A, B, C \subseteq V$, if C separates A from B relative to graph $\mathcal{G}$ then

$$A \perp\!\!\!\perp B|V \setminus (A \cup B \cup C). \tag{3.19}$$

Proof. If C separates A from B, then marginalizing out variables in C yields two disjoint subgraphs with variable sets A', B', with $A \subseteq A'$,$B \subseteq B'$, $A' \cup B' = V \setminus C$ and $\mathcal{N}(A') \cap \mathcal{N}(B') = \emptyset$. From Theorem 3.2.1, we therefore have $A' \perp\!\!\!\perp B'$. Now consider the set

$V \setminus A \cup B \cup C$ and let $\tilde{A}, \tilde{B}$ denote the partition of the set so that

$$\tilde{A} \cup \tilde{B} = V \setminus (A \cup B \cup C), \quad \tilde{A} \cap \tilde{B} = \emptyset$$
$$\tilde{A} \cap B' = \emptyset, \quad \tilde{B} \cap A' = \emptyset \tag{3.20}$$

From the axioms of conditional independence (Section 2.6.1), $A' \perp\!\!\!\perp B'$ implies $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$ since $\tilde{A} \subset A'$ and $\tilde{B} \subset B'$. □

An illustration of the above proof is provided in Figure 3.8(a). The above theorem demonstrates that if C separates A from B with respect to $\mathcal{G}$ for disjoint subsets of nodes A, B, C, then $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$.

The conditional independence property of CDNs is in fact identical to the dual Markov property of [37] for bidirected graphs

Proposition 3.2.3. Let $A, B, C \subseteq V$ be three disjoint node sets so that $V \setminus (A \cup B \cup C)$ separates A from B with respect to $\mathcal{G}$. Then $A \perp\!\!\!\perp B | C$. □

Corollary 3.2.4. Let $A, B, C \subseteq V$ be three disjoint node sets so that C separates A from B with respect to $\mathcal{G}$. Then $A \perp\!\!\!\perp B$. □

The corollary is readily proven by noting that $V \setminus (A \cup B)$ separates sets A and B with respect to $\mathcal{G}$.

In addition to the above, the conditional independence property in a CDN is closed under marginalization, so that if $\mathcal{G}$ is a CDN model for $F(\mathbf{x})$, then there is a CDN for representing any marginal CDF $F(\mathbf{x}_A) = \lim_{\mathbf{x}_{V \setminus A} \to \infty} F(\mathbf{x}_A, \mathbf{x}_{V \setminus A})$.

Proposition 3.2.5. Let $\mathcal{G} = (V, S, E)$ be a CDN and let $A, B, C \subset V$ be disjoint sets of nodes with C separating A from B with respect to $\mathcal{G}$. Let $\mathcal{G}' = (V', S', E')$ be a subgraph of $\mathcal{G}$ with $V' \subseteq V, S' \subseteq S, E' \subseteq E$. Similarly, let $A' = A \cap V', B' = B \cap V', C' = C \cap V'$ be disjoint sets of nodes with $A' \subseteq A, B' \subseteq B, C' \subseteq C$. Then C' separates A' from B' with respect to $\mathcal{G}'$. □

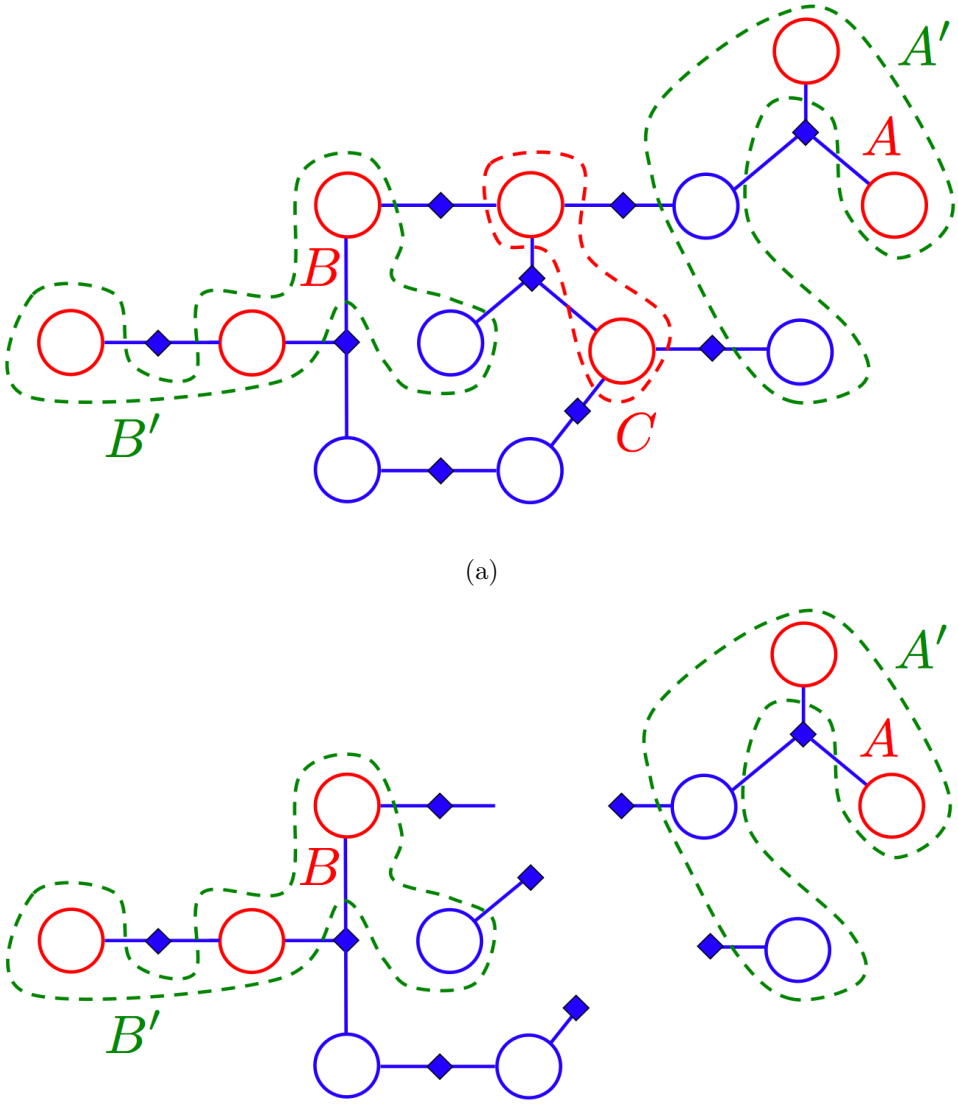


Figure 3.8: Example of conditional independence due to graph separation in a CDN. a) Given bipartite graph $\mathcal{G} = (V, S, E)$, node set C separates set A from B (nodes in red) with respect to $\mathcal{G}$. Furthermore, we have for A', B' (nodes in green dotted line) $A \subseteq A', B \subseteq B'$, $A' \cup B' = V \setminus C$ and $\mathcal{N}(A') \cap \mathcal{N}(B') = \emptyset$ as shown. b) Marginalizing out variables corresponding to nodes in C yields two disjoint subgraphs of $\mathcal{G}$ and so $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$.

As a result, the conditional independence relation $A' \perp\!\!\!\perp B' | V' \setminus (A' \cup B' \cup C')$ must also hold in $\mathcal{G}'$. The above closure property under marginalization is a property that also

holds for Markov random fields, but not for Bayesian networks (see [58]).

Proposition 3.2.6. Let $\mathcal{G} = (V, S, E)$ be a CDN and let $A, B, C \subseteq V$ be disjoint sets of variable nodes. If C separates A from B with respect to $\mathcal{G}$, then $A \perp\!\!\!\perp B | \omega\Big(\mathbf{x}_C\Big)$ where $\omega\Big(\mathbf{x}_C\Big) \equiv \{\mathbf{X}_C \leq \mathbf{x}_C\}$.

Proof. If C separates A from B with respect to $\mathcal{G}$, then we can write

$$F(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C) = g(\mathbf{x}_A, \mathbf{x}_C) h(\mathbf{x}_B, \mathbf{x}_C) \tag{3.21}$$

for some functions g, h that satisfy the conditions of Lemma 3.1.1. This then means that $F(\mathbf{x}_A, \mathbf{x}_B | \omega(\mathbf{x}_C))$ is given by

$$\begin{aligned} F(\mathbf{x}_A, \mathbf{x}_B | \omega(\mathbf{x}_C)) &= \frac{F(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C)}{F(\mathbf{x}_C)} \\ &\propto F(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C) = g(\mathbf{x}_A, \mathbf{x}_C) h(\mathbf{x}_B, \mathbf{x}_C), \end{aligned} \tag{3.22}$$

which implies $A \perp\!\!\!\perp B | \omega(\mathbf{x}_C)$. □

In the next theorem, we show that if a CDF $F(\mathbf{x})$ satisfies the conditional independence properties for a given CDN, then F can be written as a product over functions defined over neighboring sets of variable nodes in $\mathcal{G}$.

Theorem 3.2.7 (Factorization property of a CDN)**.** Let $\mathcal{G} = (V, S, E)$ be a bipartite graph. Given a CDF $F(\mathbf{x})$, suppose F satisfies all of the conditional independence properties implied by the CDN described by $\mathcal{G}$ (Theorem 3.2.2), so that for all disjoint subsets of variable nodes $A, B \subseteq V$ and separating sets of variable nodes $C \subseteq V \setminus A \cup B$ which separate A and B with respect to $\mathcal{G}$, $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$ is satisfied. Then there exist functions $\phi_s(\mathbf{x}_s), s \in S$ that satisfy the properties of a CDF so that the joint CDF $F(\mathbf{x})$ factors as $\prod_{s \in S} \phi_s(\mathbf{x}_s)$.

Proof. The proof here parallels that for the Hammersley-Clifford theorem for undirected graphical models [42]. We begin our proof by defining $\psi_U(\mathbf{x}), \zeta_U(\mathbf{x})$ as functions that

depend only on variable nodes in some set $U \subseteq V$ and that form a Möbius transform pair

$$\psi_U(\mathbf{x}) = \sum_{W \subseteq U} \zeta_W(\mathbf{x}) \tag{3.23}$$

$$\zeta_U(\mathbf{x}) = \sum_{W \subseteq U} (-1)^{|U \setminus W|} \psi_W(\mathbf{x}), \tag{3.24}$$

where we take $\psi_U(\mathbf{x}) \equiv \log F(\mathbf{x}_U)$. Now, we note that $F(\mathbf{x})$ can always be written as a product of functions $\prod_{U \subseteq V} \phi_U(\mathbf{x})$ where each function ϕ_U satisfies the properties of a CDF: a trivial example of this is to set $\phi_V(\mathbf{x}) = F(\mathbf{x})$ and $\phi_U(\mathbf{x}) = 1$ for all $U \subset V$. Since by hypothesis F satisfies all of the conditional independence properties implied by the CDN described by $\mathcal{G}$, if we take $\phi_U(\mathbf{x}) = \exp\Big(\zeta_U(\mathbf{x})\Big)$, then it suffices to show that $\zeta_U(\mathbf{x}) \equiv 0$ for subsets of variable nodes U for which any two non-neighboring variable nodes $\alpha, \beta \in U$ are separated by variable nodes in some set $C \subseteq U \setminus (\alpha \cup \beta)$. If U is such that we can find $\alpha, \beta \in U$ that are separated, then by Corollary 3.2.4, we must have both $\alpha \perp\!\!\!\perp \beta \mid U \setminus (\alpha \cup \beta \cup C)$ and thus $\alpha \perp\!\!\!\perp \beta$. Now we can write $\zeta_U(\mathbf{x})$ as

$$\begin{aligned}\zeta_U(\mathbf{x}) &= \sum_{W \subseteq U} (-1)^{|U \setminus W|} \psi_W(\mathbf{x}) \\ &= \sum_{W \subseteq U \setminus (\alpha \cup \beta)} (-1)^{|U \setminus W|} \Big(\psi_W(\mathbf{x}) - \psi_{W \cup \alpha}(\mathbf{x}) - \psi_{W \cup \beta}(\mathbf{x}) + \psi_{W \cup \alpha \cup \beta}(\mathbf{x}) \Big). \end{aligned} \tag{3.25}$$

Since we have $\alpha \perp\!\!\!\perp \beta$, we have $F(x_\alpha, x_\beta, \mathbf{x}_W) = f(x_\alpha, \mathbf{x}_W) g(x_\beta, \mathbf{x}_W)$ for $W \subseteq U \setminus (\alpha \cup \beta)$ and for functions f, g that satisfy the properties of a CDF so that

$$\begin{aligned}\psi_{W \cup \alpha \cup \beta}(\mathbf{x}) - \psi_{W \cup \alpha}(\mathbf{x}) &= \log \frac{F(x_\alpha, x_\beta, \mathbf{x}_W)}{F(x_\alpha, \mathbf{x}_W)} = \log \frac{f(x_\alpha, \mathbf{x}_W) g(x_\beta, \mathbf{x}_W)}{f(x_\alpha, \mathbf{x}_W) g(\infty, \mathbf{x}_W)} \\ &= \log \frac{f(\infty, \mathbf{x}_W) g(x_\beta, \mathbf{x}_W)}{f(\infty, \mathbf{x}_W) g(\infty, \mathbf{x}_W)} \\ &= \log F(x_\beta, \mathbf{x}_W) - \log F(\mathbf{x}_W) \\ &= \psi_{W \cup \beta}(\mathbf{x}) - \psi_W(\mathbf{x}). \end{aligned} \tag{3.26}$$

Thus if U is any set where nodes $\alpha, \beta \in U$ are separated by some set $C \subseteq U \setminus (\alpha \cup \beta)$, then for all $W \subseteq U \setminus (\alpha \cup \beta)$ we must have $\psi_W(\mathbf{x}) - \psi_{W \cup \alpha}(\mathbf{x}) - \psi_{W \cup \beta}(\mathbf{x}) + \psi_{W \cup \alpha \cup \beta}(\mathbf{x}) \equiv 0$ and so

$\zeta_U(\mathbf{x}) = 0$. Since $F(\mathbf{x}) = \exp(\psi_V(\mathbf{x})) = \exp\Big(\sum_U \zeta_U(\mathbf{x})\Big) = \prod_U \phi_U(\mathbf{x})$ where the product is taken over subsets of variable nodes U that are not separated, and noting that for each s, variable nodes in $\mathcal{N}(s)$ are not separated, we can then substitute $\phi_U(\mathbf{x}) = \phi_s(\mathbf{x}_s)$ into the product with $U = \mathcal{N}(s)$. Thus we can write $F(\mathbf{x}) = \prod_{s \in S} \phi_s(\mathbf{x}_s)$, where each function ϕ_s is defined over the set of variable nodes $\mathcal{N}(s)$. □

Thus, if $F(\mathbf{x})$ satisfies all of the conditional independence properties of a CDN, then F can be written as a product of functions of the form $\prod_{s \in S} \phi_s(\mathbf{x}_s)$. Theorems 3.2.2 and 3.2.7 show that the conditional independence property of a CDN is equivalent to the factorization property, so that graph separation of disjoint subsets A and B by set C implies $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$ if and only if $F(\mathbf{x}) = \prod_{s \in S} \phi_s(\mathbf{x}_s)$. The above closure and conditional independence properties for CDNs have also been previously shown to hold for bidirected graphs as well, as we will now discuss.

3.3 Cumulative distribution networks as bidirected graphs

The independence properties of CDNs have in fact been studied previously in the statistics literature. As shown above, the conditional independence property given in Proposition 3.2.3 for CDNs corresponds to the dual global Markov property of [37] (Theorem 2.6.3) in the context of bidirected graphical models. Furthermore, the connected set property for CDNs presented in Theorem 3.2.7 is in fact identical to the connected set property of [58], which was also derived in the context of bidirected graphical models [13, 58, 59]. This therefore suggests that graph separation in a CDN and a bidirected graph that contains the same paths between variable nodes implies the same sets of independence statements. More precisely, in a bidirected graphical model, the lack of an edge between two nodes α and β implies $\alpha \perp\!\!\!\perp \beta$. Concomitantly, we have shown that in a CDN, two nodes that do not share any function nodes in common are marginally independent. An example of a

bidirected graph and CDN that model the same set of marginal independence constraints is shown in Figures 3.9(a), 3.9(b).

(a) (b)

Figure 3.9: a) A bidirected graph over four variables X_1, X_2, X_3, X_4; b) A corresponding CDN. Graph separation of nodes in both graphs imply the marginal independence relations $X_1 \perp\!\!\!\perp X_4, X_2 \perp\!\!\!\perp X_3$.

Several parameterizations had been previously proposed for bidirected graphical models. Covariance graphs [37] were proposed in which variables are jointly Gaussian with zero pairwise covariance if there is no edge connecting the two variables in the bidirected graph. In addition, [63] proposed a mixture model with latent variables in which dependent variables in the bidirected graph can be explained by the causal influence of common components in the mixture model. For bidirected graphical models defined over binary variables, a parameterization was proposed based on joint probabilities over connected components of the bidirected graph so that the joint probability of any subset of variables could be obtained by Möbius inversion [13]. As we have shown that the above conditional independence properties in CDNs are identical to those for bidirected graphical models, CDNs supplement the above parameterizations with a class of parameterizations for bidirected graphical models that are distinct from the parameterizations proposed by [13, 37, 63].

While the conditional independence properties of continuous variable CDNs are identical to those of bidirected graphical models defined over continuous variables, in the case of graphical models defined over discrete variables there is an additional conditional

independence property that is distinct from the conditional independence properties of bidirected graphical models. For a CDN defined over discrete variables taking values in an ordered set $\mathcal{X} = \{r_1, \cdots, r_K\}$, conditioning on the event $\mathbf{X}_C = r_1\mathbf{1}$ yields conditional independence between disjoint sets $A, B, C \subseteq V$ in which C separates A, B with respect to $\mathcal{G}$. We will define the corresponding *min-independence* property below.

Definition 3.3.1 (Min-independence)**.** Let $\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C$ be sets of ordinal discrete variables that take on values in the totally ordered alphabet $\mathcal{X}$ with minimum element $r_1 \in \mathcal{X}$ defined as $r_1 \prec \alpha \; \forall \, \alpha \neq r_1, \alpha \in \mathcal{X}$. $\mathbf{X}_A$ and $\mathbf{X}_B$ are said to be *min-independent* given $\mathbf{X}_C$ if

$$\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C = r_1\mathbf{1}, \tag{3.27}$$

where $r_1\mathbf{1} = [r_1 \;\cdot\; r_1]^T$. □

Theorem 3.3.1 (Min-independence property of CDNs)**.** Let $\mathcal{G} = (V, S, E)$ be a CDN defined over ordinal discrete variables that take on values in the totally ordered alphabet $\mathcal{X}$ with minimum element $r_1 \in \mathcal{X}$ defined as $r_1 \prec \alpha \; \forall \, \alpha \neq r_1, \alpha \in \mathcal{X}$. Let $A, B, C \subseteq V$ be arbitrary disjoint subsets of V, with C separating A, B with respect to $\mathcal{G}$. Then $\mathbf{X}_A$ and $\mathbf{X}_B$ are min-independent given $\mathbf{X}_C$.

Proof. Since C separates A from B with respect to $\mathcal{G}$, we can write

$$F(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C) = \phi(\mathbf{x}_A, \mathbf{x}_C)\psi(\mathbf{x}_B, \mathbf{x}_C). \tag{3.28}$$

The conditional CDF $F(\mathbf{x}_A, \mathbf{x}_B | \mathbf{x}_C = r_1\mathbf{1})$ is then given by

$$\begin{aligned} F(\mathbf{x}_A, \mathbf{x}_B | \mathbf{x}_C = r_1\mathbf{1}) &= \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{\mathbf{X}_B \leq \mathbf{x}_B\} \cap \{\mathbf{X}_C = r_1\mathbf{1}\}\Big]}{\mathbb{P}\Big[\mathbf{X}_C = r_1\mathbf{1}\Big]} \\ &= \frac{\mathbb{P}\Big[\{\mathbf{X}_A \leq \mathbf{x}_A\} \cap \{\mathbf{X}_B \leq \mathbf{x}_B\} \cap \{\mathbf{X}_C \leq r_1\mathbf{1}\}\Big]}{\mathbb{P}\Big[\mathbf{X}_C \leq r_1\mathbf{1}\Big]} \\ &\propto \phi(\mathbf{x}_A, r_1\mathbf{1})\psi(\mathbf{x}_B, r_1\mathbf{1}), \end{aligned} \tag{3.29}$$

and so $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C = r_1\mathbf{1}$. □

Thus, in the case of a CDN defined over discrete variables where each variables can have values in the discrete poset $\mathcal{X}$, a finite difference with respect to variables $\mathbf{X}_C$, when evaluated at the vector of minimum elements $\mathbf{X}_C = r_1\mathbf{1}$ is equivalent to directly evaluating the CDF at $\mathbf{X}_C = r_1\mathbf{1}$. Thus CDNs defined over discrete variables admit an additional set of rules for assessing conditional independence between sets of variables than those for bidirected graphs defined over discrete variables. This means that in the case of models defined over ordinal discrete variables, the particular set of conditional independence relationships amongst variables in the model is determined as a function of the ordering over possible labels for each variable in the model, so that one must exercise care in how such variables are labelled and what ordering is satisfied by such labels. The above conditional independence property demonstrates however that CDNs defined over discrete variables can be used to construct probability distributions which naturally admit conditional independence relationships amongst model variables which are a function of variable configurations in the model.

3.4 Converting a cumulative distribution network to a factor graph

The connection with bidirected graphical models also allows us to establish a rule for converting between CDNs and factor graphs. In particular, [58] established rules allowing one to convert from a bidirected graphical to a canonical DAG model via the introduction of additional latent variables and directed edges. The resulting probability model obtained from marginalizing out these latent variables can be shown to satisfy the same set of conditional independence properties that result from graph separation in the original bidirected graph (see Figures 2.6(a), 2.6(b)). This transformation rule thus allows one to explain the presence of statistical dependence relationships between observable variables in the CDN through the causal influence of shared latent variables

in the corresponding factor graph. We can take advantage of the connection between bidirected graphical models and directed graphical models to construct a corresponding factor graph for a CDN in which we introduce additional latent variables into the probability model. Any probability modeled by this factor graph must satisfy the same set of independence constraints implied by graph separation of variables in the CDN. We present the corresponding representation theorem below.

Theorem 3.4.1 (Factor graph representation for a CDN)**.** Let $\mathcal{G} = (V, S, E)$ be a CDN and let $A, B, C \subseteq V$ be any disjoint subsets of nodes in $\mathcal{G}$ with C separating A from B with respect to $\mathcal{G}$. For each $s \in S$, let Y_s be a latent random variable corresponding to function ϕ_s and let $\mathbf{Y}_\alpha = \{Y_s : s \in \mathcal{N}(\alpha)\}$ be a vector of such latent random variables corresponding to functions connected to node α in $\mathcal{G}$. Let $\mathbf{Y} = \{Y_s : s \in S\}$ with $Y_s \perp\!\!\!\perp Y_{s'}\ \forall\ s \neq s'$ and let $\mathbf{Y}_U$ be the vector of variables $\{Y_s : s \in \mathcal{N}(U)\}$ for some subset of variable nodes U. Let $\mathcal{G}' = (V', S', E')$ be the factor graph modelling the joint probability density function

$$P(\mathbf{x}, \mathbf{y}) = \prod_{\alpha \in V} P(x_\alpha | \mathbf{y}_\alpha) \prod_{s \in S} P(y_s). \tag{3.30}$$

with $V' = (V, S)$ and $S' = V$. Then for any disjoint subsets $A, B, C \subseteq V$ where C separates A from B with respect to $\mathcal{G}$, the probability density $P(\mathbf{x})$ obtained by marginalizing the probability $P(\mathbf{x}, \mathbf{y})$ modeled by $\mathcal{G}'$ with respect to the variables $\mathbf{Y}$ satisfies $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$.

Proof. We begin by noting that by hypothesis, there is a one-to-one correspondence between the latent variables in the factor graph and the function nodes in the CDN and there is also a one-to-one correspondence between observable variable nodes V in the CDN and the factor nodes in the factor graph. Thus each latent variable Y_s has neighboring factor nodes f_α for $\alpha \in \mathcal{N}(s)$ in the factor graph $\mathcal{G}'$.

If C separates A from B with respect to $\mathcal{G}$, then removing nodes in C from $\mathcal{G}$ by marginalizing out the corresponding variables yields at least two disjoint subgraphs with

node sets A', B', with $A \subseteq A'$,$B \subseteq B'$, $A' \cup B' = V \setminus C$ and $\mathcal{N}(A') \cap \mathcal{N}(B') = \emptyset$. In $\mathcal{G}'$, we must therefore have $\mathbf{Y}_{A'} \cap \mathbf{Y}_{B'} = \emptyset$. The probability density $P(\mathbf{x}_{V\setminus C}, \mathbf{y})$ modeled by the factor graph $\mathcal{G}'$ with variables $\mathbf{X}_C$ marginalized out can then be written as

$$\begin{aligned} P(\mathbf{x}_{V\setminus C}, \mathbf{y}) &= \int_{\mathbf{x}_C} P(\mathbf{x}, \mathbf{y})\, d\mathbf{x}_C \\ &= \prod_{s\in S} P(y_s) \prod_{\alpha \in V\setminus C} P(x_\alpha|\mathbf{y}_\alpha) \\ &= \left(\prod_{s\in\mathcal{N}(A')} P(y_s) \right) \prod_{\alpha\in A'} P(x_\alpha|\mathbf{y}_\alpha) \prod_{s\in\mathcal{N}(B')} P(y_s) \left(\prod_{\beta\in B'} P(x_\beta|\mathbf{y}_\beta) \right) \\ &= f_{A'}(\mathbf{x}_{A'}, \mathbf{y}_{A'}) f_{B'}(\mathbf{x}_{B'}, \mathbf{y}_{B'}) \end{aligned} \tag{3.31}$$

for some functions $f_{A'}, f_{B'}$. Thus the conditional probability $P(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_{V\setminus(A\cup B\cup C)})$ must have the form

$$\begin{aligned} P(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_{V\setminus(A\cup B\cup C)}) \propto P(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_{V\setminus(A\cup B\cup C)}) &= \int_{\mathbf{y}} P(\mathbf{x}_{V\setminus C}, \mathbf{y})\, d\mathbf{y} \\ &= \int_{\mathbf{y}_{A'}} f_{A'}(\mathbf{x}_{A'}, \mathbf{y}_{A'})\, d\mathbf{y}_{A'} \int_{\mathbf{y}_{B'}} f_{B'}(\mathbf{x}_{B'}, \mathbf{y}_{B'})\, d\mathbf{y}_{B'} \\ &= \tilde{f}_A(\mathbf{x}_A, \mathbf{x}_{A'\setminus A}) \tilde{f}_B(\mathbf{x}_B, \mathbf{x}_{B'\setminus B}) \end{aligned} \tag{3.32}$$

for functions $\tilde{f}_A, \tilde{f}_B$, where in the last line we have used $A \subseteq A', B \subseteq B'$. Since $A' \cap B' = \emptyset$, the above is equivalent to $A \perp\!\!\!\perp B|V \setminus (A \cup B \cup C)$. □

Thus for any given CDN, there exists a factor graph that can be obtained by mapping each variable node in the CDN to a factor node in the factor graph and each function node in the CDN to a latent variable in the factor graph. We then connect each factor node f_α to its corresponding observable variable node α and all latent variable nodes Y_s whose CDN function nodes are connected to α in the CDN. Finally, we set each factor f_α to be equal to the conditional probability $P(x_\alpha|\mathbf{y}_\alpha)$ under the joint probability $P(\mathbf{x}, \mathbf{y})$. By doing so, if A, B, C are disjoint sets of variables where A is separated from B by C with respect to $\mathcal{G}$, then any joint probability $P(\mathbf{x})$ obtained by marginalizing

over variables $\mathbf{Y}$ in the joint probability $P(\mathbf{x}, \mathbf{y})$ modeled by the corresponding factor graph $\mathcal{G}' = ((V, S), S, E')$ will satisfy $A \perp\!\!\!\perp B | V \setminus (A \cup B \cup C)$. We note that the set of conditional independence relationships amongst variables in the CDN are not preserved in the construction if we set the factors f_α to be arbitrary functions of neighboring variables. Figures 3.10(a), 3.10(b), 3.10(c) provide examples of a CDN, a factor graph and a Bayesian network in which any joint probability modeled by any of the three graphical models must satisfy the same set of conditional independence relationships between variables X_1, X_2, X_3, X_4.

In the above construction, the CDN models the CDF $F(\mathbf{x})$ corresponding to a probability $P(\mathbf{x}, \mathbf{y})$ with latent variables $\mathbf{Y}$ marginalized out. Often, the cost of marginalizing out latent variables in probability density functions can be difficult and may require approximation schemes such as that of [50] or the use of sampling techniques [53]. The representation theorem suggests that CDNs can be used to directly construct models over observable variables without the need to explicitly introduce latent variables and then marginalize these out of the model. Equivalently, one can view CDNs as directed graphical models with latent variables implicitly marginalized out, so that we are left with a probability model defined over the remaining observable variables. Note however that the above theorem does not guarantee uniqueness of the CDN-factor graph pair, nor does it always yield a closed-form mapping between CDN functions and factors. As a result, the CDF $F(\mathbf{x})$ modeled by the CDN and the probability $P(\mathbf{x}, \mathbf{y})$ modeled by the factor graph are related by $P(\mathbf{x}) = \int_{\mathbf{y}} P(\mathbf{x}, \mathbf{y}) = \partial_{\mathbf{x}} \Big[F(\mathbf{x}) \Big]$. The above representation theorem merely guarantees the existence of a CDN-factor graph pair for which graph separation in the CDN imply the same set of conditional independence relationships between observable variables in the factor graph model. There is however a particular closed-form mapping between a CDN and factor graph in the special case where variables in the CDN are constrained to be maxima of latent variables in the factor graph. We formalize this in the following lemma.

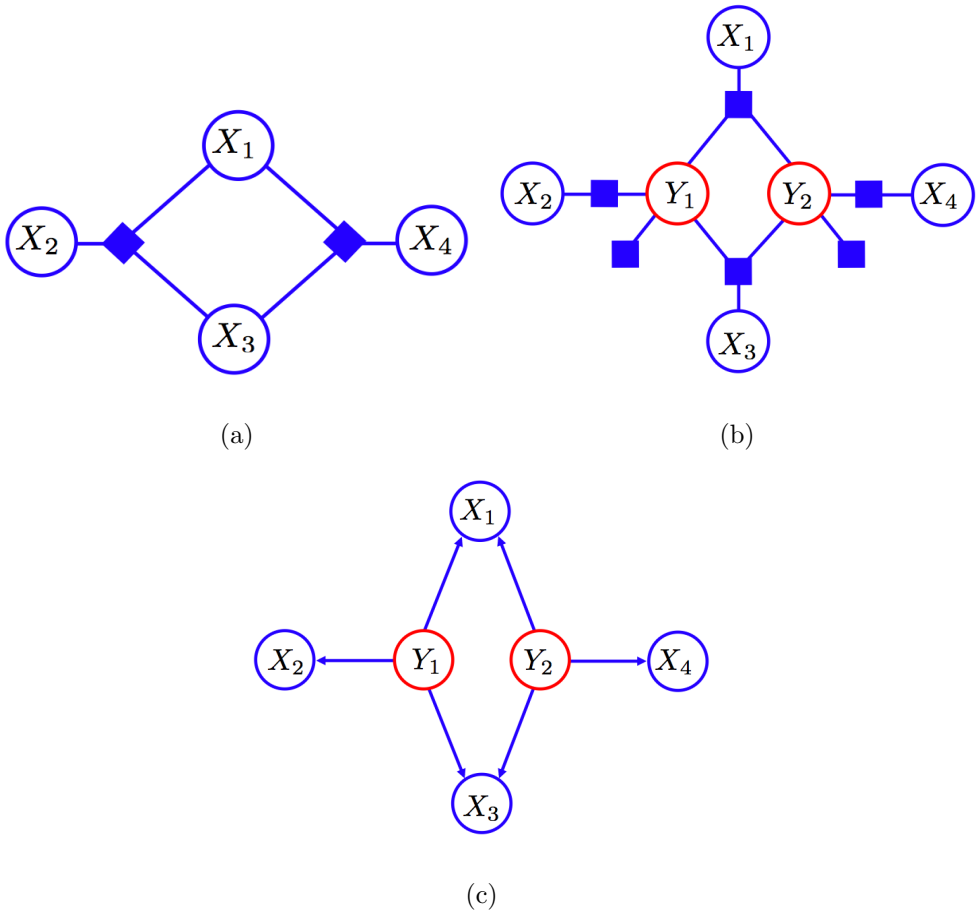


Figure 3.10: a) A CDN defined over observable variables X_1, X_2, X_3, X_4; b) A factor graph with latent variables Y_1, Y_2 introduced and factor nodes neighbouring observable variable nodes are set to conditional probabilities conditioned on the latent variables; c) The corresponding directed graphical model. Any joint probability modeled by any of three graphs satisfies the marginal independence relations $X_2 \perp\!\!\!\perp X_4$.

