Master's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Markov Logic Networks with Complex Weights and Algorithms to Train Them

Bc. Jan Tóth

Supervisor: Ing. Ondřej Kuželka, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2021

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Tóth Jan**	Personal ID number:	**457116**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Markov Logic Networks with Complex Weights and Algorithms to Train Them

Master’s thesis title in Czech:

Markovské logické sítě s komplexními váhami a algoritmy pro jejich učení

Guidelines:

1. Familiarize yourself with Markov logic networks (Richardson and Domingos, 2006) and their extension into the complex domain (Kuzelka, 2020).
2. Implement an algorithm/algorithms for marginal inference in Markov logic networks with complex-valued weights (C-MLNs) or in a tractable fragment thereof.
3. Using the solution to the previous point, design and implement a weight-learning algorithm for C-MLNs (optionally also for structure learning).
4. Compare the quality of predictions of your algorithm with classical MLNs.

Bibliography / sources:

1. Richardson, M., and Domingos, P. (2006). Markov logic networks. Machine learning, 62(1-2), 107-136.
2. Kuzelka, O. (2020). Complex markov logic networks: Expressivity and liftability. In Conference on Uncertainty in Artificial Intelligence (pp. 729-738). PMLR.
3. Schulte, O., Khosravi, H., Kirkpatrick, A. E., Gao, T., and Zhu, Y. (2014). Modelling relational statistics with bayes nets. Machine Learning, 94(1), 105-125.
4. Das, M., Wu, Y., Khot, T., Kersting, K., and Natarajan, S. (2016). Scaling lifted probabilistic inference and learning via graph databases. In Proceedings of the 2016 SIAM International Conference on Data Mining (pp. 738-746). Society for Industrial and Applied Mathematics.
5. Van Haaren, J., Van den Broeck, G., Meert, W., and Davis, J. (2016). Lifted generative learning of Markov logic networks. Machine Learning, 103(1), 27-55.

Name and workplace of master’s thesis supervisor:

Ing. Ondřej Kuželka, Ph.D., Intelligent Data Analysis, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **21.02.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **19.02.2023**

Ing. Ondřej Kuželka, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to express my deepest gratitude to my supervisor, Ing. Ondřej Kuželka, Ph.D., for his valuable advice, patience and guidance while elaborating this thesis. This work would not be possible without his collaboration.

Declaration

Abstract

This thesis studies Markov logic networks with complex weights (complex Markov logic networks, $\mathbb{C}$–MLNs). Those are an extension of Markov logic networks that achieves full expressivity in terms of count distributions. Slight modification of the $\mathbb{C}$–MLN definition is proposed attempting to solve a few identified problems. An inference procedure based on Gibbs sampling is developed for the model. Two parameter learning algorithms are proposed as well. The first one utilizes the maximum likelihood estimation. Due to the non-convexity of the problem, gradient descent-based maximization does not perform well. The other learning procedure uses the discrete Fourier transform to encode an arbitrary count distribution as a $\mathbb{C}$–MLN. However, it requires a huge data set, and it learns weights of unnecessarily large dimensions.

Keywords: Markov logic networks with complex weights, inference, Gibbs sampling, learning, maximum likelihood estimation, discrete Fourier transform

Supervisor: Ing. Ondřej Kuželka, Ph.D.

Abstrakt

Tato práce se zabývá markovskými logickými sítěmi s komplexními vahami (komplexními markovskými logickými sítěmi, $\mathbb{C}$–MLNs). Ty jsou rozšířením markovských logických sítí, které je plně expresivní v kontextu počtových distribucí. Text identifikuje několik problémů s původní $\mathbb{C}$–MLN definicí a navrhuje její úpravu. Dále je odvozena procedura pro inferenci založená na Gibbsově vzorkování. Kromě toho jsou odvozeny dva algoritmy pro učení parametrů. První z nich je založen na metodě maximální věrohodnosti. Z důvodu nekonvexity problému nepodává maximalizace založená na gradietním sestupu dobré výsledky. Druhý algoritmus využívá diskrétní Fourierovu transformaci k vyjádření libovolné počtové distribuce jako $\mathbb{C}$–MLN. Algoritmus nicméně vyžaduje velkou trénovací množinu a učí se váhy o zbytečně vysoké dimenzi.

Klíčová slova: markovské logické sítě s komplexními vahami, inference, Gibbsovo vzorkování, učení, metoda maximální věrohodnosti, diskrétní Fourierova transformace

Překlad názvu: Markovské logické sítě s komplexními váhami a algoritmy pro jejich učení

Contents

Chapter 1

Introduction

The are two main approaches to machine learning. Those are *statistical* and *symbolic*.

The statistical approach assumes a probability distribution over the data which it aims to approximate. It often makes several limiting assumptions to do so. The most common being *identically independently distributed* (IID) samples from the target distribution and a fixed-sized vector for each data point that provides all necessary information for accurate approximation of the target distribution. The statistical perspective allows us to mathematically describe the uncertainty that may be present in the data. Among other things, that allows to, relatively with ease, reconcile any apparent contradictions in a data set.

On the other hand, the symbolic approach relies on mathematical logic to describe the data and infer new information (i.e., *learn*) from it. Using logical formulas to represent provided data points allows one to avoid the assumption of a fixed-sized input. It may also lead to a more natural and more easily explainable models. However, the symbolic approach has its drawbacks, as well. For one, it cannot, in its purest form, deal with contradictions in the data set. Furthermore, *first-order logic* (FOL), which is required for any relational modelling, is only *semi-decidable* [1].

Statistical relational learning is a discipline of machine learning that aims to unite the two approaches described above. It deals with relational models that additionally manifest some uncertainty.

Several techniques for combining FOL and probability theory have been proposed. *Markov logic networks* (MLNs) are one of the most straightforward, yet the most powerful among them [2]. Some insufficiencies of MLNs in terms of their expressivity were pointed out in [3]. The paper then proposed an extension of MLNs to *complex Markov logic networks* ($\mathbb{C}$-MLNs), making them fully expressive. However, the new model is incompatible with already existing algorithms developed and implemented for classical MLNs. Some are no longer applicable at all, others must be modified.

This thesis aims to develop basic inference and learning techniques for $\mathbb{C}$-MLNs. To do so, Chapter 2 first summarizes necessary background material, definitions, and some notation that this text works with. Chapter 3 then further explores the $\mathbb{C}$-MLN model and identifies its challenges. Inference and learning are tacked in chapters 4 and 5, respectively. Finally, Chapter 6 describes implementation of the proposed techniques and summarizes experiments performed to assess their quality.

Chapter 2

Background

2.1 First-Order Logic

This text uses a function-free subset of FOL, sometimes referred to as *relational logic*. It should be noted that the following descriptions of logical constructs are brief, and they are, in no way, meant as a replacement for proper mathematical definitions. Those are usually quite cumbersome, and restating them here would not be beneficial to the remainder of this text in any way. Hence, they are omitted with a note that they can be found in more specialized literature, e.g., [1].

2.1.1 Syntax

Subsequent sections assume a finite set of (typed) constants Δ, a finite set of (typed) variables $\mathcal{V}$, and a finite set of predicates $\mathcal{P}$. For easier differentiation, in any examples on the subsequent pages, constants' names shall start with uppercase letters, whereas variables' names with lowercase letters. A term is any constant or variable. An atom (or an atomic formula) is any predicate $p \in \mathcal{P}$ applied to n terms, where n is the given predicate's arity (i.e., $p(t_1, t_2, \ldots, t_n)$ where $t_1, t_2, \ldots, t_n$ are terms). An atom is called a positive literal, and a negation of an atom is called a negative literal. A set of variables occurring in a FOL formula (or just a formula) α is denoted as $vars(\alpha)$. A formula α is called *ground* if $vars(\alpha) = \emptyset$. A substitution is a mapping from variables to terms. Given a formula α and a substitution ρ, $\alpha\,[\rho]$ denotes the application of the substitution ρ to α. Applying the substitution produces a new formula with each variable in α, for which there is an image defined in ρ, replaced by that image.

When working with logical formulas, it can be beneficial (especially in computer science), to only consider formulas in special forms. One such form is a *conjunctive normal form* (CNF), which is a conjunction of *clauses*. A clause is, in turn, a universally quantified disjunction of literals. When working with individual clauses, one can treat them as sets of literals (with their disjunction implied) rather than actual logical formulas.

2.1.2 Semantics

This work adopts *Herbrand semantics* [1] to assign meaning to the language specified in Subsection 2.1.1. In the function-free setting, the *Herbrand universe* becomes the set of all constants Δ. The *Herbrand base* (HB) is a set of all ground atoms whose arguments are only the elements of the Herbrand universe. A *Herbrand interpretation* (also called a possible world and usually denoted ω) is an arbitrary subset of the HB. Atoms contained in an interpretation are those that are considered to be true. Others are considered to be false. Ω denotes the set of all possible worlds, and it is thus the power set of the HB, i.e., 2^{HB}.

2.2 Gibbs Sampling

It is generally challenging to work with multivariate joint probability distributions. Representing them exactly would have (in the discrete case) memory requirements exponential in the number of random variables. Instead, they are often stored only implicitly with the ability to sample from them whenever drawing a data point is required.

One family of sampling algorithms are *Markov chain Monte Carlo* (MCMC) algorithms. Those attempt to generate samples from a probability distribution by constructing an *ergodic* Markov chain whose stationary distribution is the distribution to be sampled from [4]. A popular member of this family is *Gibbs sampling.*

The idea behind Gibbs sampling is that, though it is difficult to query a marginal distribution of a random variable directly, it is relatively easy to query its conditional with all other variables fixed [5]. The Gibbs sampler loops over all variables, gradually querying each of their conditional distributions with all other variables fixed; exchanging each variable for its newly sampled value. The loop is then repeated until convergence.[1]

Algorithm 1 General Gibbs sampler

Require: $\mathbf{X}$ is an n-dimensional random vector and $\mathbf{x}$ a realization thereof
Require: $q(\mathbf{X})$ is a prior distribution

```
1: function GIBBS_SAMPLER($q$)
2:     $\mathbf{x}^{(0)} \sim q(\mathbf{X})$                          ▷ Initialization
3:     for $i \leftarrow 1$ to $\infty$ do
4:         for $j \leftarrow 1$ to $n$ do
5:             $x_j^{(i)} \sim p(\mathbf{X}_j | \mathbf{X}_{<j} = \mathbf{x}_{<j}^{(i)}, \mathbf{X}_{>j} = \mathbf{x}_{>j}^{(i-1)})$
6:     return $\mathbf{x}^{(\infty)}$
```

Algorithm 1 shows the Gibbs sampling procedure in detail. The pseudocode uses convenient but perhaps a bit confusing notation. The conditional

[1]In practice, theoretical convergence is usually replaced by a fixed number of iterations.

distribution $p(\mathbf{X}_j|\mathbf{X}_{<j} = \mathbf{x}^{(i)}_{<j}, \mathbf{X}_{>j} = \mathbf{x}^{(i-1)}_{>j})$ would be more formally written as

$$p(\mathbf{X}_j|\mathbf{X}_1 = x^{(i)}_1, \ldots, \mathbf{X}_{j-1} = x^{(i)}_{j-1}, \mathbf{X}_{j+1} = x^{(i-1)}_{j+1}, \ldots \mathbf{X}_n = x^{(i-1)}_n).$$

From a theoretical point of view, the entire `GIBBS_SAMPLER` procedure produces one sample from the desired distribution p. Following the theory, when several samples from the target distribution are required, one would have to make repeated calls to the sampler, waiting for it to converge every single time. That is often not done in practice, since it usually takes much time for the sampling procedure to converge. Instead, the sampler runs only once, and several samples it produces throughout the entire computation are assumed to be samples from the target distribution p.

The first problem with such an approach is that early samples clearly do not come from the distribution p, since the Markov chain has not yet had enough time to converge. This is referred to as the *burn-in* phase. As it is difficult to estimate how much time is *enough* for the sampler to converge, the sampler is usually extended by a hyperparameter stating how many early samples belong to the burn-in phase and thus should be discarded (rejected).

Another problem arises from the very nature of the Markov chain. Consecutive samples are strongly dependent on each other, violating the assumption of sample independence in Monte Carlo methods. To tackle that, yet another hyperparameter, called *thinning*, is added. Thinning denotes how many samples should be dropped between two accepted samples in order to decrease their statistical dependence.

Gibbs sampling modified in such a way can be somewhat efficiently (depending on its convergence speed) used to compute approximate inference or expectation of a random variable or a function thereof.

2.3 Kullback–Leibler Divergence

Once a probability distribution is approximated, another difficult problem is to assess how well it approximates the underlying phenomenon. This thesis will only make that assessment in simple cases, when the target distribution is known, and its domain is small enough to be stored explicitly.

In that case, *Kullback-Leibler divergence* [6] can be used to measure the difference between the target and the approximation.

Definition 2.1. Let p and q be discrete probability distributions defined on $\mathcal{X}$.

$$\mathbb{KL}(p||q) = \sum_{x\in\mathcal{X}} p(x) \ln \frac{p(x)}{q(x)}, \tag{2.1}$$

where

$$a \ln \frac{a}{0} = \begin{cases} 0 & \text{if } a = 0. \\ \infty & \text{otherwise.} \end{cases}$$

“$\mathbb{KL}(p||q)$ is the information lost when q is used to approximate p”.

(Burnham, K. P. & Anderson, D. R. — Model Selection and Multi-Model Inference (2nd ed.,2002). Springer. p. 51.)

2.4 Complex Analysis

As the name suggests, complex Markov logic network is a probabilistic model that encompasses complex numbers. It is effectively a function of a complex variable. Hence, some basics from the theory of complex functions are also required for further calculations.

It is worth noting that in all non-trivial cases, $\mathbb{C}$-MLNs are actually multi-dimensional complex functions. Despite that, the information presented in this section is for one-dimensional case only. The reviewed theory should be demonstrative enough to help with understanding the subsequent chapters, yet still simple enough so that this chapter does not become unnecessarily cumbersome.

As in Section 2.1, the provided descriptions are brief and they rely on the reader's intuition rather than exact formulations. Proper mathematical definitions are left for more specialized literature (e.g., [7]).

2.4.1 Complex Exponential Function

The basis of all probabilistic models introduced in this text is the exponential function. One way to define the complex exponential function is with a power series. However, Definition 2.2 shows an alternative formulation, which expresses the complex exponential in terms of real exponential and real trigonometric functions. Defining the exponential in such a way will be more advantageous later on.

Definition 2.2. For all $z \in \mathbb{C}$ such that $z = x + iy$, where $x \in \mathbb{R}, y \in \mathbb{R}$ and $i^2 = -1$ is the *imaginary unit*

$$e^z = e^{x+iy} = e^x(\cos(y) + i\sin(y)). \tag{2.2}$$

2.4.2 Holomorphic Functions

In Chapter 5, an optimization-based learning procedure utilizing function's gradient will be derived. That, however, will require the ability to compute the gradient of a complex function.

An analogy of a *differentiable* real function in the complex domain is a *holomorphic* (or *complex differentiable*) function.

A function is holomorphic if two conditions hold. Assume a complex function $f(z) = f(x+iy)$, where $x, y \in \mathbb{R}$ and i is the imaginary unit. Denote

$$\mathfrak{Re}\,(f(z)) = u(x, y),$$
$$\mathfrak{Im}\,(f(z)) = v(x, y).$$

The function f can be written as

$$f(z) = u(x, y) + iv(x, y).$$

The two conditions for holomorphicity then read as follows [7]:

- The partial derivatives of u and v are continuous.
- The functions u and v satisfy the *Cauchy-Riemann equations* presented in Equation 2.3.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \tag{2.3}$$

An important consequence of the *Cauchy-Riemann equations* is the following proposition:

Proposition 2.3. *A function $f(z)$ with $z \in \mathbb{C}$ such that $\mathfrak{Im}\,(f(z)) = 0$ can only be holomorphic if it is a constant.*

Proof. We have

$$f(z) = f(x + iy) = u(x, y) + iv(x, y) = u(x, y) + 0i = u(x, y).$$

By Equation 2.3, it must hold that

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = 0 \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} = 0.$$

Hence, the function $u(x, y) = f(z)$ must be a constant with respect to both x and y. □

2.4.3 Discrete Fourier Transform

Fourier transform (FT) is an integral transformation with various applications across many disciplines. This section presents FT's discrete multi-dimensional formulation, i.e., the *discrete Fourier transform* (DFT), along with its inverse.

Let $d \in \mathbb{N}$ be a positive integer and $\mathbf{M} = [N_1, N_2, \ldots, N_d] \in \mathbb{N}^d$ a vector of positive integers. Define a set of multi-dimensional zero-based indices $\mathcal{J} = \{0, 1, \ldots, N_1 - 1\} \times \{0, 1, \ldots, N_2 - 1\} \times \ldots \times \{0, 1, \ldots, N_d - 1\}$ and assume a function $f : \mathcal{J} \mapsto \mathbb{C}$. The DFT of f, denoted as $\mathcal{F}\{f\} : \mathcal{J} \mapsto \mathbb{C}$, is defined as

$$\mathcal{F}\{f\}(\mathbf{k}) = \sum_{\mathbf{n} \in \mathcal{J}} f(\mathbf{n}) \exp(-i2\pi\langle \mathbf{k}, \mathbf{n}/\mathbf{M}\rangle), \tag{2.4}$$

where $\mathbf{n}/\mathbf{N}$ denotes component-wise division and $\langle \mathbf{a}, \mathbf{b}\rangle$ is the inner product of $\mathbf{a}$ and $\mathbf{b}$.

For a function $g = \mathcal{F}\{f\}$, the inverse DFT $\mathcal{F}^{-1} : \mathcal{J} \mapsto \mathbb{C}$ is then given as

$$\mathcal{F}^{-1}\{g\}(\mathbf{n}) = \frac{1}{|\mathcal{J}|} \sum_{\mathbf{k} \in \mathcal{J}} g(\mathbf{k}) \exp(i2\pi\langle \mathbf{n}, \mathbf{k}/\mathbf{M}\rangle). \tag{2.5}$$

It holds that $f = \mathcal{F}^{-1}\{\mathcal{F}\{f\}\}$.

2.5 Markov Logic Networks

An MLN Φ is a set of pairs (α_i, w_i) where α_i is a function-free FOL formula[2] and $w_i \in \mathbb{R}$ is its weight [2]. Informally speaking, the weight represents how strong a condition the respective formula is. With other things being fixed, the higher the weight w_i, the less probable the world that does not satisfy α_i. Setting $w_i = \infty$ (if weights from the extended real number line are allowed) is equivalent to α_i being universally quantified.

Together with a finite set of constants Δ (the domain), an MLN induces a probability distribution over possible worlds by the following formula:

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z} \exp\left(\sum_{(\alpha,w)\in\Phi} w \cdot N(\alpha,\omega) \right) \tag{2.6}$$

The function $N(\alpha,\omega)$ is the number of substitutions producing groundings of α that are true in the interpretation ω. If α is already ground, $N(\alpha,\omega)$ is defined as an indicator function signalling whether α is satisfied in ω, i.e., $N = I(\omega \models \alpha)$. Often, in literature, $N(\alpha,\omega)$ is simply referred to as a number of true groundings of α in ω. That is a substantially shorter description, however, formally speaking, it is not an accurate one since two distinct substitutions may produce the same grounding of a formula α. Noting that, this thesis consistently refers to the function $N(\alpha,\omega)$ as the number of *true-grounding* substitutions of α in ω.

