

DRAFT: PLEASE DO NOT DISTRIBUTE

# Help Notes for MRFLearning Code

**Sridevi Parise & Max Welling**  
Department of Computer Science  
University of California Irvine  
sparise,welling@ics.uci.edu

September 29, 2005

## 1 Introduction

This is a set of MATLAB functions related to learning in undirected graphical models or Markov Random Fields (MRFs). It is part of an effort to create a publicly available repository where authors can contribute code and benchmark datasets that will allow for free comparison of various methods for learning in MRFs.

### 1.1 Supported Models/Architectures

Currently we deal with binary valued ( $\{0, 1\}$  or  $\{-1, +1\}$ ) pairwise MRF models also known as Boltzmann machines. The PDF is given by,

$$P^{\text{FO}}(\mathbf{x}) = \frac{1}{Z(W, \boldsymbol{\alpha})} e^{\sum_{i,j} W_{ij} x_i x_j + \sum_i \alpha_i x_i} \quad (1)$$

where,  $W$  are the weights and  $\alpha$  the biases.

Within these models, we provide additional support for architectures that allow for generating perfect samples so that one could generate synthetic data sets and run various learning algorithms and then use statistical measures such as bias and variance of the estimated parameters to assess performance. We deal with square grids which are either fully observed or have unobserved nodes interspersed with observed nodes in such a way that the neighbors of observed nodes are all unobserved nodes and vice versa. We also consider models with only non-negative weights but arbitrary biases. With this constraint we allow arbitrary architectures if all variables are observed while in the presence of unobserved nodes we restrict to bipartite

architectures, i.e. the neighbors of unobserved nodes must be observed and vice versa.

## 1.2 Function List

- Inference and Sampling in Grid Models
  1. **runJTgrid.m**: Junction Tree (JT) inference for rectangular grid boltzmann machine
  2. **findGridMarginals.m**, **getNeighbors.m**: helper functions
  3. **JTsampleGrid.m**: Exact sampling for rectangular grid boltzmann machines using JT
- Sampling in Models with positive interactions
  1. **CFTP.m**: Exact sampling using *Coupling From The Past*
- Learning Algorithms
  1. **MLgrid.m**: Maximum Likelihood(ML) learning in fully observed rectangular grid boltzmann machines.
  2. **CD.m**: Contrastive Divergence(CD) learning for fully observed general boltzmann machines
  3. **PL.m**: Psuedolikelihood(PL) learning for fully observed general boltzmann machines
  4. **PMM.m**: Pseudo Moment Matching (PMM) learning for fully observed general boltzmann machines
  5. **CD\_RBM.m**: CD learning for partially observed restricted boltzmann machines (RBMs)
- Other
  1. **mapModelStructs.m**: convert between grid and adjacency model struct formats

## 2 Function Descriptions

This section contains notes on some of the above functions. The reader is advised to look at the comments within the code in addition to these to gain a better understanding.

## runJTgrid.m

Given an  $(RXC)$  grid with  $R$  rows and  $C$  columns, (assume  $R$  is the smaller of the two dimensions), the JT turns out to be a chain where the  $n^{th}$  ‘supernode’ in the JT chain contains  $(R+1)$  consecutive nodes starting from node  $n$  (assuming nodes are numbered column-wise). It can be shown that these super-nodes are maximal cliques under an appropriate triangulation of the original grid. Complexity is proportional to  $2^{(R+1)}$  and hence this kind of inference is practical only when the number of rows is small.

We include each of the original potentials into one of the supernode potentials and run the JT algorithm. In the end, we get the marginals of these supernodes.

Throughout the code, we use MATLAB like column-first linear indexing when using a linear representation for a multi-dimensional quantity.

## JTsampleGrid.m

Given the output from the JT inference for a grid, we can use the supernode marginals to generate exact samples. Exact sampling using junction trees seems more general since the joint can be factorized in terms of conditionals that can be expressed using only the supernode and separator marginals of the JT. In our special case of the JT chain for a grid, this is easy to see since two consecutive supernodes have all but one node in common.