Lemma 3.4.2. Let $\mathcal{G}'$ be the factor graph for modelling the joint probability density function

$$P(\mathbf{x}, \mathbf{y}) = \prod_{\alpha \in V} \delta\Big(x_\alpha, \max_{s \in \mathcal{N}(\alpha)} y_s\Big) \prod_{s \in S} \psi_s(y_s), \tag{3.33}$$

where $\psi_s(y_s) \geq 0 \; \forall y_s \in \mathbb{R}$. Then the corresponding CDF $F(\mathbf{x})$ obtained by marginalizing out variables $\mathbf{Y}$ is given by the CDN $\mathcal{G} = (V, S, E)$ for

$$F(\mathbf{x}) = \prod_{s \in S} \phi_s\Big(\min_{\alpha \in \mathcal{N}(s)} x_\alpha\Big), \tag{3.34}$$

with $\phi_s(u) = \int_{-\infty}^{u} \psi_s(t)\, dt$ and $\phi_s(\infty) = \int_{-\infty}^{\infty} \psi_s(t)\, dt = 1$.

Proof. This is a special case of the previous lemma with $P(x_\alpha|\mathbf{y}_\alpha) = \delta\Big(x_\alpha, \max_{s\in\mathcal{N}(\alpha)} y_s\Big)$ and $P(y_s) = \psi_s(y_s)$. If we write the joint CDF of variables $X_\alpha, \alpha \in V$, then we obtain

$$\begin{aligned} F(\mathbf{x}) &= \mathbb{P}\Bigg[\cap_{\alpha\in V}\,\{X_\alpha \le x_\alpha\}\Bigg] = \mathbb{P}\Bigg[\cap_{\alpha\in V}\,\{\max_{s\in\mathcal{N}(\alpha)} Y_s \le x_\alpha\}\Bigg] = \mathbb{P}\Bigg[\cap_{\alpha\in V}\cap_{s\in\mathcal{N}(\alpha)}\{Y_s \le x_\alpha\}\Bigg] \\ &= \mathbb{P}\Bigg[\cap_{s\in S}\,\{Y_s \le \min_{\alpha\in\mathcal{N}(s)} x_\alpha\}\Bigg] = \prod_{s\in S}\mathbb{P}\Big[Y_s \le \min_{\alpha\in\mathcal{N}(s)} x_\alpha\Big] \\ &= \prod_{s\in S}\phi_s\Big(\min_{\alpha\in\mathcal{N}(s)} x_\alpha\Big), \end{aligned} \tag{3.35}$$

where we have used mutual independence of the Y_s variables in the second to last line. □

The above theorem allows us to view CDNs of the form of Equation 3.34 as *graphical extreme value distributions* where each observable variable X_α in the model is equal to the maximum over some *finite* subset of latent variables Y_s. The graphical modelling framework of CDNs thus allows one to model the set of statistical dependence relationships between observable variables under the constraint that the observable variables are equal to the maxima of subsets of latent variables, as we will now show with an example.

Example 3.4.1 (CDNs as graphical extreme value distributions)**.** An example of an extreme value distribution is the Gumbel distribution [21], which is the limiting distribution of a maximum of an infinite number of random variables. Such distributions arise frequently in applications in many fields, such as in finance and climate modelling. Extreme value distributions allow us to model variables as being the result of computing the maximum of many random variables. As a result, such distributions typically possess heavy tails, as opposed to many other distributions (e.g.: Gaussians, Poisson, exponential distributions) in which tails decay exponentially.

In particular, the extreme value distributions which arise in practice are typically multivariate and can be therefore be challenging to specify. If we model the statistical dependence relationships between latent variables and observable variables as a factor

graph as per Lemma 3.4.2 and if we assume the latent variables Y_s in the factor graph are Gumbel-distributed, so that the functions $\phi_s(y_s)$ take the form of Gumbel cumulative distribution functions

$$\phi_s(y_s) = \exp\left(-\exp\left(-\frac{y_s - \mu_s}{\sigma_s}\right)\right) \; \forall y_s \in \mathbb{R}, \tag{3.36}$$

then the CDN provides a graphical representation for a multivariate Gumbel distribution, as we now show.

Theorem 3.4.3 (Graphical Gumbel distributions)**.** Let the latent variables $\{Y_s\}_{s\in S}$ be Gumbel-distributed so the marginal CDF of Y_s is given by Equation (3.36). Then the CDN functions are each given by multivariate Gumbel distributions of the form

$$\phi_s(\mathbf{x}_s) \equiv \phi_s\Big(\min_{\alpha\in\mathcal{N}(s)} x_\alpha\Big) = \lim_{t\to\infty} \exp\left(-\left[\sum_{\alpha\in\mathcal{N}(s)} \exp\Big(-\frac{x_\alpha - \mu_s}{\sigma_s}t\Big)\right]^{\frac{1}{t}}\right) \tag{3.37}$$

and the joint CDF is given by the multivariate Gumbel distribution

$$F(\mathbf{x}) = \prod_{s\in S} \phi_s(\mathbf{x}_s) = \lim_{t\to\infty} \exp\left(-\sum_{s\in S}\left[\sum_{\alpha\in\mathcal{N}(s)} \exp\Big(-\frac{x_\alpha - \mu_s}{\sigma_s}t\Big)\right]^{\frac{1}{t}}\right). \tag{3.38}$$

Note that the CDN functional form for $\phi_s(\mathbf{x}_s)$ is a generalization of the bivariate Gumbel distribution [21] with correlation coefficient $\eta = \frac{1}{t} \to 0$.

Proof. We begin by noting that $\min_{\alpha\in\mathcal{N}(s)} x_\alpha$ can be written as

$$\min_{\alpha\in\mathcal{N}(s)} x_\alpha = \lim_{t\to\infty} -\frac{1}{t}\log\sum_{\alpha\in\mathcal{N}(s)} \exp\Big(-x_\alpha t\Big), \tag{3.39}$$

so that

$$\begin{aligned} F(\mathbf{x}) &= \prod_{s\in S}\phi_s\Big(\min_{\alpha\in\mathcal{N}(s)} x_\alpha\Big) = \exp\left(\sum_{s\in S} -\exp\Big(-\min_{\alpha\in\mathcal{N}(s)}\frac{x_\alpha - \mu_s}{\sigma_s}\Big)\right) \\ &= \exp\left(\sum_{s\in S} -\exp(-\lim_{t\to\infty} -\frac{1}{t}\log\sum_{\alpha\in\mathcal{N}(s)} \exp\Big(-\frac{x_\alpha - \mu_s}{\sigma_s}t\Big)\right) \\ &= \lim_{t\to\infty}\exp\left(-\sum_{s\in S}\left[\sum_{\alpha\in\mathcal{N}(s)} \exp\Big(-\frac{x_\alpha - \mu_s}{\sigma_s}t\Big)\right]^{\frac{1}{t}}\right). \end{aligned} \tag{3.40}$$

□

The CDN thus provide a graphical framework with which one can construct multivariate extreme-value distributions in which additional statistical dependence relationships can be modeled between variables in the model.

Another consequence of the use of graphs to model CDFs is that notions of orderings of variables can be specified in terms of local functions in the graphical model. We will now describe this in the following section.

3.5 Stochastic orderings in a cumulative distribution network

The CDN, in providing a graphical model for the joint CDF over many random variables, also allows one to easily specify *stochastic ordering* constraints between subsets of variables in the model (Section 2.2). Using the graphical framework of CDNs, we can specify such constraints for variables $\mathbf{X}_A, \mathbf{X}_B$ by constraining the appropriate neighboring function nodes for the variable node sets A and B in the CDN. We will focus here on first-order stochastic ordering constraints [43, 62] of the form $X \preceq Y$ and how one can specify such constraints in terms of the CDN functions in the model. We note that such constraints are not a necessary part of the definition for a CDN or for a multivariate CDF, so that the graph for the CDN alone does not allow one to inspect stochastic ordering constraints based on graph separation of variables. However, the introduction of stochastic ordering constraints, in combination with separation of variables with respect to the graph, do impose constraints on the products of CDN functions, as we will now show.

Proposition 3.5.1. Let $\mathcal{G} = (V, S, E)$ be a CDN, with $A, B \subset V$ so that $A = \{\alpha_1, \cdots, \alpha_K\}$ and $B = \{\beta_1, \cdots, \beta_K\}$ for some strictly positive integer K. Let $\mathbf{t} \in \mathbb{R}^K$. Then A, B sat-

isfy the stochastic ordering relationship $\mathbf{X}_A \preceq \mathbf{X}_B$ if and only if

$$\prod_{s \in \mathcal{N}(A)} \lim_{\mathbf{u}_{\mathcal{N}(s)\setminus A} \to \infty} \phi_s(\mathbf{u}_{\mathcal{N}(s)\setminus A}, \mathbf{t}_{\mathcal{N}(s)\cap A}) \geq \prod_{s \in \mathcal{N}(B)} \lim_{\mathbf{u}_{\mathcal{N}(s)\setminus B} \to \infty} \phi_s(\mathbf{u}_{\mathcal{N}(s)\setminus B}, \mathbf{t}_{\mathcal{N}(s)\cap B}) \tag{3.41}$$

for all $\mathbf{t} \in \mathbb{R}^K$.

The above can be readily obtained by marginalizing over variables in $V \setminus A, V \setminus B$ respectively to obtain expressions for $F(\mathbf{x}_A), F(\mathbf{x}_B)$ as products of CDN functions. The corresponding ordering then holds from Definition 2.2.2 if and only if $F_{\mathbf{X}_A}(\mathbf{t}) \geq F_{\mathbf{X}_B}(\mathbf{t})$ for all $\mathbf{t} \in \mathbb{R}^K$.

A direct corollary of the above proposition is that we can enforce pairwise stochastic ordering constraints of the form $X_\alpha \preceq X_\beta$, which we will now present.

Corollary 3.5.2. Let $\mathcal{G} = (V, S, E)$ be a CDN in which functions satisfy Lemma 3.1.1 and let $\alpha, \beta \in V$ be two variable nodes in $\mathcal{G}$. Then the stochastic ordering relationship $X_\alpha \preceq X_\beta$ is satisfied if and only if

$$\prod_{s \in \mathcal{N}(\alpha)} \lim_{\mathbf{u}_{\mathcal{N}(s)\setminus \alpha} \to \infty} \phi_s(t, \mathbf{u}_{\mathcal{N}(s)\setminus \alpha}) \geq \prod_{s \in \mathcal{N}(\beta)} \lim_{\mathbf{u}_{\mathcal{N}(s)\setminus \beta} \to \infty} \phi_s(t, \mathbf{u}_{\mathcal{N}(s)\setminus \beta}) \tag{3.42}$$

for all $t \in \mathbb{R}$.

The above corollary can be readily obtained by marginalizing over variables in $V \setminus \alpha, V \setminus \beta$ respectively to obtain expressions for $F(x_\alpha), F(x_\beta)$ as products of CDN functions. The corresponding ordering then holds from Definition 2.2.2 if and only if $F_{x_\alpha}(t) \geq F_{x_\beta}(t)$ for all $t \in \mathbb{R}$.

3.6 Discussion

We have presented the CDN and sufficient conditions on the functions in the CDN in order for the CDN to model to a CDF. We have shown that the conditional independence relationships that follow from graph separation in CDNs are different from the relationships implied by graph separation in Bayesian networks, Markov random fields and factor

graph models. The conditional independence properties of CDNs however include those of bidirected graphs, so that CDNs provide a parameterization for such models. For ordinal discrete random variables, an additional independence property called min-independence holds in which two disjoint sets of variables A and B are conditionally independent of one another if one observes all variables in set C being equal to the minimum element of the poset.

We presented a representation theorem that allows us to construct a factor graph with additional latent variables such that any probability obtained by marginalizing out the additional latent variables satisfies the same conditional independence relationships that follow from graph separation of nodes in the CDN. We demonstrated that the mapping from a CDN to a factor graph is generally not one-to-one and generally does not admit a closed-form expression relating the CDF modeled by the CDN to the probability modeled by the factor graph. In the special case in which observable variables are constrained to be equal to the maxima of latent variables in the factor graph, this mapping is one-to-one and admits a closed form. This then allows us to construct CDNs as graphical extreme value distributions in which variables in the model correspond to the maxima over independent latent variables. Finally, we have shown how one can enforce stochastic ordering constraints amongst variables in the model through constraining products of CDN functions.

Having defined the CDN and having presented theorems on conditional independence in CDNs and on converting between a CDN and a factor graph, in the next chapter we will turn to the problem of inference and how to efficiently compute probabilities.

Chapter 4

The derivative-sum-product algorithm

In the previous chapter, we showed that within the context of a joint CDF, we could compute conditional probabilities of the forms $F(\mathbf{x}_A|\omega(\mathbf{x}_B))$, $F(\mathbf{x}_A|\mathbf{x}_B)$ and $P(\mathbf{x}_A|\omega(\mathbf{x}_B))$, $P(\mathbf{x}_A|\mathbf{x}_B)$, in addition to probabilities of the type $P(\mathbf{x}_A)$, $F(\mathbf{x}_A)$. In Bayesian networks, Markov random fields and factor graphs, computing such conditional CDFs would generally require us to integrate over variables, which may be an intractable operation requiring sampling methods or approximation schemes. However in a CDN, computing such conditionals is comparatively easier, as we can compute the relevant quantities by *differentiating* the joint CDF. In this chapter we will show that if we model the joint CDF using a CDN with a tree-structured graph, then we can derive a class of message-passing algorithms called called *derivative-sum-product* (DSP) for computing derivatives efficiently in CDNs. Throughout this chapter, we will assume that the sufficient conditions for the CDN functions $\phi_s(\mathbf{x}_s)$ hold in order for the CDN to model a valid joint CDF (Lemma 3.1.1). We will further assume that the derivatives/finite differences of CDN functions $\phi_s(\mathbf{x}_s)$ with respect to all subsets of argument variables exist and that Proposition 2.1.7 holds for each function ϕ_s so that the order of differentiation does not affect the computation of any mixed derivatives of functions in the CDN. In the case where we are differentiating with respect to a set of variables $\mathbf{X}_C$ that are observed with values $\mathbf{x}_C$, we assume that

the resulting derivative/finite difference is evaluated at the observed values $\mathbf{x}_C$. In the case where we are given a function $G(x)$ defined over a single ordinal discrete variable $x \in \mathcal{X}$ with $\mathcal{X} = \{r_0, r_1, \cdots, r_{N-1}\}$ and $r_0 < r_1 \cdots < r_{N-1}, r_i \in \mathbb{R}$, we define the finite difference of G with respect to x, evaluated at x as

$$\partial_x\Big[G(x)\Big] = \begin{cases} G(r_0) & \text{if } x = r_0 \\ G(r_i) - G(r_{i-1}) & \text{if } x = r_i,\ i = 1, \cdots, N-1 \end{cases} \tag{4.1}$$

4.1 Differentiation in cumulative distribution networks

We first consider the problem of finding the marginal cumulative distribution $F(x_\alpha)$ for particular variable X_α. We note that in the CDN, marginalization is a simple procedure that involves maximization with respect to the variables in the network, so that if we let

$$F(\mathbf{x}) = F(x_\alpha, \mathbf{x}_{V\setminus\alpha}) = \prod_{s\in\mathcal{N}(\alpha)} \phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{s\notin\mathcal{N}(\alpha)} \phi_s(\mathbf{x}_s), \tag{4.2}$$

then the marginal CDF for X_α is given by

$$F(x_\alpha) = \lim_{\mathbf{x}_{V\setminus\alpha}\to\infty} F(x_\alpha, \mathbf{x}_{V\setminus\alpha}) = \prod_{s\in\mathcal{N}(\alpha)} \phi_s(x_\alpha, \infty) \prod_{s\notin\mathcal{N}(\alpha)} \phi_s(\infty) = \prod_{s\in\mathcal{N}(\alpha)} \phi_s(x_\alpha, \infty). \tag{4.3}$$

Thus for any x_α, we can obtain any distribution of the type $F(\mathbf{x}_A)$ in time $O(|S||V|)$ by taking the product of limits of functions $\lim_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}\to\infty} \phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) = \phi_s(x_\alpha, \infty)$. Furthermore, from Theorem 2.1.4, we can compute any conditional cumulative distribution of the type $F(\mathbf{x}_A|\omega(\mathbf{x}_B))$ in the same fashion by marginalizing the joint CDF over variables in $V \setminus A \cup B$. Note that the above marginalization contrasts with the problem of exact inference in density models, where potentially intractable marginalization operations must be performed locally for each variable node in order to obtain the desired marginals.

Although obtaining marginals in the CDN is relatively simple, computing probability distributions of the form $F(\mathbf{x}_A|\mathbf{x}_B), P(\mathbf{x}_A|\omega(\mathbf{x}_B)), P(\mathbf{x}_A|\mathbf{x}_B)$ and $P(\mathbf{x}_A)$ is more involved. We have seen previously that in order to compute conditional CDFs, we must compute

corresponding higher-order derivatives with respect to these observed variables. Fortunately, computing derivatives is generally tractable compared to the marginalization operation in probability densities. Since the factorization of the joint CDF modeled by a CDN consists of a product of local functions $\phi_s(\mathbf{x}_s)$, the intuition here is that we can distribute the differentiation operation to local function nodes in the graph such that each one computes the derivatives with respect to local variables and passes the result to its neighbors. Our derivation here is analogous to the derivation for the sum-product algorithm, but with the summation operator replaced by the differentiation operator.

To begin, let $\mathcal{G} = (V, S, E)$ be a tree-structured CDN and suppose we wish to compute the joint probability $P(\mathbf{x})$ and evaluate it at observation $\mathbf{x}$. We note that we can root the graph at some node α and we can write the joint CDF as

$$F(\mathbf{x}) = \prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big), \tag{4.4}$$

where $\mathbf{x}_{\tau_s^\alpha}$ denotes the vector of configurations for all variables in the subtree τ_s^α rooted at variable node α and containing function node s (Figure 4.1), and $T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)$ corresponds to the product of all functions located in the subtree τ_s^α.

Now suppose we are interested in computing the probability

$$P(\mathbf{x}) = \partial_{\mathbf{x}}\Big[F(\mathbf{x})\Big] = \partial_{\mathbf{x}}\left[\prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)\right]. \tag{4.5}$$

Here, we take advantage of the fact that the graph has a tree structure, so that

$$\partial_{\mathbf{x}}\left[\prod_{s \in \mathcal{N}(\alpha)} T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)\right] = \partial_{x_\alpha}\left[\prod_{s \in \mathcal{N}(\alpha)} \partial_{\mathbf{x}_{\tau_s^\alpha \setminus \alpha}}\Big[T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)\Big]\right] = \partial_{x_\alpha}\left[\prod_{s \in \mathcal{N}(\alpha)} \mu_{s\to\alpha}\Big(\mathbf{x}_{\tau_s^\alpha}\Big)\right], \tag{4.6}$$

where we have introduced the set of functions $\mu_{s\to\alpha}(\mathbf{x}) \equiv \mu_{s\to\alpha}\Big(\mathbf{x}_{\tau_s^\alpha}\Big)$, each defined by

$$\mu_{s\to\alpha}(\mathbf{x}) \equiv \mu_{s\to\alpha}\Big(\mathbf{x}_{\tau_s^\alpha}\Big) = \partial_{\mathbf{x}_{\tau_s^\alpha \setminus \alpha}}\Big[T_s\Big(\mathbf{x}_{\tau_s^\alpha}\Big)\Big] \tag{4.7}$$

and we have assumed that each of the derivatives/finite differences have been evaluated at the desired values $\mathbf{x}_{\tau_s^\alpha \setminus \alpha}$. By its definition, $\mu_{s\to\alpha}(\mathbf{x})$ only depends on variables in the subtree τ_s^α and corresponds to the higher order derivative of the joint CDF with respect

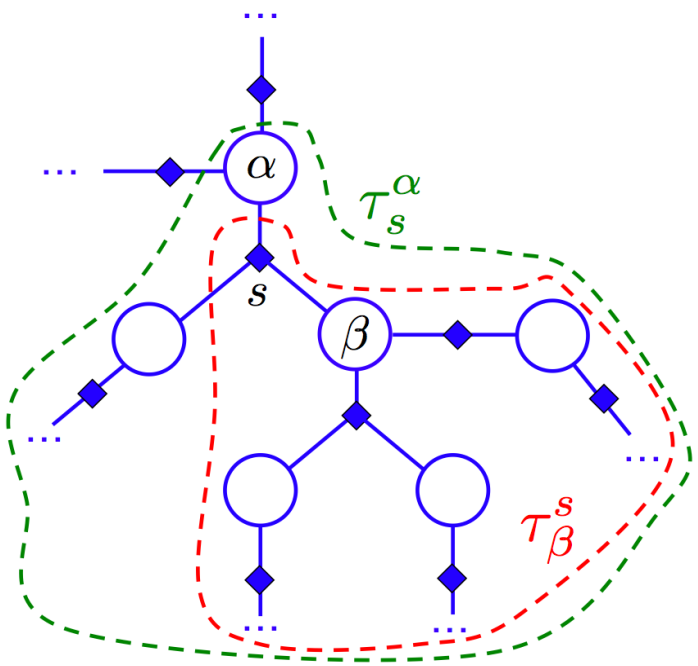


Figure 4.1: Example of the subtrees $\tau_s^\alpha, \tau_\beta^s$ for a tree-structured CDN given by the graph $\mathcal{G}$.

to variables in the subtree $\tau_s^\alpha \setminus \alpha$. We can thus view the function $\mu_{s\to\alpha}(\mathbf{x})$ as a message being passed from function node s to neighboring variable node α.

We can now write $T_s\left(\mathbf{x}_{\tau_s^\alpha}\right)$ as a product of functions owing to the tree structure of the graph $\mathcal{G}$, so that

$$T_s\left(\mathbf{x}_{\tau_s^\alpha}\right) = \phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right), \tag{4.8}$$

where $\mathbf{x}_{\tau_\beta^s}$ denotes the vector of configurations for all variables in the subtree τ_β^s that is rooted at function node s and contains node β (Figure 4.1), and T_β is the product of all functions in the subtree τ_β^s. Substituting Equation (4.8) into Equation (4.7), we obtain

$$\mu_{s\to\alpha}(\mathbf{x}) \equiv \mu_{s\to\alpha}\left(\mathbf{x}_{\tau_s^\alpha}\right) = \partial_{\mathbf{x}_{\tau_s^\alpha\setminus\alpha}}\left[\phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right)\right] \tag{4.9}$$

$$= \partial_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}}\left[\phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \partial_{\mathbf{x}_{\tau_\beta^s\setminus\beta}}\left[T_\beta\left(\mathbf{x}_{\tau_\beta^s}\right)\right]\right] \tag{4.10}$$

$$= \partial_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}}\left[\phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \mu_{\beta\to s}\left(\mathbf{x}_{\tau_\beta^s}\right)\right], \tag{4.11}$$

where we have defined the message $\mu_{\beta\to s}(\mathbf{x}) \equiv \mu_{\beta\to s}\Big(\mathbf{x}_{\tau_\beta^s}\Big)$ from variable node β to neighboring function node s. Similar to the definition for $\mu_{s\to\alpha}(\mathbf{x})$, the message $\mu_{\beta\to s}(\mathbf{x})$ only depends on variables in the subtree τ_β^s and corresponds to the higher order derivative of the joint CDF with respect to variables in the subtree $\tau_\beta^s \setminus \beta$.

Finally, to compute the message $\mu_{\beta\to s}(\mathbf{x})$ from variable node β to function node s, we can write each of the functions $T_\beta\Big(\mathbf{x}_{\tau_\beta^s}\Big)$ as a product such that

$$T_\beta\Big(\mathbf{x}_{\tau_\beta^s}\Big) = \prod_{s'\in\mathcal{N}(\beta)\setminus s} T_{s'}\Big(\mathbf{x}_{\tau_{s'}^\beta}\Big), \tag{4.12}$$

where $T_{s'}$ is defined identically to T_s above but for function node s'. Substituting this into the expression for $\mu_{\beta\to s}(\mathbf{x})$ in Equation (4.11) yields

$$\mu_{\beta\to s}(\mathbf{x}) = \partial_{\mathbf{x}_{\tau_\beta^s\setminus\beta}}\Big[T_\beta\Big(\mathbf{x}_{\tau_\beta^s}\Big)\Big] = \prod_{s'\in\mathcal{N}(\beta)\setminus s} \partial_{\mathbf{x}_{\tau_{s'}^\beta\setminus\beta}}\Big[T_{s'}\Big(\mathbf{x}_{\tau_{s'}^\beta}\Big)\Big] \tag{4.13}$$

$$= \prod_{s'\in\mathcal{N}(\beta)\setminus s} \mu_{s'\to\beta}(\mathbf{x}). \tag{4.14}$$

Thus, to compute messages from variables to functions, we simply take the product of all incoming messages except for the message coming from the destination function node. As in the sum-product algorithm, variables with only two neighboring functions simply pass messages through unchanged. We see here that the process of differentiation in a CDN can be implemented as an algorithm in which we pass messages $\mu_{\alpha\to s}$ from variables to neighboring function nodes and messages $\mu_{s\to\alpha}$ from functions to neighboring variable nodes. Messages can be computed recursively from one another: we start from an arbitrary root variable node α and propagate messages up from leaf nodes to the root node. As in the sum-product algorithm, leaf variable nodes α' send the message $\mu_{\alpha'\to s}(\mathbf{x}) = 1$ while leaf function nodes $\phi_s(x_{\alpha'})$ send the message $\mu_{s\to\alpha'}(\mathbf{x}) = \phi_s(x_{\alpha'})$.

The message-passing algorithm proceeds until messages have been propagated along every edge in the network and the root variable node has received all incoming messages

from the remainder of the network. Once all messages have been sent, we can obtain the probability of the variables in the graph from differentiating the product of incoming messages at the root node α, so that

$$P(\mathbf{x}) = \partial_{x_\alpha}\left[\prod_{s\in\mathcal{N}(\alpha)} \mu_{s\to\alpha}(\mathbf{x})\right]. \tag{4.15}$$

To illustrate the above message-passing algorithm, consider the following toy example.

Example 4.1.1. Consider the toy example of a CDN over four random variables U, X, Y, Z from Figure 4.2. The joint CDF is given by $F(u, x, y, z) = g(u, x, y)h(y, z)$. Let Z be the

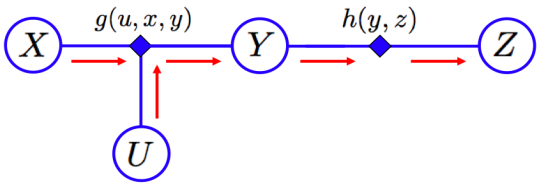


Figure 4.2: Flow of messages in the toy example of CDN defined over variables X, Y, Z, U.

root node so that X and U are leaf nodes. Then the messages from leaves to root are given by

$$\begin{aligned}
&\mu_{X\to g}(x) = 1\\
&\mu_{U\to g}(u) = 1\\
&\mu_{g\to Y}(y; u, x) = \partial_{u,x}\Big[g(u, x, y)\mu_{X\to g}(x)\mu_{U\to g}(u)\Big]\\
&\mu_{Y\to h}(y; u, x) = \mu_{g\to Y}(y; u, x)\\
&\mu_{h\to Z}(z; u, x, y) = \partial_y\Big[h(y, z)\mu_{Y\to h}(y; u, x)\Big],
\end{aligned}$$

where derivatives are computed at locally before being propagated to the next node in the graph. Figure 4.2 shows the flow of the above messages.

Once we have propagated messages from the leaf nodes to the root node, we can

evaluate the joint probability $P(u,x,y,z) = \partial_z\Big[\mu_{h\to Z}(z;u,x,y)\Big]$ at the root node so that

$$\begin{aligned}
P(u,x,y,z) &= \partial_z\Big[\mu_{h\to Z}(z;u,x,y)\Big] = \partial_z\Big[\partial_y\Big[h(y,z)\mu_{Y\to h}(y;u,x)\Big]\Big] \\
&= \partial_z\Big[\partial_y\Big[h(y,z)\mu_{g\to Y}(y;u,x)\Big]\Big] \\
&= \partial_z\Big[\partial_y\Big[h(y,z)\partial_{u,x}\Big[g(u,x,y)\mu_{X\to g}(x)\mu_{U\to g}(u)\Big]\Big]\Big] \\
&= \partial_{x,y,z,u}\Big[g(u,x,y)h(y,z)\Big] \\
&= \partial_{x,y,z,u}\Big[F(u,x,y,z)\Big]. \qquad (4.16)
\end{aligned}$$

The above example illustrates the fact that if the graph topology is a tree, then the message-passing algorithm yields the correct mixed derivatives with respect to each variable in the CDN so that we obtain the joint probability $P(\mathbf{x})$ at the root node α according to Equation (4.15).

4.2 Inference in cumulative distribution networks

Thus far we have presented an algorithm for computing derivatives of the joint CDF in order to obtain the joint PDF/PMF $P(\mathbf{x})$. In this section we will demonstrate the correspondence between computing higher order derivatives and the problem of inference in a CDN. The relation between differentiation and inference in CDNs is analogous to the relation between marginalization and inference in factor graphs. Thus, just as the sum-product algorithm allows one to compute distributions of the type $P(\mathbf{x}_A|\mathbf{x}_B)$, message-passing in a CDN allows us to compute conditional distributions of the form $F(\mathbf{x}_A|\mathbf{x}_B)$ and $P(\mathbf{x}_A|\mathbf{x}_B)$ for disjoint sets $A, B \subset V$.

In order to compute conditional distributions of the above types, we will assume that when computing a conditional distribution such as $F(\mathbf{x}_A|\mathbf{x}_B)$ or $P(\mathbf{x}_A|\mathbf{x}_B)$, we have $P(\mathbf{x}_B) = \partial_{\mathbf{x}_B}\Big[F(\mathbf{x}_B)\Big] > 0$. Now consider the problem of computing the quantity $F(\mathbf{x}_A|\mathbf{x}_B)$. We can write this as

$$F(\mathbf{x}_A|\mathbf{x}_B) = \frac{\partial_{\mathbf{x}_B}\Big[F(\mathbf{x}_A,\mathbf{x}_B)\Big]}{\partial_{\mathbf{x}_B}\Big[F(\mathbf{x}_B)\Big]} = \frac{\lim\limits_{\mathbf{x}_{V\setminus(A\cup B)}\to\infty}\partial_{\mathbf{x}_B}\Big[F(\mathbf{x})\Big]}{\lim\limits_{\mathbf{x}_{V\setminus B}\to\infty}\partial_{\mathbf{x}_B}\Big[F(\mathbf{x})\Big]} = \frac{\partial_{\mathbf{x}_B}\Big[\lim\limits_{\mathbf{x}_{V\setminus(A\cup B)}\to\infty}F(\mathbf{x})\Big]}{\partial_{\mathbf{x}_B}\Big[\lim\limits_{\mathbf{x}_{V\setminus B}\to\infty}F(\mathbf{x})\Big]}$$
$$\propto \partial_{\mathbf{x}_B}\Big[\lim_{\mathbf{x}_{V\setminus(A\cup B)}\to\infty}F(\mathbf{x})\Big], \tag{4.17}$$

so that by combining the operations of taking limits and computing derivatives/finite differences, we can compute any conditional probability of the form $F(\mathbf{x}_A|\mathbf{x}_B)$. To compute the conditional CDF for any variable node in the network, we can pass messages from leaf nodes to root and then from the root node back to the leaves. For any given variable node, we can then multiply all incoming messages to obtain the conditional CDF for that variable, up to a scaling factor. We will now demonstrate this principle using the previous toy example CDN.

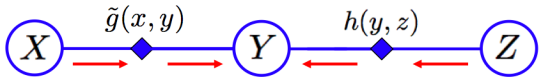


Figure 4.3: Flow of messages in the toy example CDN of Figure 4.2 with variable U marginalized out in order to compute the conditional CDF $F(y|x,z)$.

