# Compendia in Math, Statistics, Machine Learning, Robotics and Robot Learning

Compendium in the interesting fields of Math, Statistics, Machine Learning, Robotics and Robot Learning that made the years of my study a fulfilling time.

*Darmstadt 2015*
THOMAS HESSE

Hesse, Thomas
mail@thomas-hesse.eu

Faber, Natalie
mail@nf9.me

# Contents

# 1 | Notation

Scalars, vectors and matrices are mostly written in greek letters, bold font lowercase roman letters and bold font uppercase roman letters respectively.

| Symbols | Description |
|---|---|
| $X,\ Y,\ Z$ | random variables |
| $\mathbb{E}[X]$ | expectation of the random variable $X$ |
| $\mathcal{X},\ \mathcal{Y},\ \mathcal{Z}$ | sets of discrete outcomes of a random variable |
| $y$ | commonly used as output variable |
| $x$ | commonly used as input variable |
| $\mathbf{w}$ | commonly used as model parameters |
| $f$ | commonly used as a function or task |
| $p(x)$ | probability density of the variable $x$ |
| $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$ | conditional independence of the sets $\mathcal{X}$ and $\mathcal{Y}$ given set $\mathcal{Z}$ |
| $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$<br>$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \emptyset$ | (unconditional) independence of the sets $\mathcal{X}$ and $\mathcal{Y}$ |
| $\mathcal{X} \top\!\!\!\top \mathcal{Y} \mid \mathcal{Z}$ | conditional dependence of the sets $\mathcal{X}$ and $\mathcal{Y}$ given set $\mathcal{Z}$ |
| $\mathcal{X} \top\!\!\!\top \mathcal{Y}$<br>$\mathcal{X} \top\!\!\!\top \mathcal{Y} \mid \emptyset$ | (unconditional) dependence of the sets $\mathcal{X}$ and $\mathcal{Y}$ |
| $\mathrm{pa}(A)$ | the parent vertex or vertices of a vertex $A$ |
| $\mathrm{ch}(A)$ | the child vertex or children vertices of a vertex $A$ |
| $\mathrm{ne}(A)$ | the neighboring vertices of a vertex $A$ |
| $\mathcal{Q}$ | set of *easy* distributions |
| $\mathrm{KL}\left(q(x)\|\|p(x)\right)$ | the Kullback-Leibler divergence *distance measure* as defined in Equation (3.27) |
| $H(q)$ | the entropy of a distribution $q$ defined in Equation (3.28) |

# 2 | Machine Learning

## 2.1 Foundations of Machine Learning

Machine Learning is a research field that emerged from the field of artificial intelligence. There are two general definitions of machine learning from Arthur Samuel and Tom Mitchell available:

> "Field of study that gives computers the ability to learn without being explicitly programmed."
> — Arthur Samuel, 1959

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."
> — Tom Mitchell, 1997

The latter definition is more often cited and a more formal and modern definition. Here we constrain ourselves to the definition that a machine learns to perform a task from experience. More formally, learning means to adjust or to tune parameter values (often called $\theta$ or $w$), a task is considered to be a function $f$, and experience is represented by data (generally we use $\mathcal{D}$ to be our dataset because the actual form of the data set depends on the issue we are dealing with). Using the just introduced terminology, we grasp learning problems in the following mathematical manner

$$y = f(x; \mathbf{w}) \tag{2.1}$$

where $y$ is called *output variable*, $x$ is called *input variable*, and $\mathbf{w}$ are the *model parameters*. Equation 2.1 shows the special case of supervised learning.

### 2.1.1 Supervised Learning

In supervised learning, we deal with data of the form $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$. Reconsider Equation 2.1; we try to find a continuous function $f$ that maps any input to an output. Hence, the aim of supervised learning is to find a model that is sufficiently precise enough to fit the data $\mathcal{D}$ during training and generalize well to new unseen data.

**Example:** More common, we are presented some input data $x$ and its related outcome $y$. Assume that a teacher shows you multiple pictures of cars such as a red Ferraris, black Porsches, blue BMWs, and so on. In this case, our input data (or features) might be the color of the cars and the related outcome is the car name (e.g. Ferrari, Porsche, BMW, etc.). In the beginning of your supervised learning task, your teacher shows the pictures of the cars to you and let you know about the outcomes $y$. After a while, you should have learned a function $f$ that maps new unseen colored cars to their type.

### 2.1.2 Unsupervised Learning

In unsupervised learning, we deal with data of the form $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$. Given only input data or features $x$; we try to find patterns in the data.

- **Given** Input data points $x$
- **Goals**
  - **Clustering** find groups in the data
  - **Density estimation** determine the data distribution $p(x)$
  - **Dimensionality reduction** visualize the data by projections

### 2.1.3 Semi-supervised Learning

We are given two datasets

$$\mathcal{D}_l = \{(x_1, y_1), \ldots (x_k, y_k)\} \tag{2.2}$$

for learning and

$$\mathcal{D}_t = \{x_k, \ldots, x_{k+n}\} \tag{2.3}$$

for testing. Again, we seek the solution given by Equation (**??**). How can we use the extra information of $\mathcal{D}_t$ for this purpose? We shall use the test set to evaluate our model.

### 2.1.4 Reinforcement Learning

- **Given**
  - **Exploration** try out new actions
  - **Exploitation** use known actions that yield a high reward
- **Goals**
  - Find a good trade-off between **Exploration** and **Exploitation** to maximize the reward for all actions

### 2.1.5 Transductive Learning

Transductive learning is similar to semi-supervised learning of Section (2.1.3). In contrast to semi-supervised learning, we do not try to learn a function $f$ (according to Equation (**??**)) explicitly but rather try to get predicitions for the test data set given by Equation (2.3). Moreover, the test data set can be used for training.

### 2.1.6 On-line Learning

The training data is presented step-by-step; hence we are given a new datapoint of the form $x_t$ or $(x_t, y_t)$ at each new time step $t$. In this scenario, we have limited storage and need to learn in epochs where we only use a limited set of training data points (mostly those datapoints that help to achieve a certain goal).

### 2.1.7 Large-scale Learning

There is no definition for the term large-scale but it refers to *huge* amounts of data which is mostly used by fast or parallelized learning algorithms.

### 2.1.8 Active Learning

We have a data set similar to the one in Equation (**??**) and our goal is to learn $y$ as in Equation (**??**). We suppose that labeling a data point $x_i$, i.e. predict $y_i$, costs something. The goal is to keep the cost as low as possible.

### 2.1.9 Structured Output Learning

For structured output learning, we try to learn a function

$$f(x, y; \mathbf{w}) \tag{2.4}$$

which we can use to predict a certain outcome

$$y = \arg\max_{\bar{y}} \{f(x, \bar{y}; \mathbf{w})\} . \tag{2.5}$$

For the purpose to obtain an optimal $y$ we are given a data set as of the form in Equation (**??**). The wording structure in this context relates to e.g. graphical models, trees, etc..

## 2.2 Decision Theory – Inference and decision

| Inference | Decision |
|---|---|
| ◦ obtain probabilities $p(C_k\|x)$ | ◦ obtain optimal class assignment $C^*$ |

### 2.2.1 Solving decision problems

There are three approaches commonly known to tackle decision problems:

| Generative models | Discriminative models | Discriminative function |
|---|---|---|
| ◦ Infer the class conditionals $p(x\|C_k),\ k = 1, \ldots, K$ | ◦ Infer posterior probabilities $p(C_k\|x)$ directly | ◦ Find $f : \mathcal{X} \to \{1, \ldots, K\}$ by minimizing $\mathbb{E}[\Delta]$ directly where $\Delta$ is a given loss function |

| | | |
|---|---|---|
| + We can generate samples from the learned distribution | + No need to model the class conditionals $p(x\|C_k)$ | + Models the quantitiy of interest directly |
| + We are able obtain the marginal probability $p(x) = \sum_k p(x\|C_k)$ | | + Not bound to probabilities and hence no need to normalize |
| – Need for lot of data for high dimensions of $x$ to determine class conditionals | – No access to model $p(x)$ | – Need for a loss function $\Delta$ while learning and changes to the loss function $\Delta$ requires re-learning |
| – We might not be interested in all quantities and this seems to be too much | | – No probabilities mean no uncertainty and no reject option |

## 2.3 Linear Regression

The linear regression is a supervised learning method. Formulation:

- univariate
- multiple variables
- multivariate

## 2.4 Regularization

to-do

### 2.4.1 Linear Regression

### 2.4.2 Logistic Regression

## 2.5 Neural Networks

to-do

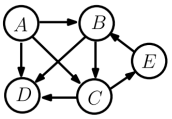
Origin? Why are they powerful, what do they do? Intuition of multiple layers?

# 3 | Graphical Models

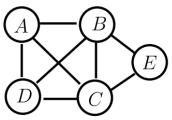
## 3.1 Modeling

Modeling our knowledge formally is essential to acquire a model language that can be used with standardized algorithms for learning and inference. In this lecture, we use graphical models, i.e. graphs, as a model language. A vertex in the graph is an event (or random variable) that may occur and an edge between vertices symbolizes a dependence.

### 3.1.1 Graph Definitions




A directed graph – directed edges.
Bayesian networks




An undirected graph – undirected edges.
Markov random fields

- **Vertex** A vertex is a single node, e.g. $A$, within a graph.
- **Edge** An edge is a tuple of vertices, e.g. $(A_1, A_2)$.
- **Directed Graph** A directed graph has directed edges.
- **Path** A path from vertex $A$ to vertex $B$ is denoted as $A \to B$, a sequence of vertices $A,\ C,\ E,\ B$.
- **Ancestor** All vertices $A$ are ancestors of $B$ iff $A \to B$ and $B \not\to A$ in directed graphs.
- **Descendant** All vertices $B$ are descendants of $A$ iff $A \to B$ and $B \not\to A$ in directed graphs.
- **Undirected Graph** An undirected graph has undirected edges.
- **Neighbor** All neighbors of a vertex $A$ in an undirected graph share an edge with $A$.
- **Clique** A clique within an undirected graph is a subset of vertices that are fully connected.
- **Maximal clique** A clique is called maximal clique if it is not extended by adding another vertex.
- **Directed Acyclic Graph (DAG)** A graph with directed edges where every vertex will be visited once along a path.

