TECHNISCHE UNIVERSITÄT BERLIN

Department of Software Engineering and Theoretical Computer Science

ARTIFICIAL INTELLIGENCE GROUP

BACHELOR'S THESIS

Parameter Estimation in Potts Models as an Instance of Probabilistic Programming via Imperatively Defined Factor Graphs

Author:
Alec HANEFELD
Student ID: 330367

Supervisors:
Prof. Dr. Manfred OPPER
Dr. Andreas RUTTOR

July 11, 2014

Abstract

To gain insights about the physical structure of protein domains, a Potts model is trained to represent pairwise statistical dependencies between amino-acids. For the first time, this Direct Coupling Analysis is performed within the framework of Imperatively Defined Factor Graphs. This enables the usage of the probabilistic programming framework FACTORIE. After laying out the problems of a maximum-likelihood based approach, the training is performed using Contrastive Divergence. The Contrastive Divergence gradients are used with the AdaGrad Stochastic Gradient Descent algorithm. To induce sparsity on the pairwise parameter estimate l1-regularization performed via Regularized Dual Averaging. The results are compared to a naïve mutual information based analysis and evaluated against radio-crystallography data.
The code can be found under: https://github.com/ahane/proteinshake

Zusammenfassung

In dieser Arbeit wird ein Pottsmodell verwendet, um Vorhersagen über die dreidimensionale Struktur eines Proteins zu treffen. Hierfür wird das Model an mehrere verschiedene Proteinsequenzen (Multiple Sequence Alingments) aus verschiedenen Organismen angepasst, um die Norm der so gefundenen Parameter als Schätzer für die physische Distanz der einzelnen Aminosäuren im Protein zu nehmen. Dieses Verfahren ist unter dem Namen Direkte Kupplungs Analyse (Direct Coupling Analysis) bekannt. Entscheidend für die Eignung der Pottsmodellparameter zur Modellierung von Aminosäuren-Interaktionen, ist das Einführen eines Strafterms auf die ℓ_1-Norm der Parameter (ℓ_1-Regularization) während des Anpassungsprozesses. Traditionelle Methoden der Modellanpassung versagen bei dem verwendeten Pottsmodell, weswegen der kontrastive Divergenz (Contrastiv Divergence) Algorithmus verwendet wird. Die Ergebnisse werden mit einem naiven Transinformationsschätzer verglichen und durch Radiokristallographiemessdaten auf ihr Korrektheit überprüft.

Eidesstattliche Erklärung

Die selbstständige und eigenhändige Anfertigung versichere Ich an Eides Statt.

Berlin den

Contents

1 Introduction

1.1 Motivation

It has become commonplace to hear talk about the abundance of data that is being generated each day. Of course the essential question that follows from this statement is: how do we gain actionable knowledge from this flood of data? This, among other trends, explains the incredible mass of research and the breakthroughs that have taken place in fields like machine learning, artificial intelligence and pattern recognition in the recent years.

Many of the methods on which the complex probabilistic models that are in common use these days are built, have their origin in multiple fields. Fundamentals where first discussed in statistical physics, while some of the biggest breakthroughs have been contributed by coding- and information theory.[1] Recently the need to gain insights from huge unstructured piles of data has seen big contributions from artificial intelligence disciplines like natural language processing and image recognition. Because of this interdisciplinary and somewhat parallel history of the field, practitioners might struggle to get a hold on the subject. Potential users in empirical disciplines which want to take advantage of these sophisticated tools are confronted with a huge mass of different models, methods, notations and terminologies to navigate. This means that far less data is being exploited with state-of-the art technology than would be possible otherwise.

Probabilistic Programming[2] promises to bridge this gap. By providing probabilistic concepts as basic constructs of a programming language, lay-people are enabled to use probabilistic models. They are freed from needing to know about implementation details and enabled to mix-and-match inference and learning algorithms to their liking.

Factor graphs[3], which have their origin in coding theory, are a very general kind of graphical models. They provide us with a common way to express graphical models such as Markov Random Fields[4], Bayes' Nets[5] and statistical physics models like the Potts Model[6].

In this work we will use a Probabilistic Programming environment based on this universality of factor graphs, FACTORIE[7], to perform a *Direct Coupling Analysis* (DCA)[8] on protein sequences. DCA aims to mute indirect correlations in protein sequences, in order to use the direct correlations for structure prediction. To do this we will derive a statistical model, train it using the Contrastive Divergence[9] algorithm and evaluate our predictions by comparing them with radio-crystallography data.

1.2 Direct Coupling Analysis

One massive data source of our time is the sequencing of proteins. A protein fundamentally is a sequence $\mathbf{x}$ of length N, with each sequence member $x_i (i \in \{1, ..., N\})$ being one of $q = 20$ different amino-acids. In a biological context sequence members are called *sites* or *positions*, amino-acids are called *residues*. Instead of a whole protein one might also look at a *domain*, which is a building block of a larger protein, but has the same properties as just defined. The one-dimensional amino-acid sequence is folded in three-dimensional space to make up the actual protein. This three-dimensional structure is of great interest to biologists, and might be observed through expensive radio-christallography. Protein sequencing is much cheaper and data abundant, so one might wonder how to infer the three-dimensional structure from the two-dimensional sequence. This task is called *Protein Structure Prediction* (PSP).

The key biological phenomenon that enables us to use statistical methods on amino-acid sequences is that there are *micro-evolutions* happening within a certain domain family. A micro-evolution is when individual or multiple sites change or drop their amino-acid, while the domain preserves its general structure. To account for the dropped amino-acids, we introduce the notion of a *gap* in a sequence and expand our $q = 21$ to account for this. The output of protein sequencing over M samples of the same protein is therefore not a unique sequence, but up to M different sequences which show a lot of similarities (called *Multiple Sequence Alignment*, MSA). Sites that interact in three-dimensional space are expected to show correlations across these different sequences. This can be explained as following: if one site out of a pair that somehow interacts with one another, is changed and the other is not, the domain might not be stable and thus not show in nature at all.

Simple methods involving the co-variance of sites have been used (see for example [10]) to infer these interactions. The problem is that these methods can not differentiate between indirect correlations and direct interactions. Two sites might show show a high correlation, simply because they both interact with a third site. An ideal method would be able to mute these indirect correlations and only present the researcher with correlations that result from direct interaction.

To tackle this problem, Weigt et al. [8] have trained a global statistical model to represent the complex interactions between multiple amino-acids. They opted for modeling pairwise and single-site probabilities, thus employing a model known as the Potts model. The norm of the pairwise parameters is then used as a predictor for the physical distance. They call this technique *Direct Coupling Analysis* (DCA), and the best way to estimate the parameters from the MSA is an active field of research.[8][11][12][13][14][15] Some of the found techniques have been made available to practitioners, mostly in the form of MATLAB scripts.[16] [17]

1.3 Enabling Code Reuse and Abstraction Through Probabilistic Programming

Employing Probabilistic Programming[2] technologies in the DCA context could offer attractive advantages. By performing DCA in an object-oriented language, the problem can be expressed in domain specific terms instead of linear-algebra concepts. This should lift some cognitive load off the researcher and allow him to tinker and think on a level of abstraction more closely resembling her prior education. By using the imperative programming language constructs to describe the model, she might even be able to include domain specific knowledge that would be hard or impossible (from the domain expert's perspective) to express in mathematical terms. Furthermore the separation of concerns between model description, model inference and model training allows for trivial reuse of implementations of algorithms. This takes further load off the domain expert. It also bears the possibility for serendipitous cross-pollination from state-of-the-art research in other fields.

1.4 Structure of this Work

To ease the understanding of the Potts model, we will start with a specialized case and historical predecessor; the Ising model. After this, some preliminaries from information theory will be introduced, as these concepts will be used later in the work.

We then take a closer look at how the Potts model can be derived from our MSA data set through the maximum-entropy principle. After discussing some model details, we shall see how we turn our parameter-matrices into scalar predictions for DCA.

As a lead-up to the discussion of Imparatively Defined Factor Graphs, a general introduction to factor graphs shall be given, along with notes on the algebraic and notational details. To complement this we will look at the idea of Probabilistic Programming and provide an overview of availiable systems. We will then proceed to introduce FACTORIE, its background, architectural concepts and available algorithms.

Next up is a close inspection of the theoretical foundations of parameter estimation and structure learning in graphical models. After explaining maximum-likelihood learning, and discussing its intractability, we will introduce Contrastive Divergence learning[9]. Contrastive Divergence is an algorithm that was developed with intractable models in mind, and was chosen as the most promising candidate among the learning algorithms available in FACTORIE. To conclude this part, we will give a brief overview of the optimization algorithm and regularization method that will be used alongside CD, namely ADAGRAD Stochastic Gradient Descent [18] and Regularized Dual Averaging[19].

After having introduced the theory behind the methods that were used in our experiment, we will briefly describe its setup. Following are our empirical results, which will be compared to a naive mutual information based measure.

In our conclusion we will asses our results, and answer the questions asked above, regarding the practicability of FACTORIE for DCA.

2 Preliminaries

2.1 The Ising Model

The Ising Model is a statistical model, proposed by Ising as an explanation for ferromagnetism in 1925 [20]. Despite, or especially because, of its simplicity it has proved over time to be of great use in areas outside of physics. It's generalization, the Potts Model, shall be the main focus of this thesis. But because of the Ising Model's intuitive nature we will take a look at it first.

Consider a two dimensional, quadratic, grid of spins $\mathbf{x}$ with size N such that each component $x_i \mid i \in \{1, ..., N\}$ can take one of two values $+1$ and -1. In a ferro-magnetic substance each pair of neighbouring spins will influence each other to take on the same value, thus making homogeneous states for $\mathbf{x}$, where all x_{ij} have the same value, more likely. Formally we can express this as an energy function:

$$E(\mathbf{x}) = -\sum_{i,j} J_{ij} x_i x_j$$

With

$$J_{i,j} = \begin{cases} J & \text{if } x_i, x_j \text{ are neighbours} \\ 0 & \text{otherwise} \end{cases}$$

and J being a positive real number, e.g. 1 which we might call *interaction strength.*

We can further introduce a *local field* h_i, indicating a preference for a given x_i for either of our two values. Our modified energy function thus looks like this:

$$E(\mathbf{x}) = -\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$$

We thus find that the energy function should have minimums at states of $\mathbf{x}$ where all the terms in the first sum evaluate to 1, which is the case if they have the same state.