Z is the so-called *partition function*, a normalizing factor that ensures $p_{\Phi,\Omega}(\omega)$ to be a probability distribution. It is defined straightforwardly as the sum of the nominators over all possible worlds:

$$Z = \sum_{\omega\in\Omega} \exp\left(\sum_{(\alpha,w)\in\Phi} w \cdot N(\alpha,\omega) \right) \tag{2.7}$$

It is worth noting that the original MLN definition introduced in [2] was described as a *Markov random field* (MRF) [8] induced by the pairs (α_i, w_i) and the domain Δ. An MRF is a model for the joint distribution of a set of random variables based on undirected graphs. It is a prominent member of the class of *probabilistic graphical models* that has found many applications, especially in the field of Computer Vision [8] . Equation 2.6 is a *log-linear* model of an MRF.

Once one crosses over from classical Markov logic networks to complex Markov logic networks, the underlying graphical structure is lost (or becomes ambiguous, at the very least). That is why this section describes MLNs by their log-linear model (which will be similar to the definition of $\mathbb{C}$-MLNs in Section 2.6) rather than by describing its underlying graph.

[2]Note, that α_i may contain variables that do not appear in any of its atoms, e.g., $\forall x \forall y p(x)$. That is an important fact when evaluating function $N(\alpha_i,\omega)$.

2.6 Complex MLNs

According to [3], classical MLNs are not fully expressive. In order to reason about the expressivity of MLNs, one must first define a suitable measure. Hence, a few necessary definitions taken from [3] follow.

Definition 2.4. Let Φ be an MLN and Δ be a finite domain. Let $\Psi = \{\alpha_1, \dots, \alpha_m\}$ be the knowledge base of Φ.

$$\mathbf{N}(\Phi, \omega) = \mathbf{N}(\Psi, \omega) = [N(\alpha_1, \omega), \dots, N(\alpha_n \omega)] \tag{2.8}$$

is a vector of the count-statistics for a given interpretation ω.

Definition 2.5. Count Distribution. Let Φ be an MLN defining a distribution $p_{\Phi,\Omega}(\omega)$ over a set of possible worlds Ω. The count distribution of Φ is the distribution of $d-$dimensional vectors of non-negative integers $\mathbf{n}$ given by

$$q_{\Phi,\Omega}(\mathbf{n}) = \sum_{\omega \in \Omega : \mathbf{N}(\Phi,\omega)=\mathbf{n}} p_{\Phi,\Omega}(\omega). \tag{2.9}$$

Definition 2.6. Support of a knowledge base. Let $\Psi = \{\alpha_1, \dots, \alpha_m\}$ be a knowledge base and Ω the set of all possible worlds. The support of Ψ on Ω is defined as

$$\mathrm{Supp}(\Psi, \Omega) = \{\mathbf{N}(\Psi, \omega) | \omega \in \Omega\}. \tag{2.10}$$

With count distributions and the support of the *knowledge base* (KB), the full expressivity can finally be defined as well.

Definition 2.7. Full Expressivity. Let Ω be a set of possible worlds and $\Psi = \{\alpha_1, \dots, \alpha_m\}$ be a set of first-order logic formulas. A class of MLNs given by Ψ is fully expressive if it holds that, for any distribution Q on $\mathrm{Supp}(\Psi, \Omega)$, there exists an MLN Φ such that its count distribution $q_{\Phi,\Omega}$ is equal to Q.

A count distribution is basically a histogram and an MLN is fully expressive if its count distribution can capture an arbitrary distribution on $\mathrm{Supp}(\Psi, \Omega)$.

A sufficient condition for an MLN Φ to be fully expressive in terms of the definition above is that for any $\omega \in \Omega$, $\omega \models \alpha_i$ is satisfied for exactly one α_i from Φ [3]. As that condition is not usually met in practice, an extension of classical MLNs to $\mathbb{C}$-MLNs was proposed in [3] to achieve full expressivity in the general case.

Definition 2.8. $\mathbb{C}$-MLN. Let Ω be a set of possible worlds. A complex Markov logic network is a set $\Phi = \{(\alpha_1, \mathbf{w}_1), \dots, (\alpha_m, \mathbf{w}_m)\}$ where α_i is a first-order logic formula and $\mathbf{w}_i \in \mathbb{C}^d$ is its weight. Φ defines a probability distribution over possible worlds as

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z} \sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right), \tag{2.11}$$

where $[\mathbf{w}_i]_k$ denotes the k-th component of the vector $\mathbf{w}_i$ and

$$Z = \sum_{\omega \in \Omega} \sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right). \tag{2.12}$$

$\mathbb{C}$-MLNs are thus defined directly in terms of a log-linear model (or a mixture thereof) without any underlying graphical model. Definitions 2.4 to 2.7 can be simply copied over from MLNs to $\mathbb{C}$-MLNs, since they only work with the KB Ψ. The change in weights and introduction of a mixture are inconsequential to them.

$\mathbb{C}$-MLNs are fully expressive as long as they contain a trivial formula that is always satisfied (i.e., the tautology denoted by $\top$). [3].

Inspecting Equation 2.11 more closely, one can notice, that $p_{\Phi,\Omega}$ may be complex-valued. Hence, one more definition is introduced.

Definition 2.9. A $\mathbb{C}$-MLN Φ is called *proper* if and only if $p_{\Phi,\Omega}(\omega) \in [0;1]$ for all $\omega \in \Omega$.

Chapter 3

Complex Markov Logic Networks

Complex Markov logic networks offer full expressivity in terms of count distributions. However, as one could expect from the added expressivity, they introduce challenges unknown to classical MLNs.

3.1 New Landscape

3.1.1 Real-Valued Weights

Classical MLNs are explicitly defined in terms of Markov random fields with the formula for the induced probability distribution following from the definition of MRFs. Due to that, they can take advantage of inference and learning techniques developed in the field of probabilistic graphical models. Said techniques include algorithms for exact and approximate inference, as well as algorithms for both parameter and structure learning.

Inference in MRFs can be computed *traditionally* by sampling from the target distribution. Nevertheless, specialized procedures such as *Junction Tree Algorithm*, *Belief Propagation* or *Loopy-Belief Propagation* also exist [8]. Those algorithms leverage the model's underlying graphical structure which allows them to either compute exact inference or, in the case of Loopy-Belief Propagation, compute approximate inference that tends to perform better in practice them sampling-based algorithms. One of the advantages of the graphical structure is that it allows considering only a subset of random variables independently of the rest given only their immediate neighbors (i.e., given their *Markov blanket*).

Learning in MRFs is then usually based on the maximization of a likelihood function of some training set or a similar quantity (e.g., pseudo-likelihood).

3.1.2 Multi-Dimensional Complex-Valued Weights

On the other hand, $\mathbb{C}$-MLNs are defined directly in terms of a probability distribution formula. The formula is similar to standard MRF log-linear models, yet quite different. It has i d-dimensional complex vectors as weights instead of i real numbers. Assuming an MRF would be constructed for a $\mathbb{C}$-MLN in the same way as for an MLN, how would one go about computing its potential? Besides the necessity to deal with complex-valued factors, it is not even clear, if a node in the graph (more accurately, the random variable it represents) would be statistically independent of all other nodes given its Markov blanket.

Inference on $\mathbb{C}$-MLNs can no longer take an advantage of an underlying graph. Nevertheless, methods based on sampling can still be used. Chapter 4 discusses that topic in greater detail.

Unlike MLNs, learning using a likelihood is not as straightforward in $\mathbb{C}$-MLNs. The likelihood function is now a multi-dimensional complex function. Optimizing those requires new considerations that do not apply in the real domain.

Problems with the likelihood more or less stem from problems with the definition itself. They are discussed in detail in Section 3.2.

Nevertheless, it might be possible to approach the learning problem from a completely different direction than from the statistical one. One might attempt to use Fourier transform to tweak network weights so that they would best accommodate training samples. That option is explored in Section 5.2.

3.2 Definition Problems

As was already stated above, when using the definition of $\mathbb{C}$-MLNs introduced in Section 2.6, some issues may be encountered. Let us use concrete examples to demonstrate them.

As was already mentioned, the *distribution* $p_{\Phi,\Omega}(\omega)$ may turn out to be complex-valued.

Example 3.1. Consider a $\mathbb{C}$-MLN Φ along with domain Δ.

$\Phi = \{(heads(x), [0, i\frac{\pi}{2}])\}$

$\Delta = \{A\}$

Then the distribution $p_{\Phi,\Omega}(\omega)$ is given by Equation 2.11 as

$$\begin{aligned} p(\{\}) &= \frac{e^0 + e^0}{Z} = \frac{2}{Z} = \frac{2}{3+i} = \frac{3}{5} - \frac{i}{5} \\ p(\{heads(A)\}) &= \frac{e^0 + (\cos\left(\frac{\pi}{2}\right) + i\sin\left(\frac{\pi}{2}\right))}{Z} = \frac{1+i}{Z} = \frac{1+i}{3+i} = \frac{2}{5} + \frac{i}{5} \\ Z &= 2 + (1+i) = 3 + i \end{aligned}$$

Definition of a proper $\mathbb{C}$-MLN may, at first glance, appear to solve that issue. One must simply select weights in such a way so that the values obtained from Equation 2.11 are valid probabilities. That, however, is not as straightforward when moving from inference to learning.

The question is, how should one even optimize such a function. It would no longer be a task of unconstrained optimization. The feasible set would be limited to all complex vectors producing a proper network. The task could be probably reformulated as a search for $2m$ weights rather than just m with $\forall i \in \{1, 2, \ldots, m\} : \mathbf{w}_i = \mathbf{w}^*_{i+m}$. That, however, does not appear to be an easy task on its own and the possibility of exploring it is left for future work.

Nevertheless, there is another problem with Definition 2.8. For some inputs, it is not even properly defined.

Example 3.2. Consider a $\mathbb{C}$-MLN Φ along with domain Δ.
$\Phi = \{(heads(x), [i\pi, i\pi])\}$
$\Delta = \{A\}$
Then the distribution $p_{\Phi,\Omega}(\omega)$ is given by Equation 2.11 as

$$\begin{aligned} p(\{\}) &= \frac{e^0 + e^0}{Z} = \frac{2}{Z} \\ p(\{heads(A)\}) &= \frac{(\cos(\pi) + i\sin(\pi)) + (\cos(\pi) + i\sin(\pi))}{Z} = \frac{-2}{Z} \\ Z &= 2 + (-2) = 0 \end{aligned}$$

In Example 3.2, we obtain that $Z = 0$ which would lead to division by zero. That might be another problem for a possible optimization procedure.

Of course, both problems shown in the examples above could be solved by an appropriate reformulation of Definition 2.9. However, that would still be only accommodating more fundamental issue with Definition 2.8.[1] In its current form, it only permits a (small) subset of all possible complex vectors as its weights. Classical MLNs suffer no such limitation. Any real number is a valid weight (even positive or negative infinity if one wishes to enforce a formula to hold universally).

An alternative approach would thus be to somehow transform the unnormalized value of $p_{\Phi,\Omega}(\omega)$ in such a way that any weights would produce a valid probability distribution (a proper $\mathbb{C}$-MLN). The remainder of this chapter attempts just that.

3.3 Alternative Definitions

When thinking about alternative definitions of $\mathbb{C}$-MLNs, this work seeks to produce a proper $\mathbb{C}$-MLN regardless of what weights are passed into the model. It is also reasonable to only consider modifications such that the proof of full expressivity of $\mathbb{C}$-MLNs still holds.

Accomplishing that, all theory derived and proven in [3] can be utilized and at the same time, the choice of weights for the model will not be restricted in any way. Any complex vector will be a feasible weight, which will allow the derivation of a gradient-based learning algorithm for $\mathbb{C}$-MLNs in Section 5.1.

3.3.1 Squared Absolute Value

A natural extension of Definition 2.8 is to take the squared absolute value of the unnormalized value of $p_{\Phi,\Omega}(\omega)$. That is analogous to how physicists

[1] It is an issue when trying to approach learning in $\mathbb{C}$-MLNs in a similar way as one approaches learning in MLNs. It is not an issue with the model design overall (except for situations demonstrated by Example 3.2 and later discussed singularities).

evaluate the probability density of particles based on a wave function [9]. The modified Equation 2.11 then reads as

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z}\left|\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right|^2. \tag{3.1}$$

And the appropriate modification of the normalizing constant is

$$Z = \sum_{\omega\in\Omega}\left|\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right|^2. \tag{3.2}$$

It is easy to see that all $\mathbb{C}$-MLNs proper before are proper still. On top of that, any improper networks will be transformed into proper ones.

However, the problem with Equation 3.1 is that such function is not holomorphic. Holomorphicity of a complex function is reviewed in Subsection 2.4.2 for one-dimensional case (i.e., $k = 1$ and $i = 1$). The theory of holomorphic functions is only more complicated for multi-dimensional complex inputs. However, since it is clear from Proposition 2.3 that Equation 3.1 is not holomorphic for $k = 1$ and $i = 1$, this text does not discuss the issue any further.

It is worth mentioning that there are techniques to compute the gradient of non-holomorphic functions [10]. That, however, is beyond the scope of this thesis and left for future work.

3.3.2 Absolute Value of the Real Part

Another possible alternative to Equation 2.11 is to take only the real part of the resulting complex number. That, on its own, is not sufficient as the situation demonstrated in Example 3.2 would still be a problem. Hence, the expression's absolute value is further taken.

The newly proposed distribution is then prescribed as

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z}\left|\mathfrak{Re}\left(\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right)\right|, \tag{3.3}$$

$$Z = \sum_{\omega\in\Omega}\left|\mathfrak{Re}\left(\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right)\right|. \tag{3.4}$$

Transforming Definition 2.8 in such a way will produce a proper network for all cases when the original definition does, with one notable exception. In the case of the real part's argument being a pure imaginary number for all $\omega \in \Omega$, Equation 2.11 would produce a proper network, whereas Equation 3.3 is not even properly defined, since all nominators will be zero, which in turn would cause the denominator to be zero as well.

However, that case is equivalent to multiplying all the nominators by the negative imaginary unit. Doing that, for a complex number $z = iy$ with $y \in \mathbb{R}$, a real number $\hat{z} = y$ is obtained. The resulting distribution will be the same for the original definition.

Naturally, the nominator cannot be arbitrarily changed for cases, when Z would be zero and remain the same for others.[2] Appendix A shows that there exists a transformation of weights that will produce the same distribution as if the real part's arguments were multiplied by $-i$.

With the weights transformed, the $\mathbb{C}$-MLN defined by Equation 3.3 will produce the same distribution as Equation 2.11 would.[3] Distributions defined by the original definition that have the nominator equal to a purely imaginary number for all possible worlds, can thus still be represented by Equation 3.3, albeit with different weights.

There are likely other cases when Equation 3.3 and also Equation 2.11 are not properly defined. Those would be the cases when the sum over the dimensions produces a zero. Such arrangements of weights can be perceived as *singularities* of the $\mathbb{C}$-MLN model. They cannot be avoided in the current setting, but none of the conducted experiments (see Chapter 6 and Appendix C) encountered them. That suggests (as well as intuition does) that the singularities are quite rare. However, a more rigorous analysis should be performed before drawing any decisive conclusions, which is another possible extension of this work.

Disregarding the singularities, Equation 3.3 ensures that the majority of weights that would not produce a proper $\mathbb{C}$-MLN with respect to Definition 2.8, will produce a proper network with respect to Equation 3.3.

Nevertheless, just like Equation 3.1, even Equation 3.3 is not a holomorphic function. However, in this case, that can be circumvented.

Let us denote the real and the imaginary part of each weight vector $\mathbf{w}_i$ as follows:

$$\mathbf{x}_i = \begin{bmatrix} \mathfrak{Re}\left([\mathbf{w}_i]_1\right) \\ \mathfrak{Re}\left([\mathbf{w}_i]_2\right) \\ \vdots \\ \mathfrak{Re}\left([\mathbf{w}_i]_d\right) \end{bmatrix} \qquad \mathbf{y}_i = \begin{bmatrix} \mathfrak{Im}\left([\mathbf{y}_i]_1\right) \\ \mathfrak{Im}\left([\mathbf{y}_i]_2\right) \\ \vdots \\ \mathfrak{Im}\left([\mathbf{y}_i]_d\right) \end{bmatrix} \tag{3.5}$$

Then, each weight can be expressed as

$$[\mathbf{w}_i]_k = [\mathbf{x}_i]_k + i\,[\mathbf{y}_i]_k\,. \tag{3.6}$$

For the sake of brevity in the remainder of the thesis, let us also introduce a special notation for two transformations that will arise later on:

$$\begin{aligned} &\operatorname{expcos}(k, \omega) = \\ &= \exp\left(\sum_{i=1}^{m} \left([\mathbf{x}_i]_k \cdot N\left(\alpha_i, \omega\right)\right)\right) \cos\left(\sum_{i=1}^{m} \left([\mathbf{y}_i]_k \cdot N\left(\alpha_i, \omega\right)\right)\right) \end{aligned} \tag{3.7}$$

[2]In principle, it could. However, that would require changing the definition, and it would pose practical problems since computing the partition function is often intractable.

[3]The network must contain the tautology formula, but since the tautology ensures full expressivity, that is not a very surprising assumption.

$$\mathrm{expsin}(k,\omega) = \\ = \exp\left(\sum_{i=1}^{m}\left([\mathbf{x}_i]_k \cdot N(\alpha_i,\omega)\right)\right)\sin\left(\sum_{i=1}^{m}\left([\mathbf{y}_i]_k \cdot N(\alpha_i,\omega)\right)\right) \tag{3.8}$$

Naturally, those are only properly defined for $k \in \{1, 2, \ldots, d\}$ in the context of a particular $\mathbb{C}$-MLN.

With the above introduced notation in mind, let us modify Equation 3.3 in the following way:

$$\begin{aligned} p_{\Phi,\Omega}(\omega) &= \frac{1}{Z}\left|\mathfrak{Re}\left(\sum_{k=1}^{d}\exp\left(\sum_{i=1}^{m}[\mathbf{w}_i]_k \cdot N(\alpha_i,\omega)\right)\right)\right| \\ &= \frac{1}{Z}\left|\sum_{k=1}^{d}\mathfrak{Re}\left(\exp\left(\sum_{i=1}^{m}[\mathbf{w}_i]_k \cdot N(\alpha_i,\omega)\right)\right)\right| \end{aligned} \tag{3.9}$$

Denote $\mathrm{A}(k,\omega) = \sum_{i=1}^{m}[\mathbf{w}_i]_k \cdot N(\alpha_i,\omega)$. Then

$$\mathfrak{Re}\left(\mathrm{A}(k,\omega)\right) = \sum_{i=1}^{m}\left([\mathbf{x}_i]_k \cdot N(\alpha_i,\omega)\right),$$
$$\mathfrak{Im}\left(\mathrm{A}(k,\omega)\right) = \sum_{i=1}^{m}\left([\mathbf{y}_i]_k \cdot N(\alpha_i,\omega)\right).$$

By applying Equation 2.2, the derivation from Equation 3.9 can then be continued as

$$\begin{aligned} p_{\Phi,\Omega}(\omega) &= \frac{1}{Z}\left|\mathfrak{Re}\left(\sum_{k=1}^{d}\exp\left(\mathrm{A}(k,\omega)\right)\right)\right| \\ &= \frac{1}{Z}\left|\sum_{k=1}^{d}\exp\left(\mathfrak{Re}\left(\mathrm{A}(k,\omega)\right)\right)\left(\cos(\mathfrak{Im}\left(\mathrm{A}(k,\omega)\right)\right)\right| \\ &= \frac{1}{Z}\left|\sum_{k=1}^{d}\mathrm{expcos}(k,\omega)\right| \end{aligned} \tag{3.10}$$

Equation 3.3 can thus be expressed equivalently as a function of only real variables:

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z}\left|\sum_{k=1}^{d}\mathrm{expcos}(k,\omega)\right|, \tag{3.11}$$

$$Z = \sum_{\omega\in\Omega}\left|\sum_{k=1}^{d}\mathrm{expcos}(k,\omega)\right|. \tag{3.12}$$

Formally speaking, Equation 3.11 is also not differentiable[4] due to the absolute value. However, in the setting of optimization, which is where this

[4]Now, only *real differentiable* as $p_{\Phi,\Omega}(\omega)$ in Equation 3.11 is no longer a complex function.

thesis will require computing the gradient of Equation 3.11, the derivative of the absolute value can be substituted by the *signum* function.