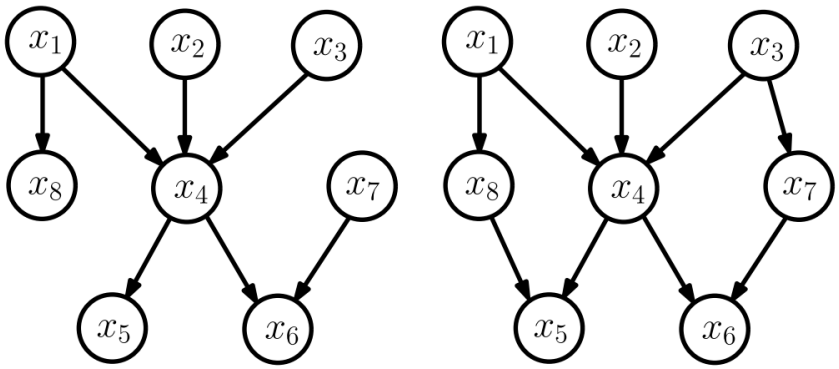
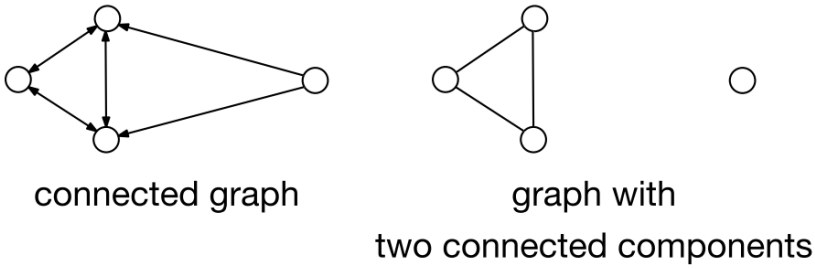


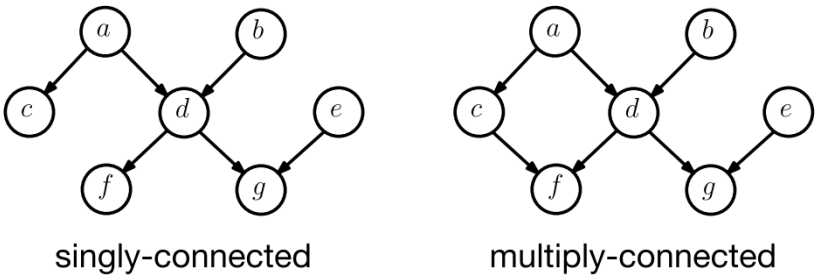

Figure 3.1: Two exemplary directed acyclic graphs (DAGs) which are sufficient to understand before introducing further terminology.

- **Parents** The parent of e.g. vertex $x_4$ are $pa(x_4) = \{x_1, x_2, x_3\}$.
- **Children** The children of e.g. vertex $x_4$ are $ch(x_4) = \{x_5, x_6\}$.

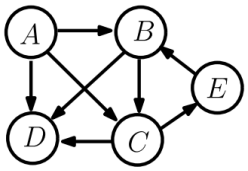
- **Family** The family of e.g. vertex $x_4$ is $\{x_4\} \cup pa(x_4) \cup ch(x_4)$.
- **Markov Blanket** The Markov blanket of a vertex is given by its parents, children and its children parents.
- **Connected graph** A graph is connected if there is a path between any two vertices, otherwise there are connected components.
- **Connected components** If a graph is no connected graph, we speak of connected components.




- **Singly-connected graph** There is at most one path between any vertices $A$ and $B$ in a singly-connected graph.
- **Multiply-connected graph** Whenever a graph is no singly-connected graph, it is a multiply-connected graph.




### 3.1.2 Bayesian Networks / Belief Networks

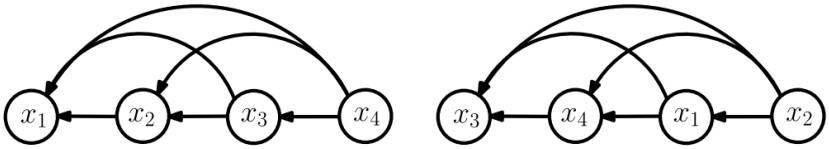



A directed graph – directed edges.
Bayesian networks

A Bayesian network or belief network is a distribution of the form

$$p(x_1, \ldots, x_D) = \prod_{i=1}^{D} p\left(x_i | pa(x_i)\right). \tag{3.1}$$

A Bayesian network can have different factorizations, e.g. a Bayesian network of the form $p(x_1, x_2, x_3, x_4)$ has two factorizations:

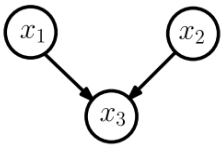



$$p(x_1|x_2, x_3, x_4)p(x_2|x_3, x_4)p(x_3|x_4)p(x_4) = p(x_1, x_2, x_3, x_4) = p(x_3|x_1, x_2, x_4)p(x_4|x_1, x_2)p(x_1|x_2)p(x_2)$$

Any distribution can be written as a factorization - a cascade form of a belief network - that becomes increasingly important when making independence assumptions. The structure of DAGs not only prevents cyclic reasoning but also corresponds to a set of conditional independence assumptions.

**Conditional independence**

Can we read off the conditional independence statements of a given graph? Is there an automatic algorithm?




To ar
netw
edge
ther
and
depe

**D-seperation**

Lets check for conditional indpendence $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$ in any belief network. For every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ check every path $\mathcal{U}$ between $x$ and $y$. A path is characterized to be blocked if there exists a vertex $u \in \mathcal{U}$ such that either

1. $u$ **is a** collider and neither $u$ nor any descendant is in $\mathcal{Z}$ or
2. $u$ **is no** collider in $\mathcal{U}$ and $u \in \mathcal{Z}$.

If we can show that all such paths $\mathcal{U}$ are blocked then $\mathcal{X}$ and $\mathcal{Y}$ are d-seperated by $\mathcal{Z}$. Consequently, the conditional indpendence $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$ does hold.

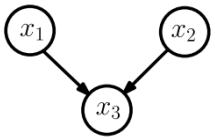
**D-connectedness**

The opposite case of D-seperation is D-connectedness. Hence, we check for conditional dependence statement $\mathcal{X} \top\!\!\!\top \mathcal{Y} \mid \mathcal{Z}$ in any belief network. We say that $\mathcal{X}$ and $\mathcal{Y}$ are d-connected by $\mathcal{Z}$ iff they are not d-seperated by $\mathcal{Z}$. Consequently, the conditional dependence $\mathcal{X} \top\!\!\!\top \mathcal{Y} \mid \mathcal{Z}$ does hold if $\mathcal{X}$ and $\mathcal{Y}$ are d-connected.

**Markov equivalence**

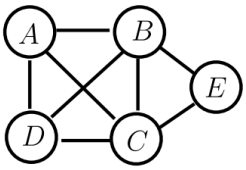
Two graphs are Markov equivalent if they represent the same set of conditional independence statements. More precisely this means that the two graphs have the same skeleton and the same set of immoralities.

- **Skeleton** resulting graph after removing all arrows of all edges
- **Immorality** the parents of a child have no connection, i.e. "are unmarried" (cf. the graph at the right as an example)




Note that the Markov equivalence holds for both, directed and undirected graphs.

### 3.1.3 Markov Random Fields




An undirected graph – undirected edges.

Markov random fields

So far, we considered factorizations of probability distributions with normalized factors so that the product will again result in the probability distribution. Alternatively we could factorize with so called potential function $\phi(x)$ as follows

$$p(a,b,c) = \frac{1}{Z}\phi(a,b)\phi(b,c) \tag{3.2}$$

with an explicit normalization constant called partition function $Z$ defined as

$$Z = \sum_a \sum_b \sum_c \phi(a,b)\phi(b,c). \tag{3.3}$$

A potential function is a non-negative function $\phi(x)$ of the input $x$ and a joint potential function is a non-negative function $\phi(x_1, \ldots, x_D)$ of a set of variables.

**Definition of a Markov Random Field**

A Markov Random Field (MRF) (or Markov Network) is defined as a product of potentials for a set of variables $\mathcal{X} = \{x_1, \ldots, x_D\}$ over the maximal cliques $\mathcal{X}_c$ of a graph and is denoted as in Equation (3.4).

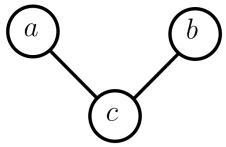
$$p(x_1, \ldots, x_D) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(\mathcal{X}_c) \tag{3.4}$$

There is the special case of the pairwise MRF where the clique sizes $\{\forall c \in C : |\mathcal{X}_c|\} = \{2\}$. However, in the case that all potentials are strictly positive, we are talking of a Gibbs distribution.

**Properties of a Markov Random Field**

We subsequently consider the Markov network depicted to the right to explain general properties of Markov networks. The Markov network has the factorization

$$p(a,b,c) = \frac{1}{Z}\phi_{ac}(a,c)\phi_{bc}(b,c) \tag{3.5}$$




**Property 1.** Marginalization of vertex $c$ in Equation (3.5) makes the vertices $a$ and $b$ "graphically" dependent, i.e. a new edge between $a$ and $b$ arises.