Example 4.2.1. Consider the toy example of a CDN over four random variables U, X, Y, Z from Figure 4.2. Suppose we wish to compute $F(y|x,z) = \lim\limits_{u\to\infty} F(u,y|x,z)$. This is equivalent to message-passing in a CDN defined over variables X, Y, Z with U marginalized out (Figure 4.3) so that $\tilde{g}(x,y) = \lim\limits_{u\to\infty} g(u,x,y)$. Thus the message updates are given by

$$\mu_{X\to\tilde{g}}(x) = 1,\ \mu_{\tilde{g}\to Y}(y;x) = \partial_x\Big[\tilde{g}(x,y)\mu_{X\to\tilde{g}}(x)\Big] = \partial_x\Big[\tilde{g}(x,y)\Big]$$
$$\mu_{Z\to h}(z) = 1,\ \mu_{h\to Y}(y;z) = \partial_z\Big[h(y,z)\mu_{Z\to h}(z)\Big] = \partial_z\Big[h(y,z)\Big]. \tag{4.18}$$

Once we have computed the above messages, we can evaluate the conditional CDF $F(y|x,z)$ at node Y as

$$F(y|x,z) = \frac{\mu_{\tilde{g}\to Y}(y;x)\mu_{h\to Y}(y;z)}{\mathcal{Z}} = \frac{\partial_z\Big[h(y,z)\Big]\partial_x\Big[\tilde{g}(x,y)\Big]}{\mathcal{Z}}. \tag{4.19}$$

Note that the normalizing constant $\mathcal{Z}$ can be readily obtained by computing

$$\mathcal{Z} = \lim_{y\to\infty} \partial_z\Big[h(y,z)\Big]\partial_x\Big[\tilde{g}(x,y)\Big] = \partial_{x,z}\Big[\lim_{y\to\infty} h(y,z)\tilde{g}(x,y)\Big], \tag{4.20}$$

so that

$$\begin{aligned} F(y|x,z) = \frac{\mu_{\tilde{g}\to Y}(y;x)\mu_{h\to Y}(y;z)}{\mathcal{Z}} &= \frac{\partial_z\Big[h(y,z)\Big]\partial_x\Big[\tilde{g}(x,y)\Big]}{\partial_{x,z}\Big[\lim_{y\to\infty} h(y,z)\tilde{g}(x,y)\Big]} = \frac{\lim_{u\to\infty}\partial_z\Big[h(y,z)\Big]\partial_x\Big[\tilde{g}(u,x,y)\Big]}{\partial_{x,z}\Big[\lim_{u,y\to\infty} h(y,z)\tilde{g}(u,x,y)\Big]} \\ &= \frac{\partial_{x,z}\Big[\lim_{u\to\infty} F(u,x,y,z)\Big]}{\partial_{x,z}\Big[\lim_{u,y\to\infty} F(u,x,y,z)\Big]}. \end{aligned} \tag{4.21}$$

The above example shows that the message-passing algorithm can be used to compute conditional CDFs of the form $F(\mathbf{x}_A|\mathbf{x}_B)$, up to a normalizing constant $\mathcal{Z}$. We can readily obtain distributions of the type and $P(\mathbf{x}_A|\mathbf{x}_B)$ from $F(\mathbf{x}_A|\mathbf{x}_B)$ by computing $\partial_{\mathbf{x}_A}\Big[F(\mathbf{x}_A|\mathbf{x}_B)\Big]$ using the above message-passing scheme and then multiplying messages together to obtain conditional CDFs. We note that computing the normalizing constant $\mathcal{Z}$ can be viewed as the result of message-passing in a CDN in which the variables $\mathbf{X}_A$ have been marginalized out in addition to variables $\mathbf{X}_{V\setminus A\cup B}$ and then evaluating the resulting messages at the observed values $\mathbf{x}_B$. Equivalently, one can compute $\mathcal{Z} = \lim_{\mathbf{x}_A\to\infty}\partial_{\mathbf{x}_B}\Big[F(\mathbf{x}_A,\mathbf{x}_B)\Big]$ after message-passing with only variables in $V\setminus(A\cup B)$ marginalized out.

4.3 Derivative-sum-product: A message-passing algorithm for inference in cumulative distribution networks

Equations (4.11) and (4.14) together define the set of messages that allow us to compute the higher order mixed derivative of the joint CDF. Because the fundamental operations required for message-passing consist of differentiation/finite differences, sums and products, we will refer to the corresponding class of algorithms as the derivative-sum-product (DSP) algorithm. For CDNs defined over discrete variables, the DSP algorithm is shown in Table 4.1. As can be seen, for graphs defined over discrete variables, the DSP algorithm is analogous to the sum-product algorithm with the summation operation replaced by a finite difference operation. While the DSP algorithm for discrete variable networks has the same order of complexity as the sum-product algorithm, the required complexity increases for CDNs defined over continuous variables. For such models, we are required to invoke the product rule of differential calculus in order to express these messages in terms of the derivatives of CDN functions and combinations thereof. To this end, we can define two additional sets of messages $\lambda_{\alpha\to s}(\mathbf{x})$ and $\lambda_{s\to\alpha}(\mathbf{x})$ that correspond to the quantities $\partial_{x_\alpha}\Big[\mu_{\alpha\to s}(\mathbf{x})\Big]$ and $\partial_{x_\alpha}\Big[\mu_{s\to\alpha}(\mathbf{x})\Big]$ respectively. We first derive the expression for $\lambda_{\alpha\to s}(\mathbf{x})$ by applying the product rule of differential calculus to Equation (4.14), bearing in mind that each of the messages $\mu_{s\to\alpha}(\mathbf{x}) \equiv \mu_{s\to\alpha}\Big(\mathbf{x}_{\tau_s^\alpha}\Big)$ depends on variable X_α. This yields

$$\lambda_{\alpha\to s}(\mathbf{x}) = \partial_{x_\alpha}\Big[\mu_{\alpha\to s}(\mathbf{x})\Big] = \partial_{x_\alpha}\left[\prod_{s'\in\mathcal{N}(\alpha)\setminus s}\mu_{s'\to\alpha}(\mathbf{x})\right] = \mu_{\alpha\to s}(\mathbf{x})\sum_{s'\in\mathcal{N}(\alpha)\setminus s}\frac{\lambda_{s'\to\alpha}(\mathbf{x})}{\mu_{s'\to\alpha}(\mathbf{x})}. \tag{4.23}$$

In order to derive the general expressions for $\mu_{\alpha\to s}(\mathbf{x})$, $\lambda_{\alpha\to s}(\mathbf{x})$, we first note that for any two differentiable multivariate functions $f(\mathbf{x}), g(\mathbf{x})$, the product rule for computing the higher order derivative of a product of functions is given by

$$\partial_{\mathbf{y}}\Big[f(\mathbf{y})g(\mathbf{y})\Big] = \sum_{\mathbf{y}_A\subseteq\mathbf{y}}\partial_{\mathbf{y}_A}\Big[f(\mathbf{y})\Big]\partial_{\mathbf{y}\setminus\mathbf{y}_A}\Big[g(\mathbf{y})\Big]. \tag{4.24}$$

In the context of computing messages $\mu_{s\to\alpha}(\mathbf{x}), \lambda_{s\to\alpha}(\mathbf{x})$ from Equation (4.11), applying the above product rule yields

$$\begin{aligned}\mu_{s\to\alpha}(\mathbf{x}) &= \partial_{\mathbf{x}_{\mathcal{N}(s)\setminus\alpha}}\left[\phi_s(x_\alpha, \mathbf{x}_{\mathcal{N}(s)\setminus\alpha}) \prod_{\beta\in\mathcal{N}(s)\setminus\alpha} \mu_{\beta\to s}(\mathbf{x})\right] \\ &= \sum_{B\subseteq\mathcal{N}(s)\setminus\alpha} \partial_{\mathbf{x}_B}\left[\phi_s(\mathbf{x}_s)\right] \prod_{\beta\in B} \mu_{\beta\to s}(\mathbf{x}) \prod_{\beta\in\mathcal{N}(s)\setminus\{\alpha\cup B\}} \lambda_{\beta\to s}(\mathbf{x}) \qquad (4.25)\\ \lambda_{s\to\alpha}(\mathbf{x}) &= \partial_{x_\alpha}\left[\mu_{s\to\alpha}(\mathbf{x})\right] \\ &= \sum_{B\subseteq\mathcal{N}(s)\setminus\alpha} \partial_{\mathbf{x}_B, x_\alpha}\left[\phi_s(\mathbf{x}_s)\right] \prod_{\beta\in B} \mu_{\beta\to s}(\mathbf{x}) \prod_{\beta\in\mathcal{N}(s)\setminus\{\alpha\cup B\}} \lambda_{\beta\to s}(\mathbf{x}), \qquad (4.26)\end{aligned}$$

where we have made use of the tree-structure of the CDN to write the derivative of a product of messages as a product of derivatives of the messages. The above updates then define the DSP algorithm for CDNs defined over continuous variables, with a total of four sets of messages defined solely in terms of the CDN functions, their derivatives and linear combinations thereof. The message-passing algorithm for continuous CDNs is summarized in Table 4.2 and is illustrated in Figure 4.4.

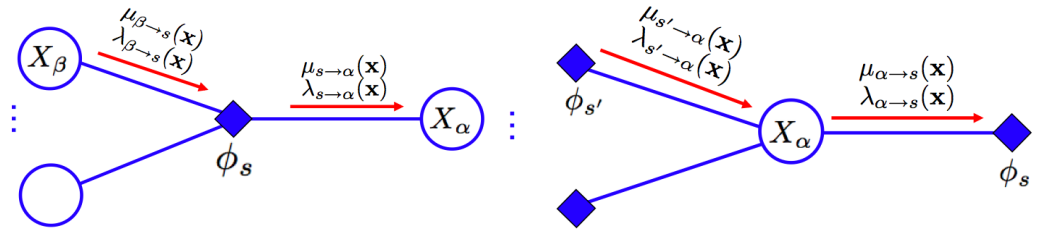


Figure 4.4: a) Computation of the message from a function node s to a variable node α; b) Computation of the message from a variable node α to a function node s.

We see from Table 4.2 that the DSP algorithm grows exponentially in complexity as the number of neighboring variable nodes for any given function increases, as the updates at function nodes require one to perform a sum over all subsets of neighboring variables. However, in many cases the computational complexity will be tractable for

sparser graphs, as demonstrated by the following example.

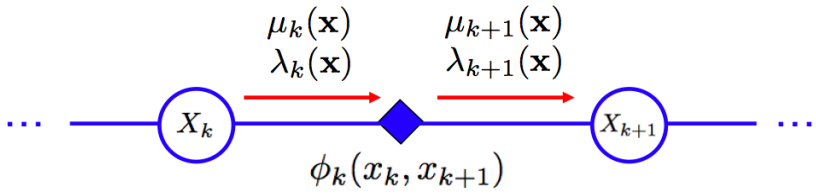


Figure 4.5: The DSP algorithm for a chain-structured CDN with continuous variables.

Example 4.3.1 (Derivative-sum-product on a linear first-order chain CDN)**.** Consider the CDN defined over K continuous variables such that the joint CDF over these variables is given by

$$F(\mathbf{x}) = \prod_{k=1}^{K-1} \phi_k(x_k, x_{k+1}), \tag{4.28}$$

so that the variable nodes are connected in the chain-structured graph shown in Figure 4.5. In this case, the DSP messages can be written simply as

$$\begin{aligned}
\mu_{k+1}(\mathbf{x}) &\equiv \mu_{\phi_k \to X_{k+1}}(\mathbf{x}) \\
&= \partial_{x_k}\Big[\phi_k(x_k, x_{k+1})\Big]\mu_k(\mathbf{x}) + \phi_k(x_k, x_{k+1})\lambda_k(\mathbf{x}), \quad k = 1, \cdots, K-1 \\
\lambda_{k+1}(\mathbf{x}) &\equiv \lambda_{\phi_k \to X_{k+1}}(\mathbf{x}) \\
&= \partial_{x_k, x_{k+1}}\Big[\phi_k(x_k, x_{k+1})\Big]\mu_k(\mathbf{x}) + \partial_{x_{k+1}}\Big[\phi_k(x_k, x_{k+1})\Big]\lambda_k(\mathbf{x}), \quad k = 1, \cdots, K-1.
\end{aligned} \tag{4.29}$$

Example 4.3.2 (Sampling from a cumulative distribution network)**.** We can further take advantage of the derivative-sum-product algorithm for generating samples from the CDF modeled by a CDN. We can proceed as follows: arbitrarily select a variable in the model, say X_1. Then, generate a sample x_1^* from its marginal CDF $F(x_1)$ (obtained by marginalizing out all other variables). Given x_1^*, we can then proceed to generate samples for its children by marginalizing out all other unobserved variables and then sampling from the conditional distribution $F(x_2|x_1^*)$. We can continue this way until we have sampled a complete configuration $\mathbf{x}^* = [x_1^*, \cdots, x_K^*]$. The algorithm for sampling from the joint CDF modeled by a CDN is then given by

- Pick a sampling ordering $X_1, X_2, \cdots, X_K$,
- For variable $X_k, k = 1, \cdots, K$, compute

$$F(x_1, \cdots, x_k) = \lim_{x_{k+1}, \cdots, x_K \to \infty} F(x_1, \cdots, x_k, x_{k+1} \cdots, x_K). \tag{4.30}$$

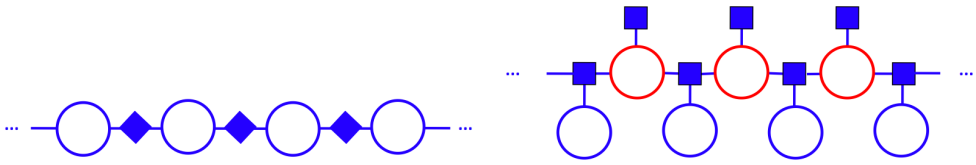
- Sample x_k^* from

$$F(x_k | x_1, \cdots, x_{k-1}) = \frac{\partial_{x_1, \cdots, x_{k-1}} \Big[F(x_1, \cdots, x_k) \Big]}{\lim_{x_k \to \infty} \partial_{x_1, \cdots, x_{k-1}} \Big[F(x_1, \cdots, x_k) \Big]}. \tag{4.31}$$

From the above we see that if the CDN has a tree structure, then we can compute the conditional CDFs $F(x_k|x_1, \cdots, x_{k-1})$ exactly via DSP. In the case of a CDN with cycles, we can always convert it to one with a tree structure by clustering variables and corresponding function nodes, as can be done in the case of factor graphs [41]. This generally incurs an increase in function node complexity, but with the benefit of being able to sample from the joint CDF defined by the CDN and not from an approximation thereof.

4.4 Complexity of inference in cumulative distribution networks and factor graphs

What is the relative computational complexity of message-passing in a CDN as compared to message-passing in a factor graph constructed using the procedure described in Theorem 3.4.1? When both the CDN and factor graph are defined over discrete variables, message-passing in either the CDN or graph will have similar computational complexity. This can be seen from the construction of the factor graph using the procedure outlined in Theorem 3.4.1, where we add a latent variable for each function node in the CDN in addition to a factor node for each variable in the CDN. However, for certain CDNs, the addition of latent variables in the factor graph will require additional computations



(a) (b)

Figure 4.6: Comparative cost of inference in a) a CDN and b) a factor graph using DSP and sum-product. The DSP algorithm here requires less floating point operations due to a simplified graphical model for the joint probability over observable variables (blue nodes), whereas sum-product requires additional computations due to the introduction of additional latent variables (red nodes).

in order to propagate all messages, whereas the CDN model for the joint probability will have fewer variables in the graph and so may simplify the number of computations needed. As a simple example of this, consider the following example of a CDN defined over discrete variables and its factor graph obtained from Theorem 3.4.1.

Example 4.4.1. Consider the CDN and the factor graph that are related according to Theorem 3.4.1 and are shown in Figure 4.6(a) and 4.6(b), where both models are defined over $|S|+1$ ordinal discrete random variables, where $|S|$ is the number of functions in the CDN graph. Suppose we are given a set of observed configurations for the model variables, which we will denote by $\mathbf{x} \in \mathcal{X}^{|V|}$ and suppose we then wish to compute all messages in the sum-product and DSP algorithms in the factor graph and CDN respectively. For ease of discussion, suppose that $\mathbf{x} > \mathbf{0}$ so that min-independence does not hold in the CDN and so the joint probabilities modeled by the two graphs satisfy identical sets of conditional independence relationships amongst model variables.

Now let $\mathbf{Y} = \{Y_s : s \in S\}$ denote the set of latent variables introduced in the factor graph according to Theorem 3.4.1 and let $\mathbf{Y}_\alpha = \{Y_s : s \in \mathcal{N}(\alpha)\}$. The joint probability

$P(\mathbf{x}, \mathbf{y})$ modeled by the factor graph can then be written as

$$P(\mathbf{x}, \mathbf{y}) = \prod_{\alpha \in V} \left(P(x_\alpha | \mathbf{y}_\alpha) \prod_{s \in \mathcal{N}(\alpha)} P(y_s) \right) = \prod_{\alpha \in V} f_\alpha(x_\alpha, \mathbf{y}_\alpha), \tag{4.32}$$

so as to not introduce additional factor nodes into the factor graph model for $P(\mathbf{x}, \mathbf{y})$.

Suppose the state space of all discrete variables in the model is L. In order to compute all messages in the CDN, one needs to perform L multiplications for each function node in order to multiply an incoming message with the function node and L subtractions per function node in order to compute the finite difference. When considering messages being passed in both directions, this yields a total of $2 \cdot |S| \cdot L$ multiplications and $2 \cdot |S| \cdot (L-1)$ additions, as finite differences require only $L - 1$ subtractions and a subtraction has the same computational cost as addition. However, in the case of the corresponding factor graph, if we assume that observable variable nodes pass the message $\mu_{\alpha \to s}(x_\alpha) = 1$, the total number of multiplications then increases to $4 \cdot (|S| + 1) \cdot L$ and the number of additions becomes $3 \cdot (|S| + 1) \cdot L$ in order to compute all messages in the graph. Thus by constructing a more compact model in the form of a CDN, one can reduce the number of additions and multiplications required in order to compute messages as compared to the amount of computation for the corresponding factor graph.

For graphs defined over continuous variables, the comparison becomes less straightforward, as computational complexity will be dominated both by graph connectivity and the complexity of differentiation/marginalization for the CDN and the corresponding factor graph. In the case of DSP, the computational complexity will be dominated by the number of neighboring variables for each function node in the CDN, as the DSP update at a function node requires one to sum over all subsets of neighboring variables. However, in the case of sparsely-connected tree-structured CDNs, it will be possible to compute $P(\mathbf{x})$ exactly with relatively low computational cost via DSP, whereas doing this in the corresponding factor graph may require either approximating the messages using methods such as expectation propagation [50] or the use of Markov Chain Monte Carlo sampling methods [53]. In general, the CDN will have the advantage of containing

only observable variables, whereas in a factor graph one will have to marginalize out additional latent variables, despite the fact that one has the freedom to select the latent variables to be either continuous or discrete with many possible numbers of states.

4.5 Discussion

We have presented the derivative-sum-product algorithm for computing derivatives in a CDN. We have shown that the algorithm is an analog of the sum-product algorithm in factor graphs, so that for tree-structured graphs the algorithm yields exact derivatives of the joint CDF. For graphs defined over continuous variables, the DSP algorithm can be implemented through two sets of messages in order to compute the higher order derivatives of the joint CDF. While we have presented the DSP algorithm for computing derivatives given a set of CDN functions, we have not addressed here the issue of how to learn these CDN functions from data. A possible method would be to run DSP to obtain the joint PDF and then maximize this with respect to model parameters. Another issue we have not addressed is how to perform inference in graphs with cycles: an interesting future direction would be to investigate exact or approximate methods for doing so and connections to methods in the literature [50, 53] for doing this in traditional graphical models. We will further discuss these issues in the concluding chapter.

Having defined the CDN and having described the DSP algorithm, we will now proceed to apply both of these to the general problem of learning to rank from examples. As we will see, the ability to model a joint CDF using a graphical framework will yield advantages in both representation and computation for this class of problems.

- For each leaf variable node α' and for all function nodes $s \in \mathcal{N}(\alpha')$, propagate $\mu_{\alpha' \to s}(\mathbf{x}) = 1$. For each leaf function node with function $\phi_s(x_{\alpha'})$, send the messages $\mu_{s \to \alpha'}(\mathbf{x}) = \phi_s(x_{\alpha'})$.

- (Messages from variables to functions) For each non-leaf variable node α and neighboring function nodes $s \in \mathcal{N}(\alpha)$,

$$\mu_{\alpha \to s}(\mathbf{x}) = \prod_{s' \in \mathcal{N}(\alpha) \setminus s} \mu_{s' \to \alpha}(\mathbf{x}).$$

- (Messages from functions to variables) For each non-leaf function node s and neighboring variable nodes $\alpha \in \mathcal{N}(s)$,

$$\mu_{s \to \alpha}(\mathbf{x}) = \partial_{\mathbf{x}_{\mathcal{N}(s) \setminus \alpha}} \left[\phi_s(\mathbf{x}_s) \prod_{\beta \in \mathcal{N}(s) \setminus \alpha} \mu_{\beta \to s}(\mathbf{x}) \right].$$

- For root node $\alpha \in V$, pass messages from α to leaf nodes according to above.

- For each node $\alpha \in V$, compute the conditional CDF $F(x_\alpha | \mathbf{x}_{V \setminus \alpha})$ as

$$F(x_\alpha | \mathbf{x}_{V \setminus \alpha}) \propto \prod_{s \in \mathcal{N}(\alpha)} \mu_{s \to \alpha}(\mathbf{x}) \tag{4.22}$$

Table 4.1: The derivative-sum-product (DSP) algorithm for inference in a CDN defined over discrete variables.

- For each leaf variable node α' and for all function nodes $s \in \mathcal{N}(\alpha')$, propagate $\mu_{\alpha' \to s}(\mathbf{x}) = \lambda_{\alpha' \to s}(\mathbf{x}) = 1$. For each leaf function node with function $\phi_s(x_{\alpha'})$, send the messages $\mu_{s \to \alpha'}(\mathbf{x}) = \phi_s(x_{\alpha'}), \lambda_{s \to \alpha'}(\mathbf{x}) = \partial_{x_{\alpha'}}\Big[\phi_s(x_{\alpha'})\Big]$.

- (Messages from variables to functions) For each non-leaf variable node α and neighboring function nodes $s \in \mathcal{N}(\alpha)$,

$$\mu_{\alpha \to s}(\mathbf{x}) = \prod_{s' \in \mathcal{N}(\alpha) \setminus s} \mu_{s' \to \alpha}(\mathbf{x}),$$
$$\lambda_{\alpha \to s}(\mathbf{x}) = \partial_{x_\alpha}\Big[\mu_{\alpha \to s}(\mathbf{x})\Big] = \mu_{\alpha \to s}(\mathbf{x}) \sum_{s' \in \mathcal{N}(\alpha) \setminus s} \frac{\lambda_{s' \to \alpha}(\mathbf{x})}{\mu_{s' \to \alpha}(\mathbf{x})}.$$

- (Messages from functions to variables) For each non-leaf function node s and neighboring variable nodes $\alpha \in \mathcal{N}(s)$,

$$\mu_{s \to \alpha}(\mathbf{x}) = \sum_{B \subseteq \mathcal{N}(s) \setminus \alpha} \partial_{\mathbf{x}_B}\Big[\phi_s(\mathbf{x}_s)\Big] \prod_{\beta \in B} \mu_{\beta \to s}(\mathbf{x}) \prod_{\beta \in \mathcal{N}(s) \setminus \{\alpha \cup B\}} \lambda_{\beta \to s}(\mathbf{x}),$$
$$\lambda_{s \to \alpha}(\mathbf{x}) = \partial_{x_\alpha}\Big[\mu_{s \to \alpha}(\mathbf{x})\Big]$$
$$= \sum_{B \subseteq \mathcal{N}(s) \setminus \alpha} \partial_{\mathbf{x}_B, x_\alpha}\Big[\phi_s(\mathbf{x}_s)\Big] \prod_{\beta \in B} \mu_{\beta \to s}(\mathbf{x}) \prod_{\beta \in \mathcal{N}(s) \setminus \{\alpha \cup B\}} \lambda_{\beta \to s}(\mathbf{x}).$$

- For root node $\alpha \in V$, pass messages from α to leaf nodes according to above.

- For each node $\alpha \in V$, compute the conditional CDF $F(x_\alpha | \mathbf{x}_{V \setminus \alpha})$ as

$$F(x_\alpha | \mathbf{x}_{V \setminus \alpha}) \propto \prod_{s \in \mathcal{N}(\alpha)} \mu_{s \to \alpha}(\mathbf{x}) \tag{4.27}$$

Table 4.2: The derivative-sum-product (DSP) algorithm for inference in a CDN defined over continuous variables.

Chapter 5

Learning to rank with cumulative distribution networks

In this chapter, we will apply CDNs and the DSP algorithm to problems of structured ranking learning in which the goal is to learn a model for ranking objects from many examples of orderings. In such applications, the CDF arises as a probability measure defined over multiple inequality events of the form $X \leq x$ with various statistical dependence relationships between events. The problem of *structured ranking learning* therefore consists of constructing a ranking model using training data whereby we wish to account for the structure, or the presence of statistical dependence relationships, between model variables.

Here we will demonstrate the advantage of modelling a joint CDF using the graphical modelling framework provided by CDNs, whereby we will be able to A) simplify the modelling of large CDFs, B) model both the notion of ordering and statistical dependence relationships between variables in the model and C) perform computations for inference and estimation in a tractable fashion. Before we proceed, we will first review some additional concepts and techniques that will be used throughout the chapter.

5.1 Background

5.1.1 Ordinal regression

In many domains, one is faced with the problem of estimating multinomial variables that can each take one of a finite number of values in some discrete set $\mathcal{X} = \{r_1, \cdots, r_K\}$ for some integer K. Such multinomial variables can then be distinguished as being of the type

- *Nominal*, or *categorical*, so that the set $\mathcal{X}$ does not admit an ordering of variable values.

- *Ordinal*, so that the set $\mathcal{X}$ admits a total ordering over variable values of the type $r_1 \prec \cdots \prec r_K$.

An example of a nominal variable is music type, such as $\mathcal{X} = \{rock, electronica, jazz\}$ and an example of an ordinal variable is a grading scheme $\mathcal{X} = \{A, B, C, D\}$ so that the possible variable values satisfy the total ordering $D \prec C \prec B \prec A$.

In ordinal regression, the goal is to predict a discrete variable $y \in \{r_1, \cdots, r_K\}$ given a set of features $\mathbf{x}$, where $r_1 \prec \cdots \prec r_K$ are an ordered set of labels. Unlike the general problem of multiclass classification in which variables to be predicted are nominal, output labels in the setting of ordinal regression are not permutation-invariant and so any model for the problem should account for the orderings of the output variable values.

One model for performing ordinal regression is the *cumulative model* [48], which relates an input vector $\mathbf{x}$ to an ordinal output y via a function f and a set of *cutpoints* $\theta(r_1) < \cdots < \theta(r_K)$ along the real line $\mathbb{R}$ so that $y = r_k$ if $\theta(r_{k-1}) < f(\mathbf{x}) + \epsilon \leq \theta(r_k)$, where ϵ is additive noise and we define $\theta(r_0) = -\infty, \theta(r_K) = \infty$ (Figure 5.1). If $P(\epsilon)$ is the

probability density function from which the noise variable ϵ is drawn, then we can write

$$\begin{aligned}\mathbb{P}\Big[y = r_k\Big] &= \mathbb{P}\Big[\theta(r_{k-1}) < f(\mathbf{x}) + \epsilon \leq \theta(r_k)\Big] \\ &= \mathbb{P}\Big[\{\theta(r_{k-1}) - f(\mathbf{x}) < \epsilon\} \cap \{\epsilon \leq \theta(r_k) - f(\mathbf{x})\}\Big] \\ &= F_\epsilon(\theta(r_{k-1}) - f(\mathbf{x})) - F_\epsilon(\theta(r_k) - f(\mathbf{x})), \end{aligned} \tag{5.1}$$

where $F_\epsilon \equiv F(\epsilon)$ is the corresponding cumulative distribution function for $P(\epsilon)$. The above equation defines a likelihood function for a given observed pair $(\mathbf{x}, y)$, so that the cutpoints $\theta(r_k)$ and the regression function $f(\mathbf{x})$ can subsequently be estimated from training data by maximizing the likelihood function with respect to the cutpoints $\theta(r_k)$ and the regression function $f(\mathbf{x})$.

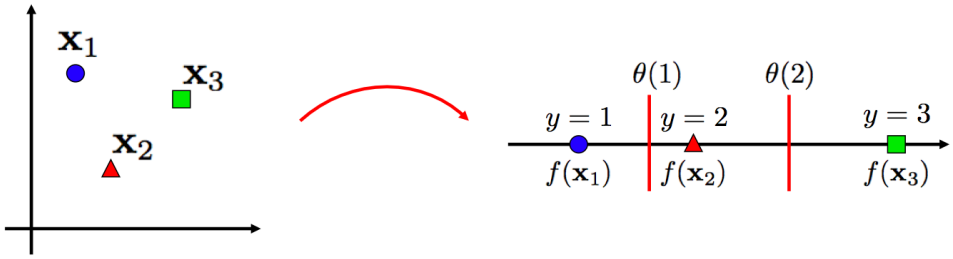


Figure 5.1: An illustration of the ordinal regression model. A given point has label $y = r_k$ if $\theta(r_{k-1}) < f(\mathbf{x}) + \epsilon \leq \theta(r_k)$, where ϵ is a noise variable.

5.1.2 Nadaraya-Watson estimators

Suppose we are given some finite sample $\mathcal{D}$ of pairs $(\mathbf{x}_n, y_n)$ for $n = 1, \cdots, N$, where $y_n \in \mathbb{R}$ is related to $\mathbf{x}_n \in \mathbb{R}^L$ via

$$y_n = f(\mathbf{x}_n) + \epsilon_n \tag{5.2}$$

and $\mathbb{E}[\epsilon_n] = 0$. A popular method for estimating $f(\mathbf{x})$ is the Nadaraya-Watson estimator, which we define below.

Definition 5.1.1 (Nadaraya-Watson estimator)**.** The Nadaraya-Watson estimator [52, 67] is the following function $\hat{f}(\mathbf{x})$ obtained from samples $(\mathbf{x}_n, y_n)$:

$$\hat{f}(\mathbf{x}) = \frac{\sum_{n=1}^{N} K(\mathbf{x}, \mathbf{x}_n) y_n}{\sum_{n=1}^{N} K(\mathbf{x}, \mathbf{x}_n)}, \tag{5.3}$$

where K is any kernel function.

Thus, the estimate of the function value $f(\mathbf{x})$ at $\mathbf{x}$ consists of a weighted sum of sample values y_n, where the weights are given by the kernel function $K(\mathbf{x}, \mathbf{x}_n)$. The kernel function can be chosen from a broad class of kernels: in this thesis we will focus primarily on Gaussian kernels of the form

$$K(\mathbf{u}, \mathbf{v}; \mathbf{A}) = \exp\Big(-\frac{1}{2}(\mathbf{u} - \mathbf{v})^T \mathbf{A} (\mathbf{u} - \mathbf{v}) \Big) \tag{5.4}$$

for some *bandwidth matrix* $\mathbf{A}$. In the simple case where $\mathbf{A} = \mathrm{diag}(\mathrm{a}_1, \cdots, \mathrm{a}_\mathrm{L})$ is diagonal with $a_l > 0,\ l = 1, \cdots, L$, we have

$$K(\mathbf{u}, \mathbf{v}; \mathbf{A}) = \prod_{l=1}^{L} K(u_l, v_l; a_l) = \exp\Big(-\frac{1}{2} \sum_l a_l (u_l - v_l)^2 \Big), \tag{5.5}$$

where a_l are bandwidth parameters that determine the smoothness of the resulting estimator. An example of such an estimator for discrete labels y_n with a Gaussian kernel for one-dimensional inputs x_n is shown in Figure 5.2, where the Gaussian kernel is defined by a single bandwidth parameter a. For a more thorough discussion of the properties of such estimators, we refer the reader to [52, 67].