It is easy to see, that $|x|' = \operatorname{sgn}(x)$ for all x where $|x|'$ exists (i.e., where $x \in \mathbb{R} \setminus \{0\}$). For $x = 0$, $|x|$ is at a local optimum, where the gradient should be zero, and it, in fact, holds that $\operatorname{sgn}(0) = 0$.

3.4 New Definition

Equation 3.3, or equivalently Equation 3.11, appears to solve the problems with Definition 2.8 outlined by examples in Section 3.2. It ensures that all properly defined $\mathbb{C}$-MLNs are necessarily proper $\mathbb{C}$-MLNs. It will also allow a fairly straightforward development of a likelihood maximization-based learning algorithm in Chapter 5.

For those reasons, a new definition of complex Markov logic networks is formulated. Unless otherwise specified, it will be used as the canonical one in the remainder of this text.

Definition 3.3. $\mathbb{C}$-**MLN**. Let Ω be a set of possible worlds. A complex Markov logic network ($\mathbb{C}$-MLN) is a set $\Phi = \{(\alpha_1, \mathbf{w}_1), \ldots, (\alpha_m, \mathbf{w}_m)\}$ where α_i is a first-order logic formula and $\mathbf{w}_i \in \mathbb{C}^d$ is its weight. Φ defines a probability distribution over possible worlds as

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z}\left|\mathfrak{Re}\left(\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right)\right|, \tag{3.13}$$

where $[\mathbf{w}_i]_k$ denotes the k-th component of the vector $\mathbf{w}_i$ and

$$Z = \sum_{\omega\in\Omega}\left|\mathfrak{Re}\left(\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)\right)\right|. \tag{3.14}$$

As it will be useful to express Equation 3.13 as a function of only real variables, let us also formally state its equivalent form.

Proposition 3.4. *Definition 3.3 may be equivalently expressed as a function of only real variables as*

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z}\left|\sum_{k=1}^{d} \operatorname{expcos}(k, \omega)\right|, \tag{3.15}$$

$$Z = \sum_{\omega\in\Omega}\left|\sum_{k=1}^{d} \operatorname{expcos}(k, \omega)\right|. \tag{3.16}$$

Proof. Proposition 3.4 is proven by the derivation shown in equations 3.9 and 3.10. □

Chapter 4

Inference in Complex MLNs

The fundamental task connected with probabilistic models is *inference*, i.e., given a model, compute the probability of a particular input (i.e., particular realizations of the random variables the model works with).

In the best-case scenario, the probability is computed exactly according to the model's formula (up to numerical errors when delegating the computation to computers). That is known as *exact inference*. Generally, exact inference is a hard problem.

In classical MLNs (which are MRFs), exact inference is a #P-complete problem [11]. There are two main issues when attempting to evaluate Equation 2.6. Firstly, one needs to compute $N(\alpha, \omega)$. That is a complicated task on its own, which is discussed in greater detail in Section 4.1. Secondly, the normalization factor Z needs to be evaluated which requires to sum over the set of all possible worlds Ω (which is exponential in the number of elements of the HB).

One method to compute the normalizer, which can be extended to $\mathbb{C}$-MLNs, is *weighted first order model counting* (WFOMC) [3]. WFOMC is particularly interesting in cases where the set of FOL formulas (the knowledge base) contains only *sentences* (i.e., formulas with no free variables), each of which containing at most two logic variables. Then, WFOMC can be computed in polynomial-time in the number of elements of the domain Δ [3, 12] . However, even WFOMC is still intractable in the general case.

Thus, it is often the case that only *approximate inference* is computed. A general MRF can take advantage of its graphical structure and compute approximate inference using specialized algorithms such as *Loopy-Belief Propagation*. Another approach is to approximate the target distribution by sampling from it. Such procedures are usually based on the MCMC paradigm.

In $\mathbb{C}$-MLNs, lacking the graphical structure, sampling-based inference appears to be a natural choice. There are several sampling algorithms. *Gibbs sampling*, introduced in Section 2.2, is a popular choice, and it is adopted by this work as well. Section 4.2 describes the instance of Gibbs sampler for the case of $\mathbb{C}$-MLNs.

4.1 Counting Substitutions

A subprocedure required by inference in (complex) MLNs is to count the number of true-grounding substitutions of the formula α in the interpretation ω with respect to the domain Δ. The function returning that count is denoted $N(\alpha, \omega)$ throughout this text (the domain Δ being implied by the network which the inference is being computed for).

At first glance, this task may appear simple, but it is actually quite difficult. The naive approach of enumerating all substitutions soon becomes infeasible, as there are exponentially many of those in the number of variables in the formula.

This section deals with devising two algorithms for the exact computation of $N(\alpha, \omega)$, that can be efficient when the KB Ψ is small.

To simplify matters a little, the remainder of the thesis further assumes all the formulas α_i to be clauses. By doing so, a particular network's knowledge base Ψ becomes a general CNF. It may limit the expressivity to some degree, that weights will only be assigned to clauses as opposed to general first-order formulas (CNFs) but such considerations are left for future work.

Straightforward implementation of $N(\alpha, \omega)$ can simply generate all possible grounding substitutions for each formula α in Φ and check how many of them produce a true grounding for the possible world ω. However, there are $2^{|HB|}$ distinct possible worlds ω, and the inference in a particular $\mathbb{C}$-MLN requires to evaluate $N(\alpha, \omega)$ for all $\omega \in \Omega$. Such an approach is thus infeasible for most practical problems.

Furthermore, α was assumed to be a clause (a disjunction). In practice, disjunctive queries often have many answers compared to conjunctive ones since relations (data sets) tend to be sparse. As an example, let us take the relation of friendship, putting together every two individuals on a planet that are friends. The number of pairs of friends will be much smaller than the number of pairs of people who are not friends.

The sparsity of relations also implies that it would be more efficient to evaluate queries without negative literals. With a query expressed as a conjunction, negative literals may be disregarded by employing *Möbius transform* adapted for relational learning [13], which is described in Subsection 4.1.2.

The conversion of a disjunctive query (a clause in the network's KB) to a conjunctive one is discussed in Subsection 4.1.1. Procedures evaluating $N(\alpha, \omega)$ for α being a negation-free conjunction are then proposed in Subsection 4.1.3.

4.1.1 Conjunctive Queries

Assuming α to be a clause, how could one evaluate $N(\alpha, \omega)$ using conjunctive queries only?

Negation of a clause is a conjunction (existentially quantified), but then $N(\neg\alpha, \omega)$ would be evaluated instead of $N(\alpha, \omega)$. However, that is not a problem as the two values are closely related.

The number of true-grounding substitutions of a formula α in a possible world ω is equal to the number of all possible substitutions without those that produce true groundings of $\neg\alpha$ in the given ω.

The total number of possible substitutions of α is given by the size of the domain Δ and the size of $vars(\alpha)$. Each variable may be mapped to any of the constants. Therefore, the total number is given as $|\Delta|^{vars(\alpha)}$.

Hence, one can work with conjunctions of literals rather than disjunctions when counting true-grounding substitutions. The following formula can then be used to appropriately transform the result:

$$N(\alpha, \omega) = |\Delta|^{vars(\alpha)} - N(\neg\alpha, \omega) \tag{4.1}$$

4.1.2 Möbius Transform

Assume a conjunctive query β.[1] Using Möbius transform [13], $N(\beta, \omega)$ can be evaluated without the need for answering any negative literals contained in β.

Denote each atom occurring in β as a_k and partition the formula into formulas A and B as follows:

$$\begin{aligned}
\beta &= \left(\bigwedge_{i=1}^{n} a_i\right) \wedge \left(\bigwedge_{j=1}^{m} \neg a_j\right) \\
A &= \bigwedge_{i=1}^{n} a_i \\
B &= \bigwedge_{j=1}^{m} a_j
\end{aligned} \tag{4.2}$$

One thing to keep in mind is that β may contain variables that do not appear in any of its literals. Those will not be included in either A or B. Furthermore, Equation 4.3 will be counting the true-grounding substitutions of formulas $A \wedge \gamma$, where $\gamma \in 2^B$.[2] $A \wedge \gamma$ may contain even fewer variables still.

All those *free* variables will, however, influence the total number of true-grounding substitutions. For any true-grounding partial substitution constructed only for variables occurring in the atoms, the *free* variables may be assigned any combination of constants from Δ. Hence, any $N(A \wedge \gamma, \omega)$ must be further multiplied by the factor $|\Delta|^{vars(\beta) - vars(A\wedge\gamma)}$.

With the above considerations in mind, Möbius transform may then be given as

$$N(\beta, \omega) = \sum_{\gamma \in 2^B} (-1)^{|\gamma|} \left(|\Delta|^{vars(\beta) - vars(A\wedge\gamma)} \cdot N(A \wedge \gamma, \omega)\right). \tag{4.3}$$

[1]The remainder of this chapter denotes the query as β rather than α to emphasize that it is a conjunction rather than a disjunction. However, the notation is naturally strictly arbitrary.

[2]Here, B is understood as a set of atoms rather than explicit conjunction thereof.

Let us demonstrate the application of Equation 4.3 more concretely by an example.

Example 4.1. Assume a formula $\beta = smokes(x) \wedge \neg cancer(x)$ and a general interpretation $\omega \in \Omega$.

Equation 4.3 gives

$$N(\beta, \omega) = N(smokes(x), \omega) - N(smokes(x) \wedge cancer(x), \omega).$$

The total number of those who smoke, yet do not have cancer, is the number of all those who smoke without those who smoke and have cancer.

4.1.3 Answering Negation-Free Conjunctions

The only missing thing to be able to evaluate $N(\alpha, \omega)$ in (complex) MLNs is to count the true-grounding substitutions of a negation-free conjunctive query (a formula) β in a database (an interpretation) ω . Subsequent paragraphs propose two methods to obtain the count.

Backtracking

A straightforward approach to evaluate $N(\beta, \omega)$ is to recursively search the space of possible variable mappings to constants in the domain. At each step, the search checks that the partial assignment is still true in ω; and if not, the procedure backtracks. Apart from it being simple and reliable, this method can also be quite efficient when the query is not overly complicated, and the database is not large.

The procedure is detailed in Algorithm 2. $\mathcal{M}$ denotes the currently considered (partial) substitution mapping variables from β to constants from Δ. $atom_\beta[\mathcal{M}]$ is then an atom from β with (partial) substitution $\mathcal{M}$ applied to it. Apart from stack primitives `EMPTY`, `PUSH` and `POP`, the pseudocode also assumes the existence of functions `UPDATE` and `REVERSE_UPDATE`. Those extend the substitution $\mathcal{M}$ so that $atom_\omega \models atom_\beta[\mathcal{M}]$ or remove the most recent changes made to the substitution $\mathcal{M}$, respectively.

Relational Database Query

Relational database tables are a natural way of representing relations in computer science. That is also the reason why a possible world ω is sometimes referred to as a *database*.

Each table in the relational database can represent one relation, i.e., predicate. Individual table entries then enumerate all tuples of constants, to which a particular predicate is applied in the interpretation. To compute $N(\beta, \omega)$ using such representation, one would have to, firstly, initialize a new database reflecting the world ω, then translate the formula β into a database query and finally count the number of entries in the resulting table. However, with the prospect of ω changing each iteration of the Gibbs sampling (see Algorithm 3), such an approach does not seem appropriate as the database would be, potentially, changing many times over the course of a single iteration.

Algorithm 2 Answering negation-free conjunctive queries

```
Require: $\beta$ is a conjunction of atoms represented as a stack
Require: $\omega \subseteq \Omega$ is a possible world

1:  function $N(\beta, \omega)$
2:      $\mathcal{M} \leftarrow$ Map<Variable, Constant>          ▷ Current substitution
3:      $count \leftarrow N(\beta, \omega, \mathcal{M})$
4:      return $count$

5:  function $N(\beta, \omega, \mathcal{M})$
6:      if EMPTY($\beta$) then
7:          return 1
8:      $count \leftarrow 0$
9:      $atom_\beta \leftarrow$ POP($\beta$)
10:     for all $atom_\omega \in \omega$ do
11:         if $\exists$ extension of $\mathcal{M}$ s.t. $atom_\omega \models atom_\beta[\mathcal{M}]$ then
12:             UPDATE($\mathcal{M}$)
13:             $count \leftarrow count + N(\beta, \omega, \mathcal{M})$
14:             REVERSE_UPDATE($\mathcal{M}$)
15:     PUSH($\beta$, $atom_\beta$)
16:     return $count$
```

An alternative is to use a database-like data structure that is being stored in memory. Changes can then be applied without much overhead, and the same (or similar) queries can still be issued. Furthermore, it might even be possible to incorporate some optimizations so that the queries do not have to be executed from scratch every time.

The procedure for transforming a formula β into a database query is omitted as that can become quite technical, and it can also differ significantly based on the actual technology used (e.g., SQL database, `pandas.DataFrame`, `DataFrames.jl` etc.). Instead, an example of using a relational database to compute $N(\beta, \omega)$ is supplied in Appendix B.

4.2 Gibbs Sampling

Section 2.2 introduced the general Gibbs sampling algorithm. Here, a specific instance of that algorithm is derived for the case of $\mathbb{C}$-MLNs.

For a network Φ and a domain Δ, Equation 3.15 specifies the probability distribution over all possible worlds. A particular $\omega \in \Omega$ can be represented as a vector of binary random variables, one for each element in the ordered HB. Denote the vector $\mathbf{x}$. If a realization x_i of a particular random variable $\mathbf{x}_i$ is true, then the i-th element of the Herbrand base (denoted a_i further on) is in the interpretation ω. To increase readability of the subsequent equations, let us also denote the unnormalized probability of a possible world as $F_{\Phi,\Omega}(\omega)$:

$$F_{\Phi,\Omega}(\omega) = Z \cdot p_{\Phi,\Omega}(\omega) = \left| \sum_{k=1}^{d} \text{expcos}(k, \omega) \right| \tag{4.4}$$

As is already stated in Section 2.2, it is often infeasible to sample directly from the joint distribution. Sampling from the joint using the marginals may also be computationally demanding since it generally requires summing over a large subset of Ω:

$$P(\mathbf{x}_i = \mathit{true}) = \sum_{\omega \in \Omega : \omega \models a_i} \frac{1}{Z} F_{\Phi,\Omega}(\omega) \tag{4.5}$$

The problem complexity decreases if all other random variables except $\mathbf{x}_i$ are fixed. If that is the case, then the conditional distribution is given as

$$P(\mathbf{x}_i = \mathit{true} | \mathbf{x}_{\neq i} = x_{\neq i}) = \frac{1}{Z} F_{\Phi,\Omega}(\omega_{\mathbf{x}}), \tag{4.6}$$

where $\mathbf{x}_{\neq i} = x_{\neq i} \iff \forall k \neq i : \mathbf{x}_k = x_k$ and $\omega_{\mathbf{x}}$ is the world precisely defined by the query and the evidence passed to the distribution (i.e., defined by the realization of the random vector $\mathbf{x}$). However, Equation 4.6 is still infeasible for most practical problems due to the normalizer Z.

To remove the normalization, one can compute the probability ratio rather than the actual probability:

$$\frac{P(\mathbf{x}_i = \mathit{true} | \mathbf{x}_{\neq i} = a_{\neq i})}{P(\mathbf{x}_i = \mathit{false} | \mathbf{x}_{\neq i} = a_{\neq i})} = \frac{\frac{1}{\cancel{Z}} F_{\Phi,\Omega}(\omega_{\mathbf{x}})}{\frac{1}{\cancel{Z}} F_{\Phi,\Omega}(\bar{\omega}_{\mathbf{x}})} = \lambda \tag{4.7}$$

The $\bar{\omega}_{\mathbf{x}}$ is an interpretation containing all atoms that are contained in $\omega_{\mathbf{x}}$, except for the one whose probability is being computed, i.e., a_i.

The normalizing factors cancel each other out, and the computation simplifies to taking the ratio of the unnormalized probabilities. The actual probability can then be recovered quite easily. For brevity, let the probability be denoted by ρ, i.e.,

$$\rho = P(\mathbf{x}_i = \mathit{true} | \mathbf{x}_{\neq i} = x_{\neq i}).$$

If $\rho = 1$, then the atom a_i will always be in the interpretation.

For cases where $\rho \neq 1$, a linear equation is to be solved:

$$\frac{\rho}{1-\rho} = \lambda \tag{4.8}$$

$$\rho = \frac{\lambda}{1+\lambda} \tag{4.9}$$

Computed ρ is a parameter of a Bernoulli distribution. Sampling from that distribution concludes a single step of one iteration of the Gibbs sampler. Algorithm 3 summarizes the entire procedure.

Algorithm 3 Gibbs sampler for $\mathbb{C}$–MLNs

Require: $\mathbf{x}^{(i)}$ is a vector of n binary random variables representing $\omega^{(i)}$

```
 1: function GIBBS_SAMPLER
 2:     $\mathbf{x}^{(0)} \leftarrow$ RANDOM_BINARY_VECTOR($n$)        ▷ Initialization
 3:     for $i \leftarrow 1$ to $\infty$ do
 4:         for $j \leftarrow 1$ to $n$ do
 5:             if $F_{\Phi,\Omega}(\omega) = 1$ then
 6:                 $x_j^{(i)} = 1$
 7:             else
 8:                 $\lambda \leftarrow F_{\Phi,\Omega}(\omega^{(i)})/F_{\Phi,\Omega}(\bar{\omega}^{(i)})$
 9:                 $\rho \leftarrow \lambda/(1+\lambda)$
10:                 $x_j^{(i)} \sim Beurnolli(\rho)$
11:     return $\mathbf{x}^{(\infty)}$
```

Chapter 5

Learning in Complex MLNs

Inference evaluates the probability of a particular input given the model. In practice, however, one must first obtain the model. Expert knowledge of the problem at hand may be sufficient to formulate the model in some cases. Nevertheless, that may not be applicable every time, and even if it were, it would likely be very tedious and time-consuming. It is a general goal of *artificial intelligence* to automate such procedures. The automation is usually done by assuming a data set of observations (samples obtained from the underlying unknown phenomenon, which should be *learned*) that capture all necessary information, i.e., the *training set.*[1] A learning algorithm then tries to guess a model, that would have generated the training data and that is the most likely mathematical description of the unknown phenomenon.