## MLgrid.m

The gradients for the ML learning are:

$$\begin{aligned}\alpha_i^{t+1} &= \alpha_i^t + \rho (\langle X_i \rangle_0 - \langle X_i \rangle_\infty) \\ &= \alpha_i^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n)} - \langle X_i \rangle_\infty \right)\end{aligned}$$

$$\begin{aligned}w_{ij}^{t+1} &= w_{ij}^t + \rho (\langle X_i X_j \rangle_0 - \langle X_i X_j \rangle_\infty) \\ &= w_{ij}^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n)} X_j^{(n)} - \langle X_i X_j \rangle_\infty \right)\end{aligned}$$

Here,  $(X_1^n, X_2^n, \dots, X_d^n)$  is the  $n^{th}$  data vector,  $N$  is the total number of data samples and,  $\langle \cdot \rangle_0$  are expectations under the empirical data distribution and

$\langle \rangle_\infty$  are expectations under the distribution using parameters at time  $t$ , which can be found by running the JT inference. For ML learning we can also monitor the objective function since the log-likelihood can be computed from the JT output similar to sampling.

### CD.m

The gradients for the  $K$ -step CD learning are:

$$\begin{aligned}\alpha_i^{t+1} &= \alpha_i^t + \rho (\langle X_i \rangle_0 - \langle X_i \rangle_K) \\ &= \alpha_i^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n)} - \frac{1}{N} \sum_{n=1}^N X_i^{(n,K)} \right) \\ w_{ij}^{t+1} &= w_{ij}^t + \rho (\langle X_i X_j \rangle_0 - \langle X_i X_j \rangle_K) \\ &= w_{ij}^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n)} X_j^{(n)} - \frac{1}{N} \sum_{n=1}^N X_i^{(n,K)} X_j^{(n,K)} \right)\end{aligned}$$

Here,  $(X_1^n, X_2^n, \dots, X_d^n)$  is the  $n^{\text{th}}$  data vector,  $N$  is the total number of data samples and,  $\langle \rangle_0$  are expectations under the empirical data distribution and  $\langle \rangle_K$  are expectations under the  $K$ -step reconstructions at time  $t$ . That is, the distribution obtained by running a Markov chain (that has the distribution using parameters at time  $t$  as the equilibrium distribution) for  $K$  steps starting at the data distribution. We run  $N$  chains starting at each data sample.  $(X_1^{(n,K)}, X_2^{(n,K)}, \dots, X_d^{(n,K)})$  is the  $K$ -step reconstruction starting at  $n^{\text{th}}$  data vector.

For the  $K$ -step reconstructions, we use node-by-node Gibbs sampling. The required conditional distributions (in the  $\{-1, +1\}$  representation) are:

$$p(X_i = -1 | X_{-i}^{(n)}) = \frac{1}{1 + e^{2(\alpha_i + \sum_{n_i} w_{in_i} X_{n_i}^{(n)})}}$$

where,  $n_i$  are neighbors of node  $i$ .

### PL.m

In PL we want to maximize,

$$\log \prod_n \prod_i p(X_i^{(n)} | X_{-i}^{(n)})$$

For boltzmann models ( $\{+1,-1\}$ ), the gradient learning equations are:

$$\alpha_i^{t+1} = \alpha_i^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n)} + 1 - \frac{1}{N} \sum_{n=1}^N \left( \frac{2}{1 + e^{-2(\alpha_i^{(t)} + \sum_{n_i} w_{in_i}^{(t)} X_{n_i}^{(n)})}} \right) \right)$$

$$w_{ij}^{t+1} = w_{ij}^t + \rho \left( \frac{1}{N} \sum_n 2X_i^{(n)} X_j^{(n)} + \frac{1}{N} \sum_n X_i^{(n)} + \frac{1}{N} \sum_n X_j^{(n)} - T_{ij}^t \right)$$