**Property 2.** Conditioning Equation (3.5) on the vertex $c$ makes the vertices $a$ and $b$ inpendent, i.e. $p(a,b \mid c) = p(a \mid c)p(b \mid c)$.

**Property 3.** The local Markov property states that each vertex has a condition on its neighbors and is independent on all other vertices. This property is formalized in Equation (3.6)

$$p(x \mid \mathcal{X}\backslash\{x\}) = p(x \mid \mathrm{ne}(x)) \tag{3.6}$$

**Property 4.** The global Markov property states that for disjoint sets of variables $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z} \tag{3.7}$$

holds where $\mathcal{Z}$ seperates $\mathcal{X}$ from $\mathcal{Y}$.

The Hammersley-Clifford Theorem states sufficient conditions to specify independence statements in an undirected graph. Thus it gives necessary and sufficient conditions under which a positive probability distribution can be represented as a Markov network.

**Hammersley-Clifford Theorem**

The Hammersley-Clifford Theorem states that a non-negative probability distribution satisfies one of the upper Markov properties with respect to an undirected graph iff it is a Gibbs random field. In this case, the density can be factorized over the cliques of the graph (or subgraph).

So how do we check for conditional independence of undirected graphs? We discussed this issue already for belief networks but this can barely be a general approach because of the directed edges. Hence we work out a general algorithm to check for conditional independence here for directed and undirected graphs. We define $\mathcal{D} = \{\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}\}$ for the subsequent algorithm.

1. **Ancestral Graph** — We can build up an ancestral graph by removing all vertices $A$ with all their edges from the graph for which hold that $A \notin \mathcal{D}$ and $A$ is no ancestor of any vertices $B \in \mathcal{D}$.
2. **Moralization** — Moralize the ancestral graph from the previous step by connecting parents with their common child and striping all the directions from the edges.
3. **Seperation** — Remove edges that link to neighbors in $\mathcal{Z}$ and if there is no path between a vertex in $\mathcal{X}$ to a vertex in $\mathcal{Y}$ we know that the conditional independence $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$ holds.

Finally, we introduce the relationship between directed and undirected graphical models. Both, directed and undirected graphical models, imply set of conditional indpendences.

### D map (dependency map)

- Every conditional independence statement, satisfied by the distribution, is reflected in the graph
- A completely disconnected graph contains all possible independence statements for its variables, i.e. is a trivial D map for any distribution

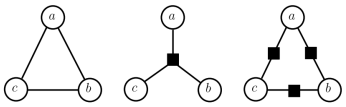
### I map (independence map)

- Every conditional independence statement, reflected in the graph, is satisfied by the distribution
- A fully connected graph contains all possible dependence statements for its variables, i.e. is a trivial I map for any distribution

We can now derive perfect maps in terms of D maps and I maps. A perfect map is, both, a D map and an I map of a distribution. More precisely, every conditional independence statement satisfied by a distribution is reflected in the corresponding graph and vice versa.

### 3.1.4 Factor Graphs

We already learned about different graph structures. They all had in common that they implicitly encode the factorization, no matter if directed or undirected. This leads to multiple factorizations of e.g. an undirected Markov random field as can be seen in the Figure to the right. However, the factorization of a factor graph (FG) is not specified by the graph itself, thus we use an (formally) explicit factorization. The factorizations are presented with an extra node or vertex; the symbol is a black square.

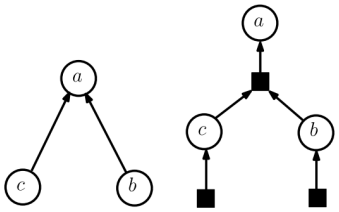



**Factor Graphs**

Given a function of the form

$$f(x_1, \dots, x_D) = \prod_i \psi_i(\mathcal{X}_i),$$

the factor graph has as abovementioned an extra node or vertex for each factor $\psi_i(\mathcal{X}_i)$ and a variable node or vertex for each variable $x_j$. Each factor is connected to the variables of which it is a function. When we use factor graphs to represent a distribution of the form




$$p(x_1, \dots, x_D) = \frac{1}{Z} \prod_i \psi_i(\mathcal{X}_i) \tag{3.8}$$

we tend to assume a normalization constant $Z$. The Figure on the right shows such an exemplary factor graph of a belief network $c \to a \leftarrow b$.

**Bipartite graph**

A bipartite graph has vertices that can be divided into two disjoint sets $U$ and $V$ of vertices such that every edge connects a vertex in the subset $U$ with a vertex of subset $V$.

## 3.2 Inference in graphical models

We exploited different structures to model probability distributions $p(x_1, \dots, x_D)$. However, how do we infer, i.e. computing functions like mean, marginal, and conditionals, in these structures?

We start with marginalization / variable elimination. Assume a Markov chain $A \leftarrow B \leftarrow C \leftarrow D$ with $A, B, C, D \in \{0, 1\}$. The distribution of this Markov chain is given by $p(A, B, C, D) = p(A \mid B)p(B \mid C)p(C \mid D)p(D)$. We seek the solution $p(A)$ which is given by

$$\begin{aligned} p(A) &= \sum_B \sum_C \sum_D p(A, B, C, D) \\ &= \sum_B \sum_C \sum_D p(A \mid B)p(B \mid C)p(C \mid D)p(D) \end{aligned} \tag{3.9}$$

which is also quite heavy in computation ($8 = 2^3$ computations). Here, we can use the linearity property of the summation and reorder the term as follows

$$p(A) = \sum_B \sum_C p(A \mid B)p(B \mid C) \sum_D p(C \mid D)p(D) \tag{3.10}$$

with subsequent functional substitution

$$\begin{aligned} p(A) &= \sum_B \sum_C p(A \mid B)p(B \mid C) \underbrace{\sum_D p(C \mid D)p(D)}_{\gamma_D(C)} \\ p(A) &= \sum_B p(A \mid B) \underbrace{\sum_C p(B \mid C)\gamma_D(C)}_{\gamma_C(B)} \\ p(A) &= \sum_B p(A \mid B)\gamma_C(B). \end{aligned} \tag{3.11}$$

Now we only need $2 + 2 + 2 = 6$ computations and for a Markov chain of arbitrary length the computation time scales linearly.

Furthermore, we will use the marginalization from above and the fact that $p(D \mid A) \propto p(A, D)$ to find the conditional marginals of the Markov chain $A \leftarrow B \leftarrow C \leftarrow D$ with $A, B, C, D \in \{0, 1\}$. The resulting term $\gamma_C(D)$ (cf. lecture slides) is no probability distribution and hence need to be normlized with the sum over all possible outcome $\left(\sum_D \gamma_C(D)\right)^{-1}$. We will end up with a representation depicted in Figure (3.2). We will interpret $\gamma_C(D)$ as message because it is not a probability distribution.

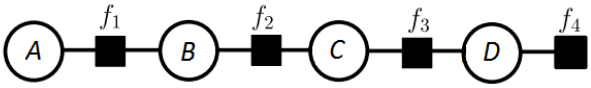



Figure 3.2: Factor graph representation of a Markov chain $A \leftarrow B \leftarrow C \leftarrow D$.

If now also take the factors into account, we end up with the following:

- We aim to compute the conditional marginal $p(D \mid A)$ as follows.

$$\begin{aligned} p(D \mid A) \propto p(A, D) &= p(A, B, C, D) \\ &= \sum_B \sum_C f_1(A, B) \cdot f_2(B, C) \cdot f_3(C, D) \cdot f_4(D) \\ &= \sum_C \underbrace{\sum_B f_1(A, B) \cdot f_2(B, C)}_{\mu_{A \to C}(C)} \cdot f_3(C, D) \cdot f_4(D) \\ &= \underbrace{\sum_C \mu_{A \to C}(C) \cdot f_3(C, D)}_{\mu_{A \to D}(D)} \cdot f_4(D) \\ &= \mu_{A \to D}(D) \cdot f_4(D) \end{aligned} \tag{3.12}$$

The advantage here, we only need to compute messages once and can reuse them at any time (as long as the outcome within these messages does not change). A message $\mu_{m \to n}(n)$ can be understood as an information carrier from $m$ to $n$ where we still have to choose $n$ (and hence it is still parameterized). But how do we compute messages efficiently?

### 3.2.1 Sum-Product algorithm / Belief propagation

We will derive the sum-product algorithm considering a singly-connected graph. We are able to compute any desired marginals with this algorithm.

1. **Initalization** Message from extremal (simplical) node factors are initialized to the factor

$$\mu_{f \to x}(x) = f(x) \tag{3.13}$$

Message from extremal (simplical) variable nodes are set to unity

$$\mu_{f \to x}(x) = 1 \tag{3.14}$$

2. **Variable-to-Factor message**

$$\mu_{x \to f}(x) = \prod_{g \in \{\mathrm{ne}(x) \setminus \{f\}\}} \mu_{g \to x}(x) \tag{3.15}$$

3. **Factor-to-Variable message**

$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \setminus \{x\}} \phi_f(\mathcal{X}_f) \prod_{y \in \{\mathrm{ne}(f) \setminus \{x\}\}} \mu_{y \to f}(y) \tag{3.16}$$

#### Message ordering

Note that there are two orderings defined on messages. This fact is important because a message always dependes on its previous message. However, extremal nodes and factors do not depend on other messages. To compute all messages in a graph, we compute from leaf-to-root (pick root node, compute messages pointing towards root) and root-to-leaf (compute messages pointing away from root).