In the specific case of $\mathbb{C}$-MLNs, there are two things that can be learned. Firstly, there are the model's parameters, i.e., the weights $\mathbf{w}_i$ and their dimensionality d. The majority of models in artificial intelligence are parameterized in some way. Thanks to that, a particular model is then, in fact, a class (a set) of models rather than just one. They all share the same theoretical approach but differ by their parameters which in turn affect how they behave. Estimating such parameters from data is called *parameter learning.*

Secondly, there are the formulas Ψ. In the simpler case, those are already provided along with the parameterized model. However, in the general case, when one attempts to truly automate the entire learning procedure, they may be unknown same as the parameters. Learning those would be *structure learning.* That, however, is a much more complicated task and is left for future work.

Even just for parameter learning, one more consideration should be made regarding the dimensionality d. Although, in theory, it is also the model's parameter, it may be viewed quite differently from the weights $\mathbf{w}_i$. The parameter d can be set to an arbitrary value without any consideration for the weights $\mathbf{w}_i$, whereas the weights cannot be set to any value without first deciding on the dimensionality. From a certain standpoint, it can thus be regarded more as the model's *hyperparameter.* In Section 5.1, a *standard*

[1]More accurate designation would be a training multiset as a sample may repeat several times in the general case.

likelihood maximization-based weight learning algorithm utilizing *gradient descent* (GD) is derived. There, d is considered as a given constant, i.e., a hyperparameter.

Section 5.2 then explores the possibility for learning in the context of $\mathbb{C}$-MLNs that arises from the complex domain and is based on the discrete Fourier transform. There, both $\mathbf{w}_i$'s and d are learned automatically viewing the dimensionality as just another (discrete) parameter.

5.1 Maximum Likelihood Estimation

The *likelihood* function $\mathcal{L}$ assumes provided data points $\mathcal{T}$ to originate from some statistical (parameterized) model (in general, each sample could originate from a different model). $\mathcal{L}$ is then defined as a function of model parameters prescribed by the joint probability distribution of the training data $\mathcal{T}$.

For $\mathcal{T} = \{\omega_1, \omega_2, \dots, \omega_{|\mathcal{T}|}\}$ and the assumption, that a $\mathbb{C}$-MLN generated $\mathcal{T}$, the likelihood is given as

$$\mathcal{L} = \prod_{j=1}^{|\mathcal{T}|} p_{\Phi,\Omega}(\omega_j|\omega_{<j}), \tag{5.1}$$

where $p_{\Phi,\Omega}(\omega_j|\omega_{<j})$ denotes the probability of the j-th sample ω_j given the previous $(j-1)$ samples $\omega_1, \omega_2, \dots, \omega_{j-1}$.

To obtain a model that would most likely generate the data set $\mathcal{T}$, the most natural course of action is to maximize $\mathcal{L}$ with respect to the parameters of the $\mathbb{C}$-MLN model. That is generally referred to as a *maximum likelihood estimation* (MLE).

To simplify the computation of the MLE, IID samples are further assumed. Also, to simplify even more, instead of likelihood $\mathcal{L}$ directly, subsequent calculations work with the *log-likelihood* $\ell = \ln \mathcal{L}$.

$$\ell = \ln \mathcal{L} = \sum_{j=1}^{n} \ln(p_{\Phi,\Omega}(\omega_j)) \tag{5.2}$$

Taking the logarithm turns the product into a sum that makes the evaluation easier and more numerically stable. Since logarithm is an injective function, maximizing ℓ is equivalent to maximizing $\mathcal{L}$.[2] One more consideration regarding the functions' domains is necessary. The likelihood, basically being probability, can take on values from $[0; 1]$. Logarithm, however, is only defined for positive numbers. Nevertheless, it is easy to see that $\mathcal{L}$ cannot be zero-valued, since it is defined by the training samples $\mathcal{T}$. Those were assumed to be generated by the underlying model, and thus none of them can have zero probability (otherwise, they would not have been generated in the first place).

[2]As long as one is only interested in the function's argument that produces the maximum value, i.e., the actual operation is $\arg\max \mathcal{L}$ rather then just $\max \mathcal{L}$.

5.1.1 Gradient Computation

The simplest way to search for the maximum of Equation 5.2, is to search for parameters, for which the gradient $\nabla\ell$ is zero. The gradient is given as

$$\begin{aligned}\nabla\ell &= \nabla\sum_{j=1}^{n}\ln(p_{\Phi,\Omega}(\omega_j))\\ &= \sum_{j=1}^{n}\nabla\ln\left(\frac{1}{Z}\left|\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right|\right)\\ &= \sum_{j=1}^{n}\left(\nabla\ln\left(\left|\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right|\right)-\nabla\ln(Z)\right).\end{aligned}\tag{5.3}$$

Taking a partial derivative from Equation 5.3 for one sample $\omega=\omega_j$ with respect to $[\mathbf{x}_i]_k$, where $i=a$ and $k=b$, gives

$$\begin{aligned}\frac{\partial\log p(\omega)}{\partial\,[\mathbf{x}_a]_b} = &\frac{\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right)\operatorname{expcos}(b,\omega)N(\alpha_a,\omega)}{\left|\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right|}\\ &-\underbrace{\frac{1}{Z}\sum_{\omega'\in\Omega}\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega')\right)\operatorname{expcos}(b,\omega')N(\alpha_a,\omega')}_{Z'}.\end{aligned}\tag{5.4}$$

Computing Z' would be intractable due to the sum over all possible worlds and the normalization by Z. Nevertheless, a simple trick may be employed to turn the value into an expectation of some value with respect to the current model (current weights):

$$\begin{aligned}Z' &= \frac{1}{Z}\sum_{\omega'\in\Omega}\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega')\right)\operatorname{expcos}(b,\omega')N(\alpha_a,\omega')\\ &= \frac{1}{\cancel{Z}}\sum_{\omega'\in\Omega}\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega')\right)\operatorname{expcos}(b,\omega')N(\alpha_a,\omega')\frac{\cancel{Z}p(\omega')}{Zp(\omega')}\\ &= \sum_{\omega'\in\Omega}p(\omega')\frac{\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega')\right)\operatorname{expcos}(b,\omega')N(\alpha_a,\omega')}{\left|\sum_{k=1}^{d}\operatorname{expcos}(k,\omega')\right|}\\ &= \mathbb{E}_p\left[\frac{\operatorname{sgn}\left(\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right)\operatorname{expcos}(b,\omega)N(\alpha_a,\omega)}{\left|\sum_{k=1}^{d}\operatorname{expcos}(k,\omega)\right|}\right]\end{aligned}\tag{5.5}$$

The expectation on its own is still intractable, of course, but it can be approximated by the Gibbs sampling-based inference technique developed in Section 4.2.

Similarly, for the partial derivative with respect to $[\mathbf{y}_i]_k$, where $i = a$ and $k = b$ (with the exact same trick for transforming the value Z''):

$$\begin{aligned}\frac{\partial \log p(\omega)}{\partial\, [\mathbf{y}_a]_b} &= \overbrace{\frac{1}{Z}\sum_{\omega'\in\Omega} \operatorname{sgn}\left(\sum_{k=1}^{d} \operatorname{expcos}(k,\omega')\right) \operatorname{expsin}(b,\omega') N(\alpha_a,\omega')}^{Z''} \\ &\quad - \frac{\operatorname{sgn}\left(\sum_{k=1}^{d} \operatorname{expcos}(k,\omega)\right) \operatorname{expsin}(b,\omega) N(\alpha_a,\omega)}{\left|\sum_{k=1}^{d} \operatorname{expcos}(k,\omega)\right|}\end{aligned} \tag{5.6}$$

$$Z'' = \mathbb{E}_p\left[\frac{\operatorname{sgn}\left(\sum_{k=1}^{d} \operatorname{expcos}(k,\omega)\right) \operatorname{expsin}(b,\omega) N(\alpha_a,\omega)}{\left|\sum_{k=1}^{d} \operatorname{expcos}(k,\omega)\right|}\right] \tag{5.7}$$

The overall result resembles the gradient of classical MLNs. For an MLN Φ, the gradient reads as follows [2]:

$$\frac{\partial \log p_{\Phi,\Omega}(\omega)}{\partial w_i} = N(\alpha_i,\omega) - \mathbb{E}_p[N(\alpha_i,\omega)]. \tag{5.8}$$

And from the computations above, for a $\mathbb{C}$-MLN Φ:

$$\begin{aligned}\frac{\partial p_{\Phi,\Omega}(\omega)}{\partial [\mathbf{x}_i]_k} &= C(k,\omega)N(\alpha_i,\omega) - \mathbb{E}_p[C(k,\omega)N(\alpha_i,\omega)] \\ \frac{\partial p_{\Phi,\Omega}(\omega)}{\partial [\mathbf{y}_i]_k} &= \mathbb{E}_p[S(k,\omega)N(\alpha_i,\omega)] - S(k,\omega)N(\alpha_i,\omega)\end{aligned} \tag{5.9}$$

The gradient can thus be interpreted as an error between the weighted[3] number of true-grounding substitutions of α_i in $\mathcal{T}$ and the expected value thereof computed with respect to the current model. At optimum, the error should be zero.

5.1.2 Gradient Ascent

Since there is no closed-form solution for the partial derivatives in Equation 5.9 to be zero, one must resort to numerical methods to find the likelihood's critical points. One of the most simple and straightforward methods for such optimization problems is to follow the direction of the steepest ascent, i.e., the gradient. Once the gradient is zero, a critical point is reached.

The function minimization technique generally called gradient descent, used already by Cauchy [14], is based on that idea. At any point of a function, the procedure follows the negative gradient, which causes the greatest decrease in the function value. The technique can easily be used for maximization instead by following the (positive) gradient in order to increase the function value as much as possible at any point. That variant is sometimes referred to as *gradient ascent*.

[3]Weighing is the only difference from classical MLNs.

In the context of $\mathbb{C}$-MLNs, the optimization is clearly non-convex. Following the gradient can lead to a local maximum, a local minimum or even a saddle point. Following that, one might argue that GD may thus not be the most suitable learning technique for the $\mathbb{C}$-MLN model. Nevertheless, GD is easy to use and simple to implement. It will be used with the caveat that the procedure should be run multiple times with different initialization at each time to maximize the chance that a (at least local) maximum was found.

5.2 DFT-Based Learning

Section 5.1 takes a traditional statistical approach to learning $\mathbb{C}$-MLNs using MLE. While that is perfectly valid, the complex domain offers another way.

The idea is already implicitly contained in [3], and it could also be perceived as an alternative proof of full expressivity of $\mathbb{C}$-MLNs. The full expressivity is defined in terms of count distributions, i.e., for any count distribution, there exists a $\mathbb{C}$-MLN whose count distribution is the selected one. Let us then first estimate the count distribution and then construct the appropriate network.

For the subsequent paragraphs, assume a KB $\Psi = \{\alpha_1, \dots, \alpha_m\}$ and data samples $\mathcal{T} = \{\omega_1, \omega_2, \dots, \omega_{|\mathcal{T}|}\}$ drawn from a set of all possible worlds Ω. Further assume that $\mathcal{T}$ contains all necessary information to reliably estimate a count distribution $\widehat{q}_{\Psi,\Omega}(\mathbf{n})$ from it. The goal is to find a $\mathbb{C}$-MLN Φ with a count distribution $q_{\Phi,\Omega}(\mathbf{n})$ such that $q_{\Phi,\Omega}(\mathbf{n}) = \widehat{q}_{\Psi,\Omega}(\mathbf{n})$. The goal can be met as long as full expressivity is assured, i.e., $\top \in \Psi$. Without lost of generality, assume $\alpha_1 = \top$.

Firstly, observe that for any $\omega, \omega' \in \Omega$ such that $\mathbf{N}(\Psi, \omega) = \mathbf{N}(\Psi, \omega')$, it holds that $p_{\Phi,\Omega}(\omega) = p_{\Phi,\Omega}(\omega')$ [3]. Following that, an *inverse* transformation from count distributions back to the original probability distribution can be derived as

$$p_{\Phi,\Omega}(\omega) = \begin{cases} \frac{q_{\Phi,\Omega}(\mathbf{N}(\Psi,\omega))}{\mathbf{MC}(\Psi,\mathbf{N}(\Psi,\omega))} & \text{if } \mathbf{MC}(\Psi, \mathbf{N}(\Psi, \omega)) \neq 0, \\ 0 & \text{otherwise}, \end{cases} \tag{5.10}$$

where $\mathbf{MC}(\Psi, \mathbf{n})$ denotes the model count, i.e., the number of possible worlds $\omega \in \Omega$ such that $\mathbf{N}(\Psi, \omega) = \mathbf{n}$.

Analogously to Equation 5.10, an estimate of the probability distribution (in terms of the count-statistics) can be defined using the estimated count distribution $\widehat{q}_{\Psi,\Omega}$:

$$\widehat{p}_{\Phi,\Omega}(\mathbf{n}) = \begin{cases} \frac{\widehat{q}_{\Psi,\Omega}(\mathbf{n})}{\mathbf{MC}(\Psi,\mathbf{n})} & \text{if } \mathbf{MC}(\Psi, \mathbf{n}) \neq 0, \\ 0 & \text{otherwise}. \end{cases} \tag{5.11}$$

Denote $\mathcal{D} = \{0, 1, \dots, |\Delta|^{vars(\alpha_1)}\} \times \ldots \times \{0, 1, \dots, |\Delta|^{vars(\alpha_m)}\}$ the domain of $\widehat{p}_{\Phi,\Omega}(\mathbf{n})$ and $\mathbf{M} = [|\Delta|^{vars(\alpha_1)} + 1, |\Delta|^{vars(\alpha_2)} + 1, \dots, |\Delta|^{vars(\alpha_m)} + 1]$ the vector holding the domain's dimensions. Note that for $\alpha_1 = \top$, $|\Delta|^{vars(\alpha_1)} = 1$ and thus $[\mathbf{M}]_1 = 2$.

$\widehat{p}_{\Phi,\Omega}(\mathbf{n})$ is a mapping $\mathcal{D} \mapsto \mathbb{C}$,[4] meaning a DFT per Equation 2.4 can be computed for it. Let $g(\mathbf{k}) = \mathcal{F}\{\widehat{p}_{\Phi,\Omega}\}$.

Let us recover $\widehat{p}_{\Phi,\Omega}(\mathbf{n})$ using the inverse DFT:

$$\begin{aligned}\widehat{p}_{\Phi,\Omega}(\mathbf{n}) &= \frac{1}{|\mathcal{D}|} \sum_{\mathbf{k}\in\mathcal{D}} g(\mathbf{k}) \exp(i2\pi\langle \mathbf{n}, \mathbf{k}/\mathbf{M}\rangle) \\ &= \frac{1}{|\mathcal{D}|} \sum_{\mathbf{k}\in\mathcal{D}: g(\mathbf{k})\neq 0} \exp(\log(g(\mathbf{k})) + i2\pi\langle \mathbf{n}, \mathbf{k}/\mathbf{M}\rangle)\end{aligned} \tag{5.12}$$

The complex logarithm used in Equation 5.12 is defined on $\mathbb{C}\setminus\{0\}$. The cases where $g(\mathbf{k}) = 0$ can be excluded, since then the product $g(\mathbf{k}) \exp(i2\pi\langle \mathbf{n}, \mathbf{k}/\mathbf{M}\rangle)$ is zero.

By the assumption, that $q_{\Phi,\Omega}(\mathbf{n}) = \widehat{q}_{\Psi,\Omega}(\mathbf{n})$, it follows that:

$$\begin{aligned}p_{\Phi,\Omega}(\omega) &= \frac{q_{\Phi,\Omega}(\mathbf{N}(\Psi,\omega))}{\mathbf{MC}(\Psi, \mathbf{N}(\Psi,\omega))} = \widehat{p}_{\Phi,\Omega}(\mathbf{N}(\Psi,\omega)) \\ &= \frac{1}{|\mathcal{D}|} \sum_{\mathbf{k}\in\mathcal{D}: g(\mathbf{k})\neq 0} \exp(\log(g(\mathbf{k})) + i2\pi\langle \mathbf{N}(\Psi,\omega), \mathbf{k}/\mathbf{M}\rangle)\end{aligned} \tag{5.13}$$

The argument of the exponential function in Equation 5.13 can be compacted into a single inner product. Firstly, denote $\mathcal{J}$ the set of indices j of elements $\mathbf{k}_j \in \mathcal{D}$, such that

$$j \in \mathcal{J} \implies g(\mathbf{k}_j) \neq 0.$$

Next, construct a vector $\mathbf{a}_j \in \mathbb{C}^m$ such that $\mathbf{a}_j = [\log(g(\mathbf{k}_j)), 0, 0, \ldots, 0]$. Finally, denote

$$\mathbf{v}_j = i2\pi\mathbf{k}_j/\mathbf{M} + \mathbf{a}_j. \tag{5.14}$$

Then Equation 5.13 can be rewritten as

$$p_{\Phi,\Omega}(\omega) = \frac{1}{|\mathcal{D}|} \sum_{j\in\mathcal{J}} \exp(\langle \mathbf{v}_j, \mathbf{N}(\Psi,\omega)\rangle). \tag{5.15}$$

The trick to defining $\mathbf{v}_j$ as is done in Equation 5.14 is in the assumption that $\alpha_1 = \top$. Then, $[\mathbf{N}(\Psi,\omega)]_1 = N(\top,\omega) = 1$. The value $\log(g(\mathbf{k}_j))$ thus gets added to a number that is multiplied by 1 in the inner product. Hence, the value is effectively added to the result of the inner product.

Equation 5.15 is a very advantageous form, since by setting

$$\begin{aligned}\mathcal{J} &= \{1, 2, \ldots, d\}, \\ \mathbf{w}_i &= [[\mathbf{v}_1]_i, [\mathbf{v}_2]_i, \ldots, [\mathbf{v}_d]_i], \\ Z &= |\mathcal{D}|,\end{aligned} \tag{5.16}$$

it can be rewritten into the original $\mathbb{C}$-MLN formula:

$$p_{\Phi,\Omega}(\omega) = \frac{1}{Z} \sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i,\omega)\right)$$

[4] More specifically, it is a mapping $\mathcal{D} \mapsto [0;1]$, but $[0;1] \subset \mathbb{C}$.

Thus, by using the DFT, not only the weights $\mathbf{w}_i$ can be learned but also the dimensionality d that had to be fixed when learning using MLE. There is no intractable inference subprocedure required. Also, the problems of the original $\mathbb{C}$-MLN definition discussed in Section 3.2 do not apply here. There was a probability distribution estimate $\widehat{p}_{\Phi,\Omega}(\mathbf{n})$ inputted into the DFT, so its inverse will again be a probability distribution taking on values from the interval $[0; 1]$.

However, there are other practical considerations to this approach. First and foremost, how to evaluate the **MC** function? That is, in fact, another intractable task. Exact evaluation requires iterating over the entire Ω set. Nevertheless, it can be approximated by (uniform) sampling. It will, however, require linear memory space in the size of $\mathcal{D}$.

Another problem is the discrete Fourier transform subprocedure. Although efficient algorithms for evaluating the DFT exist, they will still likely be overwhelmed when the problem size significantly increases.