where,

$$T_{ij}^t = \frac{1}{N} \sum_n \left( \frac{2X_j^{(n)}}{1 + e^{-2(\alpha_i^{(t)} + \sum_{n_i} w_{in_i}^{(t)} X_{n_i}^{(n)})}} \right) + \frac{1}{N} \sum_n \left( \frac{2X_i^{(n)}}{1 + e^{-2(\alpha_j^{(t)} + \sum_{n_j} w_{jn_j}^{(t)} X_{n_j}^{(n)})}} \right),$$

and  $n_i$  are neighbors of node  $i$ ,  $N$  is the sample size,  $(X_1^n, X_2^n, \dots, X_d^n)$  is the  $n^{\text{th}}$  data vector.

### PMM.m

The ‘‘pseudo-moment matching’’ (PMM) was proposed in [1]. The idea is to choose the parameters such that if we run loopy BP on these parameters we obtain the observed frequency counts on edges and nodes. For binary random variables the equations were derived in [2],

$$W_{ij} = \log \left( \frac{\xi_{ij}(\xi_{ij} + 1 - p_i - p_j)}{(p_i - \xi_{ij})(p_j - \xi_{ij})} \right)$$

$$\alpha_i = \log \left( \frac{(1 - p_i)^{|\mathcal{N}_i| - 1} \prod_{j \in \mathcal{N}_i} (p_i - \xi_{ij})}{p_i^{|\mathcal{N}_i| - 1} \prod_{j \in \mathcal{N}_i} (\xi_{ij} + 1 - p_i - p_j)} \right)$$

where  $p_i = \hat{P}(x_i = 1)$  and  $\xi_{ij} = \hat{P}(x_i = 1, x_j = 1)$  are given by empirical estimates (i.e. frequency tables).  $\mathcal{N}_i$  denotes the neighbors of node  $i$ . As it stands, this method is only defined for fully observed models.

### CD\_RBM.m

The gradients for the  $K$ -step CD\_RBM learning are:

$$\alpha_i^{t+1} = \alpha_i^t + \rho (\langle X_i \rangle_0 - \langle X_i \rangle_K)$$

$$= \alpha_i^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n),0} - \frac{1}{N} \sum_{n=1}^N X_i^{(n,K)} \right)$$

$$\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^t + \rho (\langle X_i X_j \rangle_0 - \langle X_i X_j \rangle_K) \\
&= w_{ij}^t + \rho \left( \frac{1}{N} \sum_{n=1}^N X_i^{(n),0} X_j^{(n),0} - \frac{1}{N} \sum_{n=1}^N X_i^{(n),K} X_j^{(n),K} \right)
\end{aligned}$$

This is like CD but here,  $\langle \cdot \rangle_0$  are expectations under  $p_0(v)p_\infty(h|v)$ , where  $v$  are visible nodes and  $h$  are hidden nodes.  $X_i^{(n),0}$  is the value of node  $i$  in the  $n^{\text{th}}$  data vector if  $i$  is a visible node, else it is a sample from  $p(X_i|data)$  using current model. To generate the  $K$  step reconstructions, for each data case, we start at  $\{X_i^{(n),0}\}, i = 1, 2, \dots, \#nodes$  and run a markov chain  $K$  steps. At each step, we sample alternately the visible nodes given the hidden nodes and viceversa. The required conditionals are (in  $\{-1/+1\}$  representation):

$$p(h_i = -1|\vec{v}) = \frac{1}{1 + e^{2(\sum_{n_i} w_{n_i i} v_{n_i} + \alpha_i)}}$$

The equation is similar for  $p(v_i = -1|\vec{h})$ .

## References

- [1] T.S. Jaakkola M.J. Wainwright and A.S. Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation via pseudomoment matching. In *AISTATS*, 2003.
- [2] M. Welling and Y.W. Teh. Approximate inference in boltzmann machines. *Artificial Intelligence*, 143:19–50, 2003.