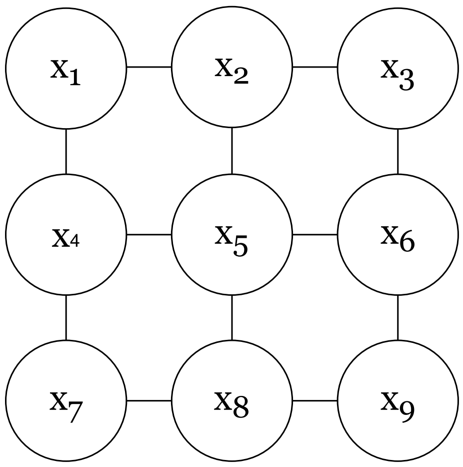
The probability distribution over all of the possible states of $\mathbf{x}$ is given by the Boltzmann distribution[21]

$$\begin{aligned} P(\mathbf{x}) &= \frac{1}{Z} e^{-E(\mathbf{x})} \\ &= \frac{1}{Z} \exp(\sum_{i,j} J_{ij} x_i x_j + \sum_i h_i x_i) \\ &= \frac{1}{Z} \prod_{i,j} \exp(J_{ij} x_i x_j) \prod_i \exp(h_i x_i) \end{aligned}$$

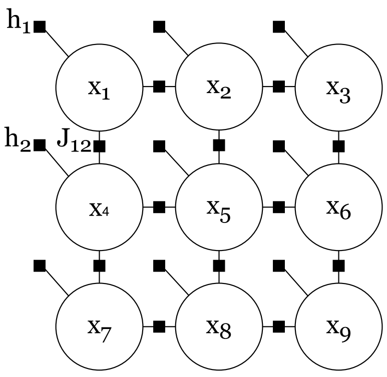
with the normalization constant or *partition function* Z over all possible states of X:

$$\begin{aligned} Z(\mathbf{x}, J, h) &= \sum_{\mathbf{x}\ in \Omega_X} \exp(\sum_{i,j} J_{ij} x_i x_j + \sum_i h_i x_i) \\ &= \sum_{\mathbf{x}\ in \Omega_X} (\prod_{i,j} \exp(J_{ij} x_i x_j) \prod_i \exp(h_i x_i)) \end{aligned}$$

Computing the partition function is NP-hard in general [22]. It involves enumerating all possible states of $\mathbf{x}$, and is therefore of complexity $O(2^n)$ for our Ising model. This intractability of the partition function is common to many complex statistical models and many approximation methods have been developed. For our problem these approximate methods unfortunately are NP-hard as well [23]. We shall therefore look at approaches which try to avoid evaluating the partition function altogether.



(a) As an undirected graphical model



(b) As a factor graph

Figure 1: A small Ising model and two different graphical representations

2.2 Entropy and Mutual Information

Definition: Entropy [24]

$$H[\mathbf{x}] = -\sum_{\mathbf{x}\ in \Omega_X} P(\mathbf{x}) ln P(\mathbf{x})$$

The entropy is non-negative and is 0 if one possible realization of $\mathbf{x} \in \Omega_X$ carries all the probability mass and all others $\Omega_X \setminus \mathbf{x}$ none. It takes its maximum value (dependent on the magnitude $|\Omega_X|$ of the outcome space) if all outcomes carry the same probability mass i.e. the uniform distribution. It can therefore be seen as a measure of how evenly the probability mass is spread out among members of the outcome space.

Definition: Kullback-Leibler divergence (KL), or relative entropy [25]

$$KL(P||Q) = -\sum_{\mathbf{x}} P(\mathbf{x}) ln[\frac{Q(\mathbf{x})}{P(\mathbf{x})}]$$

The KL divergence can be seen as a measure of how similar two distributions are. It can be said that $KL(P||Q) \geq 0$ and $KL(P||Q) = 0$ if and only if $Q(\mathbf{x}) = P(\mathbf{x})$. It should be noted that in general $KL(P||Q) \neq KL(Q||P)$

Definition: Mututal Information (MI)

$$I[\mathbf{x}, \mathbf{y}] = -\sum_{\mathbf{x} \in \Omega_X, \mathbf{y} \in \Omega_Y} P(\mathbf{x}, \mathbf{y}) ln[\frac{P(\mathbf{x})P(\mathbf{y})}{P(\mathbf{x}, \mathbf{y})}]$$

MI can be also defined as the KL of the joint distribution $P(\mathbf{x}, \mathbf{y})$ and the product of the marginals $P(\mathbf{x})P(\mathbf{y})$. It becomes obvious that $I[\mathbf{x}, \mathbf{y}] = 0$ if $\mathbf{x}$ and $\mathbf{y}$ are independent, that is $P(\mathbf{x}, \mathbf{y}) = P(\mathbf{x})P(\mathbf{y})$, and $I[\mathbf{x}, \mathbf{y}] > 0$ otherwise. It can be seen as a measure of how far $\mathbf{x}$ and $\mathbf{y}$ are from being independent. Another definition of MI is as the difference in entropy of $\mathbf{x}$ and $\mathbf{x}$ given $\mathbf{y}$. It can then be interpreted as a measure of how much information about $\mathbf{x}$ we gain by observing $\mathbf{y}$. In contrast to the Kullback-Leibler divergence MI is symmetric, that is $I[\mathbf{x}, \mathbf{y}] = I[\mathbf{y}, \mathbf{x}]$.

3 The q-State Potts Model for DCA

We shall now take a look at Weigt et al.'s derivation of the Potts Model[6] as a solution to the Protein Structue Prediction problem stated above. We want to build a global model $P(x_1, ..., x_n)$ where each x_i represents a position in the domain that can take the value of of $q = 21$ different amino-acids (and the gap). The question is thus; to what degree shall our model be consistent with the the observed data? Weight et al. opted for consistency in the single site- and pairwise frequencies.

To account for under-sampling effects the empirical local frequencies f_i and pairwise frequencies f_{ij} are adjusted with a pseudo-count ρ:

$$f_i(x_i) = \frac{1}{\rho q + M}[\rho + \sum_{a=1}^{M} \delta(x_i, x_i^a)]$$

$$f_{ij}(x_i, x_j) = \frac{1}{\rho q + M}[\frac{\rho}{q} + \sum_{a=1}^{M} \delta(x_i, x_i^a)\delta(x_j, x_j^a)]$$

With the Kronecker-delta $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise. Apart from these constraints, we want the least constrained model. This can be achieved by maximizing the entropy, as proposed by Jaynes in 1949.

Note that our x_i are now no longer scalars as in the Ising model, but rather unordered categorical values from our set of amino-acids $\mathcal{Q}$. In order to work with these values mathematically we introduce indicator functions (or *feature functions*). Single amino-acids are projected into a 21-dimensional vector space, with each one being represented by a unit vector:

$$\phi_i : \mathcal{Q} \rightarrow \mathbb{R}^{21}$$

The carthesian product of $\mathcal{Q}$ with itself represents the space of all possible pairwise observations and is projected into a 21×21-dimensional space:

$$\phi_{ij} : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{R}^{21\times 21}$$

$$\phi_{ij}(x_i, x_j) = \phi_i(x_i) \otimes \phi_j(x_j) = \phi_i(x_i)\phi_j(x_j)^\top$$

Note that our frequencies $f_i(x_i)$ are also 21-dimensional vectors, with entries for the frequencies of all possible amino-acids. The pairwise frequencies $f_{ij}(x_i, x_j)$ are 21×21 dimensional matrices.

We want the expectations of these indicator functions to be the same as our empirical frequencies and thus formulate our constraints:

$$f_i(x_i) \equiv P(x_i)\phi_i(x_i) = \sum_{x_k|k\neq i} P(x_1, ..., x_n)\phi_i(x_i)$$

$$f_{ij}(x_i, x_j) \equiv P(x_i, x_j)\phi_{ij}(x_i, x_j) = \sum_{x_k|k\neq i,j} P(x_1, ..., x_n)\phi_{ij}(x_i, x_j)$$

3.1 Maximum-Entropy Derivation

The question of what probability distribution to assume given a limited amount of data is an old problem going back to Laplace. In 1956 Jaynes proposed the *maximum entropy principle* as a solution [26]. The idea is to maximize Shannon's entropy whilst preserving constraints posed by the data. The intuition is that the probability mass thus gets spread out as evenly as possible, therefore providing us with the most unbiased prior distribution. Choosing a distribution thus turns into a constrained optimization problem. Our function to optimize is the entropy.

$$\max H[P(\mathbf{x})] = -\sum_{\mathbf{x}} P(\mathbf{x}) ln P(\mathbf{x})$$

$$s.t. \sum_{\mathbf{x}\in\Omega_X} P(\mathbf{x}) = 1 \wedge$$

$$f_i(x_i) = \sum_{x_k|k\neq i} P(x_1, ..., x_n)\phi_i(x_i) | \forall i \in \{1, ..., N\} \wedge$$

$$f_{ij}(x_i, x_j) = \sum_{x_k|k\neq i,j} P(x_1, ..., x_n)\phi_{ij}(x_i, x_j) | \forall i, j \in \{1, ..., N\}$$

We can solve this optimization problem through Lagrange's method. Introducing the Lagrange multipliers $\Theta = \{\theta_{1,2}, ..., \theta_{ij}, ..., \theta_{N-1,N}, \theta_1, ...\theta_i...\theta_N\}$ and λ we get the Lagrangian: [27]

$$L[P(\mathbf{x}), \Theta, \lambda)] =$$

$$-\sum_{\mathbf{x}\in\Omega_X} P(\mathbf{x}) ln P(\mathbf{x}) + \lambda(\sum_{\mathbf{x}} P(\mathbf{x}) - 1)$$

$$+ \sum_{i,j|i<j} [\theta_{ij} \sum_{x_k|k\neq i,j} P(\mathbf{x})\phi_{ij}(x_i, x_j) - f_{ij}(x_i, x_j)] + \sum_i [\theta_i \sum_{x_k|k\neq i} P(\mathbf{x})\phi(x_i) - f_i(x_i)]$$

Setting the derivative with respect to $P(\mathbf{x})$ to zero yields us the optimal distribution

$$P(\mathbf{x}) = \frac{1}{Z(\Theta)} \exp[\sum_{i,j|i<j} \theta_{ij}\phi_{ij}(x_i, x_j) + \sum_i \theta_i\phi(x_i)]$$

$$Z(\Theta) = \sum_{\mathbf{x}\in\Omega_X} [\exp(\sum_{i,j|i<j} \theta_{ij}\phi_{ij}(x_i, x_j) + \sum_i \theta_i\phi_i(x_i))]$$

The Lagrange factors Θ remain as parameters of the model that have to be fitted.