Next, this approach may learn the parameter d on its own, however, the parameter is not bounded by anything except the size of $\mathcal{D}$. It may happen that the DFT will learn a uselessly large value of d, producing unnecessarily intricate model.

Last but not least, $\mathcal{T}$ was assumed to contain all necessary information for accurate estimation of the count distribution. In practice, that means having a lot of samples, which may not be satisfied in many scenarios.

Chapter 6
Experiments

The techniques proposed in previous chapters were implemented and experimentally tested in the *Julia programming language* [15]. Julia is an open-source, modern, high-level, dynamically typed programming language that relies heavily on the multiple dispatch paradigm.

Experiments use formulas from the *friends and smokers* dataset [2]. A classical example often used in literature that captures smoking habits among friends.

6.1 Implementation

The implementation is mostly straightforward. Rather than provide an application-ready library full of obscure optimizations, its purpose is to be a proof of concept for the proposed algorithms, which can be used to decide their scalability and usefulness, as well as to identify possible future areas of interest.

The code is structured into a Julia package. The source files are split into several subfolders, each implementing a particular logical unit, e.g., `fol`, `inference` or `learning`.

The central type is `MLN` which is a structure holding the knowledge base and possibly weights. `MLN` can then be wrapped along with a set of constants (a domain) into a `ConcreteMLN`, which induces a particular probability distribution over possible worlds and allows sampling from it.

The sampling is performed by iterating over a structure `GibbsSampler` that takes a `ConcreteMLN` as one of its arguments upon construction. The sampling is slightly optimized by caching the results of $N(\alpha, \omega)$ for the most recently used interpretations. `LRUCache.jl` package[1] is used for the caching. The sampler is not optimized for high numerical stability nor sparsity.

GD-based learning is implemented with a fixed-sized step rather than line search. The convergence is evaluated by inspecting the gradient's norm.

For the second learning procedure, computation of the DFT is delegated to the `FFTW.jl` package [16]. The **MC** function is approximated by uniform sampling from Ω.

[1]`https://github.com/JuliaCollections/LRUCache.jl`

6.2 Counting Substitutions

Section 4.1 concluded by suggesting two procedures to exactly compute the number of true-grounding substitutions of a formula in an interpretation.

The first algorithm is based on a recursive search of the space of all possible substitutions, backtracking whenever unsatisfiability is detected. The second one relies on joining tables in a relational database.[2] As obtaining the count is generally a challenging problem, neither of those techniques is particularly useful for large-scale problems. Nevertheless, this section briefly compares their running times on small examples to demonstrate their pros and cons. The presented running times are median running times obtained using `BenchmarkTools.jl` package [17]. All the measurements were performed on a laptop with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 16GB DDR4 RAM and HDD.

Table 6.2 shows the median running times for the recursive-based (denoted R in the table) and the database-based (denoted DB) true-grounding substitutions-counting procedures. The running times are provided for several simple formulas with the interpretation fixed to atoms presented by Table 6.1[3] with $\Delta = \{Anna, Bob, Chris, Daniel, Edward, Frank, Gary, Helen\}$.

cancer
Anna
Edward

friends	
Edward	Frank
Anna	Bob
Bob	Anna
Anna	Edward
Edward	Anna
Anna	Frank
Frank	Edward
Chris	Daniel
Daniel	Chris
Gary	Helen
Helen	Gary
Gary	Anna
Anna	Gary
Frank	Anna
Bob	Chris
Chris	Bob

smokes
Anna
Edward
Frank
Gary

Table 6.1: A possible world with $|\Delta| = 8$

One can observe that the database-based approach suffers greatly from the overhead of creating new tables upon invokation. The recursive algorithm

[2]Actual implementation was done using `DataFrames.jl` package with tables stored in memory rather than in an external database. The package can be found at `https://github.com/JuliaData/DataFrames.jl`.

[3]The dataset was taken from `https://alchemy.cs.washington.edu/data/tutorial/smoking/smoking-train.db`.

Formula	N	R [ms]	DB [ms]
$\top$	1	0.23	0.23
$\bot$	0	0.23	0.23
$smokes(x)$	4	0.26	48
$cancer(x)$	2	0.26	50
$cancer(x) \land smokes(x)$	2	0.29	106
$cancer(x) \lor smokes(x)$	4	0.35	232
$\neg cancer(x) \land \neg smokes(x)$	4	0.33	246
$\neg cancer(x) \lor \neg smokes(x)$	6	0.31	111
$cancer(x) \land \neg smokes(x)$	0	0.31	174
$cancer(x) \lor \neg smokes(x)$	6	0.33	202
$\neg cancer(x) \land smokes(x)$	2	0.32	196
$\neg cancer(x) \lor smokes(x)$	8	0.40	202
$friends(x, y) \land smokes(x) \land \neg smokes(y)$	2	0.47	349
$friends(x, y) \lor smokes(x) \lor \neg smokes(y)$	50	0.54	558
$\neg friends(x, y) \land \neg smokes(x) \land smokes(y)$	14	0.51	585
$\neg friends(x, y) \lor \neg smokes(x) \lor smokes(y)$	62	0.52	356
$friends(x, y) \land smokes(x)$	10	0.36	143
$friends(x, y) \lor smokes(x)$	38	0.44	295
$\neg friends(x, y) \land \neg smokes(x)$	26	0.42	296
$\neg friends(x, y) \lor \neg smokes(x)$	54	0.40	150

Table 6.2: Running times for $N(\alpha, \omega)$ counting procedures with $|\Delta| = 8$

is much faster. However, Table 6.3 shows running times for a twice as large domain size. The exact interpretation used for those measurements is not provided as that would be extremely long and not at all demonstrative.

It should be sufficiently informative to say that a HB with atoms constituting from predicate symbols appearing in Table 6.3 and a domain with 16 elements was constructed, and one of its subsets was sampled. The possible world contained 142 *friends* predicates, 9 *smokes* predicates and 5 *cancer* predicates.

Table 6.3 demonstrates that the recursion-based procedure's running time increases radically with enlarging the domain, whereas the database-based procedure is much more resilient to such change. That is to be expected since there, the table joining is the bottleneck. Naturally, the running time would increase with adding more predicates, but that would also increase the running time of the first algorithm.

The backtracking algorithm thus appears more suitable for cases with small-sized domains. For larger domains, the database-based procedure may be more advantageous. Naturally, neither is suitable for large domains or KBs.[4]

[4]The database-based algorithm allows for further optimizations in query evaluations and table joining. Then, it can be useful for large-scale problems.

Formula	N	R [ms]	DB [ms]
$friends(x, A) \wedge smokes(x)$	6	1.98	122
$friends(x, A) \vee smokes(x)$	12	2.13	244
$\neg friends(x, y) \wedge \neg smokes(x) \wedge smokes(y)$	32	4.48	440
$\neg friends(x, y) \vee \neg smokes(x) \vee smokes(y)$	226	9.22	292
$cancer(x) \wedge \neg smokes(x)$	4	1.92	168
$cancer(x) \vee \neg smokes(x)$	10	1.92	178

Table 6.3: Running times for $N(\alpha, \omega)$ counting procedures with $|\Delta| = 16$

6.3 Learning

This section shows a few attempts to learn a $\mathbb{C}$-MLN model on simple (small) examples. Since $\mathbb{C}$-MLNs were defined to achieve full expressivity and full expressivity is defined in terms of count distributions, the examples assess the approximation quality of the count distribution rather than assessing the distribution over possible worlds directly.

Small examples allow us to store count distributions exactly and measure the information lost in approximation using Kullback-Leibler ($\mathbb{KL}$) divergence reviewed in Section 2.3. Moreover, for one-element or two-element KBs, the count distributions can even be easily visualized.[5]

As was already mentioned in Subsection 5.1.2, the function given by Equation 3.13 is non-convex. Gradient-based maximization will thus lead to, at best, local maxima. It may, however, stop at any critical point. The examples below mostly present the best-obtained results over multiple runs of the GD-based learning. Appendix C then shows other critical points found by the algorithm, indicating the sheer volume of weights that one needs to consider.

6.3.1 Non-Relational Case

To start with, let us first consider the case with a single formula $smokes(x)$ in the knowledge base.[6] Such KB cannot capture any relations in the domain. Nevertheless, it will already demonstrate some challenging properties connected with learning $\mathbb{C}$-MLNs.

It is easy to see that

$$N(smokes(x), \omega) = |\omega|. \tag{6.1}$$

The count distribution can thus be visualized using a histogram.

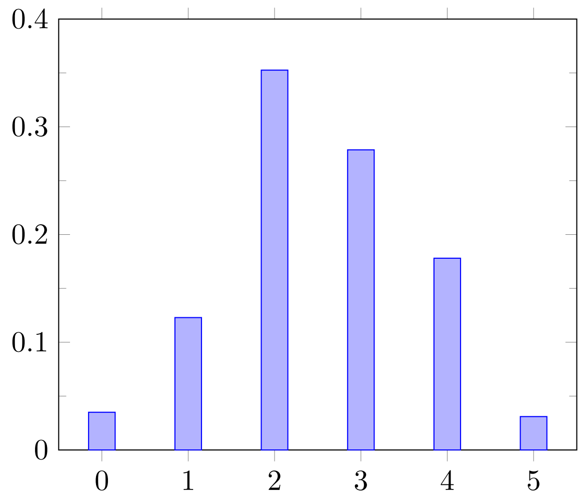
[5]When visualizing a count distribution, if the KB contains the tautology formula $\top$, the tautology's dimension can be disregarded. Assuming $\alpha_1 = \top$, count distribution can be non-zero only for inputs with the first coordinate set to one.

[6]To ensure full expressivity, the KB will be extended by $\top$ implicitly.

Example 6.1. Consider $|\Delta| = 5$ and $\mathcal{T}$ such that it specifies a binomial count distribution with the probability of each sample $p = 0.5$. One such example of $\mathcal{T}$ contains each element of Ω once.

Such count distribution is equivalent to $p_{\Phi,\Omega}(\omega)$ being a uniform distribution over Ω and it can be captured even by a classical MLN with the weight $w = 0$.

Figure 6.1 shows the best obtained GD result for $d = 2$. Except the histogram visualizing the approximated count distribution, the figure also contains the learned weights for each formula, rounded to three decimal places, and the $\mathbb{KL}$ divergence of the approximation from the true (target) count distribution, rounded to five decimal places.



$\mathbb{KL} = 0.01039$

$smokes(x)$	$\top$
$-0.155 + 0.000i$	$0.009 + 0.000i$
$0.031 + 3.142i$	$1.559 - 0.000i$

Figure 6.1: GD-approximated binomial count distribution with $d = 2$

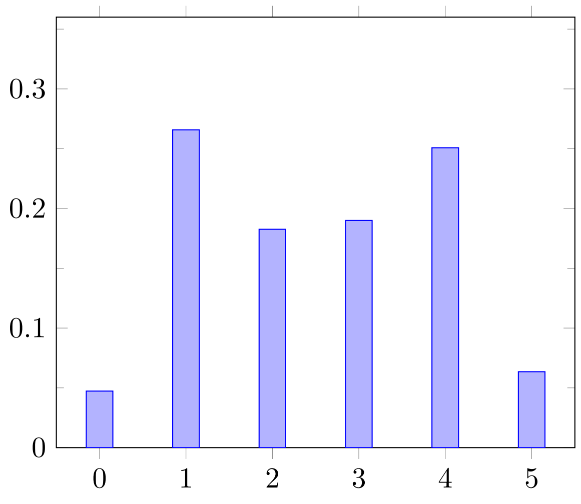
The distribution approximates the binomial distribution to some degree, but not particularly well. Some inaccuracies may be attributed to numerical errors or using samples from an unconverged Markov chain but given the fact, that a classical MLN will be able to learn such example with relative ease, it is not a very encouraging result.

As one can see from Appendix C, there are many distributions to which GD may converge. To guide the optimization a little, the imaginary parts of the tautology's weight were initialized to zero and the imaginary parts of the other formula's weight to the first factors of the DFT.[7] The real parts of the weights were initialized from $\mathcal{N}(0, \pi/2)$.

When learning with an arbitrary initialization, the learning procedure converged to even quite surprising results, such as the one shown in Figure 6.2.

[7]In the case of $\Psi = \{smokes(x)\}$ and $|\Delta| = 5$, those are $2\pi \cdot \frac{0}{6}, 2\pi \cdot \frac{1}{6}, \ldots, 2\pi \cdot \frac{d-1}{6}$.

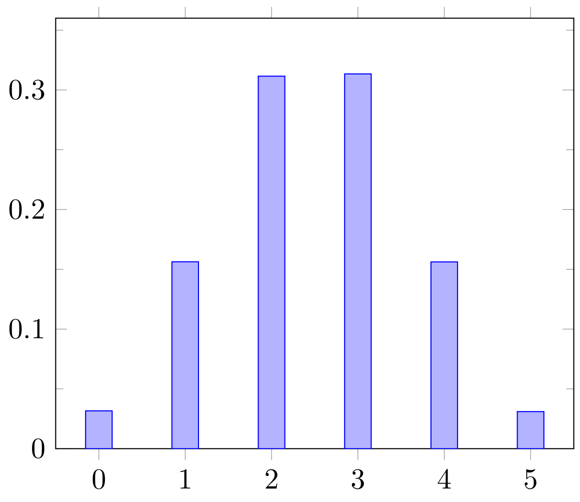
Figure 6.3 shows the resulting count distribution when learning using DFT as proposed in Section 5.2. Accounting for numerical errors, the distribution is exactly binomial. However, the learning algorithm produced 12-dimensional weights. That seems highly excessive in contrast to a single zero-valued weight in the context of classical MLNs. Nevertheless, it is another demonstration of how many solutions there are to this problem.



$\mathbb{KL} = 0.13137$

$smokes(x)$	$\top$
$-0.000 + 8.821i$	$2.352 + 3.077i$
$-0.842 + 4.910i$	$-2.643 + 1.745i$
$-1.628 - 0.182i$	$1.062 - 0.369i$

Figure 6.2: GD-approximated binomial count distribution with $d = 3$



$\mathbb{KL} = 0.00000$

$smokes(x)$	$\top$
$0.000 + 0.000i$	$-1.673 + 0.000i$
$0.000 + 1.047i$	$-8.139 - 0.627i$
$0.000 + 2.094i$	$-7.219 - 0.450i$
$0.000 + 3.142i$	$-7.762 + 0.000i$
$0.000 + 4.189i$	$-7.219 + 0.450i$
$0.000 + 5.236i$	$-8.139 + 0.627i$
$0.000 + 0.000i$	$-1.673 + 6.283i$
$0.000 + 1.047i$	$-8.139 + 5.656i$
$0.000 + 2.094i$	$-7.219 + 5.833i$
$0.000 + 3.142i$	$-7.762 + 6.283i$
$0.000 + 4.189i$	$-7.219 + 0.450i$
$0.000 + 5.236i$	$-8.139 + 0.627i$

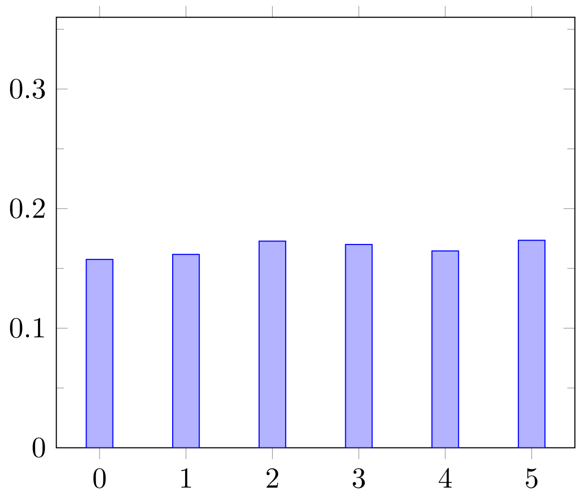
Figure 6.3: DFT-approximated binomial count distribution with $d = 12$

Example 6.2. Consider $|\Delta| = 5$ and $\mathcal{T}$ such that it specifies a uniform count distribution. One such example of $\mathcal{T}$ contains one possible world of each size.

It is already impossible to model this problem with a classical MLN (see Subsection 6.3.3).

Figure 6.4 shows the best approximation obtained with MLE. The approximation has $d = 7$. It is difficult to assess if that is the minimum required dimension for achieving such approximation, or the experiments with lower dimensions were just unlucky in finding the best possible result.

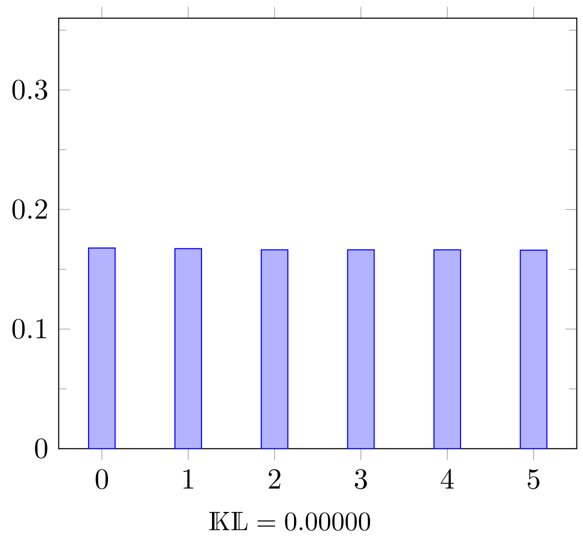
The approximation is, again, best when learning using DFT. The result in displayed in Figure 6.5. Nevertheless, DFT, again, learns seemingly large dimension $d = 12$.



$\mathbb{KL} = 0.000637$

$smokes(x)$	$\top$
$-2.345 + 0.000i$	$2.921 + 0.000i$
$0.771 + 36.717i$	$-0.015 + 11.561i$
$0.537 + 1.825i$	$0.360 + 0.048i$
$1.008 + 2.757i$	$-1.387 + 0.009i$
$0.532 + 3.600i$	$1.068 - 0.001i$
$1.103 + 4.339i$	$-1.508 - 0.039i$
$1.221 + 5.593i$	$-2.426 + 0.035i$

Figure 6.4: GD-approximated uniform count distribution with $d = 7$



$smokes(x)$	$\top$
$0.000 + 0.000i$	$-0.834 + 0.000i$
$0.000 + 1.047i$	$-1.345 + 0.520i$
$0.000 + 2.094i$	$-2.146 + 1.034i$
$0.000 + 3.142i$	$-6.464 + 0.000i$
$0.000 + 4.189i$	$-2.146 - 1.034i$
$0.000 + 5.236i$	$-1.345 - 0.520i$
$0.000 + 0.000i$	$-0.834 + 6.283i$
$0.000 + 1.047i$	$-1.345 + 0.520i$
$0.000 + 2.094i$	$-2.146 + 1.034i$
$0.000 + 3.142i$	$-6.464 + 6.283i$
$0.000 + 4.189i$	$-2.146 + 5.249i$
$0.000 + 5.236i$	$-1.345 + 5.764i$

Figure 6.5: DFT-approximated uniform count distribution with $d = 12$

6.3.2 Relational Case

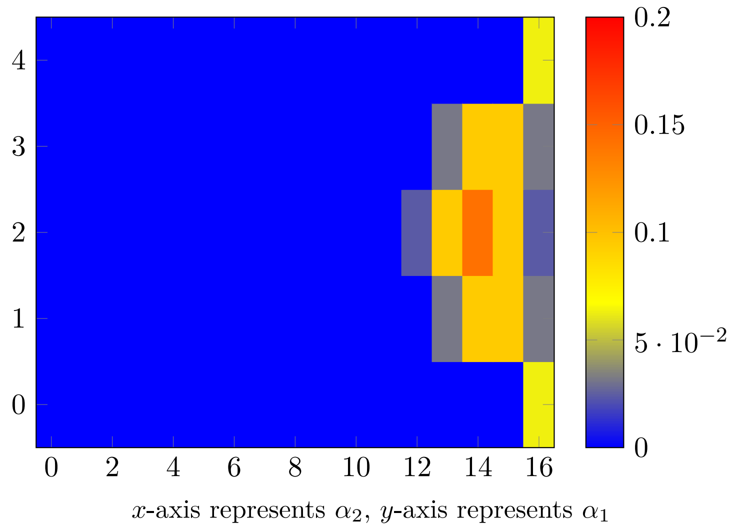
Let us now consider a relational case with formulas

$$\alpha_1 = smokes(x),$$

$$\alpha_2 = (smokes(x) \wedge friends(x, y)) \implies smokes(x).$$

With two formulas, the count distribution can still be visualized using a heatmap.