#### Log-Messages

Using logarithmic messages is quite reasonable in large graph because the messages may become very small. The belief propagation algorithm from Section (3.2.1) changes accordingly in Equations (3.15) and (3.16) as follows

- **Variable to log Factor message**

$$\lambda_{x \to f}(x) = \sum_{g \in \{\mathrm{ne}(x) \setminus \{f\}\}} \lambda_{g \to x}(x) \tag{3.17}$$

- **Factor to log Variable message**

$$\lambda_{f \to x}(x) = \log \left\{ \sum_{\mathcal{X}_f \setminus \{x\}} \phi_f(\mathcal{X}_f) \cdot \exp \left( \sum_{y \in \{\mathrm{ne}(f) \setminus \{x\}\}} \lambda_{y \to f}(y) \right) \right\} \tag{3.18}$$

with $\lambda = \log\{\mu\}$, respectively. However, the log-sum-exp term here is not so nice but we can approximate it very well with a little trick!

**Log-Sum-Exp Trick**

To get a representation which is easier to handle we can use the equality

$$\log\left\{\sum_i \exp(v_i)\right\} = \alpha + \log\left\{\sum_i \exp(v_i - \alpha)\right\} \tag{3.19}$$

where we choose $\alpha = \max\{\lambda_{y\to f}(y)\}$.

### 3.2.2 Max-product algorithm

How can we use messages and factors to find the most likely state, as shown in Equation (3.20), given a probability distribution $p(y)$?

$$y^* = \arg\max_{y_1,\dots,y_D} \{p(y_1,\dots,y_D)\} \tag{3.20}$$

1. **Initalization** Message from extremal (simplical) node factors are initialized to the factor

$$\mu_{f\to x}(x) = f(x) \tag{3.21}$$

Message from extremal (simplical) variable nodes are set to unity

$$\mu_{f\to x}(x) = 1 \tag{3.22}$$

2. **Variable to Factor message**

$$\mu_{x\to f}(x) = \prod_{g\in\{\mathrm{ne}(x)\setminus\{f\}\}} \mu_{g\to x}(x) \tag{3.23}$$

3. **Factor to Variable message**

$$\mu_{f\to x}(x) = \max_{\mathcal{X}_f\setminus\{x\}} \left\{\phi_f(\mathcal{X}_f) \prod_{y\in\{\mathrm{ne}(f)\setminus\{x\}\}} \mu_{y\to f}(y)\right\} \tag{3.24}$$

After computing all above, we can choose an arbitrary start point and use backtracking to obtain the most likely state for each $y_i^*$.

### 3.2.3 Max-sum algorithm

Again, we can take the logarithm to obtain $\log\{\max_x\{p(x)\}\} = \max_x\{\log\{p(x)\}\}$ a numerical more stable version of the max-product algorithm, the sum-product algorithm.

### 3.2.4 Loopy belief propagation

The problem of loopy belief propagation is that we have no guarantee of convergence. However, loopy belief propagation enables (approximate) inference in intractable models but we have to iterate over and over until convergence. If the loopy velief propagation does not converge, it often does oscillate between states. We generally initialize messages to a fixed value. After that we can perform messages updates in a fixed or randomized order until we converge. The Factor-to-Variable messages remain unchanged while passing messages but the Variable-to-Factor messages need to be normalized, i.e.

$$\mu_{x\to f}(x) := \frac{\prod \mu_{f\to x}(x)}{\sum_x \prod \mu_{f\to x}(x)} \tag{3.25}$$

### 3.2.5 Mean field methods

The main idea of mean field methods is to approximate the distribution $p(x)$ with the best matching distribution $q \in \mathcal{Q}$. We seek such a solution because inference over generally hard distributions $p(x)$ is hard. We need to assume that the factors of the approximate distribution $q$ are a subset of the real (too) complicated distribution $p$. Then we might be able to find the best matching distribution

$$q^* = \arg\min_{q\in\mathcal{Q}} \{\mathrm{KL}\,(q(x)||p(x))\} \tag{3.26}$$

where the KL function is the Kullback-Leibler divergence (relative entropy) defined as

$$\begin{aligned}\mathrm{KL}\,(q(x)||p(x)) &= \sum_x q(x)\cdot\log\left\{\frac{q(x)}{p(x)}\right\} \\ &= \sum_x q(x)\cdot\log\{q(x)\} - \sum_x q(x)\cdot\log\{p(x)\} \\ &= \mathbb{E}\left[\log\{q(x)\}\right] + \sum_f\sum_{\mathcal{X}_f} q(\mathcal{X}_f)\cdot\log\left\{\phi_f(\mathcal{X}_f)\right\} + \log\{Z_p\}\,.\end{aligned} \tag{3.27}$$

However, we are able to drop the term $Z_p$ because it is independent of the distribution $q$ we want to estimate. We also may rewrite the expectation, the first addend in Equation (3.27), as the entropy and thus denote the entropy as

$$H(q) = -\mathbb{E}\left[\log\{q(x)\}\right]. \tag{3.28}$$

We may start right away with the most popular mean field method called Naive Mean Field.

### Naive Mean Field

We assume or approximated distribution $q(x)$ to be i.i.d. (identically and independently distributed) so that $q(x) = \prod_i q_i(x_i)$. This will apply the i.i.d. assumption to the entropy and the factor marginals as well. Remember, that we aim to find the $q^*$ in Equation (3.26) and combine it with the so far gathered knowledge to obtain

$$\begin{aligned} q^* &= \arg\min_{q \in \mathcal{Q}} \{\mathrm{KL}\left(q(x)||p(x)\right)\} \\ &= \arg\max_{q \in \mathcal{Q}} \left\{ H(q) - \sum_f \sum_{\mathcal{X}_f} q(\mathcal{X}_f) \cdot \log\left\{\phi_f(\mathcal{X}_f)\right\} \right\} \\ &= \arg\max_{q \in \mathcal{Q}} \left\{ -\sum_i \sum_{x_i} q_i(x_i) \log\left\{q_i(x_i)\right\} - \sum_f \sum_{\mathcal{X}_f} \left( \prod_{x_i \in \mathcal{X}_f} q(x_i) \right) \cdot \log\left\{\phi_f(\mathcal{X}_f)\right\} \right\}. \end{aligned}$$

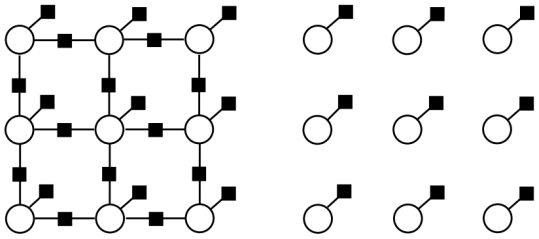
Now all we need to do is to solve for the above equation analytically for all $q_i(x_i) > 0$ with the constraint $\sum_i q_i(x_i) = 1$. However, this is not trivial to do because the second addend $-\sum_f \sum_{\mathcal{X}_f} \left(\prod_{x_i \in \mathcal{X}_f} q(x_i)\right) \cdot \log\left\{\phi_f(\mathcal{X}_f)\right\}$ is non-concave. But if we hold all but one $q_j(x_j)$ fixed, we get a concave problem of the form

$$q^* = \arg\max_{q \in \mathcal{Q}} \left\{ -\sum_i \sum_{x_i} q_i(x_i) \log\left\{q_i(x_i)\right\} - \sum_f \sum_{\mathcal{X}_f} \left( \prod_{x_i \in \mathcal{X}_f, i \neq j} \hat{q}(x_i) \right) \cdot q_j(x_j) \cdot \log\left\{\phi_f(\mathcal{X}_f)\right\} \right\}.$$

So all we need to do is

1. **Loop over all** $i$
2. **Find** $q_i^*(x_i)$

Basically, we are reducing the complexity of our problem as depicted in the image to the right. Our original problem is on the right half and what we are solving is shown in the left half. We keep every node fixed and compute the most optimal state for itself. Despite, we learned about message passing which could come in handy here to obtain a better approximated solution.



### Mean Field as Message Passing

Now let us have a look at better mean field approximation using message passing.

1. **Initalization**

   Message from extremal (simplical) node factors are initialized to the factor

   $$\mu_{f \to x}(x) = f(x) \tag{3.29}$$

   Message from extremal (simplical) variable nodes are set to unity

   $$\mu_{f \to x}(x) = 1 \tag{3.30}$$

2. **Variable-to-Factor message**

   $$\mu_{x \to f}(x) = \prod_{f_i \in \{\mathrm{ne}(x) \setminus \{f\}\}} \mu_{f \to x}(x) \tag{3.31}$$

3. **Factor-to-Variable message**

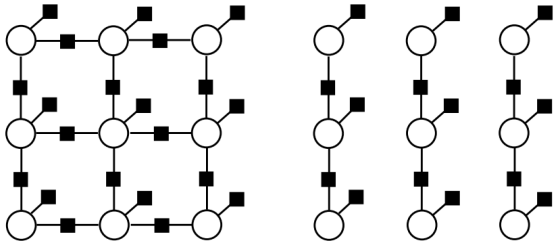
   $$\mu_{f \to x} \propto \exp\left( \sum_{\mathcal{X}_f \setminus \{x\}} \prod_{y_i \in \{\mathrm{ne}(f) \setminus \{x\}\}} \mu_{y_i \to f}(y_i) \cdot \mu_{f \to y_i}(y_i) \cdot \log\left\{\phi_f(\mathcal{X}_f)\right\} \right) \tag{3.32}$$

   This here is now a variational message because the messages $\mu_{y_i \to f}(y_i) \cdot \mu_{f \to y_i}(y_i)$ guarantee that the parents $y_i$ send their belief (i.e. the expectation of their sufficient statistics) to their children $y_j$ while the children send their natural parameter (requires messages from the co-parents).

4. **Compute the mean field approximation**

   $$q_i(x) \propto \prod_{f_i \in \mathrm{ne}(x)} \mu_{f_i \to x}(x) \tag{3.33}$$

The resulting approximation is better and still feasible because we are only looking at chains and not loopy graphs (cf. the figure to the right). Nonetheless, the chain graph in the right half of the image to the right is just an example. In fact arbitrary chain or tree structures are possible.