3.2 Differences to the Ising Model

The derived model is very similar to the Ising model discussed above. It contains several generalizations though. First, our "spins" x_i can now take on any of q values (often called colours in the Potts model context). It is therefore called a *q-state Potts model.* Our second generalization is the fact that there is no grid pattern among the variables. Every variable is potentially interacting with every other variable, the model is completely connected.

The interaction strength J_{ij} has been replaced with θ_{ij}, which is a matrix carrying weights for each of the possible amino-acid combinations for the position-pair i, j:

$$\theta_{ij} = \begin{pmatrix} \theta_{ij,(1,1)} & \theta_{ij,(1,2)} & \cdots & \theta_{ij,(1,q)} \\ \theta_{ij,(2,1)} & \theta_{ij,(2,2)} & \cdots & \theta_{ij,(2,q)} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{ij,(q,1)} & \theta_{ij,(q,2)} & \cdots & \theta_{ij,(q,q)} \end{pmatrix}$$

The local field h_i is now the local weight parameter θ_i which similarly carries weights, representing the preference for certain amino-acids at a given site i:

$$\theta_i = \begin{pmatrix} \theta_{i,1} & \theta_{i,2} & \dots & \theta i, q \end{pmatrix}^\top$$

The product between the θ_{ij} and ϕ_{ij} is the *Frobenius inner product*:
Definition: Frobenius inner product:

$$A : B = \sum_{n,m} A_{nm}B_{nm}$$

As only one entry in ϕ_{ij} is 1 and the others 0, this product essentially "selects" the right entry from the weights matrix θ_{ij}.

A small example:

$$\phi_1(x_1) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \phi_2(x_2) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\phi_{12}(x_1, x_2) = \phi_1(x_1)\phi_2(x_2)^\top = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \theta_{12} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

$$\theta_{12}\phi_{12}(x_1, x_2) = 4$$

3.3 Physical Details

The complete analytical form of the Potts model looks as following;

$$P(\mathbf{x}|\beta,\Theta) = \frac{1}{Z(\beta,\Theta)}\exp(-\beta E(\mathbf{x},\Theta))$$

with the energy function E (also called the Hamiltonian $\mathcal{H}$ [8]):

$$E(\mathbf{x},\Theta) = -\sum_{i,j|i<j}\theta_{ij}\phi_{ij}(x_i,x_j) - \sum_i \theta_i\phi_i(x_i)$$

and the partition function Z (or normalization constant):

$$Z(\beta,\Theta) = \sum_{\mathbf{x}\in\Omega_X}\exp[-\beta E(\mathbf{x},\Theta)]$$

The $\beta = \frac{1}{k_b T}$ is a constant which is composed of the temperature T and the *Boltzmann constant* k_b (which is of dimension energy divided by temperature). For our purposes we can set $\beta = 1$ and disregard it.[8] The Potts model has been thoroughly examined in statistical physics. It is used for analysing systems, making predictions about their phase transitions and computing phase transitions. Wu [28] offers a good overview. In those classical physical applications of the parameters Θ are usually known.

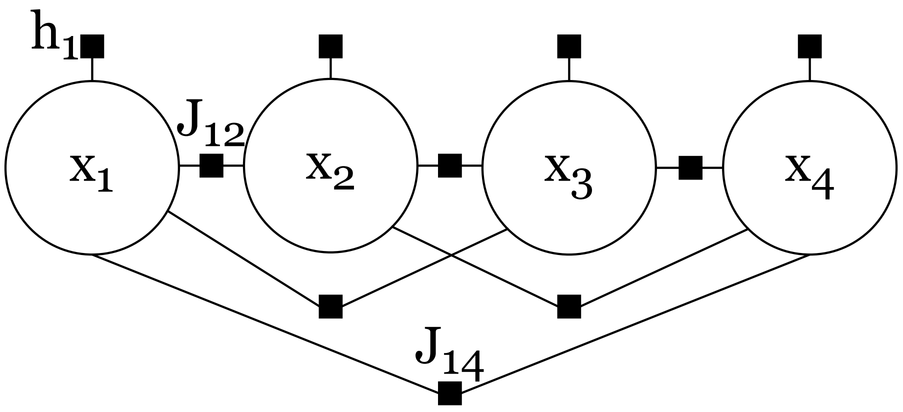


Figure 2: A completely connected Potts model in its factor graph representation (only some factors labeled)

3.4 Parameter Interpretation

Remember that our goal is to make statements about the interactions of sites in a protein domain. We are not given the values of our model parameters θ_{ij} and θ_i.

Cocco et al. [12] provide a method for interpreting the interaction parameters $\{\theta_{1,1},...,\theta_{N-1,N}\}$. Each θ_{ij} is a matrix and we want a scalar score to measure how "large" that matrix is. The *Frobenius Norm* of θ_{ij} is introduced for doing this:

$$F_{ij} = \sqrt{\sum_{a,b=1}^{q}\tilde{\theta}^2_{ij,(a,b)}}$$

with $\tilde{\theta}_{ij}$ being the transformed coupling matrices:

$$\tilde{\theta}_{ij,(a,b)} = \theta_{ij,(a,b)} - \theta_{ij,(\cdot,b)} - \theta_{ij,(a,\cdot)} + \theta_{ij,(\cdot,\cdot)}$$

The dot denotes averages over all values of q for the concerned position. This results in the sums over all rows and columns in $\tilde{\theta}_{ij}$ equaling zero. Cocco et al. explain this gauging with the intuition of "putting as much as possible" of the statistical mass into the local field parameters and "as little as necessary" into the couplings.

3.5 Proposed Estimation Strategies

How to estimate the parameters, is the key question that follows after describing a model. Before diving into the details later on, we shall give a brief overview of the different strategies devised so far. Weigt et al.[11] started off with using a loopy-belief propagation algorithm for inference, combined with gradient descent for learning. (mpDCA) Three years later, a team composed mostly of the same people proposed mfDCA[13] , a naive mean-field based approach, that essentially takes the inverse of the correlation matrix as an estimator, without having to resort to inference or optimization. Ekeberg [13] uses a pseudo-likelihood as an objective that is maximized (plmDCA) with results that surpass mfDCA. Apart from the estimation method, data preprocessing like sample re-weighting and sub-selecting can have big impact on performance and form the other pillar of research on DCA (ie. gap-optimized plmDCA [15]).

It should be noted, that both mpDCA and mfDCA are performed on column-wise frequencies calculated from the MSA. plmDCA on the other hand looks at each sample individually, which we will do as well in our method elaborated below.

4 Imperatively Defined Factor Graphs

4.1 Factor Graphs

Factor graphs are a very general kind of graphical model proposed by Kischang in 2001 [3]. They are a generalization of *Tanner graphs* [29] and thus have their origin in coding theory. They where originally thought up to discribe a generic message passing algorithm for decoding, the *sum-product algorithm*. In the same original work Kschischang et al. note that the sum-product algorithm is actually a generalization of many established message-passing algorithms, both from coding theory as well as those used in artificial intelligence (including Pearl's *belief propagation* [5], the Viterbi algorithm [12] and many others). The work can therefore be seen as a major contribution to the insight that coding theory and statistical inference form two sides of the same coin [1]. Almost in parallel Aji and McEliece proposed the *generalized distributive law* which also generalizes belief propagation [12]. Although factor graphs had been developed as a device to describe the sum-product algorithm, they have shown to be very valuable in their own right.

Factor graphs can be seen as a sort of *lingua franca* of graphical models. Bayes' nets (directed graphical models) and Markov Random Fields (undirected graphical models) can be readily transformed into Factor graphs [30]. Markov Random Fields are an important family of graphical models, interestingly also derived from the Ising model [4]. We could therefore show that our Potts model is also a MRF, we will not do so though as it very naturally translates into a factor graph.

Common to both directed and undirected graphical models is that they encode a set of conditional independence assumptions that have to be satisfied by any factorization that might be derived from the graph. In contrast to that the structure of a factor graph does not imply any conditional independence assumptions, in fact the factors do not have any direct probabilistic interpretation.

Kschischang et al. define a factor graph as following [3]:

Suppose that $g(x_1, ..., x_n)$ factors into a product of several local functions, each having some subset of $x_1, ..., x_n$ as arguments; i.e., suppose that

$$g(x_1, ..., x_n) = \prod_{j \in J} f_j(X_j)$$

where J is a discrete index set, X_j is a subset of $x_1, ..., x_n$ and $f_j(X_j)$ is a function having the elements of X_j as arguments.

Definition: A factor graph is a bipartite graph that expresses the structure of the factorization above. A factor graph has a variable node for each variable x_i, a factor node for each local function f_j, and an edge connecting variable node x_i to factor node f_j if and only if x_i is an argument of f_j

This definition is not constrained to probability distributions. In our case though, a factor graph shall refer to a factorization of a joint probability distribution over all components $x_1, ...x_n \in \mathbf{x}$. The factors may be normalized such that the sum over all possible outcomes of $\mathbf{x}$ is 1.

$$P(\mathbf{x}) = \prod_{j \in J} f_j(X_j)$$

or we may write our factorization to include a normalization constant Z

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{j \in J} f_j(X_j)$$

this does not affect our ability to draw out the factorization as a factor graph, as Z is a constant and can therefore be multiplied into any of our factors and not be represented separately in the graph.