5.1.3 Gradient-based methods for learning

In statistical machine learning, the problem of learning a parametric model from data $\mathcal{D}$ is usually formulated as a problem of optimizing some loss functional $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ of the model with respect to some parameter set $\boldsymbol{\theta}$ under some possible constraints on $\boldsymbol{\theta}$. Given a finite sample $\mathcal{D} = \{D_1, \cdots, D_N\}$ and under the assumption that samples are drawn

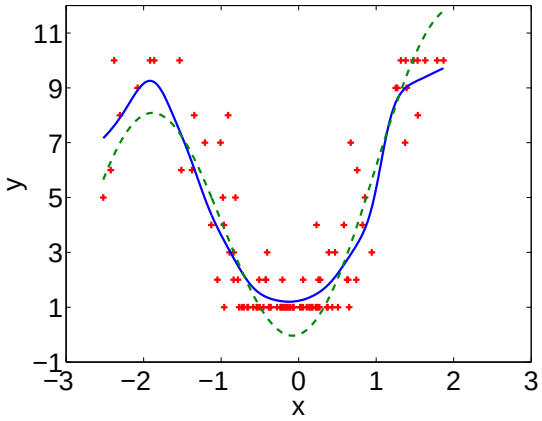


Figure 5.2: Synthetic data generated from an ordinal regression model whereby $y_n = k$ if $\theta(k-1) < f(x_n) + 2\epsilon \leq \theta(k)$ for $k = 1, \cdots, 10$ and $\theta(k) = k$ for $k = 1, \cdots, 9$ and $\theta(0) = -\infty, \theta(10) = \infty$, where the function $f(x) = 10\sin^2(0.8x) + 10\sin(0.1x)$ and ϵ is a Gaussian random variable with zero mean and unit variance for $n = 100$. The Nadaraya-Watson estimator $\hat{f}(x)$ is shown in blue, with a single bandwidth parameter a selected by cross-validation using squared loss $\mathcal{L}(a) = \sum_n (y_n - \hat{f}(x_n; a))^2$. The true function $f(x)$ is shown as the black dotted line.

independently from some probability distribution, the loss functional can often be expressed as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \equiv \sum_{n=1}^{N} \mathcal{L}(\boldsymbol{\theta}; D_n), \tag{5.6}$$

where $\mathcal{L}(\boldsymbol{\theta}; D_n)$ is the loss incurred on data sample D_n. Assuming that a (sub-)gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ of the loss functional $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ can be computed and evaluated at each data sample D_n, the problem of learning can then be solved by iterative gradient descent methods in which we begin with an initial estimate for $\boldsymbol{\theta}$ and then iteratively update $\boldsymbol{\theta}$ using the (sub-)gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$. We will now proceed to describe two classes of such learning algorithms that are often used: the reader is encouraged to refer to [6] for a more detailed discussion.

Batch gradient descent methods

In *batch* gradient descent methods, one iteratively updates the parameter vector $\boldsymbol{\theta}_t$ at iteration t of the algorithm for $t = 1, \cdots, T$ using the rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; D_n), \tag{5.7}$$

where the *learning rate* or *step size* $\eta > 0$ determines the amount by which $\boldsymbol{\theta}$ is updated. Batch gradient descent methods have been widely studied and have been shown to converge to some local optimum of the loss functional provided that η is small enough. Note that each update above requires us to compute the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; D_n)$ for each data sample D_n and then sum each of these gradients over the entire dataset $\mathcal{D}$ in order to perform one update of the parameter vector $\boldsymbol{\theta}$. This requires storing all data samples at runtime, which may become impractical for large datasets. A fast and efficient alternative is to perform updates using only one data point at a time, as we will now show.

Stochastic gradient descent methods

In the stochastic gradient descent method, one iteratively updates the parameter vector $\boldsymbol{\theta}_n$ using the rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; D_n),\ n = 1, \cdots, N,\ t = 1, \cdots, N. \tag{5.8}$$

We can also repeat the above rule for several *epochs* or passes through the training data samples. Thus, instead of summing over the entire training set $\mathcal{D}$ and then performing an update, one cycles through the data samples one by one and computes the gradient for any given sample in order to performs a single update. In contrast to the batch method for gradient descent, this does not require us to store the entire training set in memory, allowing for a smaller memory footprint and fast updates that depend only on individual data samples. Two further advantages of this approach are that the stochastic nature of the updates may allow us to avoid local minima of the function being optimized, and that

the algorithm is able to find optima faster than the batch method if there is sufficient redundancy in the data. For example, if one had a dataset consisting of M replications of a smaller dataset, the batch method would require M times more computations to find the optima, whereas the stochastic gradient descent method may require far fewer computations to reach the optima. However, depending on the size of the training set, several passes through the training data may be required.

We will now proceed to apply the framework of CDNs to the problem of ranking by considering the problem of learning to rank in multiplayer team-based gaming.

5.2 Application: Learning to rank in multiplayer team-based games

In this section, we will consider the problem of learning to rank in multiplayer team-based games, where one observes the scores achieved by several interacting players over many games $t = 1 \cdots, T$ in which players interactively compete for higher ranks in groups, or teams, that can change with each game. For any given game, players compete in teams so that at the end of each game, each player will have achieved a score as a result of actions taken by all players during the game. For example, these player scores could correspond to the number of targets destroyed or the number of flags stolen depending on the game objectives, so that a higher player score reflects a better performance for that player and for the given game objectives. Here we will define a game Γ_t as a triplet $(\mathcal{P}_t, \mathcal{T}_t, \mathcal{O}_t)$, where $\mathcal{P}_t \subset \mathcal{P}$ is a subset of the set $\mathcal{P}$ of all players and $\mathcal{T}_t$ is a partition of $\mathcal{P}_t$ into sets corresponding to teams for game Γ_t, so that if $\mathcal{T}_t = \{\mathcal{T}_t^1, \cdots, \mathcal{T}_t^N\}$ then there are N teams for game Γ_t and player $k \in \mathcal{P}_t$ is assigned to team n for game Γ_t if and only if $k \in \mathcal{T}_t^n$. For example, a game involving six players labelled $1, 2, 3, 4, 5, 6$ organized into three teams of two players each could correspond to $\mathcal{P}_t = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{T}_t = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$. Without loss of generality we will label the teams in a

game by $n = 1, \cdots, N$ where each team corresponds to a set in the partition $\mathcal{T}_t$.

In addition to the above, we will denote by $\mathcal{O}_t$ the *outcome* of a game that consists of the pair $(\mathbf{x}_{\mathcal{P}_t}, \mathbf{r}_{\mathcal{T}_t})$, where $\mathbf{x}_{\mathcal{P}_t} \in \mathbb{R}^{|\mathcal{P}_t|}$ is a vector of player scores for game Γ_t and the set $\mathbf{r}_{\mathcal{T}_t}$ is defined as an ordered set of team performances, or set of ranks for each team. Such ranks are obtained by first computing the sum of the player scores for each team $n = 1, \cdots, N$, and then ranking the teams by sorting the resulting sums. We will refer to these sums in the sequel as the team scores t_n. An example of this for the previous example of a game with six players assigned to three teams is $\mathbf{x}_{\mathcal{P}_t} = [30\ 12\ 15\ 25\ 100\ 23]^T$, so that $\mathbf{r}_{\mathcal{T}_t} = \{2,\ 1,\ 3\}$ is an ordered set of team rankings. We will also denote by $\mathbf{x}_n \in \mathbb{R}^{|\mathcal{T}_t^n|}$ as the vector of player scores for team n in game Γ_t. We note at this juncture that the above definition for a game outcome consists only of player scores and team performances, although one could extend the definition outcome to include other player-specific features such as stamina and speed. Furthermore, the team rankings are a function of unweighted sums of player scores: although there is no reason *a priori* to weight the scores of players differently for determining the rank of a team, one could extend the above scheme for determining team rankings to weight player scores according to player type or player-specific features.

Given the above, the goal is to construct a model that will allow us to predict the outcome $\mathcal{O}_t$ of the new game before it begins, given $\mathcal{P}_t$ and previous game outcomes $\mathcal{O}_1, \cdots, \mathcal{O}_{t-1}$. In particular, we wish to construct a model that will minimize the number of mis-ordered teams according to the set of team performances $\mathbf{r}_{\mathcal{T}_t}$ for game Γ_t. Here, the probability model for the given game should account for the team-based structure of games, so that the team performances for any game are determined by individual player scores in that game and a game outcome is determined by the ordering of team scores and not simply an ordering of individual player scores. We will demonstrate here that the graphical framework of CDNs makes it straightforward to model both the notion of ordering of variables in the model as well as statistical independence relationships

amongst these variables. In particular, the model we will construct here will be amenable to exact inference via the DSP algorithm.

5.2.1 Previous work

Recently, [23] presented the TrueSkill$^{\text{TM}}$ algorithm for skill rating in Halo 2$^{\text{TM}}$, whereby each player $k \in \mathcal{P}_t$ is assigned a probability distribution over latent skill variables s_k, which is then inferred from individual player scores over multiple games using the expectation propagation algorithm for approximate inference [50]. Learning of the TrueSkill$^{\text{TM}}$ model thus consists of applying expectation propagation to a factor graph for a given game in order to update probabilities over player skills. An example of such a factor graph is shown in Figure 5.3. In TrueSkill$^{\text{TM}}$, the factors connecting team-specific nodes to one another dictate a constraint on relative differences in the total player scores between teams, while factors connecting player nodes to their team-specific nodes enforce the constraint that the team score is determined by the sum of player scores. Finally, for teams $n, n+1$, there is a difference variable $H_{n,n+1}$ and a corresponding factor which declares a tied rank between two teams if the difference between the two team scores is below some threshold parameter. Having described the TrueSkill model, we will now proceed to describe an alternate model formulated using the framework of CDNs.

5.2.2 A cumulative distribution network for modelling multiplayer game outcomes

Here we will examine a model for multiplayer game outcomes that will be modeled using a CDN. The model will be designed on a game-by-game basis in which the team assignments of players for a given game determines the connectivity of the graph $\mathcal{G}$ for the CDN. In contrast to this formulation, in our model the team variables will correspond to the ranks of teams: we will call such variables *team performances* and denote these

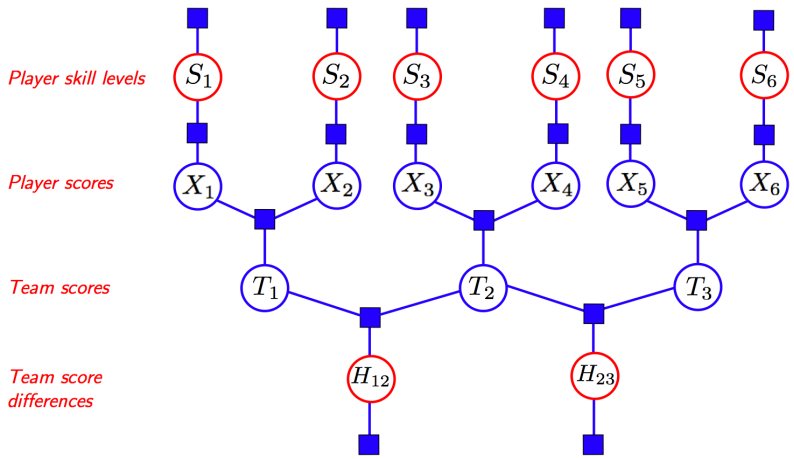


Figure 5.3: The TrueSkill$^{\text{TM}}$ factor graph for a particular Halo 2$^{\text{TM}}$ game involving three teams with two players each with the team scores $T_1 = t_1, T_2 = t_2, T_3 = t_3$ with $t_1 < t_2 < t_3$ so that team 3 here achieved the highest total of player scores. The variables H_{12}, H_{23} correspond to differences in team scores which determine the ranking of teams, so that teams n and $n+1$ are tied in their rankings if the difference in their team scores is below a threshold parameter. Here, $\mathcal{P}_t = \{1,2,3,4,5,6\}$ and $\mathcal{T}_t = \{\{1,2\},\{3,4\},\{5,6\}\}$. Latent variables correspond to nodes in red and observed variables correspond to nodes in blue. Each player $k = 1,2,3,4,5,6$ is assigned a skill function that reflects the distribution of that player's skill level S_k given past game outcomes. Each player then achieves score X_k in any given game and team scores $T_n, n = 1,2,3$ are then determined as the sum of player scores for each team.

as R_n for team n in order to contrast these with the team score variables T_n in the TrueSkill model. Our model will account for player scores X_k for each player $k \in \mathcal{P}_t$ in the game, the team performances R_n for each team $n = 1, \cdots, N$ in the game and each player's skill S_k, which is a random variable that influences that player's score for the given game. For any given game, R_n will be determined as the sum of the player scores for team n, and then sorting the resulting sums so that R_n corresponds to the rank of team n. The set of team performances $\mathbf{r}_{\mathcal{T}_t}$ will be given by the joint configuration of the R_n variables for that game. The goal will then be to learn a distribution over player skills S_k given previous game outcomes. We will design our model according to two principles.

First, the relationship between player scores and team performances is modeled as being stochastic, as player scores vary from one game to the next and team assignments change from one game to the next, so that given knowledge of the players in that game and their team assignments, there is some uncertainty in how a team will rank once the game is over. Second, team performance variables depend on those of other teams in the game, so that each team's performance should be linked to that of other teams in a game.

The CDN framework allows us to satisfy both desiderata in the form of modelling constraints on the marginal CDFs for variables in the model. To address the first point, we will require a set of CDN functions that connect player scores to team performances. Here we will make use of the cumulative model presented in Section 5.1.1 that relates a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ on inputs $\mathbf{x}$ to a single output ordinal variable $y \in \{r_1, \cdots, r_L\}$ so that $\mathbb{P}[y = r_l] = \mathbb{P}[\theta(r_{l-1}) < f(\mathbf{x}) + \epsilon \leq \theta(r_l)] = F_\epsilon(\theta(r_l) - f(\mathbf{x})) - F_\epsilon(\theta(r_{l-1}) - f(\mathbf{x}))$, where ϵ is an additive noise variable and $\theta(r_0), \cdots, \theta(r_L)$ are the cutpoint parameters of the model. Equivalently, we can write $\mathbb{P}[y \leq r_l] = \mathbb{P}[\epsilon \leq \theta(r_l) - f(\mathbf{x})]$. In the context of a multiplayer game, the set of inputs $\mathbf{x}$ will consist of a set of player scores for a given game. Thus, we learn a set of cutpoints $\theta(r_0) < \cdots < \theta(r_L)$ once using all of the games in the training data set. The above regression on team performances treats team performances as being independent: thus, we can use the CDN framework to augment the above parametric model in order to account for statistical dependencies between *multiple* team performances in any given game.

We will model multiplayer games using a CDN in which players are grouped into teams and teams compete with one another. If there are N teams for any given game, then we can assign a CDN function g_n for each team such that

$$g_n(\mathbf{x}_n, r_n) = \int_{-\infty}^{\mathbf{x}_n} F\Big(\theta(r_n); \mathbf{1}^T\mathbf{u}, \sigma^2\Big) P\Big(\mathbf{u}\Big)\, d\mathbf{u}, \tag{5.9}$$

where $F\Big(\theta(r_n); \mathbf{1}^T\mathbf{u}, \sigma^2\Big)$ is a cumulative model relating input player scores to output team performance, $\mathbf{1}$ is a vector of ones so that the regression function in the cumulative model is given by $f(\mathbf{x}) = \mathbf{1}^T\mathbf{x}$ and $\mathbf{x}_n, r_n$ are the player scores and team performance for team

n. Furthermore, $\theta(r_n)$ are "cutpoints" that define contiguous intervals in which r_n is the ranking for team n based on that team's performance and $P(\mathbf{u})$ is a probability density over the vector of player scores $\mathbf{u}$. In order to set these cutpoints, we solve the ordinal regression problem in which for any given game and for a given team n, $f(\mathbf{x}_n) = \mathbf{1}^T\mathbf{x}_n$ and the output variable is the team performance $r_n \in \{r_1, \cdots, r_L\}$. We then solve the ordinal regression problem using all of the teams in all games in the training data set in order to obtain cutpoints $\theta(r_0), \cdots, \theta(r_L)$, where we treat each team's ranking r_n as being mutually independent for the purpose of estimating the cutpoints. Once the cutpoints have been estimated, we will model the distributions $F\Big(\theta(r_n); \mathbf{1}^T\mathbf{u}, \sigma^2\Big), P\Big(\mathbf{u}\Big)$ in Equation (5.9) as

$$F\Big(\theta(r_n); \mathbf{1}^T\mathbf{u}, \sigma^2\Big) = \Phi\Big(\theta(r_n); \mathbf{1}^T\mathbf{u}, \sigma^2\Big), \tag{5.10}$$

$$P(\mathbf{u}) = Gaussian(\mathbf{u}; \mu\mathbf{1}, \sigma^2\mathbf{I}). \tag{5.11}$$

To address the fact that teams compete in any given game, we model ordinal relationships between team performance using the notion of stochastic orderings (Section 2.2), so that for two teams with team performances R_X, R_Y, $R_X \preceq R_Y$ if $F_{R_X}(t) \geq F_{R_Y}(t) \,\forall\, t \in \mathbb{R}$, where $F_{R_X}(\cdot), F_{R_Y}(\cdot)$ are the marginal CDFs of R_X, R_Y. This then allows us to design models in which we can express differences in team performances in the form of pairwise constraints on their marginal CDFs. We note at this juncture that while it is possible to model such stochastic ordering constraints between variables using directed, undirected or factor graphs, doing so introduces additional constraints that are likely to increase the difficulty of performing inference under such models. In contrast, the CDN framework here allows us to explicitly specify such stochastic ordering constraints, in addition to allowing for tractable computations in the resulting model. In particular, although the R_n variables are a deterministic function of the sum of player scores, it is easy to specify orderings amongst the R_n variables whilst modelling them as being stochastic using the framework of CDNs. By contrast, it will generally be more difficult in terms

of computation and representation to enforce constraints of the type $\left[r_n \leq r_{n+1}\right]$ for $R_n = r_n, R_{n+1} = r_{n+1}$ in a directed/undirected/factor graph model.

For the proposed CDN model, given N ranked teams, we can thus define $N-1$ functions $h_{n,n+1}$ so that

$$h_{n,n+1}(r_n, r_{n+1}) = \Phi\left(\left[\begin{smallmatrix} r_n \\ r_{n+1} \end{smallmatrix} \right]; \left[\begin{smallmatrix} \tilde{r}_n \\ \tilde{r}_{n+1} \end{smallmatrix} \right], \mathbf{\Sigma} \right), \tag{5.12}$$

where

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma^2 & \rho\sigma^2 \\ \rho\sigma^2 & \sigma^2 \end{bmatrix}$$

and $\tilde{r}_n \leq \tilde{r}_{n+1}$ so as to enforce $R_n \preceq R_{n+1}$ in the overall model. While the use of stochastic ordering constraints is admittedly weaker here than the use of deterministic constraints such as $r_n \leq r_{n+1}$, the use of stochastic constraints here simplifies model specification. Finally, we will use a *skill function* $s_k(x_k)$ for each player k to model that player's distribution over game scores given previous game outcomes. The player performance nodes in the CDN will then be connected to the team performance nodes via the above CDN functions g_n and team performance variable nodes R_n are linked to one another via the above CDN functions $h_{n,n+1}$.

The above functions and model variables jointly define the CDN for modelling multi-player games. An example is given in Figure 5.4(a) for a game with three teams and six players. One can readily verify from the CDN of Figure 5.4(a) that for the above model and for any given game, the stochastic ordering relationship $R_1 \preceq R_2 \preceq \cdots \preceq R_N$ as defined above is enforced by marginalizing over all player scores in the CDN.

It is instructive to convert the CDN to both a factor graph and a Bayesian network in order to understand the assumptions being made by the proposed CDN model. If we apply the representation theorem from Chapter 3 (Theorem 3.4.1) to the proposed CDN by introducing latent variable nodes $S_k, k \in \mathcal{P}_t$, $G_n, n = 1, \cdots, N$ and $H_{n,n+1}, n = 1, \cdots, N-1$ for each CDN function in the model, we obtain the factor graph shown in Figure 5.4(b) and the directed graphical model in Figure 5.4(c). The corresponding factor graph representation of the CDN provides us with some insight about

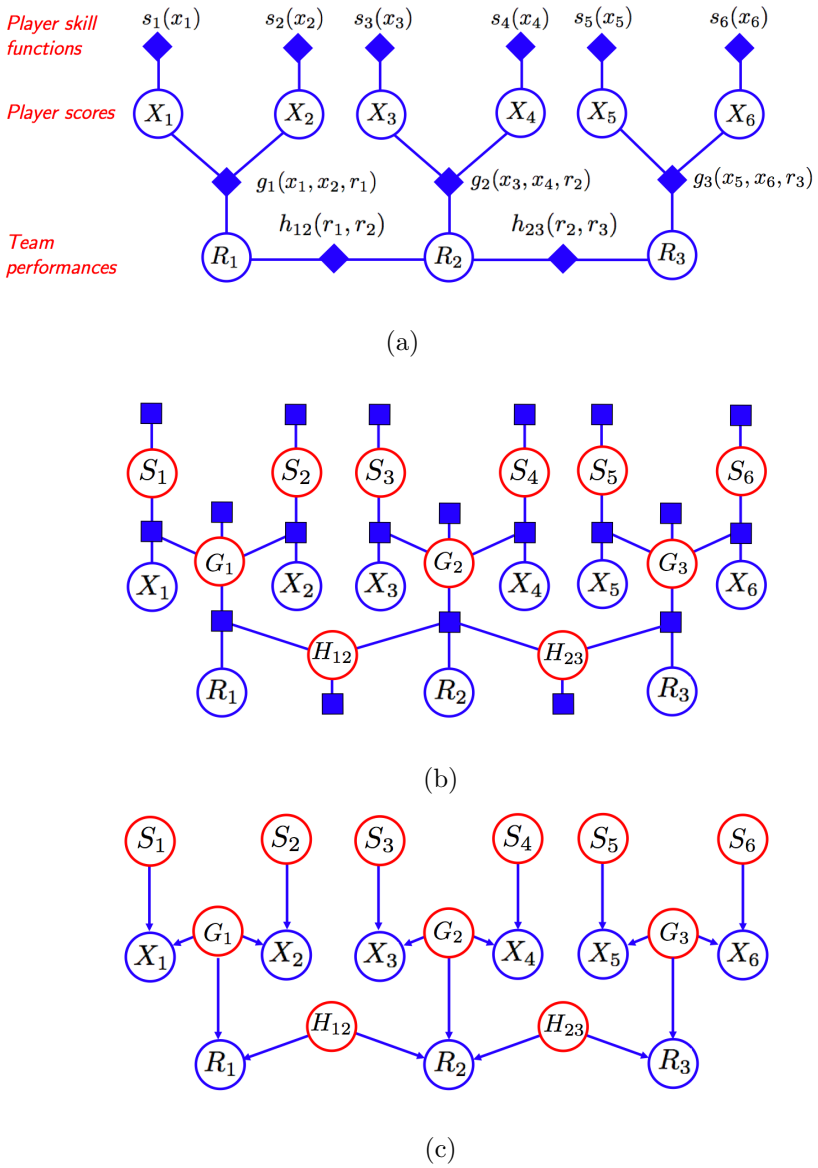


Figure 5.4: Graphical models for the player and team performances in a game of Halo 2^{TM} for three teams with two players each. Latent variables correspond to nodes in red and observed variables correspond to nodes in blue. Each player $k = 1, 2, 3, 4, 5, 6$ achieves score X_k in a match and team performances $R_n, n = 1, 2, 3$ are determined as the sum of player performances for each team. a) A model for the Halo 2^{TM} game represented as a CDN; b) The corresponding factor graph for the CDN in a) with latent variable nodes introduced according to the representation theorem; c) The corresponding directed graphical model.

the assumptions being made using the CDN model. For example, in the factor graph, each S_k variable corresponds to a variable for player k's unobserved skill level. The $H_{n,n+1}$ variables can be interpreted as *pairwise draw margin variables*, or the expected pairwise difference in team performances. The G_n variables can be interpreted as team-specific performance modifiers that impacts both team and player-specific performances. Comparing to the TrueSkill$^{\text{TM}}$ factor graph of [23] shown in Figure 5.3, we see that the CDN models the generative process describing player and team performances differently, though the the S_k and $H_{n,n+1}$ variable nodes are present in both factor graphs and model similar relationships between observable variables in the model. We can further glean insights about independence assumptions being made by the CDN by viewing the directed graphical model corresponding to the factor graph obtained above (Figure 5.4(c)). In particular, we can see that graph separation in the CDN do not imply conditional independence of player skill nodes, whilst in the TrueSkill$^{\text{TM}}$ factor graph, player skills are separated from one another by all observed variable nodes and so are conditionally *independent* given all observed variables.

Having presented the CDN for modelling multiplayer games, we will now proceed to describe a method for predicting game outcomes in which we update player skill functions after each game using message-passing.

5.2.3 Ranking players in multiplayer games using the derivative-sum-product algorithm

Here we will apply the DSP algorithm from Chapter 4 in the context of ranking players in multiplayer games with a team structure, where the problem consists of jointly predicting multiple ordinal output variables. It should be noted that while it may be possible to construct similar models using a directed, undirected or factor graph, the CDN allows us to simultaneously specify both ordinal and statistical independence relationships amongst model variables while allowing for a tractable inference algorithm.

In order to compute the DSP messages using the above CDN functions, we must compute the derivatives of all CDN functions. Since all of our functions are themselves Gaussian CDFs, the derivatives $\partial_{\mathbf{x}_A}\Big[\phi_s(\mathbf{x}_s)\Big]$ can be easily evaluated with respect to variables $\mathbf{X}_A$ as

$$\partial_{\mathbf{x}_A}\Big[\Phi\Big(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}\Big)\Big] = Gaussian\Big(\mathbf{x}_A;\boldsymbol{\mu}_A,\boldsymbol{\Sigma}_A\Big)\Phi\Big(\mathbf{x}_B;\tilde{\boldsymbol{\mu}}_B,\tilde{\boldsymbol{\Sigma}}_B\Big), \tag{5.13}$$

where

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_B \end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_A & \boldsymbol{\Sigma}_{A,B} \\ \boldsymbol{\Sigma}_{A,B}^T & \boldsymbol{\Sigma}_B \end{bmatrix},$$

$$\tilde{\boldsymbol{\mu}}_B = \boldsymbol{\mu}_B + \boldsymbol{\Sigma}_{A,B}^T\boldsymbol{\Sigma}_A^{-1}(\mathbf{x}_A - \boldsymbol{\mu}_A),$$
$$\tilde{\boldsymbol{\Sigma}}_B = \boldsymbol{\Sigma}_B - \boldsymbol{\Sigma}_{A,B}^T\boldsymbol{\Sigma}_A^{-1}\boldsymbol{\Sigma}_{A,B}.$$

Thus the message computations in the CDN are given by the updates shown in Tables 5.1, 5.2 and 5.3. We ensure that each message is properly normalized by locally computing the constant $\mathcal{Z} = \lim_{z\to\infty}\mu(z)$ for each message and multiplying each message pair μ, λ by $\mathcal{Z}^{-1}$.

Having described the above CDN model for multiplayer games, we would like to then estimate the player skill functions $s_k(x_k)$ for each player k from previous games played by that player. Denote by the set $T_k \subseteq \{1, \cdots, T\}$ the set of games in which player k participated. We then seek to estimate $s_k(x_k)$ for player k given previous team performances $\mathbf{r}_{\mathcal{T}_t}, t \in T_k$ and player scores for *all other* players $\mathbf{x}_{\mathcal{P}_t\setminus k}$ for all games $t \in T_k$ in which player k participated. Denote by $\mathcal{O}_t^{-k}$ the outcome of a game with the player score for player k removed from $\mathbf{x}_{\mathcal{P}_t}$. We will define the skill function $s_k(x_k)$ for a player to be given by

$$s_k(x_k) = F\Big(x_k|\{\mathcal{O}_t^{-k}\}_{t\in T_k}\Big) = \prod_{t\in T_k} F(x_k|\mathcal{O}_t^{-k}). \tag{5.14}$$

The above expression for the skill function $s_k(x_k)$ for player k corresponds to the conditional distribution $F\Big(x_k|\{\mathcal{O}_t^{-k}\}_{t\in T_k}\Big)$ given all past games played by player k with the

- Initialize for each player score node X_k:

$$\mu_{X_k \to g_n}(x_k) = s_k(x_k),$$
$$\lambda_{X_k \to g_n}(x_k) = \partial_{x_k}\Big[s_k(x_k)\Big].$$

- Pass messages from function node g_n to team performance node R_n for neighboring player nodes $\mathbf{X}_n$, $n = 1, \cdots, N$:

$$\mu_{g_n \to R_n}(\mathbf{r}, \mathbf{x}) = \sum_{\substack{s,t|\mathbf{X}_s \cup \mathbf{X}_t = \mathbf{X}_n \\ \mathbf{X}_s \cap \mathbf{X}_t = \emptyset}} \partial_{\mathbf{x}_s}\Big[g_n(\mathbf{x}_n, r_n)\Big] \prod_{j|X_j \in \mathbf{X}_s} \mu_{X_j \to g_n}(x_j) \prod_{j|X_j \in \mathbf{X}_t} \lambda_{X_j \to g_n}(x_j),$$
$$\lambda_{g_n \to R_n}(\mathbf{r}, \mathbf{x}) = \sum_{\substack{s,t|\mathbf{X}_s \cup \mathbf{X}_t = \mathbf{X}_n \\ \mathbf{X}_s \cap \mathbf{X}_t = \emptyset}} \partial_{\mathbf{x}_s, r_n}\Big[g_n(\mathbf{x}_n, r_n)\Big] \prod_{j|X_j \in \mathbf{X}_s} \mu_{X_j \to g_n}(x_j) \prod_{j|X_j \in \mathbf{X}_t} \lambda_{X_j \to g_n}(x_j).$$

- Set $\mu_{h_{n-1,n} \to R_n}(\mathbf{r}, \mathbf{x}) = \lambda_{h_{n-1,n} \to R_n}(\mathbf{r}, \mathbf{x}) = 1$ for $n = 1$. Pass messages from team performance node R_n to neighboring team performance nodes R_{n+1} and function nodes $h_{n,n+1}$ for $n = 1, \cdots, N$:

$$\begin{aligned}
\mu_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) &= \mu_{h_{n-1,n} \to R_n}(\mathbf{r}, \mathbf{x}) \mu_{g_n \to R_n}(\mathbf{r}, \mathbf{x}), \\
\lambda_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) &= \lambda_{h_{n-1,n} \to R_n}(\mathbf{r}, \mathbf{x}) \mu_{g_n \to R_n}(\mathbf{r}, \mathbf{x}) \\
&\quad + \mu_{h_{n-1,n} \to R_n}(\mathbf{r}, \mathbf{x}) \lambda_{g_n \to R_n}(\mathbf{r}, \mathbf{x}), \\
\mu_{h_{n,n+1} \to R_{n+1}}(\mathbf{r}, \mathbf{x}) &= \mu_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) \partial_{r_n}\Big[h_{n,n+1}(r_n, r_{n+1})\Big] \\
&\quad + \lambda_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) h_{n,n+1}(r_n, r_{n+1}), \\
\lambda_{h_{n,n+1} \to R_{n+1}}(\mathbf{r}, \mathbf{x}) &= \mu_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) \partial_{r_n, r_{n+1}}\Big[h_{n,n+1}(r_n, r_{n+1})\Big] \\
&\quad + \lambda_{R_n \to h_{n,n+1}}(\mathbf{r}, \mathbf{x}) \partial_{r_{n+1}}\Big[h_{n,n+1}(r_n, r_{n+1})\Big].
\end{aligned}$$

Table 5.1: The DSP algorithm for updating player ranks. Messages are ensured to be properly normalized locally by computing the constant $\mathcal{Z} = \lim_{z \to \infty} \mu(z)$ for each message and multiplying the message pair μ, λ by $\mathcal{Z}^{-1}$.