Example 6.3. Consider an MLN $\Phi = \{(\alpha_1, 0), (\alpha_2, 0)\}$ along with a domain Δ such that $|\Delta| = 4$. Figure 6.6 shows the induced count distribution.



α_1	α_2
0	0

Figure 6.6: Count distribution defined by an MLN

In this instance, the training data $\mathcal{T}$ was obtained by sampling from the target MLN. Experiments were conducted for $|\mathcal{T}| = 100$ and $|\mathcal{T}| = 1,000$.

Figures 6.7 and 6.8 show the count distributions estimated from $\mathcal{T}$ for both sizes. The smaller training set was apparently not rich enough to capture the true count distribution as accurately as the larger one did. Nevertheless, experiments can be performed for both cases, assessing divergence of the results from the count distribution estimates rather than from the true count distribution shown in Figure 6.6. Therefore, $\mathbb{KL}$ divergence presented in the following figures is always measured from the estimated count distribution for the particular size of the training set.

GD converged very slowly for the cases with $|\mathcal{T}| = 100$. For any of the trials, it did not even fully converge. The algorithm remained oscillating between points with $|\nabla| \in [1; 2]$. Decreasing the step size did not help. One of the count distributions from the oscillating phase is shown in Figure 6.9. As one can see, it is nowhere near the target.

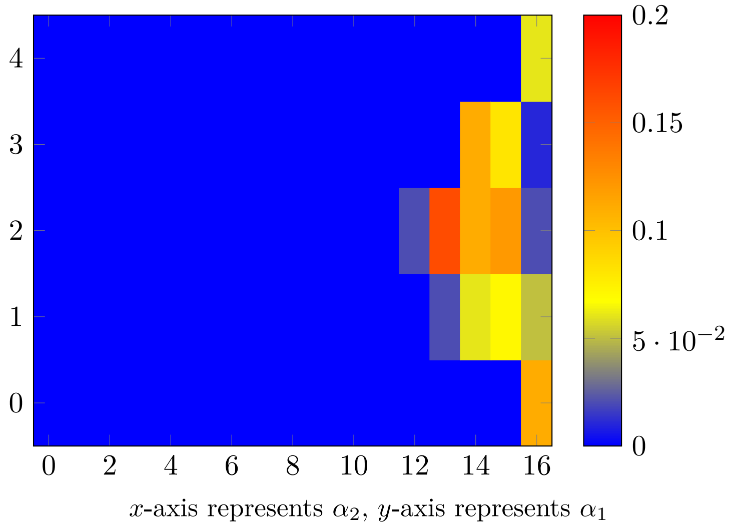


Figure 6.7: Estimate of the distribution from Figure 6.6 from 100 samples

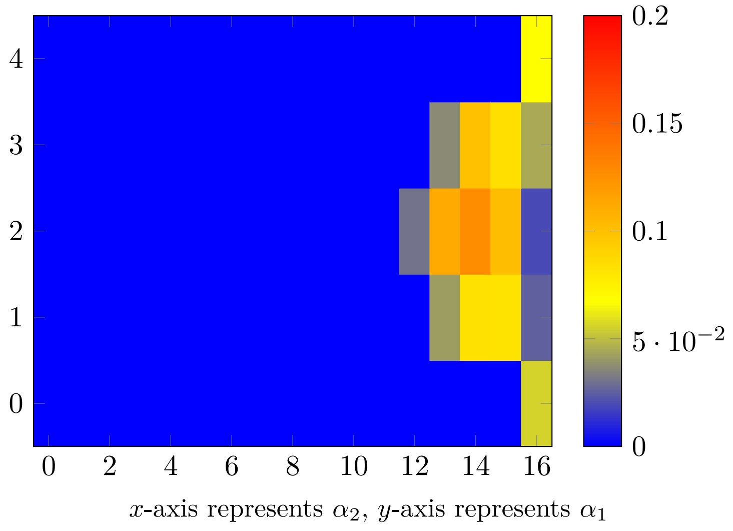
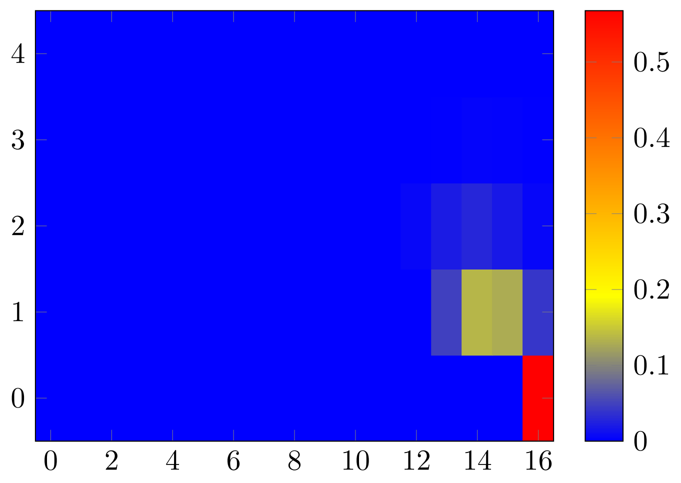


Figure 6.8: Estimate of the distribution from Figure 6.6 from $1,000$ samples

The optimization performed slightly better with the increased number of training samples. The convergence speed remained low, but, at least one, somewhat acceptable, set of weights was found. The result can be seen in Figure 6.10.

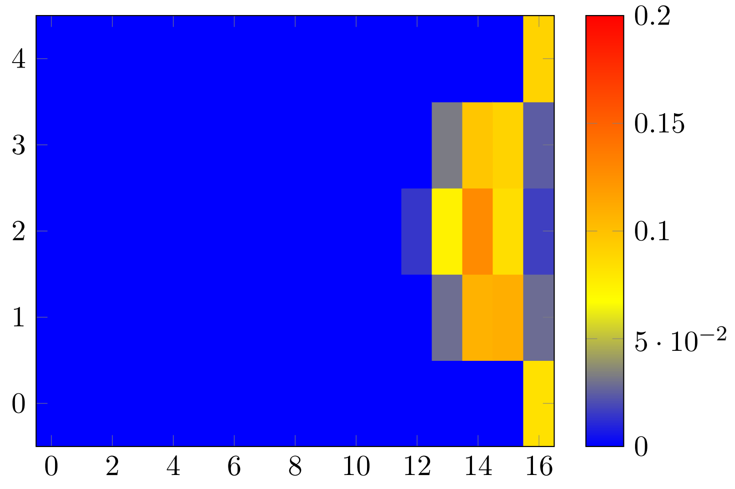
As in the previous experiments, the best approximation was obtained using the DFT. However, the dimensionality learned was 170. The result for $|\mathcal{T}| = 1,000$ can be seen in Figure 6.11. The learned weights are omitted.



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 100$
$\mathbb{KL} = 1.53673$

α_1	α_2	$\top$
$-1.961 + 0.000i$	$-0.058 + 0.000i$	$1.923 + 0.000i$
$-2.384 + 1.257i$	$-1.134 + 0.370i$	$-4.236 - 0.000i$

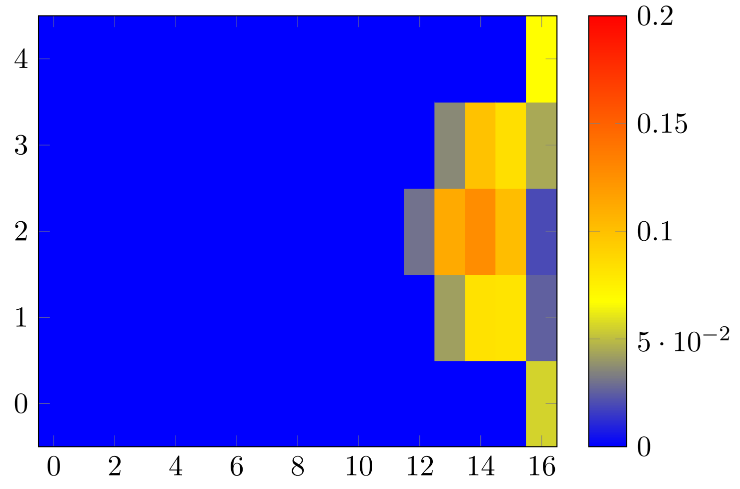
Figure 6.9: Unconverged GD approximation of Figure 6.7 with $d = 2$



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.12460$

α_1	α_2	$\top$
$1.104 + 0.000i$	$-0.163 + 0.000i$	$-1.717 + 0.000i$
$-0.399 + 0.012i$	$0.167 + 0.452i$	$-1.947 + 0.006i$

Figure 6.10: GD-approximated distribution from Figure 6.8 with $d = 2$



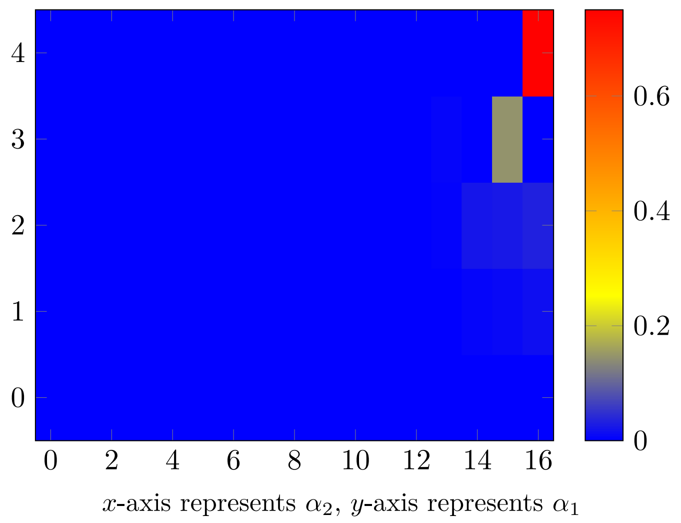
x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.00000$

Figure 6.11: DFT-approximated distribution from Figure 6.8 with $d = 170$

Example 6.4. For the final experiment, let us select a particular $\mathbb{C}$-MLN and try to learn the induced count distribution.

Assume $\Phi = \{(\alpha_1, \ln(3) + \frac{\pi}{3}i), (\alpha_2, \ln(3) + \frac{\pi}{2}i), (\top, \frac{\pi}{2}i)\}$ along with four-element domain. The situation is visualized in Figure 6.12.

Let us obtain training data by drawing $1,000$ samples from the specified $\mathbb{C}$-MLN. Estimated count distribution is shown in Figure 6.13.



α_1	α_2	$\top$
$\ln(3) + \frac{\pi}{3}i$	$\ln(3) + \frac{\pi}{2}i$	$\frac{\pi}{2}i$

Figure 6.12: Count distribution defined by a $\mathbb{C}$–MLN

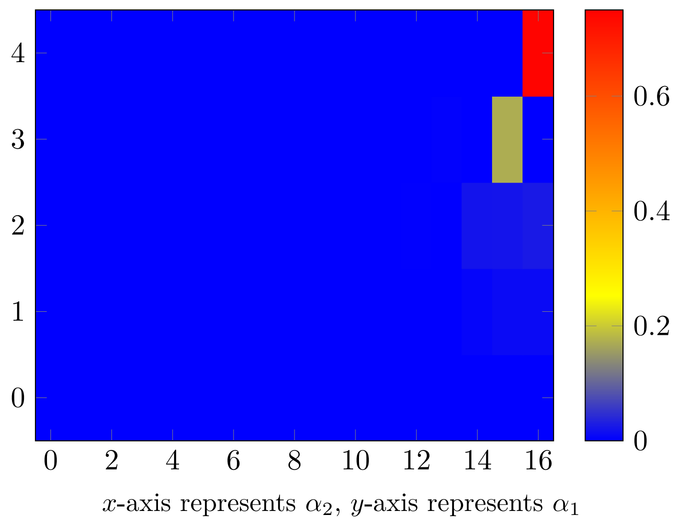
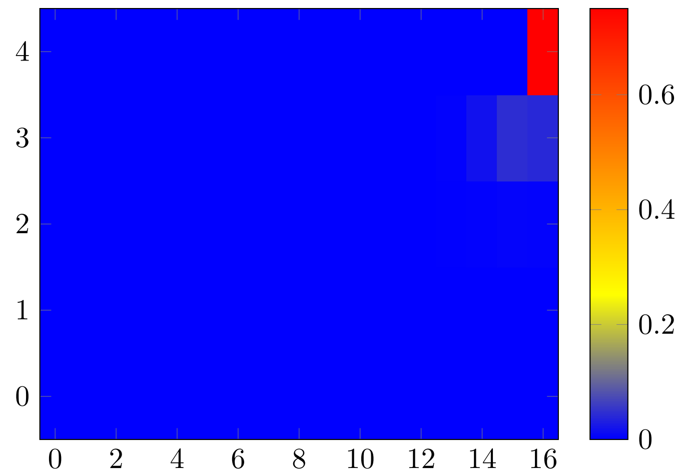


Figure 6.13: Estimate of the distribution from Figure 6.12 from $1,000$ samples

In spite of the count distribution's sparsity, gradient-based learning struggled with approximating it. The best obtained result can be found in Figure 6.14. The approximation is not very satisfying.

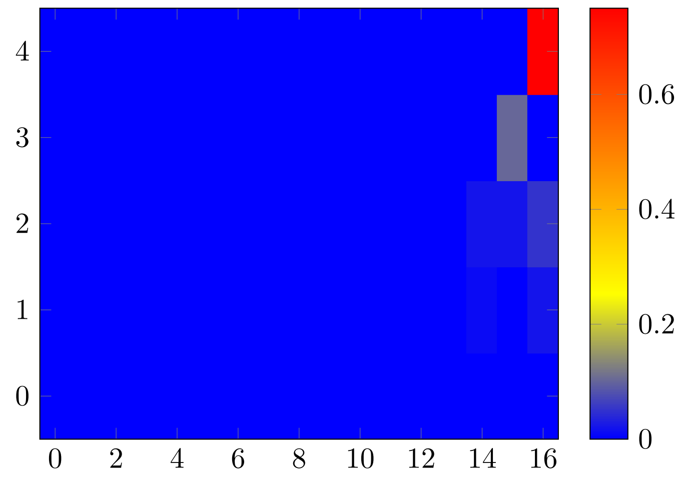
As one may already expect, DFT learning was, again, much more precise. Figure 6.15 displays the learned distribution. Learned weights, however, were, once more, 170-dimensional and they are omitted as in the previous case.



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.30938$

α_1	α_2	$\top$
$2.428 + 0.000i$	$0.981 + 0.000i$	$1.190 + 0.000i$
$0.681 + 1.257i$	$-0.556 - 0.000i$	$-2.206 - 0.000i$
$-0.348 + 0.000i$	$-0.481 + 0.370i$	$-0.494 + 0.000i$
$-1.784 + 1.257i$	$-1.807 + 0.370i$	$-0.837 - 0.000i$

Figure 6.14: GD-approximated count distribution from Figure 6.13 with $d = 4$



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.00000$

Figure 6.15: DFT-approximated count distribution from Figure 6.13 with $d = 170$

6.3.3 Classical vs. Complex MLNs

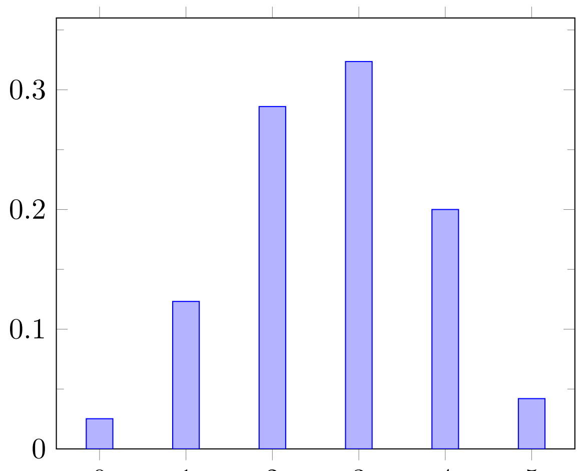
Examples 6.1 and 6.3 aim to approximate count distributions that can be modelled by classical MLNs. In the latter case, an MLN was actually the input that defined the count distribution. However, the other examples present count distributions that cannot be captured by the MLN model.

Uniform count distribution (Example 6.2) implies that the probability of the empty interpretation should equal the probability of the HB, i.e., the interpretation containing all atoms from the HB. Such equation has one solution:

$$\begin{gathered}\frac{1}{Z}\exp(w \cdot N(smokes(x), \emptyset)) = \frac{1}{Z}\exp(w \cdot N(smokes(x), HB)) \\ w \cdot 0 = w \cdot |HB| \\ w = 0\end{gathered} \tag{6.2}$$

However, $w = 0$ leads to a binomial count distribution (see Example 6.1).

Figures 6.16 and 6.17 show attempts of an MLN to learn the uniform count distribution, without and with the tautology formula $\top$ in the KB. The approximations approach binomial distribution much more closely than the uniform distribution. Moreover, they are the same. That is due to the general role of $\top$ in the KB. The tautology formula serves as a bias term that allows to *shift* the model while maintaining its *shape*.