If we now take a look at our model:

$$P(\mathbf{x}|\Theta) = \frac{1}{Z} \exp[\sum_{i,j|i<j} \theta_{ij}\phi_{ij}(x_i, x_j) \sum_i \theta_i\phi_i(x_i)]$$

we find that it naturally factorizes into local functions of pairwise interactions and single variable preferences:

$$P(\mathbf{x}|\Theta) = \frac{1}{Z} \prod_{i,j|i<j} \exp(\theta_{ij}\phi_{ij}(x_i, x_j)) \prod_i \exp(\theta_i\phi_i(x_i))$$

4.2 Probabilistic Programming

With the success probabilistic models, and graphical models in particular, have enjoyed in machine learning, the question arises how these methods can be made available to a wider audience of domain experts. Establishing an abstraction layer between model description and inference and learning procedures seems like a reasonable approach to this problem.

This idea has been called *Probabilistic Programming* and is a field of research that resides at the intersection of machine learning and language design.

Gordon, Henzinger, Nori et al. define a probabilistic program as:

[...] functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observations. [2]

Goodman gives an even simpler definition:

[...] probabilistic programming languages extend a well-specified deterministic programming language with primitive constructs for random choice. [31]

Current probabilistic programming languages could roughly be classified along two dimensions: Language paradigm (imperative, functional, logical) [2] and whether they are a self-contained language including a compiler or are implemented as an extension to an established deterministic language.

	Imperative	Functional	Logical
Self-Contained		BUGS [32] Church [34] Stan [36] IBAL [38]	BLOG [33] Alchemy [35] Tuffy [37] ProbLog [39] PRISM [40]
Libraries	FACTORIE [41] Figaro [42] Infer.NET [43]	FACTORIE Figaro	

This table is an almost comprehensive collection of probabilistic programming frameworks mentioned in [41] and [2]. It is easy to see that there is a heavy bias towards functional and logical languages. The majority of frameworks also exist in a stand-alone fashion, separated from potentially rich ecosystems of established languages. This has not been a reason for practitioners to refrain from using them of course. Especially BUGS is a widely used language for Bayesian modelling. In contrast stand the frameworks that are extensions to all-purpose, imperative languages. Infer.NET is an extension to C# developed at Microsoft Research, currently with an academic-only license. Figaro is an extension to Scala, developed by Charles River Analytics and available under a custom open-source license. In this work we shall use FACTORIE, also an extension to Scala. It is maintained by the Information Extraction and Synthesis Laboratory at the University of Massachusetts, Amherst and available under the Apache 2.0 licence.

4.3 Factorie

The expressive power of factor graphs to represent both directed and undirected graphical models has motivated a team of researches around Andrew McCallum (author of NLP toolkit MALLET [44]) to use them as a basis for a general purpose probabilistic programming framework. [7][41]. Their system is called FACTORIE (**Factor** graphs, **I**mperative, **E**xtensible) and combines multiple design decisions, many of which are motivated by McCallum et al.'s focus on NLP. FACTORIE is implemented as a library of Scala[45], a functional, interactive (including a REPL), and object-oriented language compiling to the JVM [46]. Scala stands for **sca**lable **la**nguage and was designed with parallelization and concurrency in mind. Scala has a static typing system that supports multiple inheritance through *traits*. It also allows for trivial implementation of additional operators, thus blurring the lines of libraries and *domain specific languages*. All of these language features make Scala a great fit for a probabilistic programming framework.

While factor graphs offer a concise and intuitive way to grasp and reason about graphical models, their declarative nature stands in contrast to how computer programs are usually written. McCallum et al. resolve this contrast through their approach of *imperatively defined factor graphs*. What they mean by this is that FACTORIE allows the user to use the well-known concepts of a general purpose language to describe the components of a model: variables, factors, structure, sufficient statistics etc.. These imperatively defined model components also offer an abstraction layer for developing inference and learning components that work across many different models.

FACTORIE contains modules for all sub tasks of a machine learning system, and separates them into three main steps (1) *Model Structure* (2) *Inference* and (3) *Learning*

Model Structure *Variables* in FACTORIE are typed objects in the object-oriented language. In addition to common variable kinds like *binary*, *categorical*, *real* and *ordinal*, variables can also be *strings*, *sets*, *objects* etc.. They hold a single possible value of a random variable, not distributions. Therefore they can be seen as containers of data or a single possible world. Variables have a *domain* which are also typed objects. Changes to variables are stored in *DiffLists*, which allows for very efficient implementation of sampling algorithms, as rejected sampling steps can be easily reverted.

The class hierarchy around factors is made up of the three distinct concepts of *families*, *templates* and *factors*. A *family* describes the general structure of a factor. I.e. our factors explained above which can be separated into a weight vector and a sufficient statistics function would form a family. A *template* inherits from a family. it describes how to locate neighbouring variable nodes and can have *weights*. The choice to use *templated factors*, stems from the creators' background in NLP. Factor graphs in NLP settings often use *parameter tying*, this means that the same parameter is used in different factors. *Factors* themselves are not persistent objects, but are instead instantiated by a template whenever they are needed. In our case parameter tying and templates are not necessary, so each factor is represented by a distinct family-template.

A *model* is simply a collection of templates or families (not of variables). It can be extended with the *parameters* trait, which makes all the weight vectors accesable for learning algorithms.

Inference FACTORIE supplies different inference algorithms, as well as a convenient architecture to implement additional ones. There are belief propagation variants for chains, trees [5] and loopy graphs [47]. Sampling algorithms include a ready-to-use Gibbs sampler [48] as well as a Metropolis-Hastings sampler [49] [50] which has to be supplied with a proposal distribution. Inference results in a *summary*, which is a collection of single-variable *marginals* and a value for Z.

Learning Different learning strategies can be employed through a choice of optimization algorithms, objective (or loss) function, regulatizations and learning rates. We find *Stochastic Gradient Descent* [51], ADAGRAD [18], *RDA* [19] for online-learning, and *limited-memory BFGS* [52][53] for batch learning. Objective functions are implemented through objects called *examples*, which are constructed out of a data point and are able to compute an objective value and a gradient. Available objectives which require inference as a sub-routine are maximizing the *likelihood* (Maximum Likelihood learning) and maximizing the *posterior* (MAP or Bayseian learning). We also find *Contrastive Divergence* [9] and *SampleRank* [54], which optimize different objectives that don't require inference. Different *trainers* are available, which provide convenient combinations of the methods above and take care of thread-safe parallelization.

5 Parameter Estimation Methodology

Recall that our task is to infer the pairwise connection parameters e_{ij} from data, and use these parameters as a measure for direct dependency between variables. Wu et al.[55] note that approaches to this problem of *structure learning* could be separated into two groups. The first approach is to search for possible structures based on local conditional independence tests [56]. This approach foremostly tries to decide which $\theta_{ij} \in \Theta$ should exist at all i.e. which $\theta_{ij} \neq \mathbf{0}$. This means that the problem is essentially a combinatorial search problem. The second approach would be to treat the problem as a ℓ_1 or ℓ_2 regularized maximum likelihood estimation of the parameters θ_{ij}. [57] Because the likelihood is convex, the problem reduces to a convex optimization problem.

5.1 Maximum Likelihood Estimation

Although it poses a set of challenges, which we will explore shortly, our work here shall focus on the second approach. Weigt et al.[8] use this approach without citing any prior work and might have derived it themselves. We shall follow a more formal derivation proposed in [57]. Instead of separating the steps of structure learning (deciding which interactions should exist) and parameter learning (deciding how strong interactions should be) the key idea is to subsume both of them into just learning the parameters. By enforcing some penalty on the "size" (a norm) of the parameters.

The likelihood of an outcome x given a model and parameters θ can be written as:

$$\mathcal{L}(\theta) = P(x|\theta)$$

If we assume that our training samples $D = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ to be i.i.d. (a strong assumption that doesn't necessarily hold in biological contexts) we can write the likelihood of the training dataset as following:

$$\mathcal{L}(\theta; D) = \prod_{\mathbf{x}^{(k)} \in D} P(\mathbf{x}^{(k)}|\theta) = P(D|\theta)$$

It is common to use the negative log-likelihood to turn the product into a sum:

$$l(\theta; D) = -log\mathcal{L}(\theta; D) = -\sum_{\mathbf{x}^{(k)} \in D} logP(\mathbf{x}^{(k)}|\theta)$$

If we now apply this to our Potts model from above:

$$P(d|\theta) = P(\mathbf{x}^{(d)}|\Theta)$$

$$= \frac{1}{Z(\Theta)} \prod_{i,j|i<j} \exp(\theta_{i,j}\phi_{ij}(x_i^{(d)}, x_j^{(d)})) \prod_i \exp(\theta_i \phi_i(x_i))$$

$$l(\Theta; D) = \sum_{\mathbf{x}^{(k)} \in D} [logZ(\Theta) - \sum_{i,j|i<j} \theta_{i,j}\phi_{ij}(x_i^{(k)}, x_j^{(k)}) - \sum_i \theta_i \phi_i(x_i^{(k)})]$$

$$= \; MlogZ(\Theta) - \sum_{\mathbf{x}^{(k)} \in D} [\sum_{i,j|i<j} \theta_{i,j}\phi_{ij}(x_i^{(k)}, x_j^{(k)}) + \sum_i \theta_i \phi_i(x_i^{(k)})]$$

because l is our objective function which we want to maximize, we can do linear transformations to the right-hand side, without changing the left hand side, and write:

$$l(\Theta; D) = logZ(\Theta) - \frac{1}{M}\sum_{\mathbf{x}^{(k)}\in D}[\sum_{i,j|i<j}\theta_{i,j}\phi_{ij}(x_i^{(k)}, x_j^{(k)}) + \sum_i \theta_i\phi_i(x_i^{(k)})]$$

$$= logZ(\Theta) - \langle \sum_{i,j|i<j}\theta_{i,j}\phi_{ij}(x_i^{(k)}, x_j^{(k)}) + \sum_i \theta_i\phi_i(x_i^{(k)})\rangle_D$$