- Set $\mu_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x}) = \lambda_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x}) = 1$ for $n = N$. Pass messages from team performance node R_n to neighboring team performance nodes R_{n-1} and function nodes $h_{n-1,n}$ for $n = 1, \cdots, N$:

$$\begin{aligned}
\mu_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x}) &= \mu_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x})\mu_{g_n\to R_n}(\mathbf{r},\mathbf{x}),\\
\lambda_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x}) &= \lambda_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x})\mu_{g_n\to R_n}(\mathbf{r},\mathbf{x})\\
&\quad + \mu_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x})\lambda_{g_n\to R_n}(\mathbf{r},\mathbf{x}),\\
\mu_{h_{n-1,n}\to R_{n-1}}(\mathbf{r},\mathbf{x}) &= \mu_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x})\partial_{r_n}\Big[h_{n-1,n}(r_{n-1},r_n)\Big]\\
&\quad + \lambda_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x})h_{n-1,n}(r_{n-1},r_n),\\
\lambda_{h_{n-1,n}\to R_{n-1}}(\mathbf{r},\mathbf{x}) &= \mu_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x})\partial_{r_{n-1},r_n}\Big[h_{n-1,n}(r_{n-1},r_n)\Big]\\
&\quad + \lambda_{R_n\to h_{n-1,n}}(\mathbf{r},\mathbf{x})\partial_{r_{n-1}}\Big[h_{n-1,n}(r_{n-1},r_n)\Big].
\end{aligned}$$

- Pass messages from each team performance node R_n to neighboring function nodes g_n:

$$\begin{aligned}
\mu_{R_n\to g_n}(\mathbf{r},\mathbf{x}) &= \mu_{h_{n-1,n}\to R_n}(\mathbf{r},\mathbf{x})\mu_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x}),\\
\lambda_{R_n\to g_n}(\mathbf{r},\mathbf{x}) &= \lambda_{h_{n-1,n}\to R_n}(\mathbf{r},\mathbf{x})\mu_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x})\\
&\quad + \mu_{h_{n-1,n}\to R_n}(\mathbf{r},\mathbf{x})\lambda_{h_{n,n+1}\to R_n}(\mathbf{r},\mathbf{x}).
\end{aligned}$$

Table 5.2: The DSP algorithm for updating player ranks (cont'd). Messages are ensured to be properly normalized locally by computing the constant $\mathcal{Z} = \lim_{z\to\infty}\mu(z)$ for each message and multiplying the message pair μ, λ by $\mathcal{Z}^{-1}$.

- Pass messages from function nodes g_n to neighboring player score nodes X_k:

$$\mu_{g_n \to X_k}(\mathbf{r}, \mathbf{x}) = \sum_{\substack{s,t|\mathbf{X}_s \cup \mathbf{X}_t = \mathbf{X}_n \setminus X_k \\ \mathbf{X}_s \cap \mathbf{X}_t = \emptyset}} \prod_{j|X_j \in \mathbf{X}_s} \mu_{X_j \to g_n}(x_j) \prod_{j|X_j \in \mathbf{X}_t} \lambda_{X_j \to g_n}(x_j)$$
$$\cdot \left(\partial_{\mathbf{x}_s} \Big[g_n(\mathbf{x}_n, r_n) \Big] \lambda_{R_n \to g_n}(\mathbf{r}, \mathbf{x}) + \partial_{\mathbf{x}_s, r_n} \Big[g_n(\mathbf{x}_n, r_n) \Big] \mu_{R_n \to g_n}(\mathbf{r}, \mathbf{x}) \right),$$

$$\lambda_{g_n \to X_k}(\mathbf{r}, \mathbf{x}) = \sum_{\substack{s,t|\mathbf{X}_s \cup \mathbf{X}_t = \mathbf{X}_n \setminus X_k \\ \mathbf{X}_s \cap \mathbf{X}_t = \emptyset}} \prod_{j|X_j \in \mathbf{X}_s} \mu_{X_j \to g_n}(x_j) \prod_{j|X_j \in \mathbf{X}_t} \lambda_{X_j \to g_n}(x_j)$$
$$\cdot \left(\partial_{\mathbf{x}_s, x_k} \Big[g_n(\mathbf{x}_n, r_n) \Big] \lambda_{R_n \to g_n}(\mathbf{r}, \mathbf{x}) + \partial_{\mathbf{x}_s, x_k, r_n} \Big[g_n(\mathbf{x}_n, r_n) \Big] \mu_{R_n \to g_n}(\mathbf{r}, \mathbf{x}) \right).$$

- For each player score node X_k,

$$\mu_{X_k \to s_k}(\mathbf{r}, \mathbf{x}) = \mu_{g_n \to X_k}(\mathbf{r}, \mathbf{x}),$$
$$\lambda_{X_k \to s_k}(\mathbf{r}, \mathbf{x}) = \lambda_{g_n \to X_k}(\mathbf{r}, \mathbf{x}).$$

- Update player skill functions $s_k(x_k)$ using the multiplicative rule

$$s_k(x_k) \leftarrow s_k(x_k) \mu_{g_n \to X_k}(\mathbf{x}, \mathbf{r}).$$

Table 5.3: The DSP algorithm for updating player ranks (cont'd). Messages are ensured to be properly normalized locally by computing the constant $\mathcal{Z} = \lim_{z \to \infty} \mu(z)$ for each message and multiplying the message pair μ, λ by $\mathcal{Z}^{-1}$.

assumption that team performances and player scores are independently drawn from CDFs $F(\mathbf{r}_{\mathcal{T}_t}, \mathbf{x}_{\mathcal{P}_t})$ for $t = 1, \cdots, T$. The skill function s_k can then be readily estimated by the DSP algorithm, since each game outcome is modeled by a tree-structured CDN. More precisely, we first initialize $s_k(x_k) = \Phi(x_k; \mu, \beta^2)$. For each game Γ_t we can perform message-passing to obtain the conditional CDF $F(x_k|\mathcal{O}_t^{-k}) = \mu_{g_n \to X_k}(\mathbf{r}_{\mathcal{T}_t}, \mathbf{x}_{\mathcal{P}_t \setminus k})$ for player k (assuming the message $\mu_{g_n \to X_k}$ has been properly normalized as described above) and then perform a multiplicative update $s_k(x_k) \leftarrow s_k(x_k)\mu_{g_n \to X_k}$. The skill function $s_k(x_k)$ can then be used to make predictions for player k's scores in future games. We will proceed in the next section to apply the model and the above inference procedure to the problem of modelling Halo 2™ games.

5.2.4 The Halo 2™ Beta dataset

The Halo 2™ Beta dataset (v1.1)[1] consists of player scores for four game types ("HeadToHead", "FreeForAll", "SmallTeams" and "LargeTeams") over a total of 6,465 players. The descriptions for each of the four game modes are given below.

- HeadToHead: 6227 games/1672 players, one player competing against another player
- FreeForAll: 60022 games/5943 players, up to eight players playing against each other
- SmallTeams: 27539 games/4992 players, up to four players per team, two competing teams
- LargeTeams: 1199 games/2576 players, up to eight players per team, two competing teams

[1]Credits for the use of the Halo 2™ Beta Dataset are given to Microsoft Research Ltd. and Bungie.

To construct the above CDN model, we set the cutpoints $\theta(r_n)$ in the above cumulative model using ordinal regression of team ranks on team performances for all games in the training set. We initialized all player skill functions to $s_k(x_k) = \Phi(x_k; \mu, \beta^2)$. The set of parameters $\{\mu, \rho, \beta, \sigma\}$ in the CDN model was set based on test error to $\{25, -0.95, 20, 0.25\}$ for "HeadToHead", $\{50, -0.2, 10, 0.2\}$ for "FreeForAll", $\{20, -0.1, 10, 0.027\}$ for "SmallTeams" and $\{1, -0.9, 1, 0.01\}$ for "LargeTeams" game modes. For each of these game modes, we applied the DSP algorithm as described above in order to obtain updates for the player skill functions $s_k(x_k)$. An example of such an update at the end of a game with four competing players is shown in Figure 5.5.

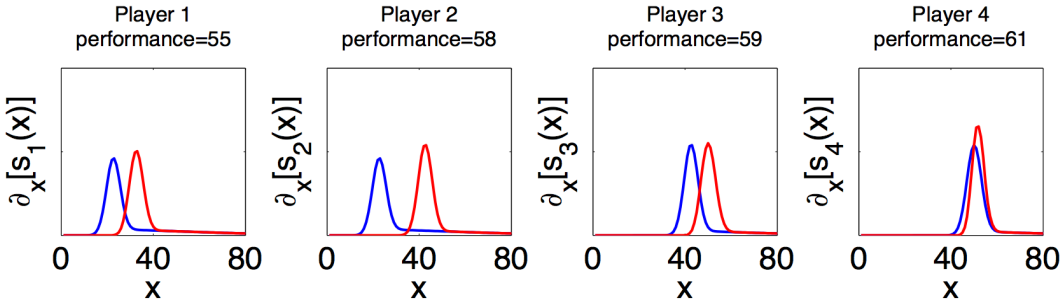
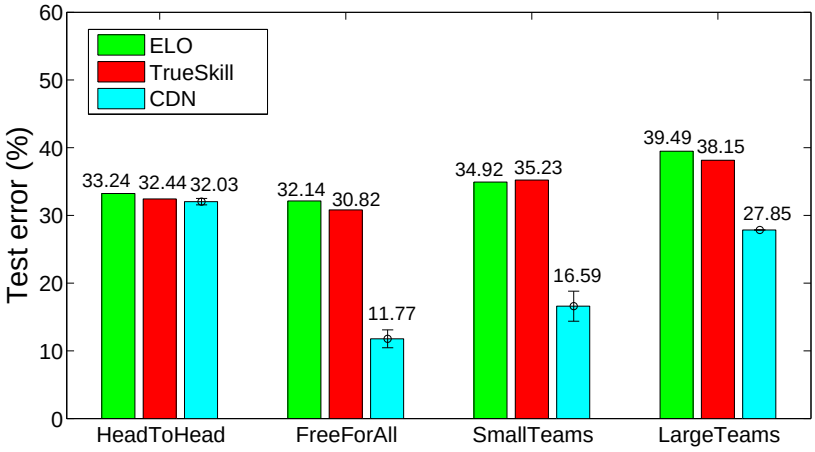


Figure 5.5: An example of derivative-sum-product updates for a four-player free-for-all game, with the derivative of the skill functions before the updates (blue) and afterwards (red).

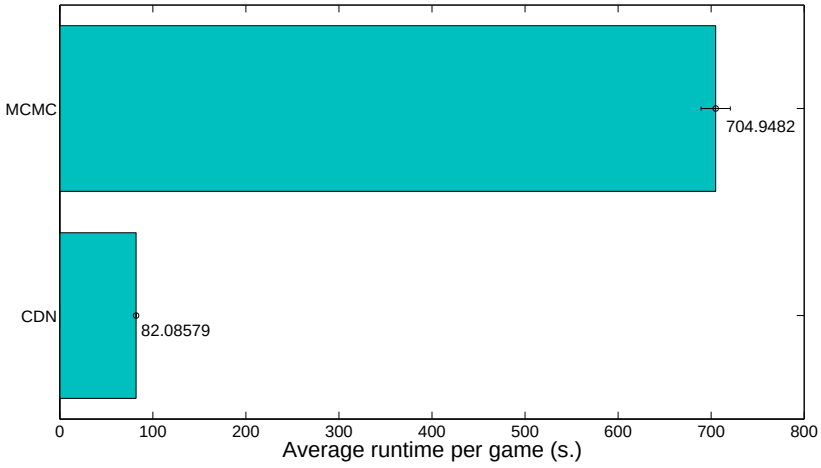
Before each game, we can predict the team performances using the player skills learned thus far via the rule $x_k^* = \arg\max_{x_k} \partial_{x_k}\Big[s_k(x_k)\Big]$. For each game, the set of team performances is then defined by the ordering of teams once the game is over, where we add the predicted player scores together x_k^* for each team and sorting the resulting sums in ascending order. For any predicted set of team performances, an error is incurred for that game if two teams for that game were misranked. One can then compute an error rate over the entire set of games for which we make predictions about team performances.

A plot showing the average prediction error rate obtained for the above CDN models over five runs of DSP is shown in Figure 5.6(a). It is worth noting that our choice of multivariate Gaussian CDFs as CDN functions in the above model requires that we use a

sampling method in order to evaluate the CDN functions, so that the error bars over the five runs are shown. In addition, Figure 5.6(a) also shows the error rates reported by [23] for TrueSkill^{TM} and ELO [14], which is a statistical rating system used in chess. Here, we see that the ability to specify both ordinal relationships and statistical dependence relationships between model variables using a CDN allows us to achieve higher predictive accuracy than either TrueSkill^{TM}, which requires approximating the true posterior distribution over team performances, or the ELO method, which does not account for the team structure of multiplayer games. As an additional comparison, we performed inference under the TrueSkill^{TM} factor graph model for the "FreeForAll" and "LargeTeams" game modes by sampling from the model using the Metropolis-Hastings (MH) algorithm where the minimum number of samples per player per game was set to be 1000. The average error rate achieved by the MH algorithm over three independent runs of sampling for the "LargeTeams" games was $28.58\% \pm 0.011\%$), which is nearly the error rate achieved by the DSP algorithm for this same task. Note however that the amount of computation required to achieve this error rate was significantly larger than that required by the DSP algorithm for the same task using a machine with 12 GB of RAM and two 2.8 GHz dual-core AMD Opteron CPUs (Figure 5.6(b)). We also found that the rejection rate in the sampling procedure was extremely high when sampling from the model for "FreeForAll" games, so that the MH algorithm failed to generate over 1000 samples from the stationary distribution after over three weeks of computation. These results suggest that while the TrueSkill^{TM} model can provide a good model for game outcomes in multiplayer games, the cost of performing exact inference under the model will generally be prohibitive. In contrast, our results suggest that the CDN model provides a model under which exact inference can be performed at a fraction of the computational cost required to perform exact inference in the factor graph model.



(a)



(b)

Figure 5.6: a) Prediction error on the Halo 2™ Beta dataset (computed as the fraction of team predicted incorrectly before each game) for DSP, ELO [14] and TrueSkill™ [23] methods. Error bars over five runs of DSP are shown. b) Average runtime per game for "LargeTeams" games using MCMC sampling in the TrueSkill™ factor graph and using the DSP algorithm in the proposed CDN model. Errorbars are shown for three independent initializations of each algorithm.

5.2.5 Discussion

In this section we presented a model and method for learning to rank in the context of multiplayer team-based games such as Halo 2™. Our model represent both statis-

tical dependence relationships and notions of orderings of variables in the model such as team performances and individual player scores. We then used the DSP algorithm from Chapter 4 to compute conditional CDFs for each player's score. Comparisons to the TrueSkill™ and ELO methods for factor graph models show that our model and method allows both for fast estimation and improved test error on the Halo 2™ Beta dataset.

While the above method has the advantage of providing a flexible probabilistic model and allowing for tractable inference, the choice of multivariate Gaussian CDFs for CDN functions requires the use of sampling methods in order to evaluate DSP messages. As shown in Chapter 3, one can find faster parameterizations of the CDN functions that do not require sampling. A particular drawback to the above method is that we have not explicitly accounted for the fact that the observed ranking values are arbitrary up to a monotonic transformation, so that our model predictions may be sensitive to such transformations. Furthermore, we have not yet presented a general framework for learning the CDN functions (such as a maximum-likelihood method for estimating model parameters), nor have we discussed how one could account for additional information in the form of features. For example, in the multiplayer gaming example, player fatigue, stamina and frequency of injury could be accounted for as additional features for the purpose of making predictions. Additionally, such features could be used to learn different rankings for different types of players (e.g.: defensive versus offensive players) and could allow for a more refined modelling of player-to-player interaction. Finally, the proposed method assumes a particular set of statistical dependence relationships that are a specific property of team-based games, but perhaps not for other problems of learning to rank. To address these issues, we will proceed in the next section to develop a general framework for learning to rank where we are given observations about partial orderings and we seek to leverage these together to predict new orderings.

5.3 Probability models over partial orderings as cumulative distribution networks

In many problems, the objects to be ranked do not necessarily exhibit a higher order structure as was the case in multiplayer team-based games, where each team's performance was a function of its member players' scores. Furthermore, unlike the multiplayer gaming setting where players were not provided with additional features, in many other problems of learning to rank, each object is provided with some features that are presumably relevant to the task of learning a ranking over objects. In this section we will formulate a general framework for learning to rank in which we will use CDNs to formulate probabilistic models for orderings over objects.

The approach we adopt here for learning to rank is to combine the graphical framework of CDNs with a ranking function ρ that independently maps each node to a scalar score value [31], so that a higher score is assigned to objects with higher ranks. The intuition here is that a sensible loss functional can be chosen as the probability $\mathbb{P}[\alpha_1 \succeq \cdots \succeq \alpha_K]$ of observing a partial ordering over nodes $\alpha_1, \cdots, \alpha_K$ for any given observation, so that CDNs provide a graphical framework whereby we can model many of the statistical dependence relationships between variables for such probabilities.

Suppose that we wish to rank nodes in the set $\mathcal{V}$ given a set of observations $D_1, \cdots, D_N$, where each observation D_n provides a partial ordering of the nodes in some subset $\mathcal{V}_n \subseteq \mathcal{V}$. For example, in an information retrieval task, a node corresponds to a document and each observation consists of a query with relevance ratings for each document that are provided by an expert. It is crucial to note here that the quantitative labels for each node, such as relevance ratings, may in general not be directly comparable from one observation to the next. In the information retrieval task, this would mean that documents with the same rating across different observations might not necessarily have the same degree of relevance for a future query.

For a given observation, we can model the ordering of nodes as a directed acyclic graph in which a directed edge $e = (\alpha, \beta)$ is drawn between two nodes α, β if and only if node α was *preferred* to node β within observation D_n. In the information retrieval task, this would mean that document α was assigned a higher relevance rating y_α than the rating y_β for document β for a given observation. The way in which a preference is defined between two objects can be application-specific and we will restrict ourselves only to modelling the resulting set of preferences. We will denote the preference between two nodes in an order graph by $\alpha \succ \beta$ and we will denote this as the *order graph* $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ for observation D_n, where $\mathcal{E}_n$ is the set of all edges in the order graph. We first note that the order graph does not correspond to a directed graphical model for some joint probability and merely conveys a set of preference relationships amongst objects to be ranked. In general, we will assume that for observation D_n we observe a partial ordering over nodes, with complete orderings being a special case. A toy example of an order graph defined over four nodes is shown in Figure 5.7. For any given order graph, the absence of an edge between two nodes α, β in the order graph indicates that we cannot assert any preference between the two nodes. From the definition of the order graph as a directed acyclic graph, it follows that transitivity of preference holds for any given observation, so that if $\alpha \succ \beta, \beta \succ \gamma$ for any given observation D_n, then $\alpha \succ \gamma$ in D_n. We note that while transitivity must hold for any given observation, we do not assume or require that transitivity hold across all observations, as different observations may contain different preference relationships between the same nodes.

We now define $\rho : \mathcal{V} \mapsto \mathbb{R}$ as a *ranking function* that assigns a real-valued number to nodes so that the score S_α for node α is given by

$$S_\alpha = \rho(\alpha) + \pi_\alpha, \tag{5.15}$$

where S_α is real-valued and π_α is a random variable. Thus by assigning a node-specific variable π_α for each node $\alpha \in \mathcal{V}$, our model allows us to account for the fact that the amount of uncertainty about a node's score may depend on unobserved features for

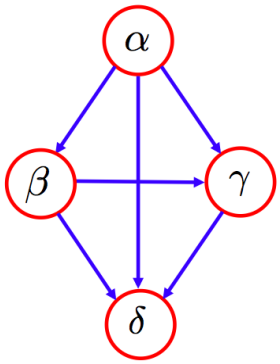


Figure 5.7: Example of an observation with four nodes $\alpha, \beta, \gamma, \delta$ with the corresponding order graph. Here, the order graph represents the set of preference relationships $\alpha \succ \beta, \alpha \succ \gamma, \beta \succ \gamma, \beta \succ \delta, \gamma \succ \delta$.

that node (e.g.: documents associated with certain keywords might have less variability in their scores than other documents associated with different keywords). Under this model, the necessary and sufficient condition for the preference relation $\alpha \succ \beta$ is then

$$\rho(\alpha) + \pi_\alpha \geq \rho(\beta) + \pi_\beta \Leftrightarrow \pi_{\alpha\beta} = \pi_\beta - \pi_\alpha \leq \rho(\alpha) - \rho(\beta). \tag{5.16}$$

where we have defined $\pi_{\alpha\beta}$ as a *preference variable* for nodes α, β. The canonical problem of ranking learning is then to learn the function ρ from the training data, where each observation consists of an order graph over some subset of the nodes to be ranked. A flowchart of the problem of learning from order graphs is shown in Figure 5.8.

For each edge $e \in \mathcal{E}_n$ in the order graph such that $e = (\alpha, \beta)$, we can define $r(\rho; e, D_n) \equiv \rho(\alpha) - \rho(\beta)$ and collect the set of such variables into the vector $\mathbf{r}(\rho; G_n) \in \mathbb{R}^{|\mathcal{E}_n|}$. Similarly, let $\pi_e \equiv \pi_{\alpha\beta}$ be a preference variable defined along the directed edge e. Then for a given observation D_n, the probability of observing the order graph $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ corresponding to the partial ordering of nodes in $\mathcal{V}_n$ is given by

$$\mathbb{P}\Big[\mathcal{E}_n | \mathcal{V}_n, \rho\Big] \equiv \mathbb{P}\Bigg[\cap_{(\alpha,\beta)\in\mathcal{E}_n} \{\alpha \prec \beta\}\Bigg] = \mathbb{P}\Bigg[\cap_{e\in\mathcal{E}_n} \{\pi_e \leq r(\rho; e, D_n)\}\Bigg] = F_{\boldsymbol{\pi}}\Big(\boldsymbol{r}(\rho; G_n)\Big), \tag{5.17}$$

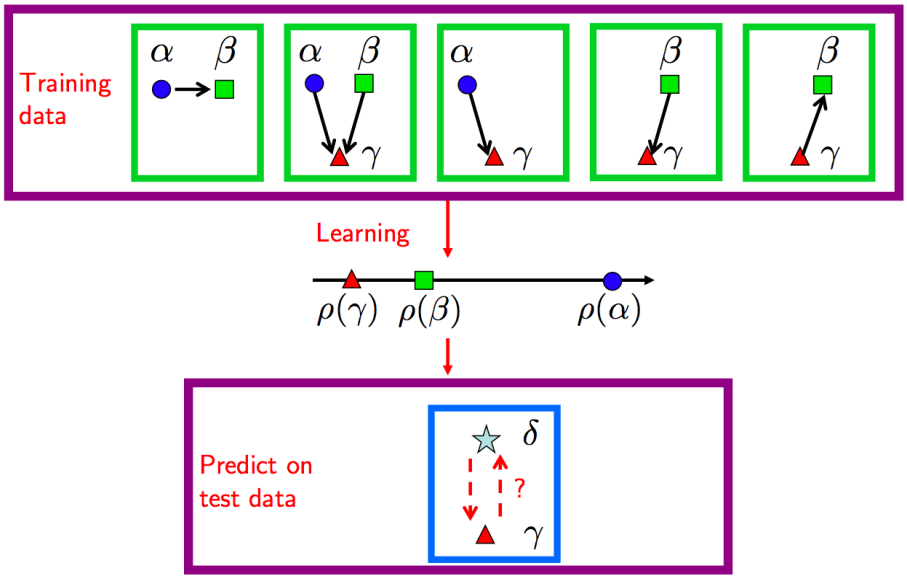


Figure 5.8: Learning the ranking function from training data. The ranking function ρ maps each node α to $\mathbb{R}$. The goal is to learn ρ such that we correctly rank the nodes in a new test observation.

where $F_{\boldsymbol{\pi}}\Big(\boldsymbol{r}(\rho; G_n)\Big)$ is the joint CDF over the preference variables. The goal given an observation D_n is therefore to learn the ranking function ρ by maximizing the probability $\mathbb{P}\Big[\mathcal{E}_n | \mathcal{V}_n, \rho\Big]$ of generating the orderings in observation D_n, given the ranking function ρ and the nodes in $\mathcal{V}_n$. Note that under this framework, the set of edges $\mathcal{E}_n$ corresponding to the set of pairwise preferences between nodes in the order graph are treated as random variables that may have a high degree of dependence between one another. The problem of learning the ranking function is then one of *structured ranking learning* in which we would like to score multiple nodes simultaneously whilst accounting for statistical dependence relationships between these scores.

Now, if we are given multiple independent observations $\mathcal{D} = \{D_1, \cdots, D_N\}$, then we can define a structured loss functional

$$\mathcal{L}(\rho, F_{\boldsymbol{\pi}}, \mathcal{D}) = -\sum_{n=1}^{N} \log \mathbb{P}\Big[\cap_{(\alpha,\beta)\in\mathcal{E}_n} \{\alpha \prec \beta\}\Big] = -\sum_{n=1}^{N} \log F_{\boldsymbol{\pi}}\Big(\boldsymbol{r}(\rho; G_n)\Big), \tag{5.18}$$

where each term $\log F_{\boldsymbol{\pi}}\Big(\boldsymbol{r}(\rho; G_n)\Big) = \log \mathbb{P}\Big[\cap_{(\alpha,\beta)\in\mathcal{E}_n} \{\alpha \prec \beta\}\Big]$ in the loss functional is the probability of a partial ordering over nodes in observation D_n. Each of these terms depends on multiple preference relationships specified by the order graph for observation D_n. Thus the structured loss functional is the probability of observing a set of partial orderings over subset of nodes to be ranked. In particular, the loss incurred for a given predicted ordering and an observed one will depend both on the ranking function ρ and the joint CDF $F_{\boldsymbol{\pi}}$. Thus, a low loss is achieved by a ranking function ρ that succeeds in minimizing the number of misordered pairs of nodes for a given observation D_n.

Having defined the structured loss functional, the problem of learning the ranking function ρ then consists of the following optimization problem:

$$\min_{\rho, F_{\boldsymbol{\pi}}} \quad \mathcal{L}(\rho, F_{\boldsymbol{\pi}}, \mathcal{D}). \tag{5.19}$$

In general, the above structured loss functional may be difficult to specify, as it takes on the form of a joint CDF over a possibly large number of preference variables, which may require a large number of parameters to specify. We can however, compactly model such a joint CDF using CDNs, as we will now show.

5.3.1 Transforming order graphs into cumulative distribution networks

We begin this section by noting that we have posited an equivalence between a pairwise preference $\alpha \prec \beta$ and the event $\pi_{\alpha\beta} \leq \rho(\alpha) - \rho(\beta)$. The representation of the structured loss functional in Equation (5.18) as a CDN then consists of transforming the order graph G_n for a given observation into a set of variable nodes in the CDN, so that for each preference relation implied by the order graph we create a corresponding preference variable. More precisely, for each edge $e = (\alpha, \beta)$ in the order graph, the preference variable $\pi_{\alpha\beta}$ is created. All such variables can then be connected to one another in a CDN with many possible connectivities (Figure 5.9). The pattern of connectivity used will

determine the set of statistical dependence relationships between these preferences $\pi_{\alpha\beta}$ as given by the marginal and conditional independence properties of the CDN explored in Chapter 3.

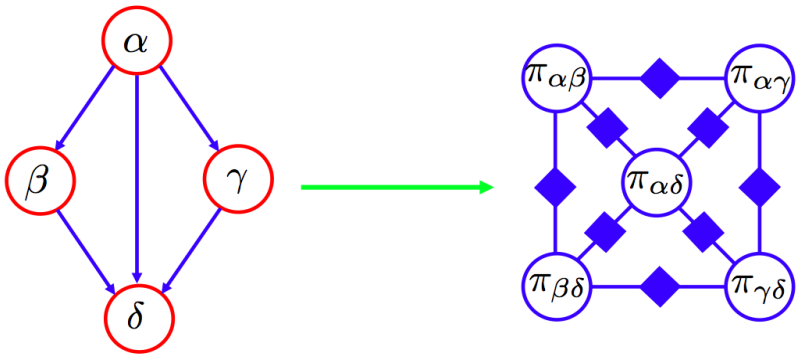


Figure 5.9: Transforming the order graph G_n into a CDN. For each edge $e = (\alpha, \beta)$ in the order graph (left), a preference variable $\pi_{\alpha\beta}$ is created. All such random variables can then be connected to one another in a CDN with different possible connectivities (right), allowing for complex statistical dependence relationships between preferences.

One possible concern is that we may require a CDN that is cumbersome to learn due to the number of CDN functions connecting pairs of preference variables in the model. In practice, because any observation will only convey information about a small subset of the nodes in $\mathcal{V}$ and because most often we will observe partial orderings of nodes, the number of preference nodes in the CDN for the given observation will be much smaller than the worst-case number of all possible pairwise preferences between nodes to be ranked. Furthermore, we do not have to store a large CDN in memory during training, as we only need to store a single CDN over a relatively small number of preference variables for the current observation. The closure property of CDNs under marginalization (Proposition 3.2.5) ensures that the set of statistical dependence relationships between preference variables remains unchanged under marginalization, so that one can view the CDN for a given observation as being a subgraph of a larger CDN defined over all possible pairwise preference variables so that graph separation of nodes in both graphs imply the same set

of independence relationships. We can thus perform structured ranking learning in an online fashion by constructing a single CDN for each observation, computing the gradient of the loss functional defined by that CDN for the given observation and then updating the parameters of the model.

5.3.2 Connections to probability models for rank data

The above formulation of the structured ranking learning framework allows us to model a joint CDF over preference variables as a CDN. We will show in this section that this formulation allows us to view probability models for rank data as CDNs with particular connectivities between preference variables. Several probability models have been developed for modelling pairwise preferences, with different models making different assumptions about the statistical independence relationships between pairwise preferences. Here we will present two statistical models for pairwise preferences that are frequently used. While other types of model have been developed to account for rank data, such as distance-based models defined over permutations [46], we will focus here on the class of models defined over pairwise preferences, due to their frequent use and their connection to the graphical modelling framework of this thesis. For a complete survey of models and methods for rank data, the reader is encouraged to consult [46].

Bradley-Terry models

Suppose we wish to rank objects $\alpha_1, \cdots, \alpha_K$. In a Bradley-Terry model [46], the probability of a preference relation $\alpha_i \succ \alpha_j$ is given by

$$\mathbb{P}\Big[\alpha_i \succ \alpha_j\Big] = \frac{\exp(\rho(\alpha_i))}{\exp(\rho(\alpha_i)) + \exp(\rho(\alpha_j))}, \tag{5.20}$$

so that a higher value of $\rho(\alpha_i)$ relative to $\rho(\alpha_j)$ corresponds to a higher degree of preference between α_i, α_j. An example of such a model arises in information retrieval, where given a query for documents, a higher $\rho(\alpha_i)$ corresponds to a higher degree of relevance

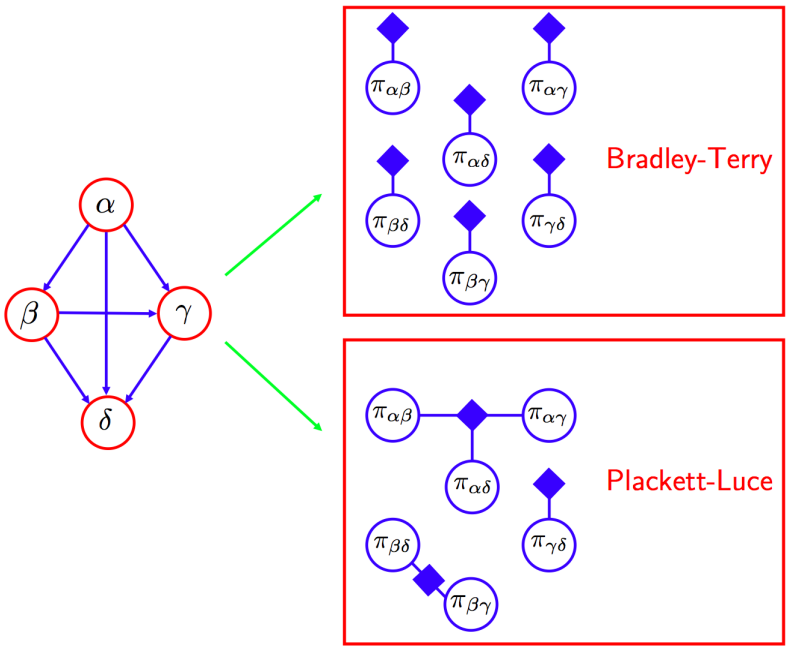


Figure 5.10: Corresponding CDNs for the Bradley-Terry and Plackett-Luce probability models for a complete ordering over four nodes $\alpha, \beta, \gamma, \delta$.

of document α_i for the given query. The probability of a partial ordering $\alpha_1 \succeq \cdots \succeq \alpha_K$ is then given by

$$\begin{aligned}
\mathbb{P}\Big[\alpha_1 \succeq \cdots \succeq \alpha_K\Big] &= \prod_{\alpha_i \succ \alpha_j} \frac{\exp(\rho(\alpha_i))}{\exp(\rho(\alpha_i)) + \exp(\rho(\alpha_j))} \\
&= \prod_{\alpha_i \succ \alpha_j} \frac{1}{1 + \exp\Big(-\big(\rho(\alpha_i) - \rho(\alpha_j)\big)\Big)} \\
&= \prod_{\alpha_i \succ \alpha_j} \frac{1}{1 + \exp\Big(-r_{ij}\Big)} = \prod_{\alpha_i \succ \alpha_j} \phi\big(r_{ij}\big),
\end{aligned} \tag{5.21}$$

where $r_{ij} \equiv \rho(\alpha_i) - \rho(\alpha_j)$ and each of the functions $\phi\big(r_{ij}\big)$ satisfy the properties of a CDF, so that we can view the joint probability $\mathbb{P}\Big[\alpha_1 \succeq \cdots \succeq \alpha_K\Big]$ as a CDF defined over preferences. Each term in the above product corresponds to the probability of observing the pairwise preference $\alpha_i \succ \alpha_j$ in the partial ordering $\alpha_1 \succeq \cdots \succeq \alpha_K$. Note that a partial ordering implies the absence of a preference for some α_i, α_j. Thus the joint probability of all pairwise preferences under a Bradley-Terry model can be viewed as a disconnected CDN in which all pairwise object preferences are treated as being independent. This

is illustrated in Figure 5.10 for an example in which four nodes $\alpha, \beta, \gamma, \delta$ have been ranked and we model the joint probability over the set of corresponding preferences $\alpha \succ \beta, \alpha \succ \gamma, \alpha \succ \delta, \beta \succ \gamma, \beta \succ \delta, \gamma \succ \delta$ using a Bradley-Terry model.