### 3.2.6 Expectation propagation

So far, we only focused on models with discrete variables but how do we handle the continuous case? Generally, we need to compute an integral for the message

$$\mu_{f\to x} = \int_{y_i\in\mathcal{X}_f\setminus\{x\}} \phi_f(\mathcal{X}_f) \prod_{y_i\in\{\mathrm{ne}(f)\setminus\{x\}\}} \mu_{y_i\to f}(y_i)\,\mathrm{d}(\mathcal{X}_f\setminus\{x\}) \tag{3.34}$$

in belief propagation. We can solve Equation (3.34) in closed form if we assume all messages and factors to be Gaussian due to the fact that products and marginals of Gaussian are Gaussian again. If we cannot make this assumption, then we must assume that the messages will be a distribution in the exponential family that is

$$m(x) \propto \exp\left(\sum_j g_j(x)\cdot\nu_j\right)$$

with sufficient statistics $g_j(x)$ and weights $\nu_j$. (The sufficient statistics for the Gaussian distribution are simply the first two statistical moments, i.e. $1, x, x^2$.) Now, approximate the true message $\mu$ (which is actually a distribution) with the message $m$ that is the closest distribution in the exponential family; thus we preserve the sufficient statistics!
**Note:** We use the $\mathrm{proj}[\,\cdot\,]$ operator to project the messages to the exponential family at each step. More precisely, minimize the Kullback-Leibler divergence $\mathrm{KL}(\mu||m)$ as we did before for the mean field methods. We end up with belief propagation with exponential family approximations

1. **Initalization** Message from extremal (simplical) node factors are initialized to the factor

$$\mu_{f\to x}(x) = f(x) \tag{3.35}$$

Message from extremal (simplical) variable nodes are set to unity

$$\mu_{f\to x}(x) = 1 \tag{3.36}$$

2. **Variable-to-Factor message**

$$\mu_{x\to f}(x) = \prod_{f_i\in\{\mathrm{ne}(x)\setminus\{f\}\}} \mu_{f_\to x}(x) \tag{3.37}$$

3. **Factor-to-Variable message**

$$\mu_{f\to x} \propto \mathrm{proj}\left[\mu_{x\to f}(x)\cdot\int_{y_i\in\mathcal{X}_f\setminus\{x\}} \phi_f(\mathcal{X}_f) \prod_{y_i\in\mathrm{ne}(f)\setminus\{x\}} \mu_{y_i\to f}(y_i)\,\mathrm{d}(\mathcal{X}_f\setminus\{x\})\right](\mu_{x\to f})^{-1} \tag{3.38}$$

### 3.2.7 Sampling

We learned a lot about exact inference in the last few chapters on inference. However, what options do we have for problems that cannot be solved with exact inference, i.e. when the model is not tree structured? There are generally two approaches on how we can tackle such problems.

1. **Deterministic approximation**
   - Approximate the quantity of interest
   - Solve the approximation analytically
   - Results depends on the quality of the approximation
2. **Sampling**
   - Take the quantity of interest
   - Use random samples to approximate it
   - Result depends on the quality and quantity of random samples

Sampling allows us to get the *golden standard* but we have to wait very long. How do we sample from $p(x)$, e.g. a graphical model? We need to estimate the cumulative sum over the distribution and can select samples $i$ with a uniform (pseudo)random number.

### Ancestral Sampling

Assume a belief network $A \leftarrow B \leftarrow C$ with the distribution

$$p(x) = \prod_i p(x_i \mid \text{pa}(x_i)) = p(A \mid B)p(B \mid C)p(C). \tag{3.39}$$

We use forward sampling to sample from the parents to the children, i.e. from the single distributions which may be difficult. **Note** that each sample drawn using forward sampling is independent! This is called perfect sampling as long as the independence of the instances hold!
A major problem of ancestral sampling is to have evidence, thus when we observe a subset of variables. We can create a workaround if we discard all the inconsistent samples with the trade-off that this fuels inefficiency.

### Rejection Sampling

We want to sample from $p(x)$ but that is too difficult. Assume that we can evaluate $p(x)$ up to a constant $Z^{-1}$. Subsequently, we evaluate the distribution as

$$p(x) = Z^{-1}\tilde{p}(x) = Z^{-1}\prod_c \phi_c(\mathcal{X}_c) \tag{3.40}$$

and choose a proposal distribution $q(x)$ from which we actually can sample, more formally $x_i \sim q(x)$. Now, we need a guarantee that the sample comes from the distribution $p(x)$ which we actually wanted to sample from!
If a uniform sample $u \sim U(0,1)$ suffices the bound

$$u < \frac{\tilde{p}(x_i)}{k \cdot q(x_i)} \tag{3.41}$$

for some $k > 1$ then we will *not* reject the sample $x_i$. The efficiency of the sampler depends on $k$, i.e. the lower the $k$ the more likely it gets that we accept more samples. Rejection sampling is usually impractical in high dimensions but however can be refined with adaptive techniques and multivariate sampling.

### Markov Chain Monte Carlo (MCMC) methods

We seek to sample from a multivariate distribution (other than in the case of rejection sampling)

$$p(x) = Z^{-1}\tilde{p}(x) \tag{3.42}$$

where the calculation of $Z$ is intractable. Here, the idea is to sample from some $q(x_{i+1} \mid x_i)$ with a stationary distribution

$$q_\infty(x') = \int_x q(x' \mid x) q_\infty(x)\, \mathrm{d}x. \tag{3.43}$$

Given the distribution $p(x)$, we try to find a proposal distribution $q(x' \mid x)$ such that $q_\infty(x) = p(x)$ holds.

### Metropolis-Hastings Sampling

The Metropolis-Hastings sampling is a Markov Chain Monte Carlo method. Assume we are in a state $x_i$ of the Markov chain and want to draw a new sample $x'$ from $p(x_i)$ using a propsal distribution $q(x_i)$. The probability that we will use the new sample $x'$ as the next state $x_{i+1}$ is given by

$$\begin{aligned}\hat{p}(x', x_i) &= \min\left(1, \frac{\tilde{p}(x')q(x_i|x')}{\tilde{p}(x_i)q(x'|x_i)}\right) \\ &= \min\left(1, \frac{p(x')q(x_i|x')}{p(x_i)q(x'|x_i)}\right)\end{aligned} \tag{3.46}$$

where $\tilde{p}(x)$ is an unnormalized probability density function. We can show that we desire a stationary distribution $q_\infty(x)$ of such Markov Chain as follows

$$\begin{aligned}p(x)q(x'|x)\hat{p}(x'|x) &= \min\left(p(x)q(x'|x), p(x')q(x|x')\right) \\ &= \min\left(p(x')q(x|x'), p(x)q(x'|x)\right) \\ &= p(x')q(x|x')\hat{p}(x|x')\end{aligned} \tag{3.47}$$

and if the proposal distribution is symmetric, i.e. $q(x_i|x') = q(x'|x_i)$, we can further simplify the equation with

$$\hat{p}(x', x_i) = \min\left(1, \frac{p(x')}{p(x_i)}\right). \tag{3.48}$$

A common choice for the proposal ditribution is a Gaussian distribution where the mean is the current state $x_i$ and the variance makes up the step size of the exploration. In this particular case, a large variance results in many rejections and a small variance results in a slow exploration.

### Gibbs Sampling

Gibbs sampling is a Markov Chain Monte Carlo method, but can also be seen as an instance of the Metropolis-Hastings. The variable $x_i$ we sample from is conditioned on its Markov blanket for sampling, thus we sample from the distribution

$$p(x_i \mid x_{\setminus\{i\}}) = Z^{-1}p(x_i \mid \text{pa}(x_i)) \prod_{j \in \text{ch}(i)} p(x_j \mid \text{pa}(x_j)) \tag{3.44}$$

with the normalisation constant

$$Z = \sum_{x_i} p(x_i \mid \text{pa}(x_i)) \prod_{j \in \text{ch}(i)} p(x_j \mid \text{pa}(x_j)). \tag{3.45}$$

The main issue with Gibbs sampling is the highly dependent states because we are sampling in terms of sequences. We can overcome this issue by considering a burn-in stage to forget our initial state but on the downside we need to sample more resulting in a longer runtime. However, Gibbs sampling is one of the most widely adopted techniques for approximate inference.

irreducibility property, mixing coefficient

### Block Gibbs Sampling

Instead of sampling from $p(x)$, we sample from $p(x, u)$ such that

$$p(x) = \int p(x, u)\, \mathrm{d}u \qquad \text{or} \qquad p(x) = \sum_u p(x, u). \tag{3.49}$$

To sample from the joint distribution $p(x, u)$, we iterate between sampling $p(u|x)$ and sampling $p(x|u)$. This is an useful sampling algorithm for certain MRFs and RBMs (Restricted Boltzmann Machines) where $p(u|x)$ makes sampling easy and $p(x|u)$ is Gaussian so makes it easy to sample as well.