This log-likelihood is a sum of convex functions, and is therefore a convex function itself. This means training our model can be seen as an unconstrained convex optimization problem.

$$\hat{\Theta} = \arg\min l(\Theta; D)$$

These can be solved through gradient descen methods. The basic idea of gradient descent is to step through the function in search of a local extreme by always stepping in the direction of the gradient of the current point: [1]

$$\Theta_{t+1} = \Theta_t + \mu\nabla l(\Theta_t)$$

μ here is the step length, also called the *learning rate*. μ can be seen as a hyperparameter of our model and should be treated accordingly. This means it should be tuned as it can affect our time to learn, the accuracy of our predictions and might even lead to pathological behaviour if chosen unwisely.

$$\frac{\partial l}{\partial\theta_{ij}} = \frac{\partial logZ(\Theta)}{\partial\theta_{ij}} - \langle\phi_{ij}(x_i, x_j)\rangle_D$$

$$\frac{\partial l}{\partial\theta_{i}} = \frac{\partial logZ(\Theta)}{\partial\theta_{i}} - \langle\phi_{ij}(x_i)\rangle_D$$

There is a fundamental hurdle in this approach though. We have to know the value of the partial derivatives of $logZ$ which are actually the expectations of the feature functions(analogusly for θ_i):[58]

$$\frac{\partial logZ(\Theta)}{\partial\theta_{ij}} = \frac{1}{Z(\Theta)}\frac{\partial Z(\Theta)}{\partial\theta_{ij}}$$

$$= \frac{1}{Z(\Theta)}\frac{\partial}{\partial\theta_{ij}}\sum_{\mathbf{x}\in\Omega_\mathbf{x}}\exp[E(\mathbf{x}|\Theta)]$$

$$= \frac{1}{Z(\Theta)}\sum_{\mathbf{x}\in\Omega_\mathbf{x}}\exp[E(\mathbf{x}|\Theta)]\phi_{ij}(x_i, x_j)$$

$$= \sum_{\mathbf{x}\in\Omega_\mathbf{x}}\frac{1}{Z(\Theta)}\exp[E(\mathbf{x}|\Theta)]\phi_{ij}(x_i, x_j)$$

$$= \sum_{\mathbf{x}\in\Omega_\mathbf{x}}P(\mathbf{x}|\Theta)\phi_{ij}(x_i, x_j)$$

$$= \langle\phi_{ij}(x_i, x_j)\rangle_{P(\mathbf{x}|\Theta)}$$

To know $logZ$ or Z exactly we would have to sum over the combinatorial state space of $\mathbf{x}$ ($|\Omega_\mathbf{x}| = Q^N$). If our factor graph was treelike there would be methods to compute Z

exactly without enumerating all possible states. For the general case, including loops in the model structure, Z is intractable. It happens that our model is loopy, as loopy as it could be: completely connected. To approximate Z, many different strategies have been devised. The best method heavily depends on the model structure, and there is still active research going on in this field. One approach would be to use a general message-passing (or belief- propagation) algorithm, which is run until a certain convergence criterion is met.[8] This loopy belief-propagation can be fruitful in many cases, but may take a very long time. An alternative approach is a sampling based approximation.

5.2 Sampling Based Approximation

The basic idea behind sampling methods (also called *Monte Carlo* methods) is simple. Assume you have a distribution $P(x)$ and you wish to compute an expectation $\langle f(x) \rangle_P = \sum_{x \in \Omega x} P(x) f(x)$ of a generic function. If you can't find a closed form solution for the expectation, you can approximate it numerically. If you have access to a sufficiently big set $\mathcal{H}$ of i.i.d. samples drawn from $P(x)$.

$$\mathcal{H} = \{x^{(1)}, ..., x^{(L)}\}$$

$$\langle f(x) \rangle_{P_\Theta^L} = \frac{1}{L} \sum_{x^{(i)} \in \mathcal{H}} f(x^{(i)})$$

Note that our samples in $\mathcal{H}$ can be synthetic samples. They have do be drawn from $P(x)$ but they don't have to be from an empirical dataset. We wrote the approximated expectation with respect to P_Θ^L, this denotes that we took the expectation with respect to a data distribution of L samples. Because our samples are i.i.d. and because of the law of large numbers, it can be shown that $\lim_{L \to \infty} P_\Theta^L = P_\Theta^\infty = P(\mathbf{x}|\Theta)$ [59].

How to generate these i.i.d. samples from $P(x)$ is where the actual challenge lies. We don't have access to the normalized probabilities and our original goal was to avoid enumerating the whole state space of x. This is where *Markov Chain Monte Carlo* (MCMC) methods come in. We shall focus on *Gibbs sampling* [48] which is a special case of the much older and more general Matropolis-Hastings method [49] [50].

The idea common to all MCMC methods is that we produce our samples in a stochastic process in which each sample $x^{(t+1)}$ is dependent only on it's predecessor $x^{(t)}$, hence *Markov chain.* The process has to be chosen in such a way that its stationary distribution is $P(\S)$.

Gibbs sampling is one of these processes that is intuitive and works well in high dimensional graphical models. Assume a multi-dimensional distribution $P(\mathbf{x}) = P(x_1, ..., x_n)$ is given by a factor graph $P(\mathbf{x}) = \frac{1}{Z} \prod_{j \in J} f_j(X_j)$ Sampling from $P(\mathbf{x})$ is impossible, but sampling from the conditionals distributions $P(x_i|\{x_1, ..., x_n\} \setminus x_i)$ is easy. If we fix all components apart from x_i this is equivalent to removing all factors which don't have x_i as an argument.

$$P(x_i|\{x_1, ..., x_n\} \setminus x_i) = \frac{1}{Z} \prod_{j \in J | x_i \in X_j} f_j(X_j)$$

The normalization constant of this distribution is easy to compute as we only have to sum over the Q possible states of x_i.

To generate a new sample $\mathbf{x}^{(t+1)}$ each component x_i of $\mathbf{x}^{(t)}$ is simply updated one by one. The conditioning is done with respect to the new state of already updated components and the old state of components yet to come.

$$x_1^{(t+1)} \sim P(x_1^{(t)}|\{x_2^{(t)}, ..., x_n^{(t)}\})$$

$$x_2^{(t+1)} \sim P(x_2^{(t)}|\{x_1^{(t+1)}, x_3^{(t)} ..., x_n^{(t)}\}) \text{ etc.}$$

Once we have obtained our *trace* $\mathcal{H}$, we could compute the estimated expectations of the feature functions needed for our gradient:

$$\frac{\partial l}{\partial \theta_{ij}} = \langle \phi_{ij}(x_i, x_j) \rangle_{P^L} - \langle \phi_{ij}(x_i, x_j) \rangle_D$$

Similar to message-passing, there are problems with this approach though. For once, there is the problem of convergence. It might take a very long time until the samples obtained through our Gibbs sampler are actually drawn from $P(\mathbf{x})$. Diagnosing if the chain has converged is hard enough to make up its own field of research. Once the chain is converged, we have to spend even more resources on drawing a sufficiently big number of samples.

5.3 Contrastive Divergence

Because of these problems in computing the partial derivatives, alternatives to ML learning have been proposed. Hinton [9] suggested a method he named *Contrastive Divergence* (CD) for random fields of the form $P(\mathbf{x}|\mathbf{\Theta}) = \frac{1}{Z(\mathbf{\Theta})} e^{-E(\mathbf{x},\mathbf{\Theta})}$.

Above we found that we need the following derivative:

$$\frac{\partial l}{\partial \theta_{ij}} = \langle \phi_{ij}(x_i, x_j) \rangle_{P_\Theta^\infty} - \langle \phi_{ij}(x_i, x_j) \rangle_{P^0}$$

P^0 shall denote our data distribution, denoted above with D, and $P_\Theta^\infty = P(\mathbf{x}|\Theta)$. We tried to approximate it by running our Gibbs sampler for very many intervals L and using the distribution of our trace P_Θ^L in place of P_Θ^∞.

Hinton's key idea is that it might be sufficient to use a small number K instead of a very large L to give us a general idea in which direction to walk. In fact he proposes that $L = 1$ might often be enough. So the gradient used in Contrastive Divergence looks like this:

$$\frac{\partial l}{\partial \theta_{ij}} = \langle \phi_{ij}(x_i, x_j) \rangle_{P_\Theta^K} - \langle \phi_{ij}((x_i, x_j) \rangle_{P^0}$$

With P_Θ^K being the the the data distribution generated by running K steps of Gibbs sampling on our original D given the current parameters Θ.

In practice, these gradients are sufficient for parameter learning, without any need to look at the objective function that is being optimized. It should be noted though that Contrastive Divergence learning minimizes the difference between $KL(P^0, P_\Theta^\infty)$ and $KL(P^K, P_\Theta^\infty)$ [9]. This objective function is not identical with the one ML-learning optimizes and CD-learning does not converge to the same parameter estimates. In general CD-learning shows lesser performance than ML-learning, but on average is very close to it. [60]

5.4 Stochastic Gradient Descent

Let Φ be the set of all indicator functions $\Phi = \{\phi_{1,1}, ..., \phi_{ij}, ...\phi_{N-1,N}, ..\phi_1, ...\phi_i, ...\phi_N\}$. If we now combinine the Contrastive Divergence gradient with the gradient descent updating term from above we get:

$$\Theta_{t+1} = \Theta_t + \mu(\langle\Phi(D)\rangle_{P^K_\Theta} - \langle\Phi(D)\rangle_{P^0})$$

This straightforward way to implement our Contrastive Divergence learning would use *batch gradient descent*[51], i.e. do K sampling steps on all samples in D and perform the above sum to get our gradient for a given step. Summing over all training samples at each updating step can be costly though, which is why alternatives have been explored.

Stochastic gradient descent (SGD) (also *online gradient descent*) is one of these alternatives [51]. Instead of computing the average gradient over our training data, we randomly step through our training samples, compute a sub-gradient for each one of them and update the parameters immediately. Our update step for one sample $\mathbf{x}^{(t)}$ thus looks like this:

$$\Theta_{t+1} = \Theta_t + \mu(\Phi(\mathbf{x}^{(t)}_{P^K_\Theta}) - \Phi(\mathbf{x}^{(t)}))$$

After all training samples have been passed over once, the process is usually repeated multiple times. Each time the order of the samples is randomly chosen (the *stochastic* aspect) and the number of these iterations is a hyperparameter of SGD learning.