Plackett-Luce models

The Plackett-Luce model generalizes the Bradley-Terry model to multiple pairwise comparisons between objects to be ranked, so that the joint probability of a partial ordering as

$$\begin{aligned}\mathbb{P}\Big[\alpha_1 \succeq \cdots \succeq \alpha_K\Big] &= \prod_{k=1}^{K} \frac{\exp(\rho(\alpha_k))}{\exp(\rho(\alpha_k)) + \sum\limits_{\alpha_j \succ \alpha_k} \exp(\rho(\alpha_j))} \\ &= \prod_{k=1}^{K} \frac{1}{1 + \sum\limits_{\alpha_j \succ \alpha_k} \exp\Big(-\big(\rho(\alpha_k) - \rho(\alpha_j)\big)\Big)} \\ &= \prod_{k=1}^{K} \phi_k(\boldsymbol{r}_k), \end{aligned} \tag{5.22}$$

where $\boldsymbol{r}_k$ is a vector whose elements consists of pairwise differences $\rho(\alpha_k) - \rho(\alpha_j)$ and each of the functions $\phi_k(\boldsymbol{r}_k)$ also satisfy the properties of a CDF, so that we can again view the joint probability $\mathbb{P}\Big[\alpha_1 \succeq \cdots \succeq \alpha_K\Big]$ as a joint CDF over preferences that can be viewed as a CDN with functions $\phi_k(\boldsymbol{r}_k)$. Each term in the Plackett-Luce model corresponds to the probability that α_k is ranked first in the sequence followed by $\alpha_{k+1}, \cdots, \alpha_K$, so that the joint probability of the partial ordering $\alpha_1 \succeq \cdots \succeq \alpha_K$ is given by the product of such terms. Thus the Plackett-Luce model incorporates additional constraints between pairwise preferences that are not present in the Bradley-Terry model, at the cost of a slightly more complex model. As a consequence of such additional constraints, the corresponding CDN adopts additional structure as compared to the CDN for the Bradley-Terry model. The CDN for the Plackett-Luce model corresponding to the previous example of four nodes $\alpha, \beta, \gamma, \delta$ is shown in Figure 5.10. Thus, we have shown that CDNs provide a graphical representation for the above models of rank data, so that

the Bradley-Terry and Plackett-Luce models can be viewed as different instantiations of a joint CDF over pairwise preferences and hence as particular types of CDNs. This suggests that we can formulate alternate probabilistic models for rank data by considering joint CDFs defined over pairwise preferences using different types of CDN functions and connectivities for modelling different conditional independence relationships amongst variables.

Having presented the framework of structured ranking learning using CDNs, we will now apply the framework to the problem domains of document retrieval and regulatory sequence search in computational biology, where the problem of learning to rank arises.

5.4 Application: Document retrieval

The problem of learning to rank is at the heart of many information retrieval applications such as collaborative filtering [57] and document retrieval [44]. In this section we will apply the above structured ranking learning framework to the latter problem, whereby we are given queries with documents labelled by relevance and we wish to learn a model to score documents by relevance for a new query.

5.4.1 Previous work

Several approaches to the problem of ranking for document retrieval have been proposed previously. One approach has been to treat the problem of learning to rank as one of structured classification [36, 66], where the aim is to directly optimize ranking measures of loss. However, typical measures of loss are difficult to optimize directly [8], making the problem of learning in this setting computationally difficult. Another approach has been to approximate these ranking measures with smooth differentiable loss functionals by formulating probabilistic models on pairwise preferences between objects (RankNet; [7]), or on ordered lists of objects (ListNet and ListMLE; [9, 70]). It is worth noting that the

RankNet, ListNet and ListMLE methods are examples of Bradley-Terry and Plackett-Luce models respectively. In the case of the RankNet model [7], the corresponding probability over a pairwise preference $\alpha \succ \beta$ is modeled by a logistic function so that if each node to be ranked is provided with a feature vector $\mathbf{x}_\alpha$ and $\rho(\alpha) \equiv \rho(\mathbf{x}_\alpha)$, we obtain

$$\mathbb{P}[\alpha \succ \beta] = \frac{1}{1 + \exp\Big(- \big(\rho(\mathbf{x}_\alpha) - \rho(\mathbf{x}_\beta)\big)\Big)}. \tag{5.23}$$

The problem of learning in RankNet consists then of minimizing cross-entropy loss

$$\mathcal{L}(\rho) = (1 - \hat{P}_{\alpha\beta})\big(\rho(\mathbf{x}_\alpha) - \rho(\mathbf{x}_\beta)\big) + \log\left(1 + \exp\Big(- \big(\rho(\mathbf{x}_\alpha) - \rho(\mathbf{x}_\beta)\big)\Big)\right), \tag{5.24}$$

across all (α, β) pairs, where $\hat{P}_{\alpha\beta}$ denotes the target probability of $\alpha \succ \beta$. We see here that while the loss function used by RankNet is cross-entropy loss, the underlying probability model corresponds to a Bradley-Terry model, which assumes that preferences are independent of one another.

In the case of ListNet [9] and ListMLE [70], the probability of observing a partial ordering $\alpha_1 \succeq \cdots \succeq \alpha_K$ objects are defined as products of functions of the type

$$\mathbb{P}(\alpha_1 \succeq \cdots \succeq \alpha_K | D) = \prod_{k=1}^{K} \frac{\exp(\rho(\mathbf{x}_k))}{\sum_{j=k}^{K} \exp(\rho(\mathbf{x}_j))} = \prod_{k=1}^{K} \frac{1}{1 + \sum_{j=k+1}^{N} \exp\Big(- \big(\rho(\mathbf{x}_k) - \rho(\mathbf{x}_j)\big)\Big)}, \tag{5.25}$$

which is a Plackett-Luce model. Thus we see here that previous methods for learning to rank for document retrieval can be reformulated and viewed as particular instances of the structured ranking learning framework using CDNs.

5.4.2 Learning to rank documents from queries

In the context of ranking documents in web search, the goal is to learn a ranking function such that for a given query, the ranking function assigns a score to each document. Under such a model, documents that are assigned a high score for a given query are likely to be relevant to the query. Here we will apply the proposed structured ranking framework to

the problem of scoring documents in databases or in the context of web search. We will focus on the OHSUMED dataset, which is a part of the Learning to Rank (LETOR) 2.0 benchmark [44]. The dataset consists of a set of query-document pairs, with a feature vector and relevance judgment provided for each pair. Under our framework, we observe the preference $\alpha \succ \beta$ for a given observation if and only if document α received a higher relevance rating y_α than the rating for document β. Under our framework, the goal is to find some function that takes the set of features and maps these to the real line so that for a new query-document pair, a high score means the document is likely to be relevant to the query. The proposed structured ranking learning framework allows one to define a ranking function over query-specific features, so that for any query D_n we will suppose that there is a query-specific feature vector $\mathbf{x}_\alpha^n \in \mathbb{R}^L$ for each document α to be ranked. Although each document α can receive different feature vectors for different queries, the ranking function ρ will be shared across queries, enabling us to learn a model for scoring documents. Furthermore, we will assume that the feature set will be shared across all queries, with no missing values to be accounted for.

We will parameterize the ranking function using a parameter vector $\mathbf{a}$, so that $\rho \equiv \rho(\mathbf{x}; \mathbf{a})$ with $-\infty < \rho(\mathbf{x}; \mathbf{a}) < \infty$. Here we will choose $\rho(\mathbf{x}; \mathbf{a})$ to be a Nadaraya-Watson [52, 67] estimator with a Gaussian kernel, as such a ranking function allows for non-linear estimates of document scores. The ranking function thus takes the form

$$\rho(\mathbf{x}; \mathbf{a}) = \frac{\sum\limits_{\alpha \in \mathcal{V}_n} K(\mathbf{x}_\alpha, \mathbf{x}; \mathbf{a}) y_\alpha}{\sum\limits_{\alpha \in \mathcal{V}_n} K(\mathbf{x}_\alpha, \mathbf{x}; \mathbf{a})}, \tag{5.26}$$

$$K(\mathbf{x}_\alpha, \mathbf{x}; \mathbf{a}) = \exp\left(-\frac{1}{2}\Big(\mathbf{x} - \mathbf{x}_\alpha\Big)^T \mathbf{A} \Big(\mathbf{x} - \mathbf{x}_\alpha\Big)\right), \tag{5.27}$$

with $\mathbf{A} = \text{diag}(a_1^2, \cdots, a_L^2)$. For the chosen ranking function, we will assume that document labels are comparable across queries, although this assumption in general is not necessary and could be relaxed.

Consider a directed edge $e = (\alpha, \beta)$ in the order graph G_n for observation D_n and

define

$$r_e \equiv r_e(\mathbf{a}; D_n) = \rho(\mathbf{x}_\alpha^n; \mathbf{a}) - \rho(\mathbf{x}_\beta^n; \mathbf{a}). \tag{5.28}$$

For the given order graph, we will model the structured loss functional $\mathcal{L}(\boldsymbol{\theta}; D_n)$ using a CDN so that

$$\mathcal{L}(\boldsymbol{\theta}; D_n) = -\log F_{\boldsymbol{\pi}}\Big(\boldsymbol{r}(\rho; G_n)\Big) = -\sum_{e,e' \in \mathcal{E}_n} \log \phi(r_e(\mathbf{a}; D_n), r_{e'}(\mathbf{a}; D_n)), \tag{5.29}$$

where $\boldsymbol{\theta}$ is the parameter vector containing $\mathbf{a}$ and parameters for the CDN functions in the network. While we can choose any CDN topology for connecting preference variables to one another, we will model the joint CDF over preferences as a CDN in which there is a function node for every pair of preference variables, so that each function is of the form $\phi(r_1, r_2)$ so that every preference variable is connected to all other preference variables via these functions. Thus, each pair of edges e, e' in the order graph G_n has a CDN function node $\phi(r_e, r_{e'})$ connecting the corresponding preference variable nodes $\pi_e, \pi_{e'}$ in the CDN. In order for the CDN functions to satisfy the sufficient conditions for the CDN to model a valid CDF (Lemma 3.1.1), we will opt for each function $\phi(r_1, r_2)$ is be a bivariate sigmoidal function so that

$$\phi(r_1, r_2) = \frac{1}{1 + \exp(-u_1 r_1) + \exp(-u_2 r_2)}, \quad u_1, u_2 \geq 0, \tag{5.30}$$

as this will simplify both representation and computation. For the given CDN and ranking functions, the learning problem then becomes

$$\begin{aligned} \min_{\mathbf{a}, u_1, u_2} \sum_{n=1}^{N} \sum_{e,e' \in \mathcal{E}_n} \log \left(1 + \exp\Big(-u_1 r_e(\mathbf{a}; D_n)\Big) + \exp\Big(-u_2 r_{e'}(\mathbf{a}; D_n)\Big)\right) \quad \text{s.t.} \quad & \mathbf{a} \geq 0 \\ & u_1, u_2 \geq 0 \\ & \|\mathbf{a}\|_1 \leq t, \end{aligned} \tag{5.31}$$

where we have added an L_1-norm regularizer on the ranking function parameters $\mathbf{a}$ using the constraint $\|\mathbf{a}\|_1 \leq t$. In order to minimize the above loss functional, we will use

a stochastic gradient descent algorithm [6]. For a given observation D_n, the gradient $\nabla_{\mathbf{a}}\mathcal{L}(\boldsymbol{\theta}; D_n)$ is given by

$$\nabla_{\mathbf{a}}\mathcal{L}(\boldsymbol{\theta}; D_n) = -\sum_{e,e' \in \mathcal{E}_n} \frac{1}{\phi(r_e, r_{e'})}\Big(\partial_{r_e}\Big[\phi(r_e, r_{e'})\Big]\nabla_{\mathbf{a}} r_e(\mathbf{a}; D_n) + \partial_{r_{e'}}\Big[\phi(r_e, r_{e'})\Big]\nabla_{\mathbf{a}} r_{e'}(\mathbf{a}; D_n)\Big), \tag{5.32}$$

with

$$\begin{aligned}
\partial_{r_e}\Big[\phi(r_e, r_{e'})\Big] &= -u_1 \exp(-u_1 r_e)\ \phi(r_e, r_{e'}) \\
\partial_{r_{e'}}\Big[\phi(r_e, r_{e'})\Big] &= -u_2 \exp(-u_2 r_{e'})\ \phi(r_e, r_{e'}) \\
\nabla_{\mathbf{a}} r_e(\mathbf{a}; D_n) &= \nabla_{\mathbf{a}}\rho(\mathbf{x}_\alpha; \mathbf{a}) - \nabla_{\mathbf{a}}\rho(\mathbf{x}_\beta; \mathbf{a}) \\
\nabla_{\mathbf{a}}\rho(\mathbf{x}; \mathbf{a}) &= \mathbf{a}\frac{\sum_{\alpha \in \mathcal{V}_n}\Big(y_\alpha - \rho(\mathbf{x}; \mathbf{a})\Big)\|\mathbf{x} - \mathbf{x}_\alpha\|^2 K(\mathbf{x}_\alpha, \mathbf{x}; \mathbf{a})}{\sum_{\alpha \in \mathcal{V}_n} K(\mathbf{x}_\alpha, \mathbf{x}; \mathbf{a})}.
\end{aligned}$$

The derivatives with respect to the CDN function weights u_1, u_2 are then given by

$$\begin{aligned}
\partial_{u_1}\Big[\mathcal{L}(\boldsymbol{\theta}; D_n)\Big] &= -\sum_{e,e' \in \mathcal{E}_n} r_e \exp\Big(-u_1 r_e\Big)\ \phi(r_e, r_{e'}) \\
\partial_{u_2}\Big[\mathcal{L}(\boldsymbol{\theta}; D_n)\Big] &= -\sum_{e,e' \in \mathcal{E}_n} r_{e'} \exp\Big(-u_2 r_{e'}\Big)\ \phi(r_e, r_{e'}).
\end{aligned}$$

5.4.3 Performance measures for ranking

In order to assess the utility of the proposed framework and to compare the performance of our proposed framework to other methods, we will use the following three metrics commonly in use in information retrieval research:

1. Precision For a given query, the precision at k, or $P(k)$ measures how many of the top-ranking k documents are relevant to the query. More formally,

$$Precision(k) = \frac{\text{Number of relevant documents in the top } k \text{ results}}{k}. \tag{5.33}$$

For the OHSUMED collection, the categories *definitely relevant* and *partially relevant* are both counted as relevant for the purposes of computing the above.

2. Mean Average Precision (MAP) For a given query with K labelled documents, the average precision (AP) is the average of the precision values for all relevant documents:

$$AP = \frac{\sum_{k=1}^{K} P(k) \cdot \mathbb{I}(k^{th}\text{document is relevant})}{\text{Number of relevant documents for the observation}}, \tag{5.34}$$

where $\mathbb{I}[\cdot]$ is equal to 1 if its argument holds true and is 0 otherwise. The MAP score is then computed by averaging the AP scores over all queries.

3. Normalized Discounted Cumulative Gain (NDCG) Recently, this performance measure has been proposed [34] to address the fact that the above metrics only distinguish between *relevant* and *not relevant* categorical labels, although different levels of relevance exist for any given set of documents. In addition, the NDCG also puts more weight on highly relevant documents than marginally relevant ones, as the less relevant a document is, the less likely a user is to examine it. The NDCG metric for a ranked list of K documents with labels $r(j)$ is given by

$$NDCG(k) = Z_k \sum_{j=1}^{k} \frac{2^{r(j)} - 1}{\log_2(j+1)}, \tag{5.35}$$

where Z_k is a normalization constant to ensure that $NDCG(k) = 1$ for the perfect ordering of documents.

5.4.4 The OHSUMED dataset

The OHSUMED collection [24] is a standard benchmark dataset for information retrieval research that is provided as a part of the LETOR 2.0 benchmark [44]. The OHSUMED collection is a subset of MEDLINE, a database on medical publications. The collection consists of 348,566 records (out of over 7 million) from 270 medical journals. The fields of a record include title, abstract, MeSH indexing terms, author, source, and publication type. There are 106 queries, each with a number of associated documents. Each query is

related to a medical search need, and thus is also associated with patient information and topic information. The relevance degrees of documents with respect to the queries are judged by humans, on three levels: *definitely relevant*, *partially relevant* or *not relevant*. There are a total of 16,140 query-document pairs with relevance judgments and 25 query-specific features for each document-query pair including term frequency, document length, BM25 and LMIR features as well as combinations thereof [2, 60, 71].

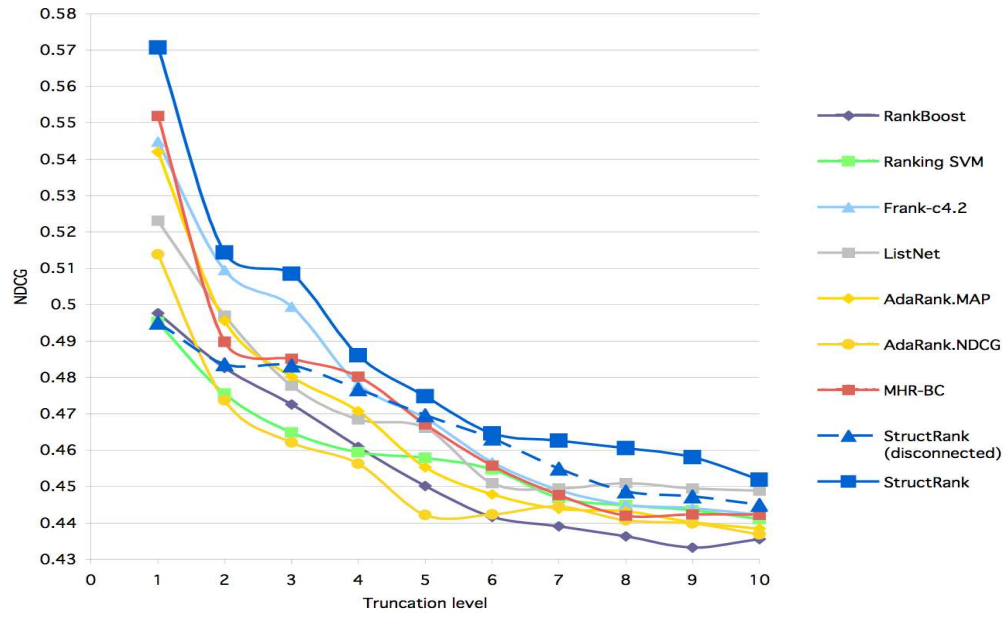
5.4.5 Experimental setup

The OHSUMED dataset is provided in the form of five training/validation/test splits with 63/21/22 observations each. To ensure that features are comparable across all observations, for each observation we normalized each feature vector within the observation so that for feature i,

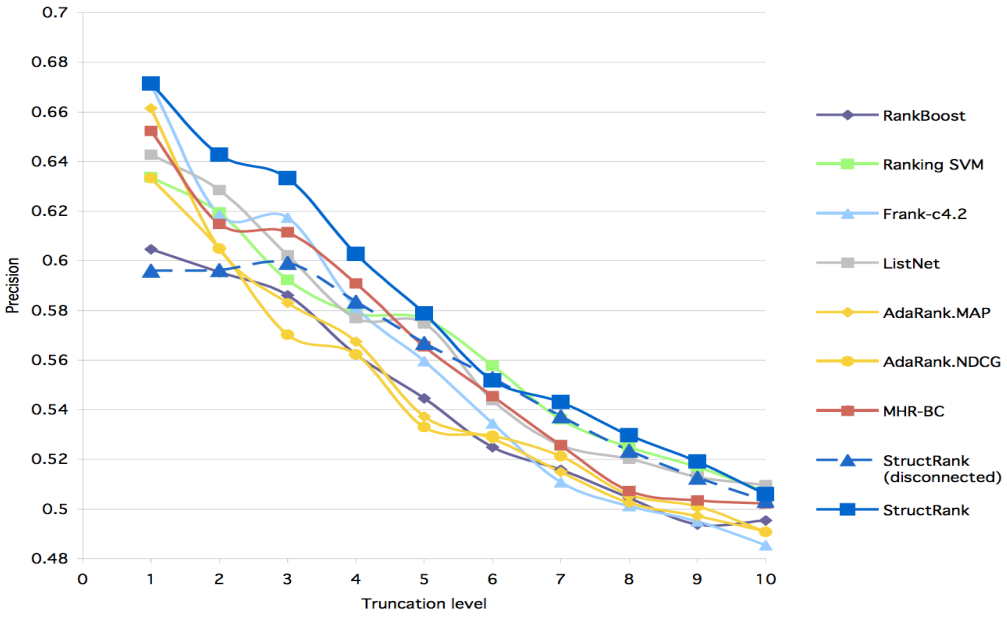
$$\tilde{x}^n_{\alpha,i} = \frac{x^n_{\alpha,i} - \min_{\beta \in \mathcal{V}_n} x^n_{\beta,i}}{\max_{\beta \in \mathcal{V}_n} x^n_{\beta,i} - \min_{\beta \in \mathcal{V}_n} x^n_{\beta,i}}. \tag{5.36}$$

We performed learning of our model using a constrained stochastic gradients algorithm where we prevent updates from violating the inequality constraints in the optimization problem defined by Equation (5.31). In the case where an update is infeasible, our algorithm iteratively reduces the learning rate η until either the update becomes feasible or some maximum number of iterations have passed, at which point the algorithm performs no update. We set the default learning rate to $\eta = 0.5$ and we randomly initialized the model parameters $\mathbf{a}, u_1, u_2$ in the range $[0, 1]$. The above optimization procedure was carried out for 10 epochs and η was scaled by $\frac{1}{\sqrt{2}}$ at the end of each epoch. We performed learning for the range of regularization parameters $t = \{25, 26, \cdots, 40\}$. Once we have finished learning from the training data for a given value of t, we score the documents in the validation set. We then select the ranking function defined by $\mathbf{a}^*$ corresponding to the regularization parameter t^* that maximizes the MAP metric on the validation set. Finally, we score all the documents in the test set using $\mathbf{a}^*$, after which we compute the

above performance metrics.



(a)



(b)

Figure 5.11: a) Average NDCG as a function of truncation level k for the OHSUMED dataset. NDCG values are averaged for test data over five cross-validation splits; b) Average precision as a function of truncation level n for test data averaged over five cross-validation splits.

We tested the CDN in which the joint CDF over preference variables is given by a product over pairwise functions $\phi(r_1, r_2)$ where all pairs of preference variables in the model have a corresponding function node in the CDN graph. We also tested a Bradley-Terry model that models preferences as being pairwise independent. The above performance metrics are shown in Figures 5.11(a), 5.11(b) and 5.12 in addition to the performances of seven methods that are part of the LETOR 2.0 benchmark. With the exception of ListNet and ListMLE, the above methods do not explicitly model statistical dependence relationships between pairwise preferences. As can be seen, accounting for additional statistical dependence relationships between pairwise preferences provides a significant gain in performance compared to modelling preferences as being independent using the disconnected CDN. This can be seen by noting that the problem of ranking contains statistical dependence relationships amongst pairwise preferences, so that the score S_α assigned to one node α can only be determined given the scores of all other nodes. Thus by using a CDN in which preferences are connected to one another, we are able to capture such dependence relationships, whereas in a disconnected CDN we lose the ability to do so.

5.4.6 Discussion

We have described here an application of the structured ranking learning framework to the problem of document retrieval. Here we chose to make use of multivariate sigmoids as CDN functions and a Nadaraya-Watson estimator as a ranking function. It is interesting to note that the hinge loss used by other methods for learning to rank such as that of [22] is a limiting case of a sigmoid function. An example of this for a single preference variable is shown in Figure 5.13. The relationship between the hinge loss and the sigmoidal function suggests that multivariate CDN functions can be viewed as being differentiable analogs of multivariate hinge loss functions. An interesting avenue for future work would involve developing a large-margin analog to the structured ranking learning framework proposed here and comparing this to the structured large margin framework of [66].

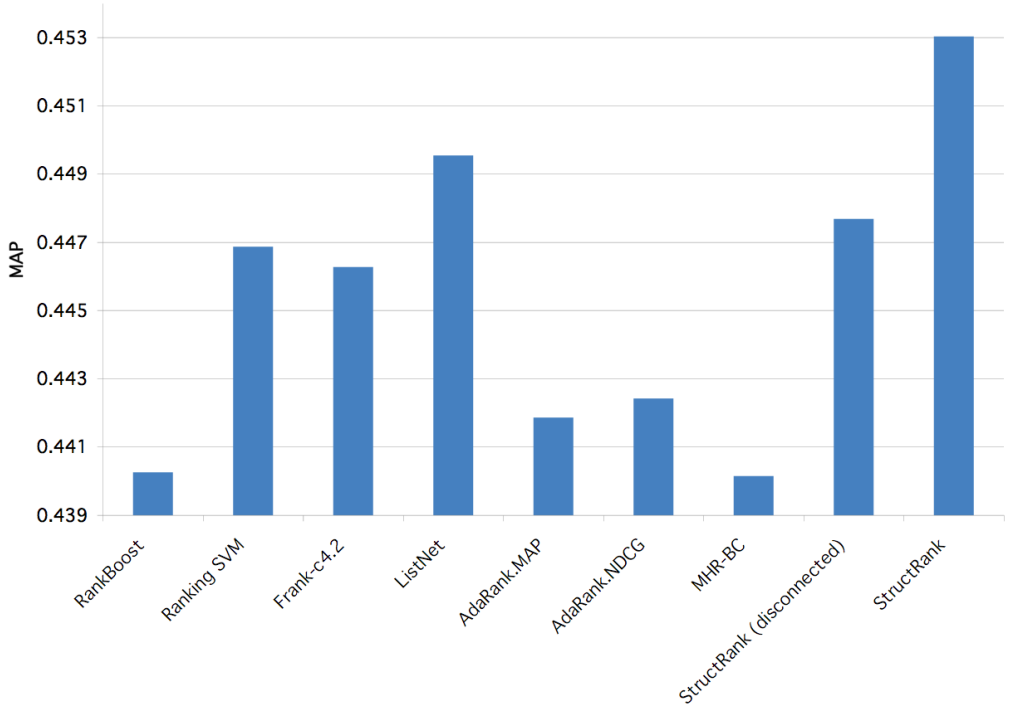


Figure 5.12: Mean average precision value for test data for several methods on the OHSUMED benchmark.

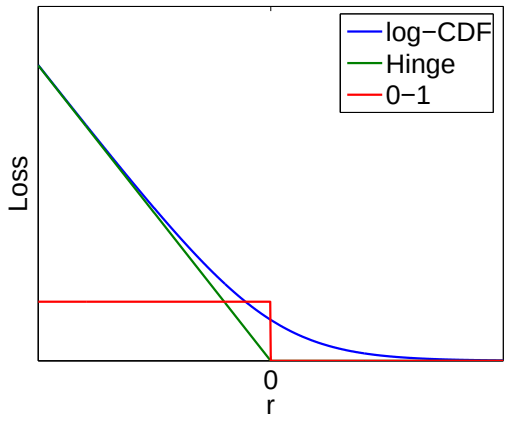


Figure 5.13: Comparison of the 0-1 loss, hinge loss and log-CDF loss for a single preference r, where the CDF is modeled as a sigmoid $\frac{1}{1+\exp(-r)}$.

Having applied the structured learning framework to the problem of document retrieval, we will now turn to the problem of regulatory sequence search in computational systems biology. For this class of problems, the semantics of learning to rank will prove

useful in formulating efficient methods for discovering regulatory sequences using multiple data sources.

5.5 Application: Regulatory sequence search in computational systems biology

A fundamental problem in computational systems biology is to discover the complex roles of biological sequences such as DNA binding sites [11, 61], microRNA target sites [27, 29] and genetic variants [28]. In this setting, we wish to discover some subset of the set of sequences $\mathcal{S}$ that are relevant for a particular biological process given information pertaining to each sequence. Thus the problem of discovering relevant sequences can be viewed as a problem of ranking, to which we have applied the structured ranking framework using CDNs. In this section we will model a sequence as an ordered set of elements of an alphabet $\mathcal{A}$, so that any sequence s of length L satisfies $s \in \mathcal{A}^L$. In the case where the sequence is one of nucleotides, the alphabet is given by $\mathcal{A} = \{A, C, G, T\}$ so that an example of a sequence is $ACCGTGACTG$.

We will assume that each sequence in the set $\mathcal{S}$ may also be provided with additional features. For example, in the case where we wish to discover microRNA targets, a sequence may correspond to the entire 3' untranslated region (3'UTR) for a particular gene, so that one has access to the sequence of the 3'UTR, as well as other features for the 3'UTR sequence. These can include its level of expression across many tissues, the abundance of proteins that are translated from the sequence preceding the 3'UTR and the expression of a microRNA that putatively targets a site in the 3'UTR sequence. An example of how such features would be extracted is given in Figure 5.14 where we have assigned a node α for the pair $s_\alpha, \mathbf{x}_\alpha$. Thus we can assume that each node α has a corresponding sequence $s_\alpha \in \mathcal{S}$ and a corresponding set of features $\mathbf{x}_\alpha$. The features $\mathbf{x}_\alpha$ can include quantitative features such as the target sequence accessibility [38], the

neighboring context of sequences [20] or other quantitative profiling data [11, 15, 27, 61, 65]. The problem we wish to solve here consists of combining features from multiple types and sources of data for the purpose of discovering some subset of the regulatory sequences $\mathcal{S}$ that are relevant for a given biological process. In addition to accounting for diverse features, incorporating the large number of computational predictions already available is also desirable.