**Slice Sampling**

A disadvantage of the Metropolis-Hastings algorithm is the dependence of finding a good proposal distribution that involves setting step-size parameters of such a distribution to appropriate values. Slice sampling tries to overcome this issue by directly sampling uniformly under the curve of an unnormalized probability density function $\tilde{p}(x) \propto p(x)$ of the probability density function $p(x)$, i.e. drawing samples $(x', u') \sim p(x, u)$. We sample from the joint distribution

$$p(x,u) = \begin{cases} Z_{\tilde{p}(x)}^{-1}, & \text{if } 0 \leq u \leq \tilde{p}(x) \\ 0, & \text{otherwise.} \end{cases} \tag{3.50}$$

with normalization constant $Z_{\tilde{p}(x)}^{-1}$ corresponding to $\tilde{p}(x)$. After a sample from the above distribution has been drawn, we can obtain a sample from the marginal distribution $p(x)$ by dropping the value $u$. We show that this holds as follows

$$\int p(x,u)\,\mathrm{d}u = 0 + \int_0^{\tilde{p}(x)} Z_{\tilde{p}(x)}^{-1}\,\mathrm{d}u + 0 = Z_{\tilde{p}(x)}^{-1}\tilde{p}(x) = p(x). \tag{3.51}$$

Let us take a step back and focus on how to sample from the joint distribution $p(x, u)$. We use Gibbs sampling and alternating draw samples from distributions as follows

$$u \sim p(u|x) = U(0, \tilde{p}(x))$$
$$x \sim p(x|u) = \begin{cases} 1, & \text{if } \tilde{p}(x) \geq u \\ 0, & \text{otherwise.} \end{cases}$$

Here the sample set $\{x \,:\, u \leq \tilde{p}(x)\}$ defines a *slice* and hence the name of the sampling algorithm.

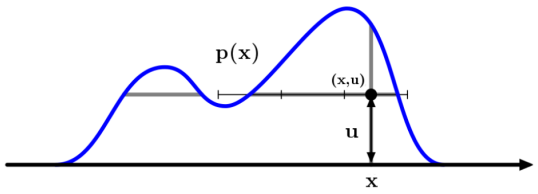



Figure 3.3: Slice sampling from the area underneath the curve and dropping $u$.

**Hybrid / Hamiltonian Monte Carlo**

Metropolis-Hastings sampling explores the proposal distribution with random walk. This approach is not very efficient for the exploration of strongly correlated distributions. We aim to take the model distribution $p(x)$ into account for proposal distributions. We start with considering the energy corresponding to the distribution

$$p(x) \propto \exp\{-E(x)\} \tag{3.52}$$

with the energy term

$$E(x) = -\log\{\tilde{p}(x)\}\,. \tag{3.53}$$

The energy $E(x)$ describes the potential energy of a physical system. We aim to describe the total energy (Hamiltonian)

$$H(x, v) = E(x) + K(\boldsymbol{v}) \tag{3.54}$$

where $K(\boldsymbol{v}) = (2)^{-1}\boldsymbol{v}^\top \boldsymbol{M}^{-1}\boldsymbol{v}$ is the kinetic energy parameterized by the velocity. Furthermore, we assume unit mass, i.e. $\boldsymbol{M} = \boldsymbol{I}$, of the kinetic energy. A particle that moves through the system (e.g. transition in the Markov chain) preserves the total energy $H(\boldsymbol{x}, \boldsymbol{v}) = H(\boldsymbol{x}', \boldsymbol{v}')$ and if we reverse the velocity $\boldsymbol{v}'$, the particle will return from $\boldsymbol{x}'$ to $\boldsymbol{x}$ with velocity $-\boldsymbol{v}$ at arrival. The essential idea is to define a joint distribution

$$p(\boldsymbol{x}, \boldsymbol{v}) \propto \exp\{-E(\boldsymbol{x}) - K(\boldsymbol{v})\} \tag{3.55}$$

following the Hamiltonian energy with Gaussian distributed velocity $\boldsymbol{v}$ that is independent of position $\boldsymbol{x}$. To transition in our system, we do the following:

1. Sample the velocity

$$\boldsymbol{v} \sim p(\boldsymbol{v}) \propto \exp\{-K(\boldsymbol{v})\} \tag{3.56}$$

2. Simulate Hamiltonian dynamics for some time interval arriving at $(\boldsymbol{x}', \boldsymbol{v}')$ with the property

$$q(\boldsymbol{x}', \boldsymbol{v}'|\boldsymbol{x}, \boldsymbol{v}) = q(\boldsymbol{x}, \boldsymbol{v}|\boldsymbol{x}', \boldsymbol{v}') \tag{3.57}$$

3. Accept the move to $(\boldsymbol{x}', \boldsymbol{v}')$ with maximal probability because following holds

$$p(\boldsymbol{x}', \boldsymbol{v}') = p(\boldsymbol{x}, \boldsymbol{v}) \tag{3.58}$$

However, we cannot simulate Hamiltonian dynamics exactly but we can use numerical methods to solve the differential equations (for step 2), i.e. the leapfrog method, and Metropolis-Hastings sampling to accept or reject the transition to the new state (step 3).

## 3.3 Learning in graphical models

The focus shifts from inference in graphical models to learning parameters of graphical models. In the last chapter, we presented many algorithms to infer model distributions directly and approximately. Now we need to tweak the (hyper)parameters of such model distributions. We will now learn about algorithms to do so for the different model distributions.

### 3.3.1 Learning in belief networks

A belief network will be specificed by a model distribution $p(x)$ after inference. However, we need to estimate optimal parameters $\theta$ to shape our model distribution to our needs. This change in shape can be understand in terms of plasticity and we tend to call this issue learning. For the basic setup, we are given the parameterized model distribution

$$p(x \mid \theta) \tag{3.59}$$

and training data $\mathcal{D}$ corresponding to the distribution or our problem task. Basically, there are two different approaches on how to obtain the most optimal parameters $\theta^*$

1. Point estimators (finding exactly one $\theta^*$) and

2. Bayesian learning (integrating out all $\theta$).

The most common point estimator technique is the maximum likelihood estimate approach. Recall that maximizing the maximum likelihood of Equation (3.1) corresponds to minimizing the Kullback-Leibler divergence of the empirical distribution $q(x)$ (from our data) and our model distsribution $p(x)$.

$$\begin{aligned}
\arg\min_p \{\mathrm{KL}\left(q||p\right)\} &= \arg\min_p \left\{-\mathbb{E}_{q(x)}\left[\sum_{i=1}^{K}\log\left\{p(x_i \mid \mathrm{pa}(x_i))\right\}\right] + \text{const}\right\} \\
&= \arg\min_p \left\{-\sum_{i=1}^{K}\mathbb{E}_{q(x_i|\mathrm{pa}(x_i))}\left[\log\left\{p(x_i \mid \mathrm{pa}(x_i))\right\}\right] + \text{const}\right\} \\
&= \arg\min_p \left\{\sum_{i=1}^{K}\left[\mathbb{E}_{q(x_i|\mathrm{pa}(x_i))}\left[\log\left\{p(x_i \mid \mathrm{pa}(x_i))\right\}\right] - \mathbb{E}_{q(x_i|\mathrm{pa}(x_i))}\left[\log\left\{q(x_i \mid \mathrm{pa}(x_i))\right\}\right]\right]\right\} \\
&= \arg\min_p \left\{\sum_{i=1}^{K}\mathrm{KL}\left(q(x_i \mid \mathrm{pa}(x_i))||p(x_i \mid \mathrm{pa}(x_i))\right)\right\}
\end{aligned} \tag{3.60}$$

The maximum likelihoood estimate approach is a heuristic. To use bayesian learning we need to define a prior on the parameters and then compute the posterior

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)p(\theta) \tag{3.61}$$

Here we need to learn a parameter for each factor because each factor corresponds to a probability distribution and the prior distribution decomposes (and hence also the posterior distribution decomposes) into different parts due to independence. For binary events, we shall assume the Binomial distribution for the likelihood but how do we model the prior distribution? We shall choose a beta distribution

$$p(\theta) = B\left(\theta|\alpha,\beta\right) = \frac{\Gamma(\alpha)\cdot\Gamma(\beta)}{\Gamma(\alpha+\beta)}\theta^{\alpha-1}\cdot(1-\theta)^{\beta-1} \tag{3.62}$$

with the Gamma function

$$\Gamma(n) = \int_0^\infty t^{n-1}\exp\left\{-t\right\}\,\mathrm{d}t = (n+1)! \tag{3.63}$$

for the prior because it represents our belief for binary events and is also a conjugate prior. If we choose now to model multiple binary events, as it is the case for a whole bayesian network, we can model our prior with a Dirichlet distribution and the likelihood with a Multinomial distribution which is both the multivariate equivalent of the Beta distribution and the Binomial distribution, respectively.

### 3.3.2 Learning in Markov networks

**Bayesian Learning of Belief Networks** A Markov Network is defined on (not necessarily maximal) cliques that is

$$p(x|\theta) = Z(\theta)^{-1}\prod_c \phi_c(x_c|\theta_c) \tag{3.64}$$

with clique variables $x_c$ and partition function

$$Z(\theta) = \int \prod_c \phi_c(x_c|\theta_c)\,\mathrm{d}x. \tag{3.65}$$

Now we are given training data $\mathcal{D}$ again to do either maximum likelihood or Bayesian learning. For belief networks, the posterior decomposed into different parts due to the independence of prior but this is not the case for Markov Networks in general. The main difficulty is to estimate the unknown partition function $Z(\theta)$.