The learning rate μ is another hyperparameter in SGD and can severely impact the outcome.

5.5 Adaptable Learning Rate

As Weigt et al.[11] note, our protein dataset suffers from a sampling bias. Because of a necessary sub-choice of organisms to sample from, and because of *phylogenetic* relatedness between these organisms, our samples can not be seen as independent. Essentially this means that across the sequences we will see a lot of similarities, which in turn endanger swamping the actually interesting micro-mutations by sheer mass. [11] resolve this issue by stepping through the sequences and reweighing them with the inverse of the count of sequences that are $> 80\%$ identical with the sequence at hand.

We shall attempt to address this issue by using the AdaGrad algorithm [61]. In Duchi et al. own words: "[AdaGrad] give[s] frequently occurring features very low learning rates and infrequent features high learning rates, where the intuition is that each time an infrequent feature is seen, the learner should "take notice." Thus, the adaptation facilitates finding and identifying very predictive but comparatively rare features." [61]. In practice this means that our learner keeps a running sum of the squares of all previous gradients. The means the update for one of our features looks as follows: [62].

$$\theta_{ij,t+1} = \theta_{ij,t} - \frac{\mu}{\sqrt{\sum_{t'=1}^{t} g^2_{ij,t'}}} g_{ij,t}$$

$$g_{ij,t} = \phi(x^{(t)}_{ij,P^K_\Theta}) - \phi(x^{(t)}_{ij})$$

5.6 Regularized Dual Averaging

As in [57] we want to use ℓ_1-regularization to induce sparsity. Unfortunately classical regularization, which introduces a penalty term $\lambda||\theta||_1$ to the objective function is not applicable to SGD. Xiao [19] proposes a method he calls *Regularized Dual Averaging* (RDA) to enable ℓ_1 and ℓ_2 regularization in online learning settings. The idea of RDA is to keep a running average of the sub-gradients:

$$\overline{g}_{ij,t} = \frac{1}{t}\sum_{t'=1}^{t} g_{ij,t'}$$

for each feature and only update the parameter should its absolute become bigger than the ℓ_1 hyperparameter λ:

$$\theta_{ij,t+1} = \begin{cases} 0 & \text{if} |\overline{g}_{ij,t}| \leq \lambda \\ -\text{sgn}(\overline{g}_{ij,t})\mu\sqrt{t}(|\overline{g}_{ij,t}| - \lambda) & \text{otherwise} \end{cases}$$

Combining this with the learning rate term from AdaGrad we get the following update step per feature:

$$\theta_{ij,t+1} = \begin{cases} 0 & \text{if} |\overline{g}_{ij,t}| \leq \lambda \\ -\text{sgn}(\overline{g}_{ij,t})\frac{\mu}{\sqrt{t\sum_{t'=1}^{t} g_{ij,t'}^2}}(|\overline{g}_{ij,t}| - \lambda) & \text{otherwise} \end{cases}$$

[62]

As our regularization happens in a per-sample basis, the we want to take number of samples M into account. Our overall hyperparameter therefore is $\lambda = \Lambda/M$

6 Experiment Setup

6.1 Note on Choice of Method

One of the main goals of this work is to explore the practicability of probabilistic programming with Factorie in an bioinformatics context. The algorithms explained above where all pre-implemented in Factorie and were deemed the most promising and interesting combination for our problem at hand. Especially the similarity of our model with *Restricted Boltzmann Machines* and Contrastive Divergence's success in training these (see for example [63]) was an inspiration.

6.2 Modelling Details

As neighbouring (low distance $|i - j|$) amino-acids are expected to show high correlations as well and are logically close in physical space, they are except from our analysis. Following Ekeberg's [source] example we opt to only include pairs with a $|i - j| > 4$.

We follow Weigt et al.'s[8] example in only passing interaction pairs into our Contrastive Divergence, that pass a certain mutual information threshold. The threshold used in [8] was 0.26, but as they used corrected mutual information measure, we opted for a lower value. Based on a visual inspection of the mutual information distribution, we decided on (0.06)which is where visible gap in the histogram lies. Tests found thresholds of 0.0 or > 0.08 to impact our performance in a negative way.

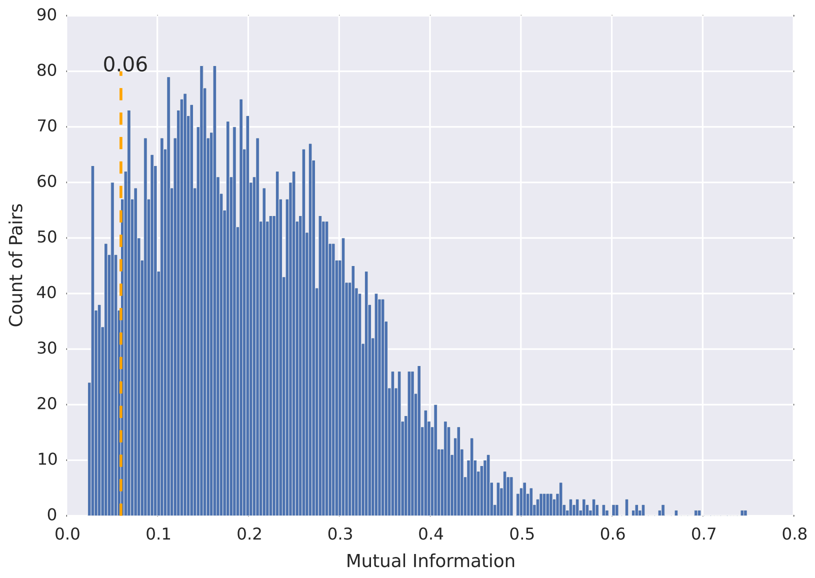


Figure 3: The histogram of the MI of all the pairs in PF00014, PF00017 and PF00035

Another important detail is the initial values of our parameters. Here we again follow Weigt et al. and initialize the pairwise parameters θ_{ij} with zero and the local parameters θ_i with the empirical log-frequencies $log(f_i(x_i))$.

6.3 Dataset

Our data consists of Pfam [64] multiple sequence alignments and actual distances between positions calculated from PDB [65] radio-crystallography datasets. This dataset was originally composed by Weigt et al.[11] based on the Pfam v24 and updated to match the Pfam v26 indexes by Magnus Ekeberg [13]. Because of limited computational resources, we will look at domain with relatively short length: Pf00014, Pf00017 and Pf00035.

6.4 Evaluation Metric

We will follow Ekberg's methodology and consider all position pairs ij with a measured distance $< 8.5\mathring{A}$ as *contacts* and in the set of positives $\mathcal{P}$.

Remember that we turned our matrices θ_{ij} into a predictor scalar by calculating its Frobenius norm F_{ij}, which is simply the square root of the sum of the squares of all entries. (See **3.4**)

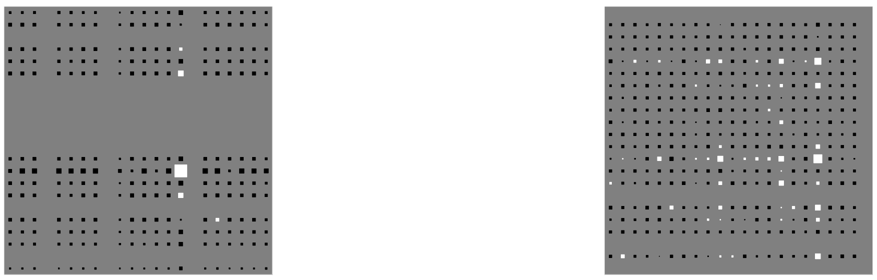
To evaluate how many contacts we predicted, we will order our pairwise interaction scores $F_{ij} \in \mathcal{F}$ descendingly and look at the true positive rates in the subset $\mathcal{F}_n$ of the n strongest predictions.

$$\mathcal{TP}_n = \mathcal{F}_n \cup \mathcal{P}$$

$$TPR_n = \frac{|\mathcal{TP}_n|}{n}$$

and the overall TP-Rate for all $p = |\mathcal{P}|$ positives.

$$TPR = \frac{|\mathcal{TP}_p|}{p}$$



(a) $\theta_{14,21}$ with $F_{14,21} = 0.1341$ (b) $\theta_{6,57}$ with $F_{6,57} = 0.09129$

Figure 4: Hinton Diagrams[66] of two weight matrixes from a model trained on PF00017. The size of the boxes represents the norm of the entry, the color its sign. The left matrix carries "more" weight overall.

6.5 Hyperparameters

We have 4 hyperparameters that have to be tuned: the learning rate μ, the ℓ_1 parameter Λ, the number of steps our CD sampler should take K and the number of times our SGD algorithm should pass over the dataset S. To not over-complicate our hyperparameter search, we decide on $K = 1$ (the canonical Contrastive Divergence K) and $S = 3$ the default value in FACTORIE.

To fnd appropriate μ and Λ we use FACTORIE's hyperparameter infrastructure to randomly visit the search space 50 times.

In particular we log-uniformly draw μs from $[10^{-3}, 10]$ and Λs from $[10^{-9}, 1]$. Our target measure for selecting "good" values was the TPR. Another interesting measure would have been to find the highest n for which $TPR_n = 1$. We will use PF00017 for this tuning and then see if the found hyperparemeters are robust enough to generate resonable predictions for PF00014 and PF000035.