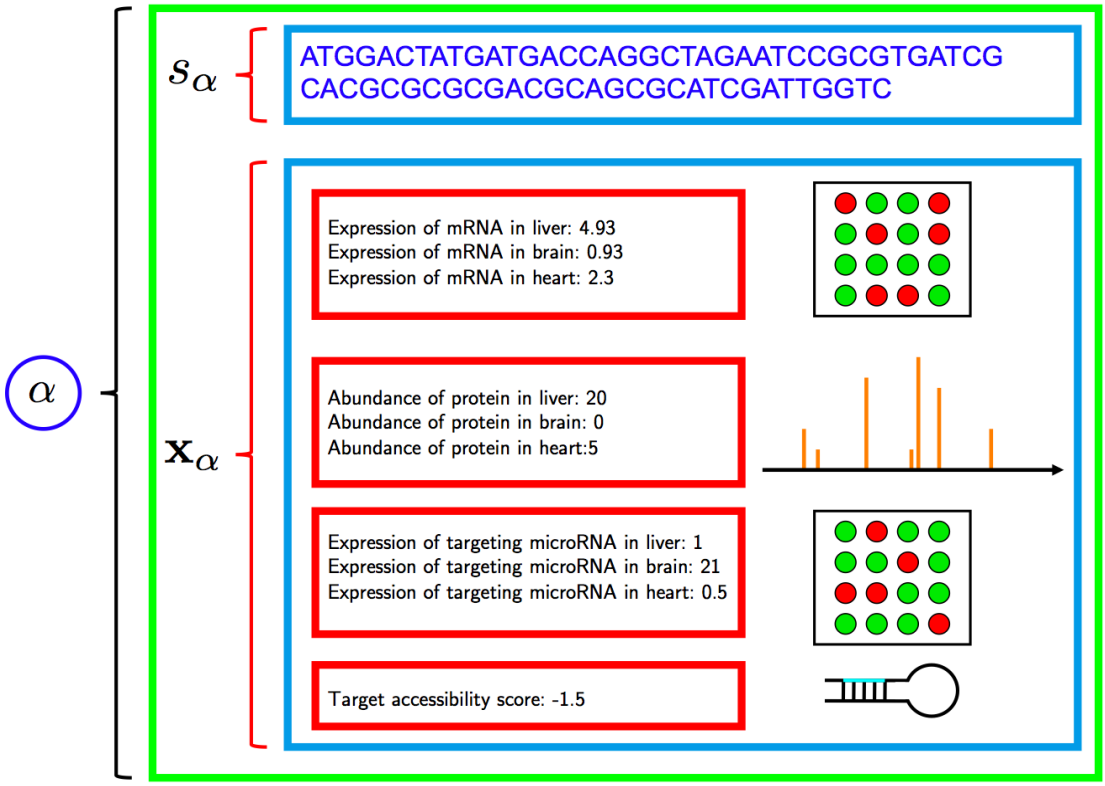


Figure 5.14: Feature extraction for nodes in the order graph. Each node α has a corresponding sequence s_α and a set of corresponding features that are relevant to ranking the sequence. For the example shown, the sequence s_α may correspond to the sequence for the entire 3' untranslated region (3'UTR) of a gene, so that the feature vector $\mathbf{x}_\alpha$ include the expression of the gene carrying the sequence, the abundance of protein produced from the coding region for the gene carrying the sequence and the expression of a putative microRNA that targets the sequence.

5.5.1 Previous work

In recent years, many different methods have been proposed to address the problem of integrating together heterogeneous datasets in the context of discovering regulatory sequences. In particular, probabilistic generative models have been proposed in which sequence discovery consists of inference and learning [27, 61] given sequence and expression data. While such methods do explicitly model the impact of sequences on gene expression in the presence of many latent random variables, a major challenge is to account for newer datasets as well as new sources of regulatory variability. Each additional dataset to be analyzed is likely to introduce a significant number of additional parameters and hidden variables, dramatically increasing the cost and complexity of inference and learning. As the number of types and sizes of data continue to increase, it is likely that both model misspecification and prohibitive computational complexity will hamper the practicality of probabilistic latent variable models. Owing to the difficulty in developing purely sequence-based models of regulatory sequences, a major challenge is then to incorporate additional data types under a unified tractable and principled framework.

5.5.2 Discovering regulatory sequences as a problem of learning to rank

A strategic approach to the above problem can be obtained by noting that the problem of discovering regulatory sequences is inherently a problem of learning to rank, whereby we are given a large number of candidate sequences and only some relatively small number are of biological significance. Furthermore, there is often a well-defined notion of preference between sequences. An example of this arises in the binding of transcription factors to binding sites, whereby some sites are more strongly bound than other sites by certain transcription factors. Thus when discovering sequences, it is desirable to explicitly model the fact that sequences do not fall into two distinct categories of positives and negatives

but instead have different degrees of significance attached to them, so that a plausible model should assign a higher score for sequences with higher importance.

Some methods have in fact formulated the problem of discovering sequences as one of ranking sequences, or *sequence search*, so that they assign a score to each sequence with the assumption that high-scoring sequences are more likely to be biologically relevant than low-scoring ones. The idea of discovering sequences using an explicit ranking formulation has been explored previously by [4, 11, 65] in the context of using the orderings obtained from microarray intensities to learn position-specific scoring matrices (PSSMs) for transcription factor binding sites. This was shown to significantly improve predictive accuracy with respect to other model-based methods, as no assumptions on the functional relationship between intensities and sequences needed to be made in order to learn to rank sequences. The improved accuracy of such ranking-based methods with respect to model-based methods suggests that a good method for discovering sequences would be one specifically tailored to the problem of learning to rank.

Given the above methods for ranking sequences, our goal in this section is to expand on previous work along three fundamental directions. First, we address the structured nature of the problem of ranking, since the rank of one sequence can only be determined given the ranks of all sequences so that complex statistical dependence relationships exist between variables in the model. Second, the scoring function used by the previous methods of [4, 11, 65] was parameterized by a PSSM and so was restricted to sequence inputs. Here we will expand on this idea to allow for rich feature spaces obtained from quantitative measurements such as expression profiling data. Lastly, by formulating the sequence search problem as one of ranking, we can leverage information across several experimental datasets and diverse prediction methods via the orderings over sequences that each provides. Under the framework of learning to rank, orderings provided by diverse computational methods and those provided by experimental data are all comparable and readily accounted for, even if scores and labels between different prediction methods and

datasets may be difficult to compare. Thus, given that we observe many different partial orderings provided by diverse datasets and prediction methods, our aim will be to predict orderings over sequences so that sequences that are often highly ranked across observations should also be highly ranked by our method. The proposed framework of ranking then offers three significant advantages over previous model-based approaches for sequence search. First, as in [4, 11, 65], it makes minimal assumptions about the relationships between sequences and measured/predicted labels for the sequences and so largely avoids the issue of model misspecification. Second, it allows us to leverage orderings provided by heterogeneous datasets and prediction methods that may have little overlap with one another in the sequences they contain, but may nevertheless informative when combined together under a single model. Lastly, predictive accuracy is improved by explicitly modelling the statistical dependence relationships involved in learning to rank.

To model the structured nature of the ranking problem, we can take advantage of the structured ranking learning framework. In the context of discovering sequences, we can then interpret a set of prediction methods and a set of experimental measurements as observations that convey partial orderings over some subset of the sequences of interest. Thus we present STORMSeq, a method formulated using the structured ranking learning framework that scores sequences given a set of features and a set of orderings over subsets of the sequences to be ranked. The outline of the method is shown in Figure 5.15. Our method expands the RankMotif++ method of [11] to a structured learning setting where we can A) account for the statistical dependence relationships in the problem of ranking, B) we can incorporate rich feature spaces such as quantitative measurements of mRNA and protein expression in addition to sequence data, and C) we can account for diverse computational prediction methods as additional sources of data for learning. We will apply the proposed framework to the problems of scoring transcription factor binding sites and microRNA targets, although the framework is general enough to be applied

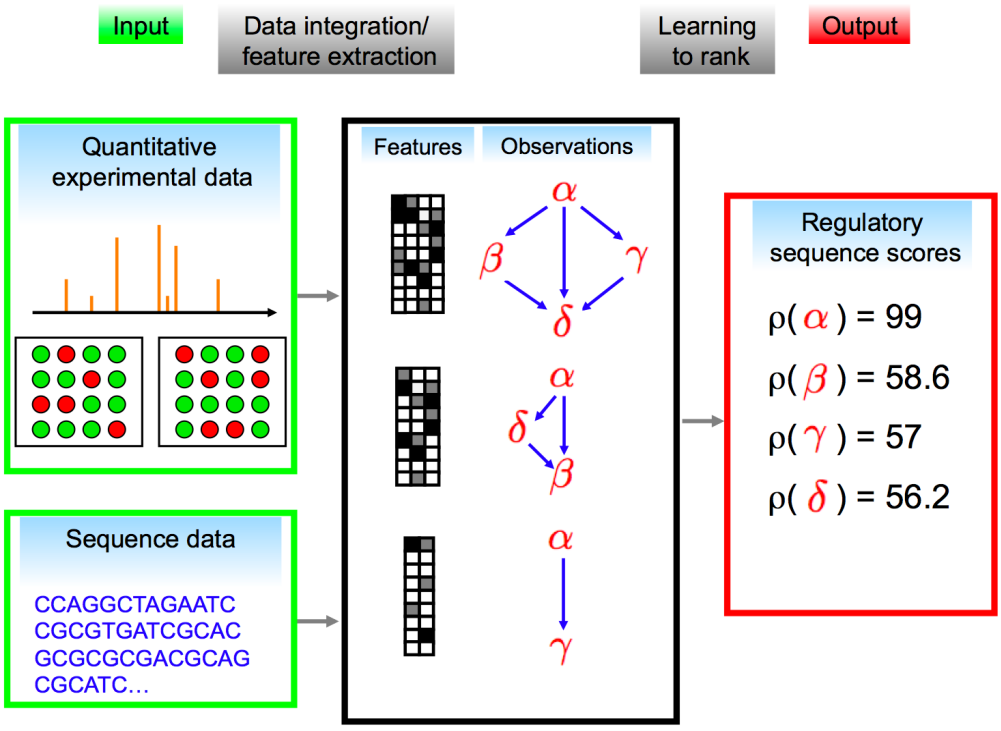


Figure 5.15: The STructured ranking of Regulatory Motifs and Sequences (STORMSeq) method. Given multiple independent observations conveying various orderings over sequences and given the observed sequences and input features extracted for each observation (e.g.: mRNA, microRNA and protein measurements, sequence context features), STORMSeq learns a ranking function such that the probability of generating the observed orderings is maximized.

to a wide variety of bioinformatics problems, such as ranking therapeutic drug targets, finding genetic associations or scoring protein-protein interactions.

5.5.3 STORMSeq: STructured ranking of Regulatory Motifs and Sequences

Suppose that we are given a set of N observations $\mathcal{D} = \{D_1, \cdots, D_N\}$, where each observation D_n provides an ordering of the sequences in some subset $\mathcal{S}_n \subseteq \mathcal{S}$. Here, an observation corresponds to any set of quantitative values that convey some meaningful

ordering of the sequences to be ranked. For example, in the context of scoring sequences bound by transcription factors, orderings might be provided by microarray intensities [4], which provide a measurement of the concentration of transcription factors binding to any given sequence. For a given observation, we can then represent the ordering of sequences as an order graph as described previously. The structured ranking learning proposed in Section 5.3 can then be directly applied in which we learn a ranking function $\rho(\alpha)$ from the observations $\mathcal{D} = \{D_1, \cdots, D_N\}$. In the next section we will describe how the ranking function ρ can be made to account for additional information that may be provided for each sequence to be ranked,.

5.5.4 Ranking using sequence and quantitative features

In order to adapt the structured ranking learning framework to the problem of ranking sequences where we are given both sequence information s_α and a feature vector $\mathbf{x}_\alpha$, the ranking function $\rho(\alpha)$ will be given the general form

$$\rho(\alpha) = \rho_{seq}(s_\alpha; \mathbf{M}) + \rho_{quant}(\mathbf{x}_\alpha; \mathbf{w}), \tag{5.37}$$

where ρ_{seq}, ρ_{quant} are functions that assign scores to the sequence s_α and its corresponding feature vector $\mathbf{x}_\alpha$. Here, it is possible to specify different parametric forms for $\rho(\alpha)$ that assign scores to sequences under various assumptions. In order to score any given node α based on sequence s_α alone, we will consider the sum of contributions of subsequences of s_α under the assumption that each subsequence contributes independently to the overall score for s_α. Suppose we are given a sequence s_α of length L_α. Let $s_\alpha^{k:k+K-1}$ be a subsequence of length K of s_α starting at position k of s_α and let $s_\alpha^j \in \mathcal{A}$ be the symbol observed at position j in sequence s_α. Given a *position-specific scoring matrix* (PSSM) $\mathbf{M} \in \mathbb{R}^{K \times |\mathcal{A}|}$ of length K where $M_{k,a}$ is equal to the probability of emitting symbol $a \in \mathcal{A}$ at position k of the PSSM, we can define the score for s_α as the probability of emitting

at least one subsequence of length K in s_α, so that

$$\rho_{seq}(s_\alpha; \mathbf{M}) = \log \left(1 - \prod_{k=0}^{L_\alpha - K + 1} (1 - P(s_\alpha^{k:k+K-1} | \mathbf{M})) \right), \tag{5.38}$$

where $P(s_\alpha^{k:k+K-1}|\mathbf{M}) = \prod_{j=k}^{k+K-1} M_{j,s_\alpha^j}$ is the probability of binding to subsequence $s_\alpha^{k:k+K-1}$ according to the distribution specified by $\mathbf{M}$. In the case where we are provided with quantitative features in the form of a feature vector $\mathbf{x}_\alpha$, we can define the ranking function $\rho_{quant}(\mathbf{x}_\alpha; \mathbf{w})$ to be a linear function of the quantitative features, so that $\rho_{quant}(\mathbf{x}_\alpha; \mathbf{w}) = \mathbf{w}^T \mathbf{x}_\alpha$. Thus the above parameterizations for the ranking functions $\rho_{seq}(s_\alpha; \mathbf{M}), \rho_{quant}(\mathbf{x}_\alpha; \mathbf{w})$ allow us to account for the influence of sequence information in addition to quantitative features on the score to be assigned to any given sequence.

To address the problem of learning the functions ρ_{seq}, ρ_{quant}, we will use the stochastic gradient descent algorithm which was also applied to the document retrieval task. For the function ρ_{seq}, the derivative of the ranking function $\rho_{seq}(s_\alpha; \mathbf{M})$ with respect to the parameter $M_{k,a}$ is given by

$$\frac{\partial \rho_{seq}(s_\alpha; \mathbf{M})}{\partial M_{k,a}} = \frac{1 - \exp\left(\rho_{seq}(s_\alpha; \mathbf{M})\right)}{\exp\left(\rho_{seq}(s_\alpha; \mathbf{M})\right)} \left(\sum_i \frac{P(s_\alpha^{i+1:i+K}|\mathbf{M})}{1 - P(s_\alpha^{i+1:i+K}|\mathbf{M})} \left([s_\alpha^{i+k} = a] - P(a|\mathbf{M}) \right) \right). \tag{5.39}$$

For the ranking function ρ_{quant}, the gradient is given simply by $\nabla_{\mathbf{w}} \rho_{quant}(\mathbf{x}_\alpha; \mathbf{w}) = \mathbf{x}_\alpha$. Once we have computed both of the above gradient vectors, we can form the gradient of the ranking function ρ as

$$\nabla_{\boldsymbol{\theta}} \rho(\alpha; \boldsymbol{\theta}) = \begin{bmatrix} \nabla_{\mathbf{M}} \rho_{seq}(s_\alpha; \mathbf{M}) \\ \nabla_{\mathbf{w}} \rho_{quant}(\mathbf{x}_\alpha; \mathbf{w}) \end{bmatrix}, \tag{5.40}$$

where $\boldsymbol{\theta}$ is the parameter vector whose elements correspond to elements of $\mathbf{M}$ and $\mathbf{w}$. Thus given the above parameterizations, a sequence s_α will have a higher score if both ρ_{seq} and ρ_{quant} assign high scores based on sequence s_α and features $\mathbf{x}_\alpha$ respectively. Given the above gradient vector, we can then estimate $\boldsymbol{\theta}$ in addition to the CDN function parameters using a stochastic gradient descent algorithm where we iteratively update the parameter vector $\boldsymbol{\theta}$ for each observation in our training set.

To summarize, given a training set of observations consisting of orderings over sequences to be ranked and their associated quantitative features, the goal is to learn a ranking function $\rho(\alpha)$ that maximizes the probability of generating the observed orderings by assigning higher scores to those sequences that are most consistently highly ranked in the training set of observations. The ranking function accounts for sequence data in addition to other quantitative features such as expression measurements. The structured loss functional then allows us to account for the statistical dependence relationships between preference variables. We emphasize at this juncture that STORMSeq has been formulated in a general way so that it is applicable to several problems in which we wish to learn a ranking function for sequences using both multiple instances of orderings, sequence data and other quantitative features. To illustrate how STORMSeq might be used in practice, we will apply the proposed structured ranking learning framework to the problem of searching for transcription factor binding sites. Before we proceed, it will be instructive to study the relation between STORMSeq and a previous method for learning to rank sequences from orderings over sequences obtained from microarray measurements.

5.5.5 The RankMotif++ model as a cumulative distribution network

It is worth noting that in the RankMotif++ model of [11], the objective being minimized corresponds to the log of a joint CDF over preferences, under the assumption that the preferences are independent. More precisely, in RankMotif++ the loss function is given by $\mathcal{L}(\boldsymbol{\theta}) = \log F_{\boldsymbol{\pi}}(\mathbf{r}(D_n))$, where the probability over all pairwise preferences $\alpha \succ \beta$ is modeled by a product over logistic functions of $r_{\alpha\beta} = \rho(\alpha) - \rho(\beta)$ so that

$$F_{\boldsymbol{\pi}}(\mathbf{r}(D_n)) \equiv \mathbb{P}\Big[\boldsymbol{\pi} \leq \mathbf{r}(D_n)\Big] = \prod_s \frac{1}{1+\exp(-\nu r_s)} = \prod_{\alpha\succ\beta} \frac{1}{1+\exp\Big(-\nu\Big(\rho(\alpha)-\rho(\beta)\Big)\Big)}, \tag{5.41}$$

with $\rho(\alpha) = \rho_{seq}(s_\alpha)$ and $\nu > 0$. We have previously identified probability models of this type as Bradley-Terry models (Section 5.3.2), so that the above loss function

can be modeled using a disconnected CDN where each function node corresponds to $\phi_s(r_s) = \left(1 + \exp(-\nu r_s)\right)^{-1}$ and all pairwise object preferences are modeled as being independent of one another.

5.5.6 Discovering transcription factor binding profiles

Transcription factors (TFs) are of significant interest in molecular biology and immunology, as they consist of proteins that bind to specific nucleotide sequences in order to regulate the activity of genes carrying these sequences [11]. Due to the short nature of these regulatory sequences in comparison to the size of a typical genome, the problem of discovering transcription factor binding sites remains a significant challenge in computational molecular biology. Here we will apply the proposed structured ranking learning framework to the problem of ranking sequences bound by TFs using measurements from protein binding microarray (PBM) experiments. In this problem, we will make use of sequence data alone so that a fair comparison can be made with existing methods for discovering regulatory motifs from PBM data using sequence.

Data processing

We obtained PBM data from the Supplementary Material section of [4], which consisted of intensity measurements for 35-mer nucleotide probes bound by five different transcription factors Cbf1, Ceh-22, Oct-1, Rap1, Zif268 across two experimental replicate arrays *Array 1* and *Array 2*. The PBM data consisted of intensity measurements y_α for a set of sequences $s_\alpha \in \mathcal{S}$, where each probe on the array is indexed by α, so that s_α denotes the nucleotide sequence of a given probe on the array. Thus for this problem the set $\mathcal{A} = \{A, C, G, T\}$. We used the array labeled *Array 1* as our training data and the probe measurements from *Array 2* as test data. We normalized the microarray intensity data in both sets by first shifting microarray intensities such that the minimum intensity was equal to one, then applying a log-transformation, as was done in [11]. We labelled the

250 probe sequences that had the highest measured intensity as positives and the 250 sequences with the lowest normalized intensities as negatives. We then constructed the order graph over these 500 sequences based on preferences assessed using the criteria used by [11]. Briefly, we computed the median absolute deviation (MAD) m of the 500 normalized intensities and asserted $\alpha \succ \beta$ if $y_\alpha > y_\beta + 3\sigma$ and at least one of s_α, s_β were labelled as positive sequences as described above, where $\sigma = m/0.6745$ and 0.6745 is the MAD of the standard normal distribution. Given order graphs constructed in this fashion, the goal is to then learn a ranking function that assigns scores under the assumption that higher scores should indicate an increased affinity of a TF for a given sequence.

Experimental setup

Using the sequence ranking function $\rho_{seq}(s_\alpha; \mathbf{M})$ for a fixed PSSM of length K, we ran STORMSeq and RankMotif++ using three random initializations each, whereby we then selected the model that maximized the Spearman correlation with the training data as was done in [11]. For each initialization, the PSSM $\mathbf{M}$ was initialized to a set of random positive values and then normalized so that $\sum_{a \in \mathcal{A}} M_{k,a} = 1 \; \forall \; k = 1, \cdots, K$. The MatrixREDUCE, MDScan and Prego methods [15, 45, 65] were then applied to the training data using the parameters specified in [11] and the resulting PSSM models were selected using the same Spearman correlation criterion as above. For all models, we varied K from 7 to 13 and selected the value of K that optimized the above Spearman correlation criteria. We ran STORMSeq for 100 epochs using the constrained stochastic gradients optimization method from Section 5.4.2. The learning rate was set to $\eta_t = 0.1$ with a decay rate of $1/t$ at the end of each epoch t. Our loss functional was chosen to be the one proposed in Equation (5.18) with a CDN containing pairwise functions $\phi(r_1, r_2)$ as in the previous information retrieval application where all preference variables are connected to one another via the functions $\phi(r_1, r_2)$. Here we imposed the additional constraint on the

CDN parameters so that $u_1 = u_2 = \nu$. In order to provide regularization on the CDN parameter ν, we set the constraint $0 < \nu \leq 1$. We also enforced the constraints that $0 < M_{k,a} < 1$ for all $k = 1, \cdots, K$ and $a \in \mathcal{A}$ and $\sum_{a \in \mathcal{A}} M_{k,a} = 1$.

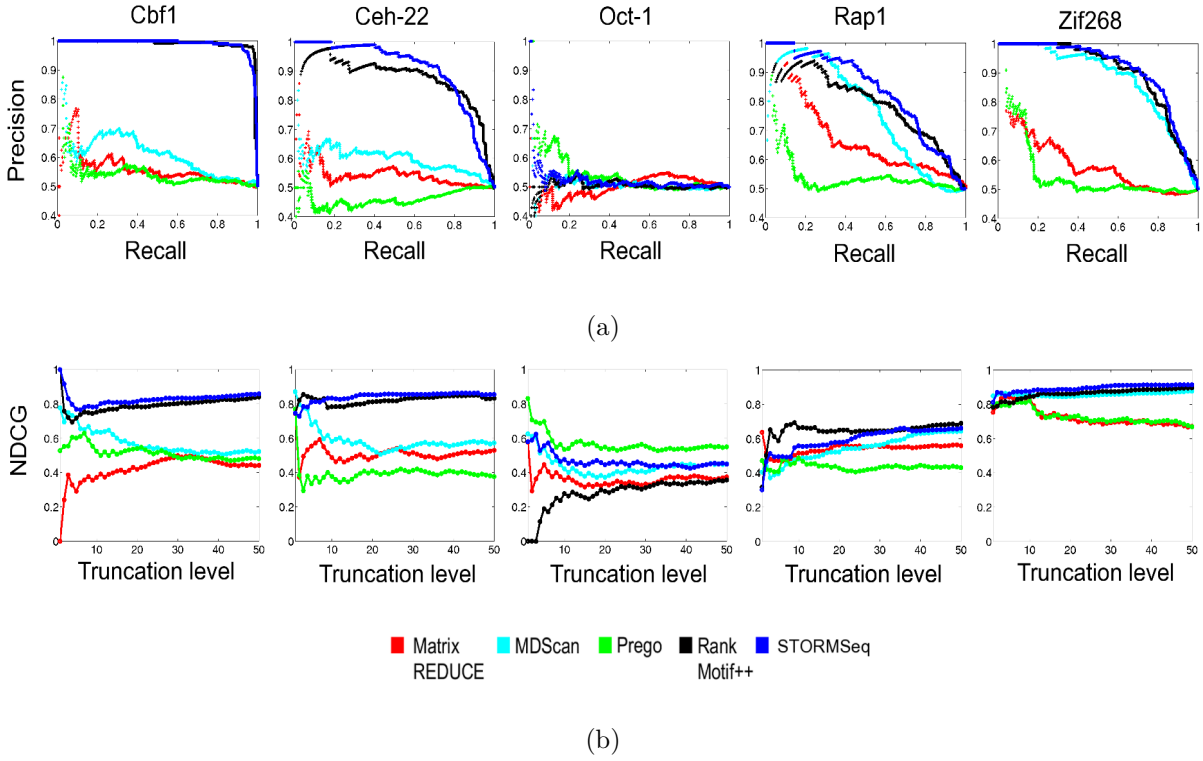


Figure 5.16: a) Out-of-sample precision versus recall using five different methods for the Cbf1, Ceh-22, Oct-1, Rap1, Zif268 transcription factors studied in [4, 11]. The methods shown are MatrixREDUCE (red), MDScan (cyan), Prego (green), RankMotif++ (black) and STORM-Seq (blue); b) The corresponding curves showing Normalized Discounted Cumulative Gains (NDCG) versus the truncation level, or the number of top-ranking sequences. Both a) and b) show that by ranking in a structured learning setting using STORMSeq, we generally improve predictive accuracy, in terms of precision, recall and NDCG, with respect to the other unstructured learning methods shown here.

The performance of all five methods for the above five TFs are summarized in Figures 5.16(a) and 5.16(b) using precision/recall for assessing performance We also assessed predictive performance using the NDCG metric for ranking that we used in the document

retrieval task. The use of the NDCG metric here is well-suited to the problem at hand, as the truncation level n can be interpreted as the number of sequences to be further validated or analyzed, so that a higher NDCG value is obtained if the most significant sequences appear at the top of the list in the correct order of significance. In the context of PBM array experiments, the significance of a sequence is determined by the strength with which a transcription factor binds to it, so that the highest score should be assigned to the most strongly bound sequence. In addition to the these plots, we also show the Area Under the precision-recall Curve (AUC) in the heatmap display of Figure 5.17.

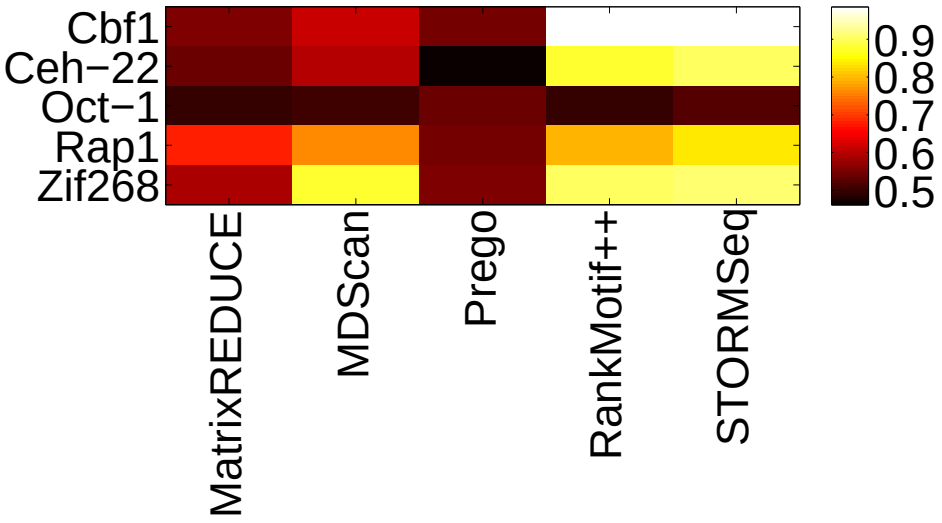


Figure 5.17: Heatmap of the Area Under the precision-recall Curve (AUC) for the five transcription factors and for the five methods.

The corresponding PSSMs found by each of the above methods are shown in Figure 5.18. As can be seen, the PSSMs learned by STORMSeq are generally consistent with those found by the other methods as well as with PSSMs previously reported for this dataset [4, 11]. Figures 5.16(a), 5.16(b) and 5.17 demonstrate that by ranking in a structured setting and by making no particular assumption about the relationship between sequence PSSMs and measured PBM intensities, we can increase predictive accuracy as

measured by precision, recall and NDCG compared to the other unstructured prediction methods such as RankMotif++. In particular, the AUC's achieved by our method exceeds that of RankMotif++ for four of the five transcription factors and equals its performance for the Cbf1 transcription factor. Furthermore, according to the NDCG metric, our method of ranking also has increased accuracy in terms of the ranking itself, so that sequences with higher intensities are more likely to be ranked higher by STORMSeq than by the other models.

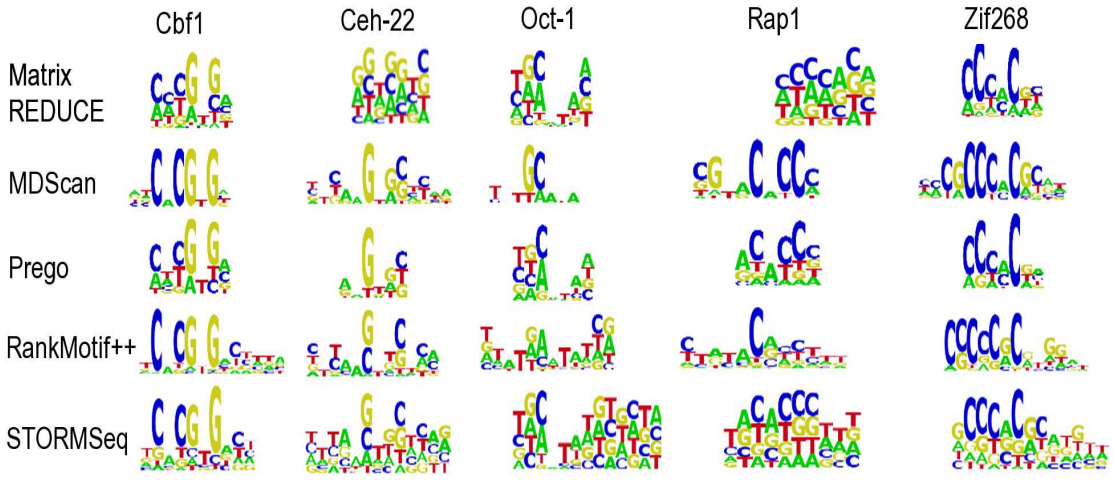


Figure 5.18: Position weight matrices found by the MatrixREDUCE, MDScan, Prego, RankMotif++ and STORMSeq methods (rows) for each of the five transcription factors Cbf1, Ceh-22, Oct-1, Rap1, Zif268.

Having applied the structured ranking learning framework to the problem of discovering transcription factor binding sites, we will also demonstrate the usefulness of STORMSeq for discovering microRNA targets, which also consist of short nucleotide sequences that regulate the activity of genes.

5.5.7 Discovering microRNA targets

MicroRNAs consist of molecules of 22-25 nucleotides that target mRNA transcripts through complementary base-pairing to short target sites, in a fashion analogous to the

operation of transcription factors. However, unlike transcription factors, microRNAs are generally inhibitory in their activity, so that microRNA activity generally represses the activity of their target genes either by reducing the abundance of their target mRNA transcripts or by repressing translational activity of their target mRNAs [1, 27]. There is substantial evidence that microRNAs are an important component of the cellular regulatory network, providing a post-transcriptional means to control the amounts of mRNA transcripts and their protein products [1, 20, 27, 38]. As a consequence of their important role in gene regulation, many previous methods have been proposed for finding the targets of microRNAs [20, 27, 33, 40]. In this section, we will formulate the problem of finding microRNA targets as one of ranking and we will apply the structured ranking learning framework for the let-7b microRNA and its putative targets. In addition to sequence data, we will make use of mRNA, microRNA and protein abundance measurements in order to rank microRNA-target interactions.

Data processing

Here we will use a dataset profiling the expression of human mRNAs after transfection of a synthetic RNA duplex of the mature let-7b hairpin into WERI-Rb1 retinoblastoma samples [27]. Under the assumption that microRNA regulation causes a reduction in mRNA expression, pairwise preferences between sequences were asserted using the MAD criterion used for the above analysis of transcription factors, but using *negative* log-expression-ratios of expression from the let-7b transfections instead. Thus the score assigned to a sequence by a good model should correlate with the amount of down-regulation by let-7b.

We focused here on the human genes in the WERI-Rb1 transfection experiment of [27] that A) had 3'UTR sequence information provided by Ensembl and B) were provided with both mRNA expression and protein abundance data in 3,636 paired mRNA-protein expression profiles obtained from cDNA microarray and mass-spectrometry across brain,

heart, liver, lung and placenta tissue pools in mouse [39, 72]. This yielded a total of 799 human 3'UTR sequences that satisfied the above desiderata. We then selected the 400 sequences with the lowest log-expression ratios as positives and labelled the other 399 genes as negatives. As with the preceding analysis of transcription factor binding sites (Section 5.5.6), we wish to assess the out-of-sample predictive performance of our method, so that we selected a random sample of 250 positive sequences for our training data and the remainder for the test data. Similarly, we selected 250 sequences from the negative group for our training set and the rest for the test data. We thus formed five independent training/test splits in this fashion.