The most common point estimator technique is the maximum likelihood estimate approach and hence we will use the log variant of it here (for numerical stability). The likelihood of a Markov network is defined in Equation (3.64) and the log variant derives as follows

$$\begin{aligned}\log\{p(x \mid \theta)\} &= \sum_n \sum_c \log\{\phi_c(x_c^n)|\theta_c\} - \sum_n \log\{Z(\theta)\} \\ &= N \cdot N^{-1} \sum_n \sum_c \log\{\phi_c(x_c^n)|\theta_c\} - N \cdot \log\{Z(\theta)\} \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\sum_c \log\{\phi_c(x_c)|\theta_c\}\right] - N \cdot \log\{Z(\theta)\}\end{aligned} \tag{3.66}$$

where $q(x;\mathcal{D}) = N^{-1}\sum_n \delta(x - x^n)$ is the empirical distribution. To obtain the optimal parameter $\theta^*$, we need to take the derivative of the log-likelihood with respect to $\theta$ (and setting it to zero) and we obtain

$$\begin{aligned}\frac{\partial}{\partial\theta_d}\log\{p(x \mid \theta)\} &= \frac{\partial}{\partial\theta_d}\left[N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\sum_c \log\{\phi_c(x_c)|\theta_c\}\right] - N \cdot \log\{Z(\theta)\}\right] \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\sum_c \frac{\partial}{\partial\theta_d}\log\{\phi_c(x_c)|\theta_c\}\right] - N \cdot \frac{\partial}{\partial\theta_d}\log\{Z(\theta)\} \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot \frac{\partial}{\partial\theta_d}\log\left\{\int\prod_c \phi_c\left(x_c|\theta_c\right)\,\mathrm{d}x\right\} \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot Z(\theta)^{-1} \cdot \frac{\partial}{\partial\theta_d}\int\prod_c \phi_c\left(x_c|\theta_c\right)\,\mathrm{d}x \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot Z(\theta)^{-1} \cdot \int\prod_c \frac{\partial}{\partial\theta_d}\phi_c\left(x_c|\theta_c\right)\,\mathrm{d}x \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot Z(\theta)^{-1} \cdot \int\frac{\partial}{\partial\theta_d}\phi_d\left(x_d|\theta_d\right)\cdot\prod_{c\neq d}\phi_c\left(x_c|\theta_c\right)\,\mathrm{d}x \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot Z(\theta)^{-1} \cdot \int\frac{\partial}{\partial\theta_d}\log\{\phi_d\left(x_d|\theta_d\right)\}\cdot\prod_c\phi_c\left(x_c|\theta_c\right)\,\mathrm{d}x \\ &= N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d(x_d)|\theta_d\}\right] - N \cdot \mathbb{E}_{p(x|\theta)}\left[\frac{\partial}{\partial\theta_d}\log\{\phi_d\left(x_d|\theta_d\right)\}\right].\end{aligned} \tag{3.67}$$

However, this requires inference because we need to take an average over the model distribution $p(x|\theta)$. We can extend this even further with an energy view of Markov Random Fields.

**Point estimator: Maximum Likelihood for Markov networks**

We can use energy to express correlation in a system. Therefore, we describe a Markov Network by a Gibbs distribution / Boltzmann distribution

$$p(x \mid \theta) = Z(\theta, T)^{-1} \exp\left\{-T^{-1} E(x; \theta)\right\} \tag{3.68}$$

with temperature $T$ and the potentials defined as

$$E(x; \theta) = \sum_c E_c(x_c; \theta_c). \tag{3.69}$$

The partition function is given by the sum over all $x$ of Equation (3.68). We write the log likelihood gradient as

$$\frac{\partial}{\partial \theta_d} p(x \mid \theta) = -N \cdot \mathbb{E}_{q(x;\mathcal{D})}\left[\frac{\partial}{\partial \theta_d} E(x; \theta)\right] + N \cdot \mathbb{E}_{p(x|\theta)}\left[\frac{\partial}{\partial \theta_d} E(x; \theta)\right] \tag{3.70}$$

There is a closed form solution available for maximum (log) likelihood in decomposable Markov networks. The type of solution can be characterized by unconstrained potentials and exponential family models. Hence, we cannot obtain a closed form solution for non-decomposable Markov networks with constrained forms of the potential and are limited to approximate inference with sampling or approximate the parameter with moment matching methods.

**Energy of Markov Networks**

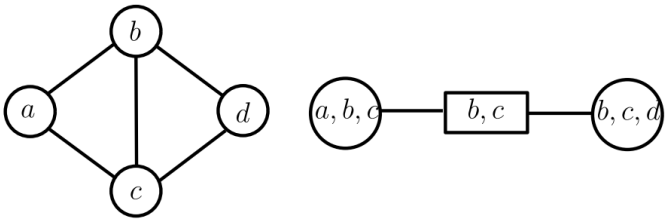
**Unconstrained potentials**

Unconstrained potentials are important to obtain a closed form maximum likelihood solution for Markov networks are no constraints on the potentials, i.e. we can obtain the gradient of real distribution average and do gradient Otherwise, if we have constraints on the potentials, we need to go for constraint optimization techniques.

A Markov network si called decomposable when we can write

$$p(x_1, \ldots, x_n) = \frac{\prod_c p(x_c)}{\prod_s p(x_s)} \tag{3.71}$$

where $p(x_c)$ is a maximal clique and $p(x_s)$ is called a seperator. An example for a decomposable Markov network with its Clique Graph (at the right; also called junction tree) and the decomposable Markov network is shown in the following figure.




The clique graph of the Markov network in the left is given by

$$p(a, b, c, d) = Z^{-1}\phi(a, b, c)\phi(b, c, d) \tag{3.72}$$

where we can write the marginals over $a$ and $d$ as

$$p(b, c, d) \cdot Z = \phi(b, c, d) \sum_a \phi(a, b, c)$$
$$p(a, b, c) \cdot Z = \phi(a, b, c) \sum_d \phi(b, c, d)$$

respectively to conclude that the product of the two marginals

$$\begin{aligned} Z^2 \cdot p(a, b, c)p(b, c, d) &= \left(\phi(b, c, d) \sum_a \phi(a, b, c)\right) \cdot \left(\phi(a, b, c) \sum_d \phi(b, c, d)\right) \\ &= Z^2 \cdot p(a, b, c, d) \sum_{a,d} p(a, b, c, d) \end{aligned}$$

leads to the representation of the Markov network (in the right half of the above figure) as follows

$$p(a, b, c, d) = \frac{p(a, b, c)p(b, c, d)}{p(b, c)}. \tag{3.73}$$

The most important insight here is that we can rewrite the distribution of a Markov network in terms of its marginals on the variables in the original cliques. The structure remains the same and all that has changed is that the original clique potentials have been replaced by the marginals of the distribution and the seperator by the marginal defined on the seperator variables.

**Decomposable Markov Networks**

### 3.3.3 Conditional Random Fields

A Conditional Random Field is defined as a conditional distribution

$$p\left(\boldsymbol{y}|\boldsymbol{x}\right) = Z(\boldsymbol{x}, \theta)^{-1} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{y}, \theta)\right\} \tag{3.74}$$

where the partition function now depends on the input vector

$$Z(\boldsymbol{x}, \theta) = \sum_{\boldsymbol{y} \in \mathcal{Y}} \exp\left\{-E(\boldsymbol{x}, \boldsymbol{y}, \theta)\right\}. \tag{3.75}$$

A Conditional Random Field is similar to a Markov Random Field (special case of the Markov Network), but the condition on the input $\boldsymbol{x}$ makes a huge difference for the partition function! However, the potentials are - as in the Markov Random Fields - implicitly modeled in the energy term. The difference between the Conditional Random and Markov Random Fields is that the Conditional Random Fields describe a discriminant approach while the Markov Random Fields describe a generative approach. The first advantage of learning $p(\boldsymbol{y}|\boldsymbol{x})$ instead of $p(\boldsymbol{x}, \boldsymbol{y})$ is that the learning process is easier in general and only depends on the $\boldsymbol{y}$'s and the second advantage is that we do not need very high-dimensional potentials. However, the discriminative approach is tied to the predicition problem $p(\boldsymbol{y}|\boldsymbol{x})$.

The maximum likelihood for Conditional Random Fields (or conditional likelihood) is given by

$$\begin{aligned} p(\theta; \boldsymbol{x}, \boldsymbol{y}) &= \sum_n \log \left\{ p(\boldsymbol{y}_n | \boldsymbol{x}_n, \theta) \right\} \\ &= -\sum_n E(\boldsymbol{y}_n, \boldsymbol{x}_n, \theta) - \log \left\{ Z(\boldsymbol{x}_n, \theta) \right\} \end{aligned} \tag{3.76}$$

where, again, knowing the partition function $Z$ makes learning tractable in tree-like structures. One ususally uses exponential family Conditional Random Fields that are of the form

$$p(\boldsymbol{y}|\boldsymbol{x}, \theta) = Z(\boldsymbol{x}, \theta)^{-1} \exp \left\{ \theta^\top \phi(\boldsymbol{x}, \boldsymbol{y}) \right\} \tag{3.77}$$

with $\phi(\boldsymbol{x}, \boldsymbol{y})$ being some fixed feature functions. The maximum likelihood problem then becomes

$$\theta^* = \underset{\theta \in \mathbb{R}^D}{\arg\max} \left\{ \sum_n \theta^\top \phi(\boldsymbol{x}_n, \boldsymbol{y}_n) - \sum_n \log \left\{ Z(\boldsymbol{x}_n, \theta)^{-1} \right\} \right\} \tag{3.78}$$

and with this choice, the problem becomes linear in the parameters $\theta$ and hence convex.