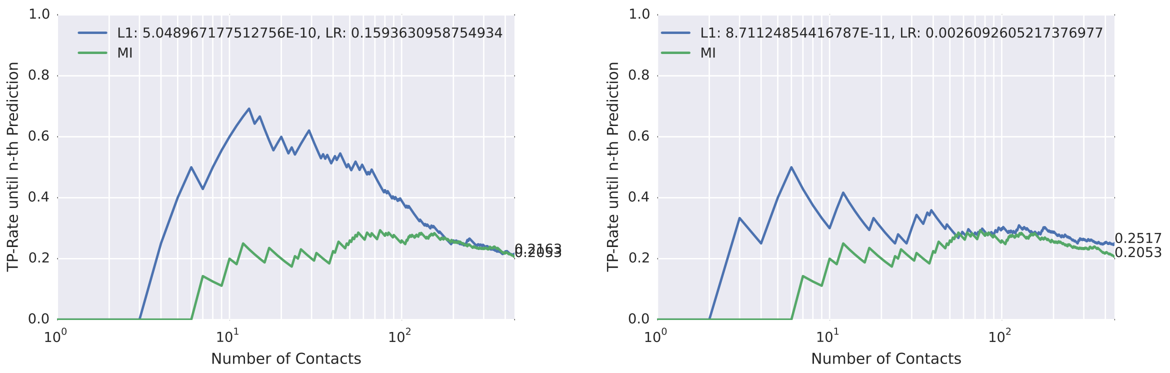


Figure 5: Two bad choices for hyperparamters, with performance barely better than the MI approach. [PF00017]

6.6 Results

Out of these 50 trails the best combination showed to be $\mu = 0.001257$ and $\Lambda = 0.002163$ with a $TPR_{CD} = 0.3532$ compared to $TPR_{MI} = 0.2075$.

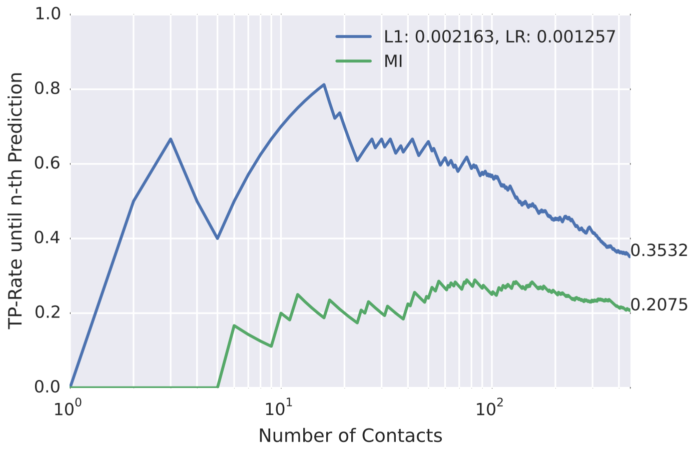


Figure 6: Our best result out of 50 trails with $\mu = 0.001257$ (lr) and $\Lambda = 0.002163$ (l1) [PF00017, $N = 78$, $M = 4403$]

We can see that these hyperparamters also produce better predictions than MI on PF00014 and PF00035, with a TPRs that are higher by 0.12 and 0.07 respectively.

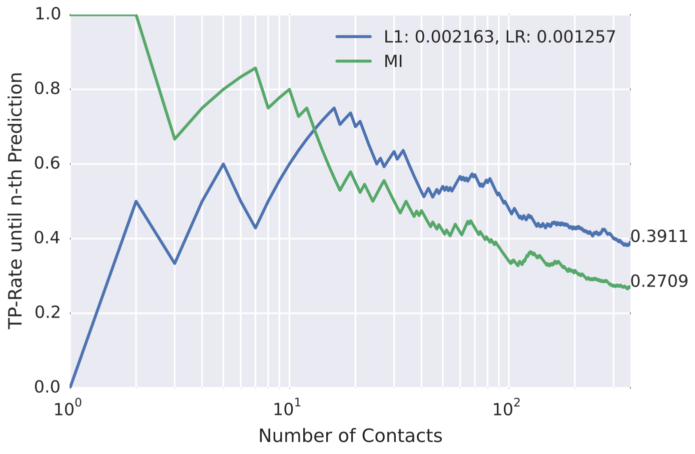


Figure 7: PF00014, $N = 54$, $M = 3090$

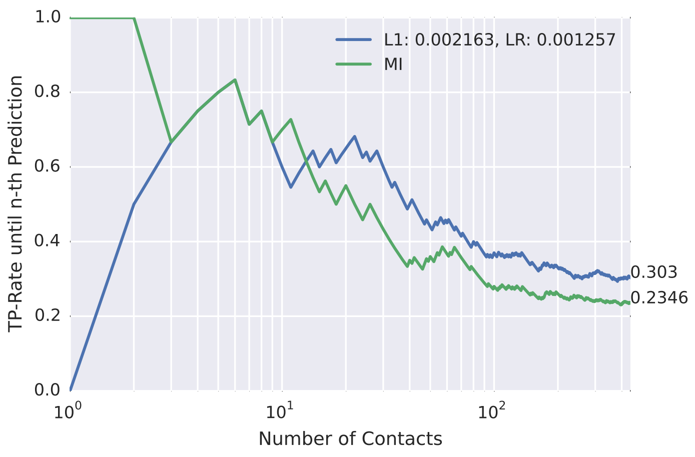


Figure 8: PF00035, $N = 68$, $M = 5584$

7 Conclusion

Our method succeeded in producing better contact predictions than the naive mutual information based approach. Our results are somewhat worse than other DCA approaches, such as mfDCA and plmDCA, which could have multiple reasons. One big aspect of this shortcoming is that we didn't perform much data preprocessing. Weigt et al. and Ekeberg took care of redundancy in the MSA by re-weighting of the sequences based on their similarity to the others, which we hoped to tackle through the AdaGrad algorithm. This might simply not be an appropriate method for our problem. Another aspect would be that Contrastive Divergence is ill-suited for the relatively large ($q = 21$) categorical domain of our variables.

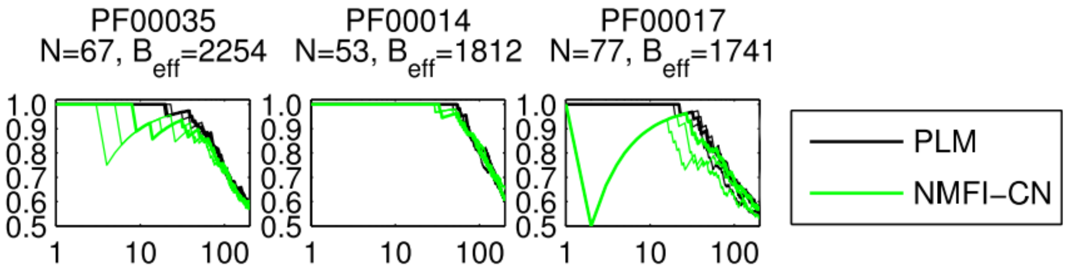


Figure 9: The results of Ekeberg (plmDCA) compared to mfDCA. Taken from [13]

From a software engineering perspective, we succeeded in building a high-level, object-oriented Potts model trainer. Factorie managed to supply us with efficient and thoroughly tested modules for optimization, regularization, sampling and hyperparameter tuning. There was no need for third-party software (safe for data preprocessing which was done in Python) at any of the major steps in constructing our system. The model construction itself took up the biggest chunk of time and lines-of-code for this work. The Imparatively Defined Factor Graphs framework worked well for modelling our problem, save for one cavet. The class hierarchy around factors (Families, Templates, Factors and Weights) was much to convoluted for our problem. This class design is very powerful and convenient in Natural Language Processing contexts, where factor templating and shared weights among different factors is common. In our context, where all four of the above mentioned classes had a 1:1 relationship, it caused some cognitive overhead. As our model structure was most easy to reason about in set theory concepts, the imperative nature of our model declaration was not really necessary. One of the logic-based probabilistic programming languages might have been a better fit.

8 Acknowledgements

I want to thank my loving parents Nicola and Wilfried Hanefeld for always nurturing my curiosity and giving me access to computers from an early age. I want to thank Pieter Abeel for getting me in artificial intelligence and the IPython[67] development team for powering both the development of the model and the writing of this document. I also want to thank Andreas Ruttor and Florian Stimberg for supervising me.

References

[1] D. J. MacKay, *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[2] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, “Probabilistic Programming,” in *International Conference on Software Engineering*, 2014.

[3] F. Kschischang, B. Frey, and H.-a. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[4] R. J. L. S. Kindermann, “Markov Random Fields and Their Applications,” *Contemporary Mathematics*, vol. 1, 1980.

[5] J. Pearl, A. Science, and L. Angeles, “Reverend Bayes on Inference Engines: A Distributied Hierarchical Approach,” in *AAAI-82*, 1982, pp. 133–136.

[6] R. B. Potts, “Some generalized order-disorder transformations,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 48, no. 01, pp. 106–109, 1952. [Online]. Available: http://journals.cambridge.org/article_S0305004100027419

[7] A. Mccallum, K. Schultz, and S. Singh, “FACTORIE: Probabilistic Programming via Imperatively Defined Factor Graphs,” *Neural Information Processing Systems (NIPS)*, 2009. [Online]. Available: http://www.cs.umass.edu/~kschultz/publications/factorie-nips-2009.pdf

[8] M. Weigt, R. a. White, H. Szurmant, J. a. Hoch, and T. Hwa, “Identification of direct residue contacts in protein-protein interaction by message passing.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, no. 1, pp. 67–72, 2009.

[9] G. E. Hinton, “Training products of experts by minimizing contrastive divergence.” *Neural computation*, vol. 14, no. 8, pp. 1771–800, Aug. 2002. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/12180402

[10] G. M. Süel, S. W. Lockless, M. a. Wall, and R. Ranganathan, "Evolutionarily conserved networks of residues mediate allosteric communication in proteins." *Nature structural biology*, vol. 10, no. 1, pp. 59–69, Jan. 2003. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/12483203

[11] M. Weigt, F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, and T. Hwa, "PNAS Plus: Direct-coupling analysis of residue coevolution captures native contacts across many protein families," *Proceedings of the National Academy of Sciences*, vol. 108, no. 49, pp. E1293–E1301, Dec. 2011. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3241805&tool=pmcentrez&rendertype=abstract

[12] S. Cocco, R. Monasson, and M. Weigt, "Inference of Hopfield-Potts patterns from covariation in protein families: calculation and statistical error bars," *Journal of Physics: Conference Series*, vol. 473, p. 012010, Dec. 2013. [Online]. Available: http://stacks.iop.org/1742-6596/473/i=1/a=012010?key=crossref.621b977590d0ac796c0d937b7c215a19

[13] M. Ekeberg, "Detecting contacts in protein folds by solving the inverse Potts problem – a pseudolikelihood approach," Ph.D. dissertation, Royal Institute of Technology, Stockholm, Sweden, 2012. [Online]. Available: http://www.math.kth.se/matstat/seminarier/reports/M-exjobb12/120522.pdf

[14] L. Burger and E. van Nimwegen, "Disentangling direct from indirect co-evolution of residues in protein alignments." *PLoS computational biology*, vol. 6, no. 1, p. e1000633, Jan. 2010. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2793430&tool=pmcentrez&rendertype=abstract

[15] C. Feinauer, M. J. Skwark, A. Pagnani, E. Aurell, P. Torino, and C. Science, "Improving contact prediction along three dimensions," 2014.