In contrast to the previous problem, where we had relatively few sources of data variability, here we are provided with *in vivo* expression measurements of genes that may have several different regulators, some of which may themselves be regulated by let-7b. The problem of scoring microRNA targets is thus an example of the type of the problem commonly encountered in genomics, where the goal is to discover sequences in the presence of many sources of *in vivo* regulatory variability. The hypothesis here is that we can leverage additional information in the form of independent quantitative measurements and computational predictions in order to better account for the variability in orderings over sequences.

To learn to rank microRNA targets, we will use human 3'UTR sequence data, mouse mRNA expression [72], mouse let-7b expression [1] and mouse protein abundance data [39] across brain, heart, liver, lung and placenta tissue pools, whereby the mouse mRNAs being profiled are homologs of the human mRNAs from the WERI-Rb1 assay. Furthermore, the expression for the let-7b microRNA in the above tissue pools corresponds to that of mouse homolog for let-7b. Expression and abundance values from the above three datasets were processed according to the guidelines in [1, 39, 72].

In addition to expression features, we would also like to account for other contextual sequence features, such as microRNA site accessibility as measured by the energy score

$\Delta\Delta G$. To this end, we ran the algorithm of [38] for computing $\Delta\Delta G$ for each 3'UTR sequence given the mature let-7b sequence using the default algorithm settings provided by the authors of [38]. To obtain a $\Delta\Delta G$ score for the entire 3'UTR, we summed the $\Delta\Delta G$ scores for each putative microRNA target site in the 3'UTR. Combined with the above mRNA, microRNA and protein abundance features, this yielded a total of 16 quantitative features for each sequence to be scored. Thus for this problem, each 3'UTR sequence corresponds to a putative let-7b-target interaction so that let-7b putatively targets at least one target site in the 3'UTR sequence. The above 16 features thus form the feature vector $\mathbf{x}_\alpha$, which we will use for learning to rank targets.

Incorporating diverse computational predictions

In addition to the above features, we would like to also incorporate computational target predictions for let-7b from the PicTar [40], TargetScan [20] and RNA22 [33] sequence-based methods. Each of these methods makes use of various criteria such as conservation and contextual sequence features to assign scores to candidate microRNA targets. The scores output by any of these prediction methods can be thus used to generate an order graph over sequences, so that each method provides a partial ordering over some subset of microRNA-target interactions. The proposed framework of structured ranking learning then allows us to combine diverse computational predictions under one unified probabilistic framework.

We downloaded computational microRNA target predictions for let-7b from the Supplementary Data resources for the TargetScan [20], PicTar [40] and RNA22 [33] algorithms. Preferences were established between 3'UTR sequences by summing over microRNA-target site scores within the given 3'UTR sequence for a given prediction method and then comparing total scores between 3'UTR sequences. Thus, for a given computational prediction method, the preference $\alpha \succ \beta$ was established between two 3'UTR sequences s_α, s_β if s_α had a higher score than s_β according to the prediction

method and at least one of s_α, s_β were labelled as a positive sequence as described above.

For each of the five training datasets, we ran STORMSeq with $K = 7$ and selected the best model out of three random restarts via the Spearman correlation between the learned ranking function scores and the orderings seen in the training data. We used the same experimental configuration as in the case where we ranked sequences using PBM data, with the additional constraint on weights $\mathbf{w}$, so that we set an additional L_1-norm constraint of $\|\mathbf{w}\|_1 \leq 50$. As before, the loss functional was chosen to be the one proposed in Equation (5.18) with a CDN containing pairwise functions $\phi(r_1, r_2)$ connecting all preference variables to one another, with the additional constraint on the CDN parameters so that $u_1 = u_2 = \nu$.

Given all of the above, we applied STORMSeq under three settings, where A) we use only sequence information, B) we use both sequence and quantitative features (mRNA and microRNA expression, protein abundance and $\Delta\Delta G$) and C) we use both quantitative features and sequence in addition to information provided by diverse computational prediction methods. For each of the five train/test datasets, we computed precision and recall for each of these experimental settings. The resulting precision and recall curves, averaged over the five test sets, is shown in Figure 5.19. As can be seen, incorporating sequence data, quantitative features and computational predictions yields an improvement in predictive accuracy compared to using sequence alone or sequence in tandem with quantitative features. This indicates that by leveraging multiple sources of information about microRNA regulation, we can significantly increase the accuracy with which we discover microRNA targets. In order to establish a reference method with wh9ich to compare the above settings for STORMSeq, we also applied a simple technique for scoring 3'UTR sequences by counting the occurrence of different 7-mers in each sequence and then ranking the 3'UTRs by the counts. Thus we examined counts of all possible 7-mers for the 3'UTRs in each training set, then selected the 7-mer which achieved the best Spearman correlation with the mRNA expression measurements in the training

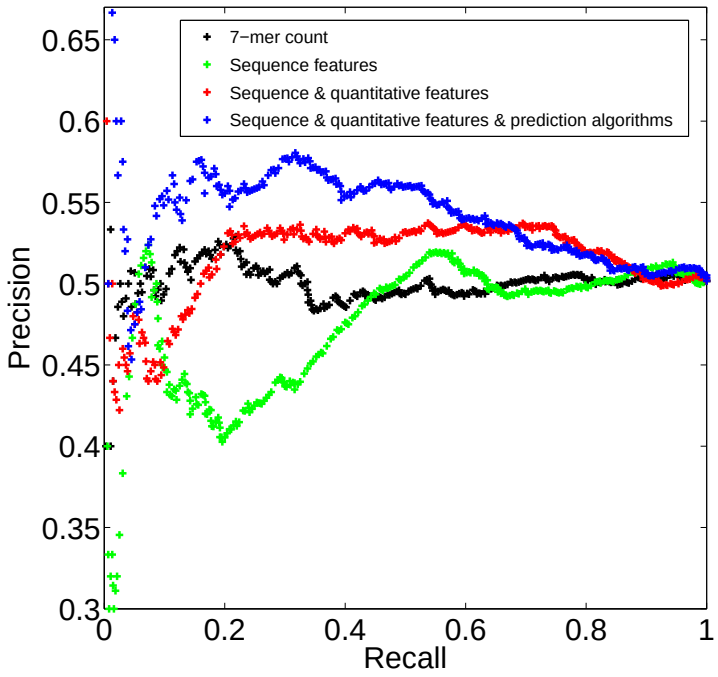


Figure 5.19: Average out-of-sample precision versus recall for different STORMSeq learning configurations using expression data for mRNAs in response to let-7b transfection [27]. Curves are shown as the average precision and recall over five training and test data sets. By incorporating additional sources of sequence information, sequence context and quantitative profiling features, STORMSeq achieves higher accuracy (blue) than using sequence data alone (green), sequence data combined with quantitative features without computational predictions as additional data (red), or by using counts of 7-mers in the 3'UTR sequences (black).

data. The average precision and recall of the resulting 7-mer on test data, averaged over five test datasets, is shown in Figure 5.19. Here we see that while this simple technique can outperform STORMSeq using only sequence information, but underperforms with respect to STORMSeq using additional quantitative features and prediction methods.

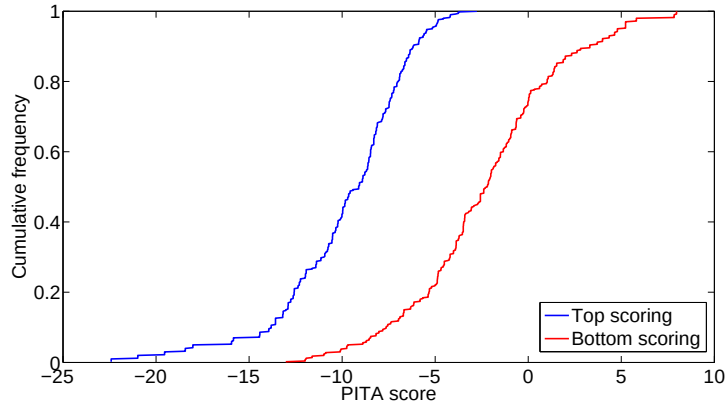
As additional validation, we show the cumulative distribution of $\Delta\Delta G$ scores between the let-7b microRNA and target mRNAs in our dataset for the top and bottom 100 targets ranked according to STORMSeq (Figure 5.20(a)). We expect *a priori* that sequences with lower $\Delta\Delta G$ score are more likely to be bound by a targeting microRNA than not. As

can be seen, high-scoring targets have a significantly lower average $\Delta\Delta G$ value than low-scoring targets ($P < 10^{-20}$, Wilcoxon-Mann-Whitney), demonstrating that the targets discovered by STORMSeq are likely to be genuinely targeted by let-7b. Furthermore, the protein abundances for the top and bottom 100 targets differed significantly ($P = 7.73 \times 10^{-4}$), adding support for the hypothesis that the targets that receive a high score under STORMSeq are *bona fide*, as microRNA activity generally leads to lower protein abundance and mRNA transcript abundance [1, 20, 27, 38].

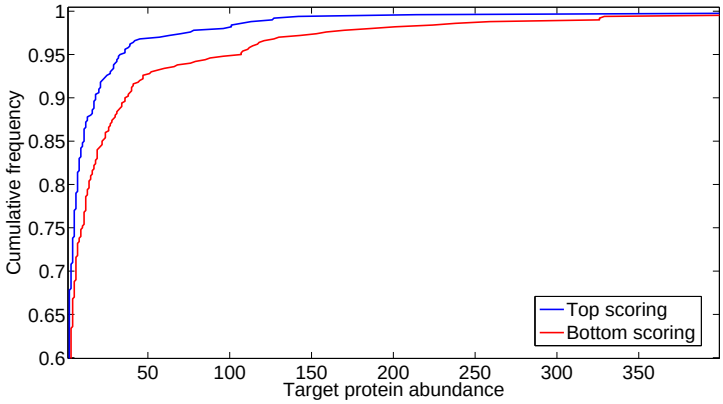
To further assess the use of purely sequence-based methods for this problem, we also ran the MEME [3] and AlignACE [32] algorithms using default settings on the 250 positive sequences for each training set and examined the resulting PSSMs reported by both algorithms. Each PSSM $\mathbf{M}$ obtained from these methods can then be used to rank sequences using the ranking function $\rho_{seq}(s_\alpha; \mathbf{M})$ as described previously. For all five training/test datasets, we found that neither of the PSSMs discovered by MEME and AlignACE led to any significant ability to rank let-7b targets (data not shown), suggesting that without additional information in the form of sequence conservation or quantitative measurements, *de novo* approaches to scoring sequences are significantly more likely to find poor models by virtue of overfitting, using only sequence information or due to model misspecification.

5.5.8 Discussion

We have presented the STORMSeq method for learning to rank regulatory sequences by combining heterogeneous datasets and diverse computational prediction methods. STORMSeq learns a model for ranking sequences given many observations, even if measurements or scores from different observations are not directly comparable. The explicit formulation of sequence search as a problem of ranking accounts for the fact that different sequences can have multiple levels of significance and any method for ranking should correctly order sequences by assigning a high score to biologically significant sequences. In particular, by accounting for the structure inherent to the problem of ranking, STORM-



(a)



(b)

Figure 5.20: a) Cumulative frequency plots of the $\Delta\Delta G$ scores on the top and bottom 100 targets as ranked by STORMSeq. High-scoring STORMSeq targets generally have higher target site accessibility and so have a lower $\Delta\Delta G$ value compared to low-scoring targets ($P < 10^{-20}$, Wilcoxon-Mann-Whitney); b) Cumulative frequency plots of protein abundances for top and bottom 100 targets as ranked by STORMSeq. High-scoring STORMSeq targets have significantly lower target protein abundance ($P = 7.73 \times 10^{-4}$) as a result of microRNA repressive activity.

Seq improves predictive performance over other unstructured methods for learning to rank. In addition, STORMSeq largely avoids many of the issues of model misspeci-

fication and complex inference that may arise when modelling multiple heterogeneous datasets. As STORMSeq is formulated in fairly general terms, it can also be applied to other problems of sequence search such as ranking drug targets, discovering genetic associations or scoring protein-protein interactions, although we have not focused on such applications here.

We have applied STORMSeq to the problems of scoring sequences bound by transcription factors and sequences bound by microRNAs. In the latter case, the use of CDNs to model statistical dependence relationships amongst preferences, in tandem with the ability to combine different data types with computational predictions, were both shown to improve predictive accuracy. This suggests that STORMSeq may also be useful for problems in comparative genomics as a principled means for combining diverse datasets. Other interesting extensions of the STORMSeq would include scaling the proposed framework to genome-wide detection of regulatory sequences as well as using alternate parametric forms for the ranking function that could account for interactions between the sequences to be ranked.

In the case of ranking microRNA-target interactions we have shown that incorporating diverse computational predictions increases predictive accuracy as measured by precision and recall. It should be noted that one must exercise care in what additional sources of computational predictions are incorporated into the analysis. We found that by incorporating computational prediction methods that had inherently low accuracy (e.g.: using methods which make predictions in a *de novo* fashion using sequence data alone), we could in fact decrease the predictive accuracy of our method (data not shown). In our case, particular computational prediction methods were included in our analysis on the basis of a previous study conducted in [27], which gauged the predictive accuracy of a variety of microRNA-target prediction methods according to a variety of metrics. We caution that in the case in which data is relatively limited in size, including computational predictions from methods that have low accuracy can adversely impact the

accuracy of STORMSeq. A possible extension to the framework proposed here is to allow for outlier detection so that the model can discount the impact of outlier observations.

An important issue which arises often in practice concerns the tractability of the proposed framework. In a setting in which one is given a large number of sequences to be ranked for a single observation, the number of edges in an order graph may in the worst case reach $O(n^4)$, where n is the number of objects in the observation. As storing and processing such a large observation may be outright intractable, we have made use of the MAD criterion for asserting preference relationships, which has the effect of reducing the number of pairwise preferences to be modeled. One can devise similar schemes to reduce the number of pairwise preferences to be modeled, as many of these will represent pairwise ordering constraints between very highly relevant sequences and irrelevant ones. We have also found that one can randomly break up an observation defined over many sequences into a set of multiple observations defined over smaller subsets of the sequences. Each of these observations could then be tractably modeled using the proposed method. In addition or as an alternative to the above, one can choose a CDN graph which is tractable and amenable to fast computations. An advantage of the proposed framework is that it is possible to use sparser CDN graphs which tradeoff the presence of dependencies between pairwise preferences for tractability and speedups in computation time.

Chapter 6

Conclusion

We have proposed the CDN as a graphical model for joint CDFs over many variables. In Chapter 3, we have shown that the conditional independence properties of a CDN are distinct from the independence properties of directed, undirected and factor graphs. However, these properties include those for bidirected graphs as a subset, so that CDNs provide a parameterization for bidirected graphical models. Furthermore, we presented a method for constructing a factor graph with additional latent variables for which graph separation in the corresponding CDN implies conditional independence of the separated variables in the joint probability obtained from marginalizing out the latent variables. As a result of this theorem we can always view a CDN as a factor graph with latent variables introduced. In the particular case in which variable nodes in the CDN correspond to maxima over latent variables, we provided a closed-form mapping from a CDN to a factor graph. We have then demonstrated in Chapter 4 that inference in a CDN corresponds to computing derivatives/finite differences. Chapter 4 described the DSP algorithm for computing such derivatives/finite differences by passing messages in the CDN where each message corresponds to local derivatives of the joint CDF.

In Chapter 5, we used the graphical framework provided by CDNs to formulate models and methods for learning to rank in a structured setting in which we must account for

statistical dependence relationships between model variables. We first applied the DSP algorithm to the problem of ranking in multiplayer gaming where players compete in teams. The DSP algorithm allowed us to compute distributions over player scores given previous game outcomes while accounting for the team-based structure of the games, whereby we were able to show improved results over previous methods. The CDN framework was then used to construct loss functionals for structured ranking learning where we wish to account for statistical dependence relationships that arise in ranking a set of objects. We showed that many probability models for rank data can be viewed as particular CDNs with different connectivities between pairwise object preferences. We then applied the proposed framework to the problems of document retrieval and sequence search in computational systems biology. In the latter problem, the proposed framework allows one to flexibly define structured loss functionals and ranking functions, which then allows one to integrate together a variety of data sources, including both experimental measurements and computational predictions.

Based on the work and results presented in this thesis, we can recommend future directions of research pertaining to the methods presented in previous chapters.

6.1 Future work

6.1.1 Construction of CDN models

In this thesis we have presented a few parameterizations for CDN functions such as multivariate Gaussian CDFs, copula functions and multivariate sigmoidal functions. Future work could focus on investigating the applicability of additional parameterizations for such functions. Additionally, future work could focus on the implications of the min-independence property for CDNs defined over discrete variables, whereby the set of conditional independence relationships is a function of the particular ordering over the possible variable configurations. One could investigate how this impacts the selection of

CDN functions and how this affects the design of CDNs in general.

6.1.2 Learning in cumulative distribution networks

While in Chapter 5 we presented a framework for learning a CDN in which the likelihood was of the form $\log F(\mathbf{x}|\boldsymbol{\theta})$, there may be applications in which one may wish to instead optimize the log-probability density $\log P(\mathbf{x}|\boldsymbol{\theta})$, such that learning corresponds to maximum-likelihood learning. In Chapter 4, we presented the DSP algorithm for both discrete and continuous-variable networks and we showed how DSP could be used to compute the probability density $P(\mathbf{x}|\boldsymbol{\theta})$ from the joint CDF $F(\mathbf{x}|\boldsymbol{\theta})$ modeled by the CDN. In order to perform maximum likelihood learning in which we wish to maximize the log-likelihood $\mathcal{L}(\mathbf{x};\boldsymbol{\theta}) = \log P(\mathbf{x}|\boldsymbol{\theta})$ with respect to a parameter vector $\boldsymbol{\theta}$ for a given set of observed variables $\mathbf{x}$, one can use modified versions of DSP messages in order to compute the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathbf{x};\boldsymbol{\theta})$ of the log-likelihood. The guiding principle here is that the gradient operator can be distributed amongst local functions in the CDF, much like the differentiation operation in DSP, so that by modifying DSP messages appropriately we can obtain the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathbf{x};\boldsymbol{\theta})$. Once computed, the gradient vector can then be used in a gradient-descent algorithm to optimize the log-likelihood. Future research in this direction could be directed at establishing what class of graphs can yield tractable gradient computations, as well as the complexity/accuracy tradeoffs involved in computing gradients in graphs with cycles. Another interesting direction would be to examine the problem of structure learning of CDNs via maximum-likelihood, which we have not addressed in this thesis. This could perhaps be accomplished by message-passing for maximum-likelihood, or perhaps by using other measures of loss for learning CDNs.

6.1.3 Derivative-sum-product in graphs with cycles

We have shown that our message-passing algorithm leads to the correct set of derivatives of the joint CDF provided that the underlying graph is a tree. As with the sum-product

algorithm for factor graphs, if the graph contains cycles, then the derivative-sum-product is no longer guaranteed to yield the correct mixed derivatives of the joint CDF, so that messages may begin to 'oscillate' as they propagate around cycles in the graph. One important direction to pursue is to establish conditions under which the presence of cycles will not lead to oscillations in messages: one could resort to a similar methodology as that employed by [68], where a graph with cycles is "unwrapped" and the resulting messages are analyzed.

6.1.4 Approximating derivative-sum-product in continuous variable models

In Chapter 4, we showed that for graphs defined over continuous variables, the complexity of computing DSP message updates at a given function node increased exponentially with the number of neighboring variable nodes, as one has to sum over products of messages incoming from all subsets of variables connected to the function node. However, it may be possible to approximate messages using simpler, tractable forms such as conditional univariate Gaussian CDFs. Future work here would be to establish tractable methods for performing such approximations and gauge the performance of such an approximate scheme for inference in CDNs on both synthetic and real-world data.

6.1.5 Extending the structured ranking learning framework

In Chapter 5 we applied the structured ranking learning framework to problems of learning to rank in document retrieval and ranking sequences in computational systems biology. One could easily explore the use of different CDN topologies, CDN functions and ranking functions for these same problems and the tradeoff in accuracy versus computational cost. Other areas in which the problem of learning to rank arises is in collaborative prediction [57], where any model should account for both the ordinal and discrete non-metric nature of user ratings. Regarding applications to computational systems biology, an open question is what other problems would be amenable to the proposed structured

ranking learning framework where one would like to integrate together multiple datasets and computational predictions in the setting of learning to rank.

6.1.6 Constructing mixed graphical models

As we have demonstrated in this thesis, the graph separation criterion for assessing conditional independence in CDNs include those for bidirected graphs [58]. As such graphs are a special case of mixed graphs containing undirected, directed and bidirected edges, a future avenue of research would be to investigate whether one can tractably construct such mixed graphical models using a hybrid graphical formulation combining the CDN model with that of factor graphs for joint probability density/mass functions. The Bayesian learning approach adopted by [64] could provide a good framework with which to qualitatively and quantitatively compare the use of CDNs for constructing such mixed graphical models.

Bibliography

[1] Babak, T., Zhang, W., Morris, Q.D., Blencowe, B.J. and Hughes, T.R. (2004) Probing microRNAs with microarrays: Tissue specificity and functional inference. *RNA*, **10**, 1813-1819.

[2] Baeza-Yates, R., Ribeiro-Neto, B. (1999) *Modern Information Retrieval*. Addison Wesley.

[3] Bailey, T.L., Williams, N., Misleh, C. and Li, W.W. (2006) MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Research* , **34**, W369-W373.

[4] Berger, M.F., Philippakis, A.A., Qureshi, A.M., He, F.S., Estep III, P.W. and Bulyk, M.L. (2006) Compact, universal DNA microarrays to comprehensively determine transcription factor binding specificities. *Nature Biotechnology*, **24**, 1429-1435.

[5] Bishop, C. (2006) Pattern recognition and machine learning. *Springer*.

[6] Bottou, L. and Le Cun, Y. (2004) Large scale online learning. *Advances in Neural Information Processing Systems*, **16**, MIT Press, Cambridge, MA.

[7] Burges, C.J.C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N. and Hullender, G. (2005) Learning to rank using gradient descent. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*, 89-96.

[8] Burges, C.J.C., Ragno, R. and Le, Q.V. (2007) Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems (NIPS)*, **19**, 193-200.

[9] Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F. and Li, H. (2007) Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*, 129-136.

[10] Casella, G. and Berger, R.L. (2002) *Statistical Inference. (2nd Ed.)*, Duxbury Press.

[11] Chen, X., Hughes, T.R. and Morris, Q.D. (2007) RankMotif++: a motif-search algorithm that accounts for relative ranks of K-mers in binding transcription factors. *Bioinformatics*, **23**, i72-i79.

[12] Dawid, A.P. (1979) Conditional independence in statistical theory (with discussion). *Journal of the Royal Statistical Society, Series B*, **41**, 1-31.

[13] Drton, M. and Richardson, T.S. (2008) Binary models for marginal independence. *Journal of the Royal Statistical Society, Series B*, **70**, 287-309.

[14] Elo, A.E. (1978) *The Rating of Chess Players: Past and Present*. Arco Publishing.

[15] Foat, B.C. *et al.* (2006) Statistical mechanical modeling of genome-wide transcription factor occupancy data by MatrixREDUCE. *Bioinformatics*, **22**, e141-e149.

[16] Frey, B.J. (2003) Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 257-264.

[17] Frey, B.J. and Dueck, D. (2007) Clustering by passing messages between data points. *Science*, **315**, 972-976.

[18] Frey, B.J., Kschischang, F.R., Loeliger, H-A. and Wiberg, N. (1997) Factor graphs and algorithms. In *Proceedings of the Thirty-Fifth Allerton Conference on Communications, Control and Computing*.

[19] Frey, B.J. and Mackay, D.J.C. (1997) A revolution: Belief propagation in graphs with cycles. In *Advances in Neural Information Processing Systems*, **10**, MIT Press.

[20] Grimson, A. *et al.* (2007) MicroRNA targeting specificity in mammals: determinants beyond seed pairing. *Molecular Cell*, **27**, 91-105.

[21] Gumbel, E. J. and Mustafi, C. K. (1967) Some analytical properties of bivariate extreme distributions. *Journal of the American Statistical Association*, **62**, 569-588.

[22] Herbrich, R., Graepel, T. and Obermayer, K. (2000) Large margin rank boundaries for ordinal regression. *In* Smola, A., Bartlett, P., Scholkopf, B. and Schurmanns, D., eds. *Advances in Large Margin Classifiers*, MIT Press, 115-132.

[23] Herbrich, R., Minka, T.P. and Graepel, T. (2007) TrueSkill$^{\text{TM}}$: A Bayesian skill rating system. In *Advances in Neural Information Processing Systems (NIPS)*, **19**, 569-576.

[24] Hersh, W.R., Buckley, C., Leone, T.J. and Hickam, D.H. (1994) OHSUMED: An interactive retrieval evaluation and new large test collection for research. *In Proceedings of the 17th Annual ACM SIGIR Conference*, 192-201.

[25] Huang, J.C., Morris, Q.D., and Frey, B.J. (2006) Detecting microRNA targets by linking sequence, microRNA and gene expression data. In Apostolico, A., Guerra, C., Istrail, S., Pevzner, P., Waterman, M., eds., *Research in Computational Molecular Biology (RECOMB)*, Lecture Notes in Computer Science **3909**, Springer Berlin/Heidelberg, 114-129.

[26] Huang, J.C., Morris, Q.D., and Frey, B.J. (2007) Bayesian learning of microRNA targets from sequence and expression data. *Journal of Computational Biology*, **14(5)**, 550-563.

[27] Huang, J.C., Babak, T., Corson, T.W. , Chua, G. Khan, S., Gallie, B.L., Hughes, T.R., Blencowe, B.J., Frey, B.J., and Morris, Q.D. (2007) Using expression profiling data to identify human microRNA targets. *Nature Methods*, **4(12)**, 1045-1049.

[28] Huang, J.C., Kannan, A. Winn, J. (2007) Bayesian association of haplotypes and non-genetic factors to regulatory and phenotypic variation in human populations. *Bioinformatics*, **23**, i212-221.

[29] Huang, J.C., Frey, B.J. and Morris, Q.D. (2008) Comparing sequence and expression for predicting microRNA targets using GenMiR3. *Pacific Symposium for Biocomputing (PSB)*, **13**, 52-63.

[30] Huang, J.C. and Frey, B.J. (2008) Cumulative distribution networks and the derivative-sum-product algorithm. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, 290-297.

[31] Huang, J.C. and Frey, B.J. (2009) Structured ranking learning using cumulative distribution networks. *Advances in Neural Information Processing Systems (NIPS)*, **21**, 697-704.

[32] Hughes, J.D., Estep III, P.W., Tavazoie, S. and Church, G.M. (2000) Computational identification of cis-regulatory elements associated with groups of functionally related genes in *Saccharomyces cerevisiae*. *Journal of Molecular Biology*, **296(5)**, 1205-1214.

[33] Huynh, T., Miranda, K., Tay, Y., Ang, Y.-S., Tam, W.-L., Thomson, A. M., Lim, B. and Rigoutsos, I. (2006) A pattern-based method for the identification of microRNA-target sites and their corresponding RNA/RNA complexes. *Cell*, **126**, 1203-1217.

[34] Jarvelin, K. and Kekalainen, J. (2002) *Cumulated evaluation of IR Techniques*. ACM Information Systems.

[35] Joachims, T. (2002) Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 133-142.

[36] Joachims, T. (2005) A support vector method for multivariate performance measures. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*, 377-384.

[37] Kauermann, G. (1996) On a dualization of graphical Gaussian models. *Scandinavian Journal of Statistics*, **23**, 105-116.

[38] Kertesz, M., Iovino, N., Unnerstall, U., Gaul, U. and Segal, E. (2007) The role of site accessibility in microRNA target recognition. *Nature Genetics*, **39**, 1278-1284.

[39] Kislinger, T., Cox, B., Kannan, A. *et al.* (2006) Global survey of organ and organelle protein expression in mouse: combined proteomic and transcriptomic profiling. *Cell*, **125**, 173-186.

[40] Krek, A. *et al.* (2005) Combinatorial microRNA target predictions. *Nature Genetics*, **37**, 495-500.

[41] Kschischang, F.R., Frey, B.J. and Loeliger, H.-A. (2001) Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, **47(2)**, 498-519.

[42] Lauritzen, S. (1996) *Graphical models.* Oxford Science Publications.

[43] Lehmann, E. L. (1955) Ordered families of distributions. *The Annals of Mathematical Statistics*, **26**, 399-419.

[44] Liu, T-Y., Xu, J., Qin, T., Xiong, W. and Li, H. (2007) LETOR: Benchmark dataset for research on learning to rank for information retrieval. *LR4IR 2007, in conjunction with SIGIR 2007.*

[45] Liu, X.S., Brutlag D.L. and Liu, J.S. (2002) An algorithm for finding protein-DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nature Biotechnology*, **20**, 835-839.

[46] Marden, J. I. (1995) *Analyzing and Modeling Rank Data.* CRC Press, 1995.

[47] Matúš, F. (1992) Ascending and descending conditional independence relations. In *transactions of the Eleventh Prague conference on information theory, statistical decision functions and random processes*, 189-200, Academia, Prague.

[48] McCullagh, P. (1980) Regression models for ordinal data. *Journal of the Royal Statistical Society, Series B (Methodological)*, **42(2)**, 109-142.

[49] Mézard, M., Parisi, G. and Zecchina, R. (2002) Analytic and algorithmic solution of random satisfiability problems. *Science*, **297**, 812-815.

[50] Minka, T.P. (2001) Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 362-369.

[51] Moussouris, J. (1974) Gibbs and Markov random systems with constraints. *Journal os Statistical Physics*, **10**, 11-33.

[52] Nadaraya, E.A. (1964) On estimating regression. *Theory of Probability and its Applications*, **9(1)**, 141-142.

[53] Neal, R.M. (1993) Probabilistic inference using Markov chain Monte Carlo methods, Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

[54] Nelsen, R.B. (1999) *An Introduction to Copulas.* Springer.

[55] Papoulis, A. and Pillai, S.U. (2001) Probability, random variables and stochastic processes. *McGraw Hill.*

[56] Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann.

[57] Rennie, J. and Srebro, N. (2005) Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of Twenty-Second International Conference on Machine Learning (ICML)*, 713-719.

[58] Richardson, T.S. and P. Spirtes (2002) Ancestral graph Markov models. *Annals of Statistics*, **30**, 962-1030.

[59] Richardson, T.S. (2003) Markov properties for acyclic directed mixed graphs. *Scandinavian Journal of Statistics*, **30**, 145-157.

[60] Robertson, S.E. (1997) Overview of the OKAPI projects. *Journal of Documentation*, **53 (1)**, 3-7.

[61] Segal, E., Raveh-Sadka, T., Schroeder, M., Unnerstall, U. and Gaul, U. (2008) Predicting expression patterns from regulatory sequence in *Drosophila* segmentation. *Nature*, **451**, 535-540.

[62] Shaked, M. and Shanthikumar, J.G. (1994). *Stochastic orders and their applications.* Academic Press.

[63] Silva, R. and Ghahramani, Z. (2009) Factorial mixture of Gaussians and the marginal independence model. In *Proceedings of the Twelfth Annual Conference on Artificial Intelligence and Statistics (AISTATS)*, *JMLR: W & CP*, **5**, 520-527.

[64] Silva, R. and Ghahramani, Z. (2009). The hidden life of latent variables: Bayesian learning with mixed graph models. *Journal of Machine Learning Research.*

[65] Tanay, A. (2006) Extensive low-affinity transcriptional interactions in the yeast genome. *Genome Research*, **16**, 962-972.

[66] Tsochantaridis, I., Hofmann, T., Joachims, T. and Altun, Y. (2004) Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, 104-112.

[67] Watson, G.S. (1964) Smooth regression analysis. *The Indian Journal of Statistics, Series A*, **26**, 359-372.

[68] Weiss, Y. and Freeman, W.T. (2001) Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, **13**, 2173-2200.

[69] Wermuth, N, Cox, D. and Pearl, J. (1994) Explanations for multivariate structures derived from univariate recursive regressions. Technical Report 94-1, University of Mainz.

[70] Xia, F., Liu, T.-Y., Wang, J., Zhang,W. and Li, H. (2008) Listwise approach to learning to rank - theory and algorithm. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*, 1192-1199.

[71] Zhai, C. and Lafferty, J. (2001) A study of smoothing methods for language models applied to ad-hoc information retrieval. In *Proceedings of SIGIR 2001*, 334-342.

[72] Zhang, W., Morris, Q.D. *et al.* (2004) The functional landscape of mouse gene expression. *Journal of Biology*, **3**, 21-43.