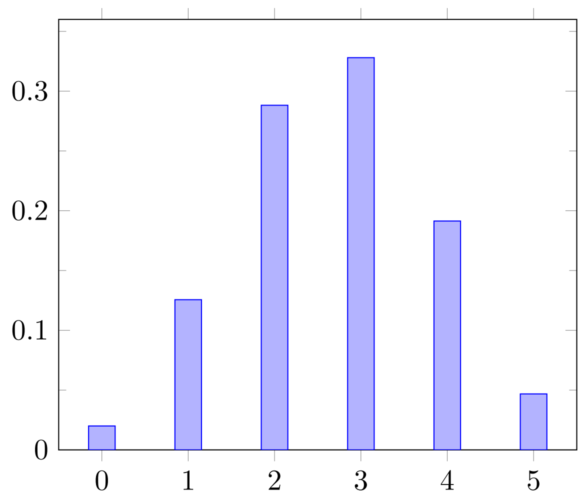


$\mathbb{KL} = 0.38504$

$smokes(x)$
0.146

Figure 6.16: MLN approximation of uniform count distribution

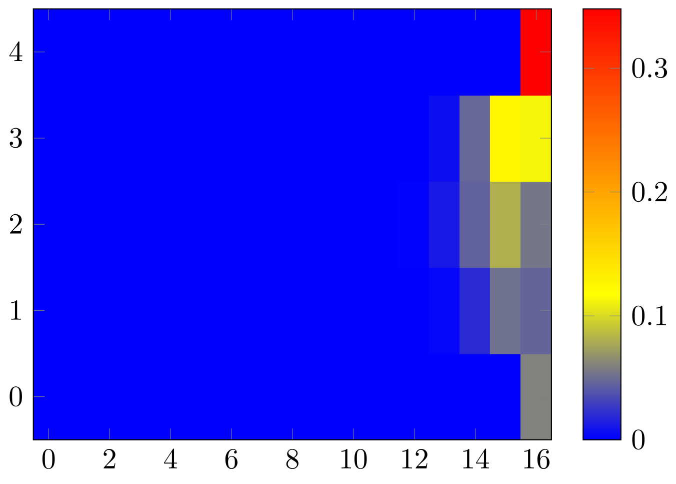
Figure 6.18 visualizes the MLN approximation of the distribution shown in Figure 6.13 in Example 6.4. Similarly to the uniform distribution above, the *shape* of the MLN approximation there approaches more the count distribution of a network with the same KB, yet zero-valued weights (i.e., network from Example 6.3).



$\mathbb{KL} = 0.38504$

$smokes(x)$	$\top$
0.135	−0.688

Figure 6.17: MLN approximation of uniform count distribution with tautology



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1000$
$\mathbb{KL} = 0.53860$

α_1	α_2	$\top$
0.446	0.990	−0.167

Figure 6.18: MLN approximation of Figure 6.13

Chapter 7
Conclusion

Complex Markov logic networks are a model from the area of statistical relational learning, that allows expressing compactly an arbitrary count distribution. They are an extension of Markov logic networks, a simple, well established and extensively studied method for combining first-order logic and probability theory.

Compared to MLNs, $\mathbb{C}$-MLNs lack underlying graphical structure, making it impossible to reuse graph-based inference algorithms developed for MLNs. The sampling-based inference is still possible. In Chapter 4, a sampling algorithm for $\mathbb{C}$-MLNs based on the popular Gibbs sampler was developed. The algorithm was implemented and used as a subprocedure while computing the gradient. It performed reasonably well on small examples.

The original $\mathbb{C}$-MLN definition [3] specified the model as a mixture of complex exponential functions. Although that is similar to the log-linear model of MLNs, it introduces new challenges. One of the most notable being that learning can no longer be formulated as a problem of unconstrained optimization in the real domain. The weights may be complex vectors rather than real number as it is in the case of classical MLNs. Furthermore, Equation 2.11 permits only a small subset of all possible weights so that a proper probability distribution is produced. However, the definition can be modified (see Section 3.4) so that its expressivity is not lost and it becomes a real function differentiable on the majority of its domain.

With the new definition, the gradient can be computed and used for learning. This work used the simple technique of gradient descent. However, the learning did not perform particularly well.

The negative log-likelihood is not a convex function. As one can observe from Chapter 6 and Appendix C, there are many critical points that the GD may converge to. The optimization-based learning algorithm was executed several times with different initialization weights. The results were then compared to the target count distribution using Kullback-Leibler divergence to decide on their quality. The experiments were conducted only on small examples so such comparisons would be possible. That, however, is not a practical approach, since for real-life datasets, the target distribution is unknown. Moreover, the best-found weights were often of a higher dimension than was clearly necessary.

Apart from the non-convexity, some issues with convergence were also encountered in Example 6.3, in spite the fact, that the count distribution there could be represented even by classical MLNs.

Section 5.2 approached the learning problem differently. Using discrete Fourier transform, an arbitrary count distribution was directly encoded as a $\mathbb{C}$-MLN (the original one). However, that approach has two drawbacks. Firstly, it learns not only weights but also their dimensionality, which may grow extremely large even for simple examples. Secondly, a sufficiently large training data set is required that allows an accurate estimation of the count distribution.

7.1 Future Work

There are many ways in which this thesis can be expanded.

Firstly, GD could be replaced by a more sophisticated optimization technique, perhaps even higher-order one, that might be better at learning mixture models. One possibility could be the L-BFGS algorithm, which is a popular choice for parameter learning [18].

However, learning as is developed in Section 5.1 might be overall too inspired by learning in MLNs. Even the modifications proposed in Section 3.3 to enable computation of gradient may not be the best approach. Learning directly the original $\mathbb{C}$-MLN model (which was also derived in Section 5.2 using DFT) could be a better strategy. The learning could be possibly done by utilizing $\mathbb{CR}$-Calculus [10] or algorithms from constrained optimization.

Besides refining parameter learning, examining the possibilities of structure learning is another possible extension.

Going back to Chapter 4, this work assumed the formulas in the knowledge base to be clauses. The question remains if assigning weights to clauses limits expressivity when compared to assigning weights to general FOL formulas.

Regarding the $\mathbb{C}$-MLN model itself, the matter of singularities discussed briefly in Chapter 3 should be inspected further. They were not encountered during experiments presented in Chapter 6; that, however, does not mean that they can't become a problem in other cases.

Apart from its singularities, the model appears to be overparameterized. Even simple experiments showed that there exist distinct sets of weights that approximate particular count distribution with similar accuracy. Furthermore, individual dimensions are mutually independent, meaning that coordinates may be shuffled, yet the distribution will remain the same. The possibility of some $\mathbb{C}$-MLN regularization, which could also be beneficial for the DFT-based learning, should thus also be explored.

Bibliography

[1] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer New York, 1996.

[2] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 02 2006.

[3] Ondrej Kuzelka. Complex markov logic networks: Expressivity and liftability. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 729–738, Virtual, 03–06 Aug 2020. PMLR.

[4] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[5] Alan E. Gelfand and Adrian F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990.

[6] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[7] H. A Priestley. *Introduction to complex analysis*. Clarendon, Oxford, 1985.

[8] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. 01 2009.

[9] David J. Griffiths. *Introduction to Quantum Mechanics (2nd Edition)*. Pearson Prentice Hall, 2nd edition, April 2004.

[10] Ken Kreutz-Delgado. The complex gradient operator and the cr-calculus, 2009.

[11] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996.

[12] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. 12 2013.

[13] O. Schulte, H. Khosravi, A. Kirkpatrick, Tianxiang Gao, and Yuke Zhu. Modelling relational statistics with bayes nets (poster presentation srl workshop). 2012.

[14] A. Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *C. R. Acad. Sci Paris*, 25(1):536–538, 1847.

[15] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[16] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[17] Jiahao Chen and Jarrett Revels. Robust benchmarking in noisy environments. *arXiv e-prints*, Aug 2016.

[18] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING-02, page 49–55, USA, 2002. Association for Computational Linguistics.

Appendix A

Weights' Transformation

Proposition A.1. *Let Φ be a $\mathbb{C}$-MLN with formulas $\Psi = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$, where $\alpha_1 = \top$, and with weights $\mathbf{w}_i$ for each α_i. Let $\omega \in \Omega$ be a possible world.*

Multiplying

$$\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right)$$

by the negative imaginary unit is equivalent to subtracting $\frac{\pi}{2}i$ from the tautology weight along each dimension.

Proof. Denote $z_k(\omega) = \sum_{i=1}^{m}[\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)$, and let $x_k(\omega)$ and $y_k(\omega)$ be its real and imaginary part, respectively. For brevity, let us also drop the argument and assume the numbers z_k, x_k, y_k for any $\omega \in \Omega$.

Then

$$\sum_{k=1}^{d} \exp\left(\sum_{i=1}^{m} [\mathbf{w}_i]_k \cdot N(\alpha_i, \omega)\right) = \sum_{k=1}^{d} \exp(z_k).$$

Multiplying the sum by $-i$ produces:

$$\begin{aligned} -i\sum_{k=1}^{d} \exp(z_k) &= -i\sum_{k=1}^{d} \exp(x_k)(\cos(y_k) + i\sin(y_k)) \\ &= \sum_{k=1}^{d} \exp(x_k)(\sin(y_k) - i\cos(y_k)) \end{aligned} \tag{A.1}$$

Remember elementary properties of the sine and cosine functions:

$$\begin{aligned} \cos(x) &= \sin\left(x + \frac{\pi}{2}\right) \\ \sin(-x) &= -\sin(x) \\ \cos(-x) &= \cos(x) \end{aligned} \tag{A.2}$$

Using Equation A.2:

$$
\begin{aligned}
-i\sum_{k=1}^{d}\exp\left(z_k\right) &= \sum_{k=1}^{d}\exp(x_k)(\sin(y_k) - i\cos(y_k)) \\
&= \sum_{k=1}^{d}\exp(x_k)\left(\cos\left(y_k - \frac{\pi}{2}\right) - i\cos\left(-y_k\right)\right) \\
&= \sum_{k=1}^{d}\exp(x_k)\left(\cos\left(y_k - \frac{\pi}{2}\right) - i\sin\left(-y_k + \frac{\pi}{2}\right)\right) \\
&= \sum_{k=1}^{d}\exp(x_k)\left(\cos\left(y_k - \frac{\pi}{2}\right) + i\sin\left(y_k - \frac{\pi}{2}\right)\right) \\
&= \sum_{k=1}^{d}\exp(z_k - i\frac{\pi}{2})
\end{aligned}
\tag{A.3}
$$

Returning to the weights:

$$
\begin{aligned}
z_k - i\frac{\pi}{2} &= x_k + i\left(y_k - \frac{\pi}{2}\right) \\
&= x_k + i\left(-\frac{\pi}{2} + \sum_{i=1}^{m}[\mathbf{y}_i]_k \cdot N\left(\alpha_i, \omega\right)\right)
\end{aligned}
\tag{A.4}
$$

Since $\alpha_1 = \top$ and $N(\top, \omega) = 1$ for any $\omega \in \Omega$, Equation A.4 can be rewritten as

$$
z_k - i\frac{\pi}{2} = x_k + i\left(([\mathbf{y}_1]_k - \frac{\pi}{2}) \cdot N(\alpha_1, \omega) + \sum_{i=2}^{m}[\mathbf{y}_i]_k \cdot N\left(\alpha_i, \omega\right)\right). \tag{A.5}
$$

Computing $-i\sum_{k=1}^{d}\exp\left(z_k\right)$ is thus equivalent to subtracting $\frac{\pi}{2}i$ from each dimension of the tautology weight. $\square$

Appendix B

Answering Query with SQL

Example B.1. Assume a conjunctive query β and a database ω.

$$\beta = smokes(x) \wedge friends(x, y)$$
$$\omega = \{smokes(A), smokes(B), smokes(E), friends(A, B), friends(E, B)\}$$

First of, initialize a fresh database so that it reflects the possible world ω:

```
CREATE TABLE smokes(
          id VARCHAR(1) NOT NULL,
          PRIMARY KEY (ID)
);
INSERT INTO smokes (ID) VALUES ('A');
INSERT INTO smokes (ID) VALUES ('B');
INSERT INTO smokes (ID) VALUES ('E');

CREATE TABLE friends(
          id_1 VARCHAR(1) NOT NULL,
          id_2 VARCHAR(1) NOT NULL,
          PRIMARY KEY (id_1, id_2)
);
INSERT INTO friends (id_1, id_2) VALUES ('A', 'B');
INSERT INTO friends (id_1, id_2) VALUES ('E', 'B');
```

Next, translate the formula β into an SQL select query:

```
SELECT smokes.id AS X, friends.id_2 AS Y
FROM smokes JOIN friends ON smokes.id = friends.id_1;
```

X	Y
A	B
E	B

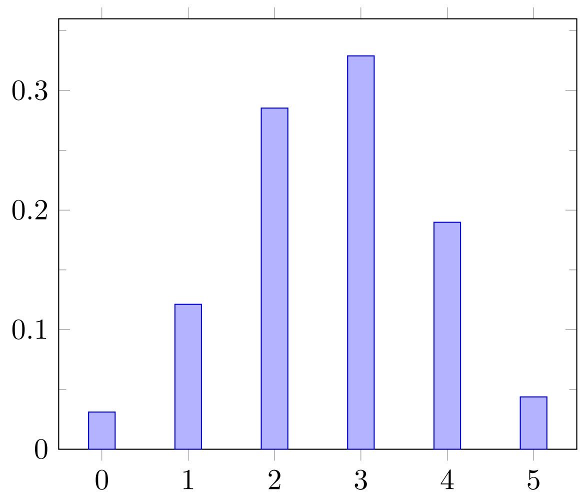
Table B.1: Result of an SQL select query

Hence $N(\beta, \omega) = 2$, since there are 2 entries in the resulting table.

Appendix C

Additional Learning Results

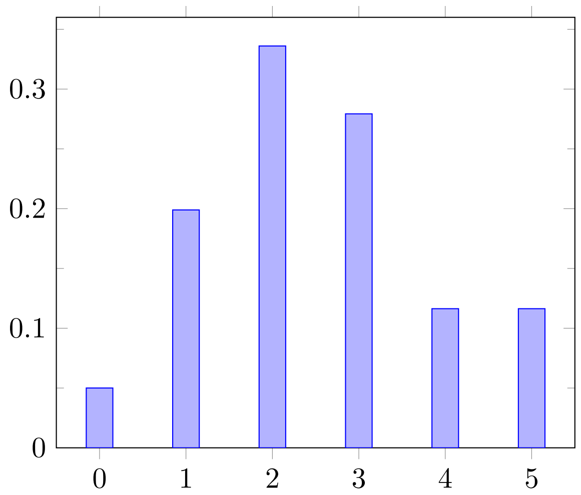
C.1 Example 6.1



$\mathbb{KL} = 0.01147$

$smokes(x)$	$\top$
$-3.015 + 0.000i$	$-0.946 + 0.000i$
$0.143 + 3.142i$	$-0.151 - 0.000i$

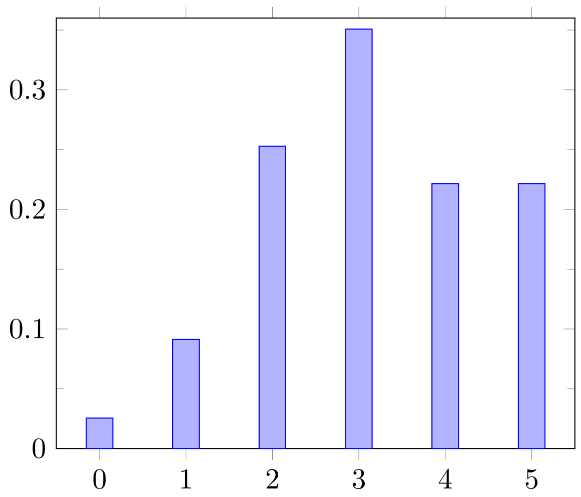
Figure C.1: GD-approximated binomial count distribution with $d = 2$



$\mathbb{KL} = 0.02100$

$smokes(x)$	$\top$
$-2.046 - 3.726i$	$-2.266 + 1.142i$
$-0.186 + 0.015i$	$0.208 - 0.232i$

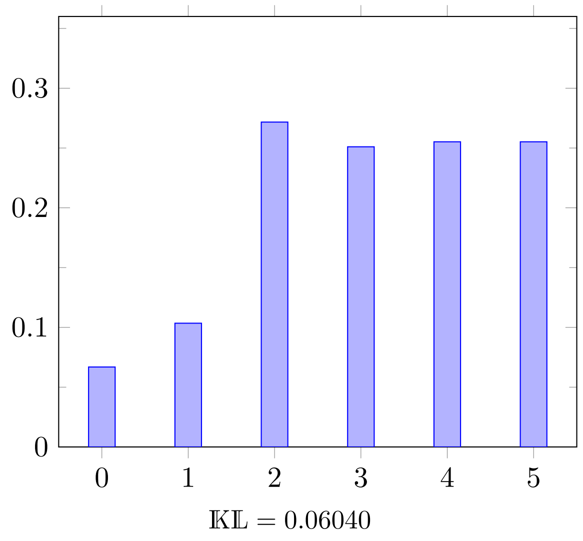
Figure C.2: GD-approximated binomial count distribution with $d = 2$



$\mathbb{KL} = 0.04659$

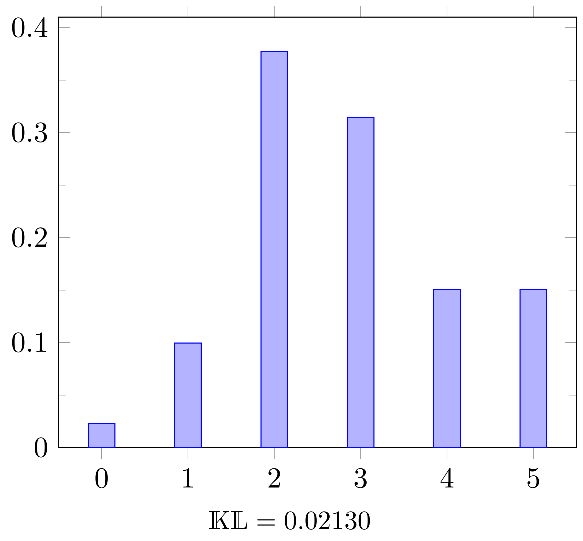
$smokes(x)$	$\top$
$0.267 + 0.000i$	$-0.342 + 0.000i$
$-1.677 + 2.091i$	$-1.202 - 0.004i$
$-0.473 + 4.189i$	$-1.756 + 0.005i$

Figure C.3: GD-approximated binomial count distribution with $d = 3$



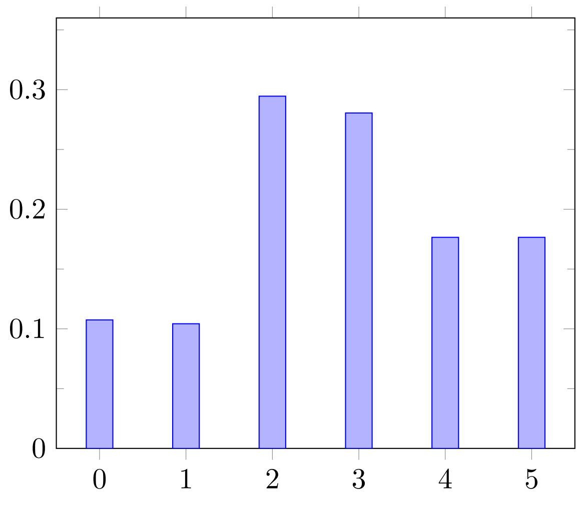
$smokes(x)$	$\top$
$-0.618 + 1.425i$	$-2.038 - 0.566i$
$0.003 + 3.722i$	$-0.247 + 3.347i$
$-0.355 - 0.755i$	$-0.783 - 2.194i$

Figure C.4: GD-approximated binomial count distribution with $d = 3$



$smokes(x)$	$\top$
$-0.628 + 0.000i$	$-1.893 + 0.000i$
$0.508 - 1.334i$	$2.071 - 0.786i$
$0.978 + 0.875i$	$-0.165 - 0.173i$
$0.648 + 2.693i$	$0.727 + 0.039i$
$-1.112 + 3.590i$	$-2.698 - 0.000i$
$-0.409 + 4.485i$	$-1.246 - 0.003i$
$-0.490 + 5.379i$	$-0.268 - 0.007i$