[16] E. A. C. Feinauer, M.J. Skwark, A. Pagnani, "gplmDCA, "Improving contact prediction along three dimensions"," 2014. [Online]. Available: http://gplmdca.aurell.org/download

[17] M. Morcos, F., Pagnani, A., Lunt, B., Bertolino, A., Marks, D.S., Sander, C., Zecchina, R., Onuchic, J.N., Hwa, T., Weigt, "mfDCA, "Direct-coupling analysis of residue coevolution captures native contacts across many protein families."," 2011. [Online]. Available: http://dca.upmc.fr/DCA/DCA.html

[18] J. Duchi and E. Hazan, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *The Journal of Machine Learning Research*, pp. 1–23, 2011.

[19] L. Xiao, "Dual Averaging Method for Regularized Stochastic Learning and Online Optimization," *Advances in Neural Information Processing Systems*, pp. 2116–2124, 2009.

[20] E. Ising, "Beitrag zur Theorie des Ferromagnetismus," *Zeitschrift fur Physik*, vol. 31, pp. 253–258, 1925.

[21] C. M. Bishop, “Pattern Recognition and Machine Learning,” in *Pattern Recognition*, ser. Information science and statistics, M. Jordan, J. Kleinberg, and B. Schölkopf, Eds. Springer, 2006, vol. 4, no. 4, ch. 8.3, p. 385. [Online]. Available: http://www.library.wisc.edu/selectedtocs/bg0137.pdf

[22] F. Barahona, “On the computational complexity of Ising spin glass models,” *Journal of Physics A: Mathematical and General*, vol. 15, no. 10, p. 3241, 1982. [Online]. Available: http://stacks.iop.org/0305-4470/15/i=10/a=028

[23] M. Jerrum and A. Sinclair, “Polynomial-time Approximation Algorithms for the Ising Model,” in *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming.* New York, NY, USA: Springer-Verlag New York, Inc., 1990, pp. 462–475. [Online]. Available: http://dl.acm.org/citation.cfm?id=90397.92354

[24] C. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal, The*, vol. 27, no. 3, pp. 379–423, July 1948.

[25] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 03 1951. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177729694

[26] E. T. Jaynes, “Information Theory and Statistical Mechanics,” *The Physical Review*, vol. 106, no. 4, pp. 620–630, 1957.

[27] D. Topics, D. M. Universit, and W. Semester, “Topic III.2: Maximum Entropy Models,” 2012. [Online]. Available: http://www.mpi-inf.mpg.de/~pmiettin/dtdm/slides/TIII_2.pdf

[28] F. Y. Wu, “The Potts model,” *Reviews of Modern Physics*, vol. 54, no. 1, pp. 235–268, 1982.

[29] N. Wiberg, *Codes and Decoding on General Graphs*, 1996, no. 440.

[30] C. M. Bishop, “Pattern Recognition and Machine Learning,” in *Pattern Recognition*, ser. Information science and statistics, M. Jordan, J. Kleinberg, and B. Schölkopf, Eds. Springer, 2006, vol. 4, no. 4, ch. 8.4.3, p. 738. [Online]. Available: http://www.library.wisc.edu/selectedtocs/bg0137.pdf

[31] N. D. Goodman, “The principles and practice of probabilistic programming,” *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL ’13*, p. 399, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2429069.2429117

[32] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter, “A language and program for complex Bayesian modelling. 43,” *The Statistician*, vol. 43, no. 1, pp. 169–177, 1994.

[33] B. Milch, B. Marthi, S. Russell, and C. S. Division, “BLOG: Relational Modeling with Unknown Objects,” 2006. [Online]. Available: http://www.cs.berkeley.edu/~russell/papers/ijcai05-blog.pdf

[34] N. D. Goodman, V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," 2008. [Online]. Available: http://arxiv.org/pdf/1206.3255.pdf

[35] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, and H. Poon, "The Alchemy System for Statistical Relational AI: User Manual," 2007. [Online]. Available: http://alchemy.cs.washington.edu/

[36] Stan Development Team, *Stan Modeling Language Users Guide and Reference Manual, Version 2.3*, 2014. [Online]. Available: http://mc-stan.org/

[37] F. Niu, A. Doan, and J. Shavlik, "Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS," in *Proceedings of the VLDB Endowment*, 2011, pp. 373–384.

[38] A. Pfeffer, "IBAL: An Expressive, Functional Probabilistic Modeling Language," *Introduction to Statistical Relational Learning*, p. 399, 2007.

[39] A. Kimmig, B. Demoen, L. D. Raedt, and R. Rocha, "On the Implementation of the Probabilistic Logic Programming Language ProbLog," *Theory and Practice of Logic Programming*, vol. 11, no. 235-262, 2011.

[40] T. Sato and Y. Kameya, "PRISM: a language for symbolic-statistical modeling," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[41] A. Mccallum, K. Rohanemanesh, M. Wick, K. Schultz, and S. Singh, "FACTORIE: Efficient Probabilistic Programming for Relational Factor Graphs via Imperative Declarations of Structure, Inference and Learning," *Advances on Neural Information Processing Systems*, pp. 1–3, 2008.

[42] A. Pfeffer, "Figaro: An Object-Oriented Probabilistic Programming Language," https://www.cra.com/commercial-solutions/probabilistic-modeling-services.asp, pp. 1–9, 2009.

[43] T. Minka, J. M. Winn, J. P. Guiver, and D. A. Knowles, "Infer.NET 2.5," http://research.microsoft.com/infernet, 2012.

[44] A. K. McCallum, "MALLET: A Machine Learning for Language Toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu

[45] M. Odersky and Al., "An Overview of the Scala Programming Language," www.scala-lang.org/docu/files/ScalaOverview.pdf, EPFL, Lausanne, Switzerland, Tech. Rep. IC/2004/64, 2004.

[46] T. Lindholm and F. Yellin, *Java Virtual Machine Specification*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[47] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized Belief Propagation," *Advances in Neural Information Processing Systems*, vol. 13, 2001.

[48] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images." *IEEE transactions on pattern analysis and machine intelligence*, vol. 6, no. 6, pp. 721–41, Jun. 1984. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/22499653

[49] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, 1953.

[50] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications." *Biometrika*, vol. 57, pp. 97–109, 1970.

[51] L. Bottou, *Online Learning and Stochastic Approximations*, D. Saad, Ed. Cambridge, UK: Cambridge University Press, 1998.

[52] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," Sep. 1995.

[53] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-newton matrices and their use in limited memory methods," 1994.

[54] M. Wick, K. Rohanimanesh, A. Culotta, and A. Mccallum, "SampleRank: Learning Preferences from Atomic Gradients," *Advances in Ranking*, p. 69, 2008.

[55] R. Wu, R. Srikant, and J. Ni, "Learning loosely connected Markov random fields," *Stochastic Systems*, vol. 3, no. 2, pp. 362–404, 2012.

[56] P. Abbeel, D. Koller, and A. Y. Ng, "Learning Factor Graphs in Polynomial Time and Sample Complexity," *Journal of Machine Learning Research*, vol. 7, pp. 1743–1788, 2006.

[57] V. Ganapahthi, D. Koller, and S.-i. Lee, "Efficient Structure Learning of Markov Networks using L 1 -Regularization," in *Conference on Neural Information Processing Systems*, no. Ml, 2006.

[58] O. Woodford, "Notes on Contrastive Divergence," [accessed 09-July-2014. [Online]. Available: http://www.robots.ox.ac.uk/~ojw/files/NotesOnCD.pdf

[59] D. J. MacKay, *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003, ch. 29.

[60] A. Miguel and G. E. Hinton, "On Contrastive Divergence Learning," in *Tenth International Workshop on Artificial Intelligence and Statistics*, vol. 0, 2005, pp. 33–40.

[61] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[62] C. Dyer, "Notes on AdaGrad," [accessed 09-July-2014]. [Online]. Available: http://www.ark.cs.cmu.edu/cdyer/adagrad.pdf

[63] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. [Online]. Available: http://www.nowpublishers.com/product.aspx?product=MAL&doi=2200000006

[64] R. D. Finn, A. Bateman, J. Clements, P. Coggill, R. Y. Eberhardt, S. R. Eddy, A. Heger, K. Hetherington, L. Holm, J. Mistry, E. L. L. Sonnhammer, J. Tate, and M. Punta, "Pfam: the protein families database," *Nucleic Acids Research*, vol. 42, no. D1, pp. D222–D230, 2014. [Online]. Available: http://pfam.sanger.ac.uk/

[65] B. Fc, T. K. Koetzle, G. J. Williams, E. E. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasum, "The Protein Data Bank: A Computer-based Archival File For Macromolecular Structures," *J. of. Mol. Biol*, vol. 112, pp. 535+, 1977.

[66] F. Bremner, S. Gotts, and D. Denham, "Hinton diagrams: Viewing connection strengths in neural networks," *Behavior Research Methods, Instruments, & Computers*, vol. 26, no. 2, pp. 215–218, 1994.

[67] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[68] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information science and statistics, M. Jordan, J. Kleinberg, and B. Schölkopf, Eds. Springer, 2006, vol. 4, no. 4. [Online]. Available: http://www.library.wisc.edu/selectedtocs/bg0137.pdf