Instead of solving Equation (3.78), we can solve the regularized maximum conditional likelihood

$$\begin{aligned} \theta^* &= \underset{\theta \in \mathbb{R}^D}{\arg\max} \left\{ p(\theta) \prod_{n=1}^N p(\boldsymbol{y}_n | \boldsymbol{x}_n, \theta) \right\} \\ &= \underset{\theta \in \mathbb{R}^D}{\arg\min} \left\{ -\log \left\{ p(\theta) \right\} - \sum_{n=1}^N \theta^\top \phi(\boldsymbol{x}_n, \boldsymbol{y}_n) + \sum_n \log \left\{ Z(\boldsymbol{x}_n, \theta) \right\} \right\} \end{aligned} \tag{3.79}$$

which is more robust to overfitting due to the introduced prior. For a Gaussian prior, we obtain the following solution

$$\theta^* = \underset{\theta \in \mathbb{R}^D}{\arg\min} \left\{ \lambda ||\theta||^2 - \sum_{n=1}^N \theta^\top \phi(\boldsymbol{x}_n, \boldsymbol{y}_n) + \sum_n \log \left\{ Z(\boldsymbol{x}_n, \theta) \right\} \right\}. \tag{3.80}$$

The Ising / Potts model can be used for image denoising applications. Here, we assume a Markov Random Field with observed noisy pixels $x_i \in \{-1, +1\}$ and true underlying image pixel $y_i \in \{-1, +1\}$. We *restrict* ourselves to three parameters $h, \beta, \eta$, i.e. the energy function is of the form

$$E(\boldsymbol{x}, \boldsymbol{y}) = h \sum_i \boldsymbol{y}_i - \beta \sum_{i \sim j} \boldsymbol{y}_i \boldsymbol{y}_j - \eta \sum_i \boldsymbol{x}_i \boldsymbol{y}_i \tag{3.81}$$

and the complete Ising model has the form

$$p(\boldsymbol{x}, \boldsymbol{y}) = Z^{-1} \exp\{-E(\boldsymbol{x}, \boldsymbol{y})\}. \tag{3.82}$$

However, we implicitly model $p(\boldsymbol{y}|\boldsymbol{x})$ because we have no knowledge about the true underlying image $\boldsymbol{y}$. And hence we seek the solution given by

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} \{p(\boldsymbol{y}|\boldsymbol{x})\}. \tag{3.83}$$

We cannot obtain the soltuion directly, i.e. in closed form. So, how do we infer the distribution then?

We can obtain an approximated solution with the Iterated Conditional Means method. Given the noisy image $\boldsymbol{x}$, we simply initialize our true underlying image with the noisy image $\boldsymbol{y} = \boldsymbol{x}$. Now it is crucial to pick solely one pixel after another in random or fixed order and minimize the energy for this specific pixel for all possible pixel values (here: $\boldsymbol{y}_i \in \{1, +1\}$). Iterate with this procedure until convergence (choose an appropriate convergence criterium).
**Note:** Iterated Conditional Means will only find a local maximum solution. In general we can do better with minimizing the (true) posterior expected loss

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}'} \left\{ \int_{\boldsymbol{y}} L(\boldsymbol{y}', \boldsymbol{y}) p(\boldsymbol{y}|\boldsymbol{x}) \right\} \tag{3.84}$$

where we still need to be careful with the choice of the loss function $L$! However, this approach will likely only succeed in finding better local maxima for the 0/1 loss which must be sufficiently appropriate for the given application!

### 3.3.4 Latent Variable Models

**Iterated Conditional Means (ICM)** In Latent Variable Models, there are unobserved variables during the training for which we seek a solution. There are two possible solutions presented here to solve for Latent Variable Models. The first solution would be to margnialize out the unobserved variables which may lead to complex dependencies (and other problems). The other approach would include the following algorithms.

The idea of Gaussian Mixture Models is to either estimate a probability density or cluster data where

$$p(\boldsymbol{x}|\theta) = \sum_{j=1}^{m} p(\boldsymbol{x}|\theta_j)p(j) \tag{3.85}$$

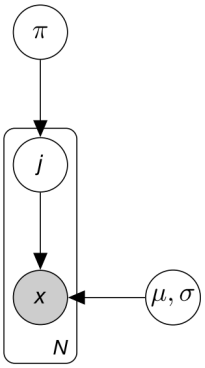
is a smooth density approximation with $m$ many Gaussians. We identify the different terms formally as

$$p(\boldsymbol{x}|\theta) = \sum_{j=1}^{m} p(\boldsymbol{x}|\theta_j)p(j) \tag{3.86}$$

$$p(\boldsymbol{x}|\theta_j) = \mathcal{N}(\boldsymbol{x}|\mu_j, \sigma_j^2) \tag{3.87}$$

$$p(j) = \pi_j \text{ with } 0 \leq \pi_j \leq 1, \sum_{j=1}^{m} \pi_j = 1 \tag{3.88}$$

Hence there are $3 \cdot m$ parameters to learn and the mixture density is a probability distribution!
We can view the Gaussian Mixture Model as a particular Bayesian network (see the figure to the right). But how do we sample from such a mixture model? Basically, we want to sample from $p(\boldsymbol{x}|\theta) = \sum_j \pi_j p(\boldsymbol{x}|\theta_j)$ and hence we first draw a $j \sim p(j)$ and then draw a $x \sim p(\boldsymbol{x}|\theta_j)$. This is Ancestral sampling and we can use it for approximate inference of the latent distribution. The maximum likelihood solution then yields for the paarameters




$$\boldsymbol{\pi}_j = {n_j}^{-1} \cdot n \tag{3.89}$$

$$\boldsymbol{\mu}_j = {n_j}^{-1} \sum_{i:l_i=j} \boldsymbol{x}_i \tag{3.90}$$

$$\boldsymbol{\Sigma}_j = {n_j}^{-1} \sum_{i:l_i=j} (\boldsymbol{x}_i - \boldsymbol{\mu}_i)(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^\top \tag{3.91}$$

with labels $l_i$ of sample $\boldsymbol{x}_i$, $n$ as the total number of labels and $n_j$ as the number of samples with label $j$.
But here we do not know anything about the class label assignments. We can *simply* guess them, based on the current estimated mixture distribution, as

$$\hat{p}(j|i) = p(l_i = j|\boldsymbol{x}_i, \theta), \qquad \sum_{j=1}^{m} \hat{p}(i|j) = 1, \quad \forall i = 1, \ldots, n \tag{3.92}$$

with

$$p(l_i = j|\boldsymbol{x}_i, \theta) = \frac{p(l_i = j, \boldsymbol{x}|\theta)}{p(\boldsymbol{x}|\theta)} = \frac{p(\boldsymbol{x}|\theta_j)p(l_i = j)}{\sum_{j=1}^{m} p(\boldsymbol{x}|\theta_j)p(j)}. \tag{3.93}$$

Given this soft assignment estimates, we can formulate the Expectation Maximization (EM) algorithm for this issue where in the E-step we softly assign samples to mixture components

$$\hat{p}(j|i) \leftarrow p(l_i = j|\boldsymbol{x}_i, \theta) \qquad \forall j = 1, \ldots, m \quad \forall i = 1, \ldots, n \tag{3.94}$$

and in the M-step we re-estimate parameters based on the soft assignments

$$\hat{n}_j \leftarrow \sum_{i=1}^{n} \hat{p}(j|i) \qquad \hat{\boldsymbol{\pi}}_j \leftarrow \frac{\hat{n}_j}{n} \tag{3.95}$$

$$\hat{\boldsymbol{\mu}}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j|i)\boldsymbol{x}_i \qquad \hat{\boldsymbol{\Sigma}} \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j|i)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)^\top. \tag{3.96}$$

**Gaussian Mixture Models (GMM) / Mixture of Gaussians (MoG)** Next, we are going to introduce the Latent Semantic Analysis for which we need to define some more terminology. We define the term frequency (TF) of one document $d$ as

$$\mathrm{tf}(d, w) = \frac{n(d, w)}{\sum_k n(d, k)} \tag{3.97}$$

with $n(d, w)$ as the number of occurences of word $w$ in document $d$. Furthermore, the inverse document frequency (IDF) of a corpus

$$\mathrm{idf}(w) = \log \left\{ \frac{n}{|\{d|n(d, w) \neq 0\}|} \right\} \tag{3.98}$$

can be used to obtain a weighting of entries according to importance

$$\text{tf-idf}(d, w) = \mathrm{tf}(d, w) \cdot \mathrm{idf}(d). \tag{3.99}$$

Often, the document-term matrices are huge, e.g. the internet. The challenges we encounter are the following.

- Compactness: few search terms with rare redundancy

- Variability: synonyms and semantically related terms, expression, writing styles, etc.
- Ambiguity: Terms with multiple senses (polysems)
- Quality & Authority: Correlation between quality and relevance

Assume we are given a matrix $\boldsymbol{A}$ that encodes the data in some way, e.g. co-occurence counts. This matrix will be in general too large, or too complicated, or may have missing or noisy entries, or lack structure. So we try to explore a latent structure underlying the data to explain the entries. We expect to obtain a structure that e.g. encodes semantic topics. The common approach to reveal such a structure is to approximately factorize the matrix, i.e.

$$\boldsymbol{A} \approx \underbrace{\tilde{\boldsymbol{A}}}_{\text{approximation}} = \underbrace{\boldsymbol{L}}_{\text{left factor}} \underbrace{\boldsymbol{R}}_{\text{right factor}} . \tag{3.100}$$

We can obtain such a factorization with the singular value decomposition

$$\text{SVD}(\boldsymbol{A}) = \underbrace{\boldsymbol{L}}_{\text{left singular eigenvectors}} \underbrace{\boldsymbol{\Sigma}}_{\text{singular values}} \underbrace{\boldsymbol{R}}_{\text{right singular eigenvectors}} . \tag{3.101}$$

Here, $\boldsymbol{\Sigma}$ is a diagonal matrix where we set the smallest singularvalues (the ones which are close to zero) to zero, such that we can solve our approximation problem

$$\boldsymbol{A}^* = \underset{\hat{\boldsymbol{A}}: rank(\hat{\boldsymbol{A}})=q}{\arg\min} \left\{ ||\boldsymbol{A} - \hat{\boldsymbol{A}}|| \right\} \tag{3.102}$$

We can do even better than with Latent Semantic Analysis by reformulating the Latent Semantic Analysis with probabilities. For the probabilistic Latent Semantic Analysis (pLSA), we try to maximize the quantity

$$p(R_d = 1|r) \propto p(r|R_d = 1)p(R_d = 1) \tag{3.103}$$

where $R_d$ is the document relevance and $r$ is a query or set of words. We cannot obtain a solution directly but we can use the Expectation Maximization (EM) algorithm to maximize the likelihood.