Figure C.5: GD-approximated binomial count distribution with $d = 7$

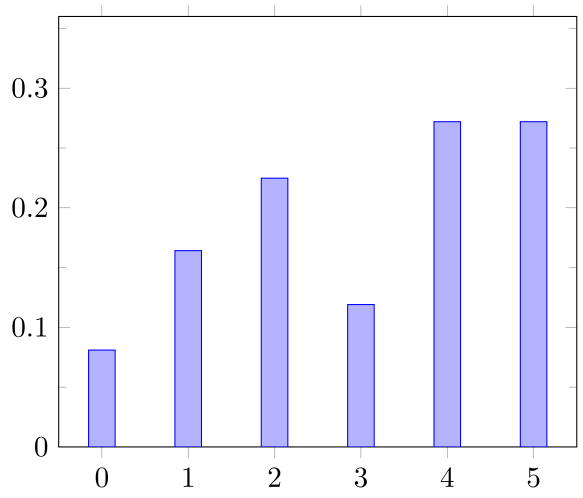


$\mathbb{KL} = 0.05278$

$smokes(x)$	$\top$
$-0.048 + 0.000i$	$1.145 + 0.000i$
$-0.444 + 0.890i$	$-0.254 + 0.009i$
$0.667 + 2.504i$	$-0.772 + 0.266i$
$-0.784 + 2.869i$	$2.567 + 0.129i$
$1.017 + 4.146i$	$-1.973 + 0.067i$
$-0.402 + 4.414i$	$0.157 - 0.063i$
$-0.674 + 5.386i$	$-0.609 - 0.007i$

Figure C.6: GD-approximated count binomial distribution with $d = 7$

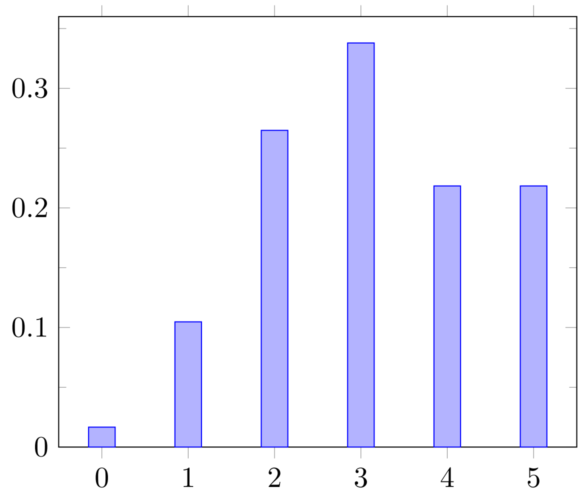
C.2 Example 6.2



$\mathbb{KL} = 0.07758$

$smokes(x)$	$\top$
$1.005 + 0.000i$	$-1.940 + 0.000i$
$-0.818 + 3.142i$	$2.526 - 0.000i$

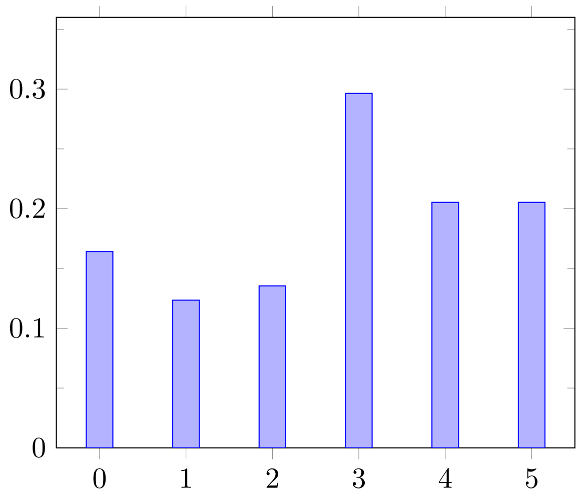
Figure C.7: GD-approximated uniform count distribution with $d = 2$



$\mathbb{KL} = 0.39869$

$smokes(x)$	$\top$
$0.634 + 0.000i$	$-2.266 + 0.000i$
$0.216 + 0.000i$	$1.440 + 0.000i$

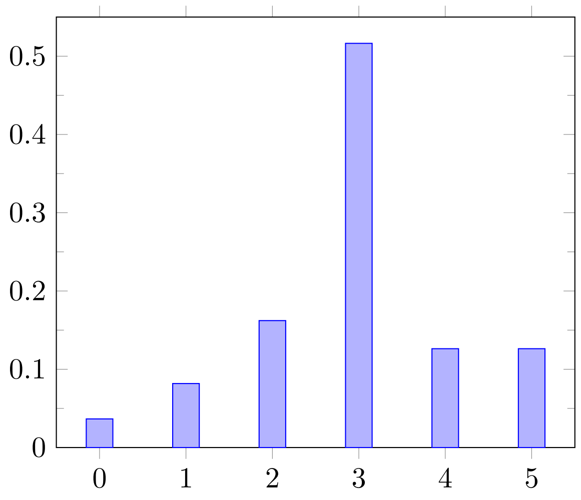
Figure C.8: GD-approximated uniform count distribution with $d = 2$



$\mathbb{KL} = 0.08902$

$smokes(x)$	$\top$
$0.552 + 0.000i$	$-0.976 + 0.000i$
$-0.113 + 2.156i$	$-1.223 + 0.029i$
$-1.352 + 3.990i$	$2.518 - 0.146i$

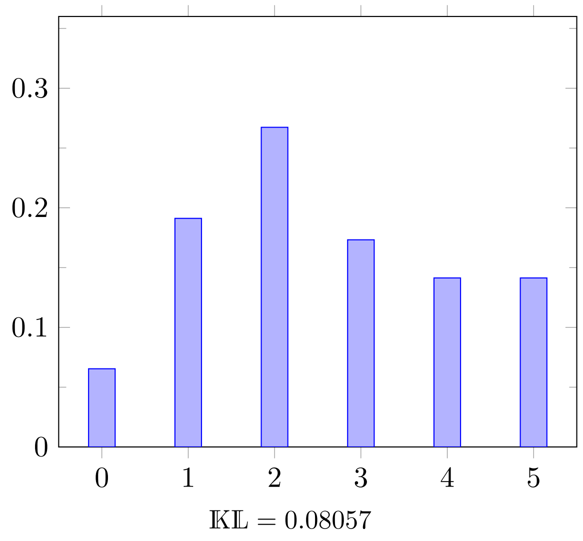
Figure C.9: GD-approximated uniform count distribution with $d = 3$



$\mathbb{KL} = 0.36295$

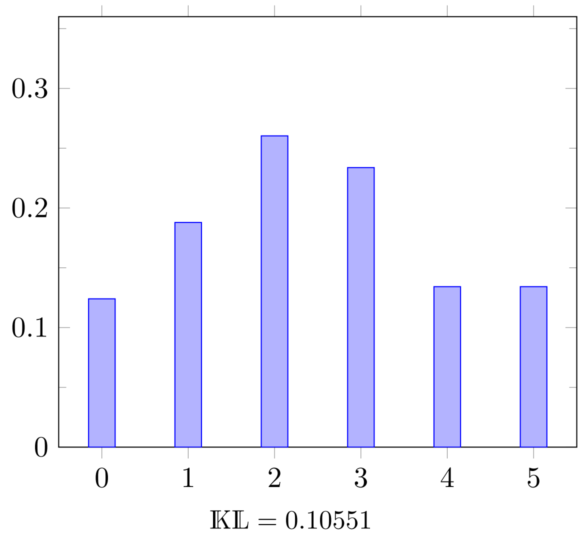
$smokes(x)$	$\top$
$-0.645 + 0.000i$	$1.129 + 0.000i$
$-0.262 + 2.209i$	$2.404 + 0.100i$
$0.445 + 4.268i$	$1.846 - 0.089i$

Figure C.10: GD-approximated uniform count distribution with $d = 3$



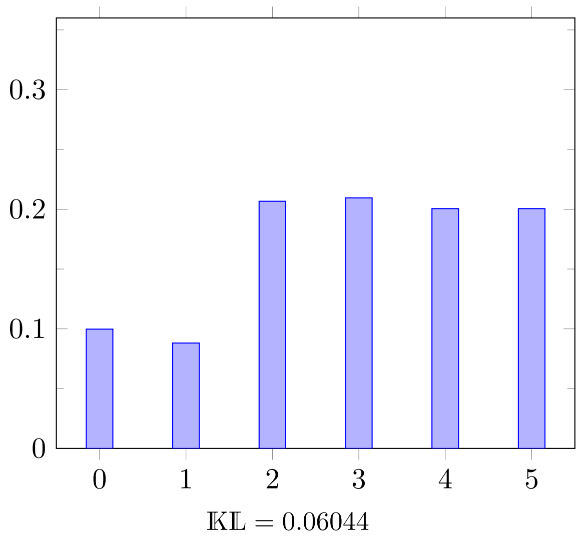
$smokes(x)$	$\top$
$-0.152 + 0.000i$	$2.657 + 0.000i$
$-0.998 + 1.258i$	$0.287 + 0.000i$
$1.752 + 2.749i$	$-3.709 + 0.070i$
$-0.328 + 3.770i$	$-2.718 + 0.000i$
$-0.630 + 5.015i$	$1.964 - 0.005i$

Figure C.11: GD-approximated uniform count distribution with $d = 5$



$smokes(x)$	$\top$
$-1.105 + 0.000i$	$-0.571 + 0.000i$
$0.055 + 1.384i$	$1.023 + 0.031i$
$-0.417 + 2.580i$	$1.963 + 0.034i$
$-0.224 + 3.742i$	$0.526 - 0.013i$
$0.103 + 4.927i$	$0.505 - 0.023i$

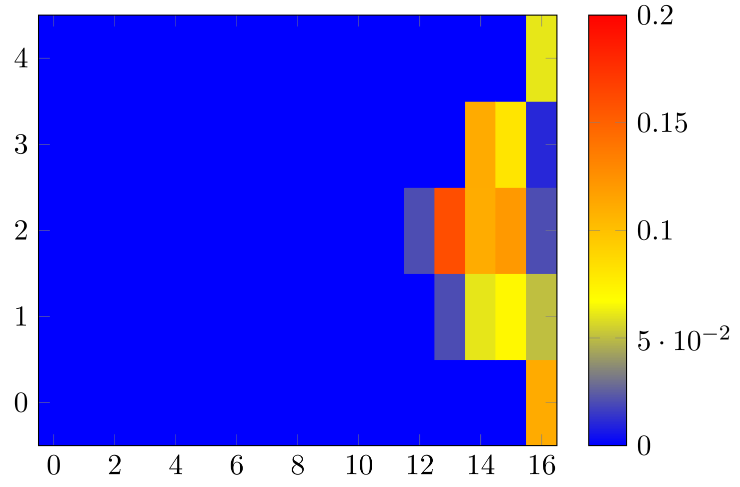
Figure C.12: GD-approximated uniform count distribution with $d = 5$



$smokes(x)$	$\top$
$-2.340 + 0.000i$	$2.909 + 0.000i$
$0.833 + 36.783i$	$0.014 + 11.582i$
$0.560 + 1.784i$	$0.370 + 0.023i$
$0.999 + 2.771i$	$-1.394 + 0.009i$
$0.482 + 3.625i$	$1.031 + 0.025i$
$1.131 + 4.361i$	$-1.502 - 0.028i$
$1.264 + 5.574i$	$-2.416 + 0.027i$

Figure C.13: GD-approximated uniform count distribution with $d = 7$

C.3 Example 6.3



x-axis represents α_2, y-axis represents α_1

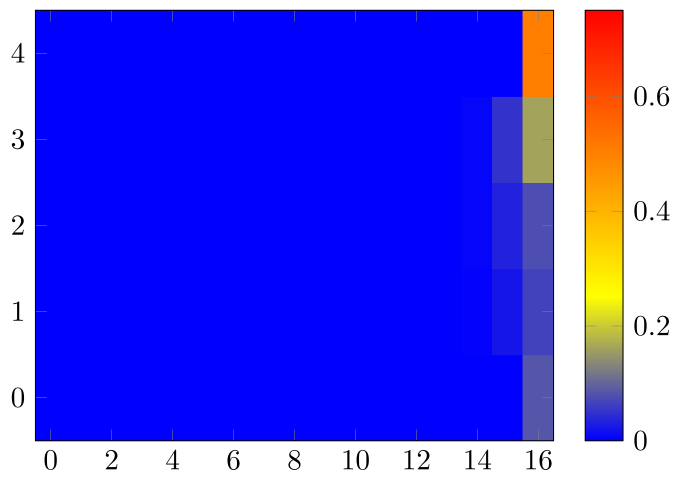
$|\mathcal{T}| = 100$

$\mathbb{KL} = 0.00000$

weights are omitted

Figure C.14: DFT-approximated distribution from Figure 6.7 with $d = 170$

C.4 Example 6.4



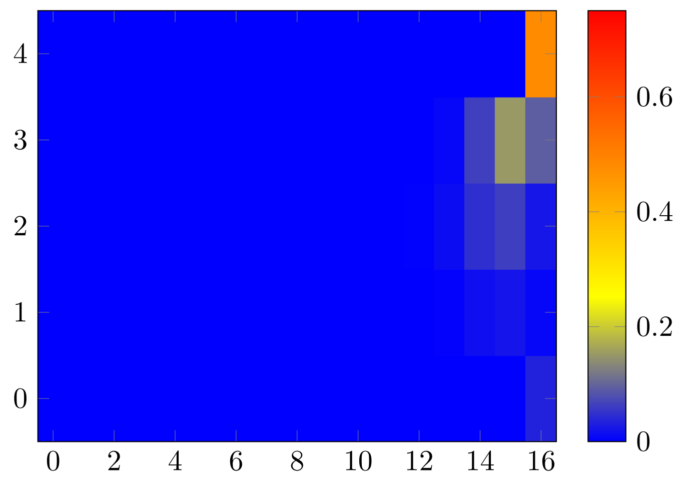
x-axis represents α_2, y-axis represents α_1

$|\mathcal{T}| = 1,000$

$\mathbb{KL} = 0.46821$

α_1	α_2	$\top$
$0.455 + 0.000i$	$2.250 + 0.000i$	$0.251 + 0.000i$
$-2.492 + 1.257i$	$2.111 + 0.002i$	$-1.282 + 0.000i$

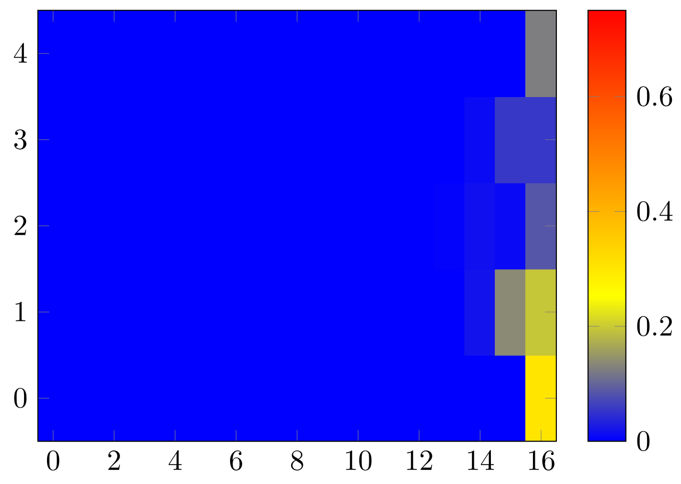
Figure C.15: GD-approximated distribution from Figure 6.13 with $d = 2$



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.36975$

α_1	α_2	$\top$
$-1.480 + 0.000i$	$-3.144 + 0.000i$	$-1.184 + 0.000i$
$-2.446 + 1.257i$	$0.953 - 0.001i$	$-0.161 - 0.000i$
$0.847 + 0.397i$	$0.718 + 2.636i$	$2.831 + 0.154i$
$-1.871 + 1.257i$	$-1.045 + 0.370i$	$0.850 - 0.000i$

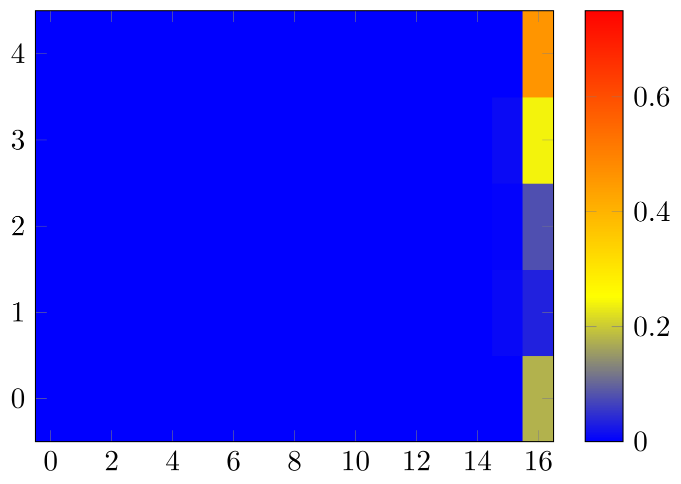
Figure C.16: GD-approximated distribution from Figure 6.13 with $d = 4$



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 1.4530$

α_1	α_2	$\top$
$-2.278 + 0.000i$	$-2.580 + 0.000i$	$2.079 + 0.000i$
$1.225 + 1.257i$	$-0.421 - 0.000i$	$1.657 - 0.000i$
$-0.940 - 0.000i$	$0.888 + 0.370i$	$-2.368 - 0.000i$
$-0.428 + 1.766i$	$1.659 + 0.712i$	$0.373 + 0.021i$

Figure C.17: GD-approximated distribution from Figure 6.13 with $d = 4$



x-axis represents α_2, y-axis represents α_1
$|\mathcal{T}| = 1,000$
$\mathbb{KL} = 0.77133$

α_1	α_2	$\top$
$0.452 + 0.000i$	$-0.387 + 0.000i$	$1.940 + 0.000i$
$-0.059 + 1.257i$	$0.308 - 0.000i$	$-0.579 - 0.000i$
$-2.205 - 0.000i$	$1.163 + 0.370i$	$-1.131 - 0.000i$
$0.690 + 1.257i$	$-1.083 + 0.370i$	$-0.228 - 0.000i$
$-0.721 + 1.257i$	$-2.766 + 0.739i$	$-1.085 - 0.000i$
$0.194 + 2.513i$	$2.247 + 0.370i$	$0.415 + 0.000i$
$0.301 + 0.868i$	$3.742 - 5.446i$	$1.792 - 0.382i$

Figure C.18: GD-approximated distribution from Figure 6.13 with $d = 7$

Appendix D

Enclosed Source Codes

```
ComplexMLN
├─ benchmark
├─ src
│  ├─ fol
│  ├─ groundings
│  ├─ inference
│  ├─ learning
│  ├─ network
│  ├─ parsing
│  └─ ComplexMLN.jl           - main package file
├─ test
├─ Manifest.toml              - locked versions of used dependencies
└─ Project.toml               - package (project) file
```

Appendix E

List of Abbreviations

$\mathbb{C}$-MLN	complex Markov logic network
CNF	conjunctive normal form
DFT	discrete Fourier transform
FOL	first-order logic
FT	Fourier transform
GD	gradient descent
HB	Herbrand base
IID	identically independently distributed
KB	knowledge base
MCMC	Markov chain Monte Carlo
MLE	maximum likelihood estimation
MLN	Markov logic network
MRF	Markov random field
WFOMC	weighted first order model counting

Appendix F

List of Algorithms

Appendix G

List of Figures

Appendix H

List of Tables