CS 475 Machine Learning: Homework 6
Graphical Models 2
Due: Monday December 8, 2014, 11:59pm
50 Points Total Version 1.1

Late hours cannot be used on this assignment.
Late assignments will not be accepted.
Make sure to read from start to finish before beginning the assignment.

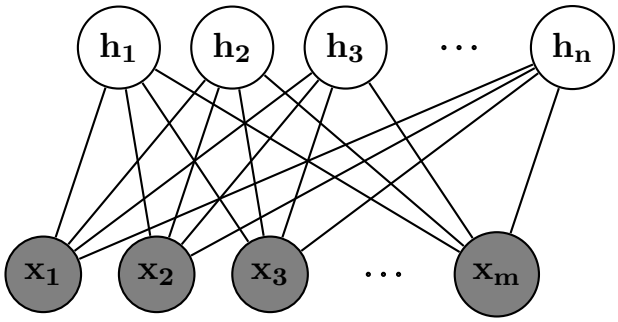
1 Programming (30 points)

A popular type of recurrent neural network is the Boltzmann machine, the development of which goes back to the mid 1980s. Boltzmann machines are neural networks that can learn internal representations. While learning Boltzmann machines are challenging, a common variant, the Restricted Boltzmann Machine (RBM) allows for efficient learning algorithms. For this reason, and others, many deep networks are composed of RBMs. A RBM is characterized by two layers: a hidden layer and an observed layer. These two layers form a bipartite graph, i.e., there are no connections within a layer. The observed layer is the input and the learned representation is captured by the hidden layer.

We have done exact inference on RBM when learning its parameters, which involves exponential number of summations. It is tractable only when the graph is very small. In this assignment, you will only implement inference – *not learning* – for a given RBM. Rather than rely on exact inference, you will implement an approximate inference algorithm based on a stochastic sampling algorithm. Specifically, you will compute the marginal probability of a hidden node in a given RBM network and parameters. As a reminder, we will first briefly introduce RBM, and then describe how to compute marginals by sampling.

1.1 RBM

Consider the following RBM configuration with n latent variables and m observed variables, all of which are binary valued.



Probability in RBM models is often expressed as "energy", where maximizing the probability means minimizing the energy. The joint probability distribution for the variables in this RBM is given by:

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{h})\} \tag{1}$$

where $E(\mathbf{x}, \mathbf{h})$ is the energy function and Z is the partition function that normalizes the distribution. The energy function is defined as:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{x}^T \mathbf{W} \mathbf{h} - \mathbf{b}^T \mathbf{x} - \mathbf{d}^T \mathbf{h} \tag{2}$$

where $\mathbf{W} = (w_{i,j})$ is a matrix of weights associated with the connection between the visible unit x_i and the hidden unit h_j. $\mathbf{b}$ and $\mathbf{d}$ are vectors, where the ith position of $\mathbf{b}$ (b_i) and and the jth position of $\mathbf{d}$ (d_j) correspond to the bias parameters for the observed variable x_i and the latent variable h_j respectively.

The partition function Z is:

$$Z = \sum_{(\mathbf{x}, \mathbf{h})} \exp\{-E(\mathbf{x}, \mathbf{h})\} \tag{3}$$

which sums over all possible combinations of the vectors $\mathbf{x}$ and $\mathbf{h}$.

An RBM is a Markov Random Field (MRF), so the Markov blanket of a node is simply all of its neighbors. In the case of the RBM, the Markov blanket of an observed node is the set of connected latent nodes (i.e., all of them) and the Markov blanket for the latent node is all connected observed nodes (i.e., all of them.) This yields the following conditional independence probabilities:

$$p(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{m} p(x_i|\mathbf{h}) \tag{4}$$

$$p(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} p(h_j|\mathbf{x}) \tag{5}$$

The conditional probability of a single variable (e.g., $p(x_i = 1|\mathbf{h})$, where the "=1" is usually implied) is represented using a sigmoid activation function $\sigma(z)$:

$$p(h_j|\mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{W}_{-,j} + d_j) \tag{6}$$

$$p(x_i|\mathbf{h}) = \sigma(\mathbf{h}^T \mathbf{W}_{i,-}^T + b_i) \tag{7}$$

where $\mathbf{W}_{-,j}$ is the jth column of $\mathbf{W}$, $\mathbf{W}_{i,-}$ is the ith row of $\mathbf{W}$, and $\sigma(z)$ is defined as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{8}$$

The likelihood of a single example $\mathbf{x}$ is given by:

$$p(\mathbf{x}) = \sum_{\mathbf{h}} \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{h})\} \tag{9}$$

You will be asked to compute the likelihood of an example. However, rather than summing over the hidden variables, which would be prohibitively expensive, you will use a sampling algorithm.

1.2 Sampling

The idea behind a sampling algorithm is to generate samples from the true distribution. If you generate enough samples (possibly an infinite number) then you can compute the true distribution by counting how many samples you obtained for each outcome. In practice, we use a finite number of samples and settle for an approximation of the true distribution. In this case, you will be approximating the value of $p(h_j)$, the marginal of h_j given the parameters.

The sampler will have two steps, which will be repeated many times to obtain a set of samples. Assume we have values for the input $\mathbf{x}$, the first step will be to sample the configuration of $\mathbf{h}$ given $\mathbf{x}$.

1. Generate one sample of $\mathbf{h}$, $\mathbf{h}^{(t)} \sim p(\mathbf{h}|\mathbf{x} = \mathbf{x}^{(t-1)})$, according to Eq. 5 and Eq. 6:

$\mathbf{h}^{(t)}$ indicates the tth sample of $\mathbf{h}$ and $\mathbf{x}^{(t-1)}$ is the value of $\mathbf{x}$ from the previous sample. In the first step ($t = 1$), $\mathbf{x}$ will be initialized in some way. To create a sample, you will draw a random number (0 or 1) from a distribution where the probability of drawing a 1 (for each position in the vector) is given as $p(\mathbf{h}|\mathbf{x} = \mathbf{x}^{(t-1)})$ (see details below). Note that because of conditional independence, the draw of each position of $\mathbf{h}$ is conditionally independent.

Using your sample for $\mathbf{h}$, you will draw a new sample for $\mathbf{x}$.

2. Generate a sample of $\mathbf{x}$, $\mathbf{x}^{(t)} \sim p(\mathbf{x}|\mathbf{h} = \mathbf{h}^{(t)})$, according to Eq. 4 and Eq. 7

Sampling $\mathbf{x}$ follows the same procedure as sampling $\mathbf{h}$.

At the conclusion of these two steps, you will have a pair $(\mathbf{x}^{(t)}, \mathbf{h}^{(t)})$, a sample for both $\mathbf{x}$ and $\mathbf{h}$. Repeating this procedure for many T iterations will yield T samples. Note that you will only need $\mathbf{x}$ below for computing $p(\mathbf{x})$, but $\mathbf{h}$ is used to draw new samples of $\mathbf{x}$ in each round.

After you generate T samples of $(\mathbf{x}^{(t)}, \mathbf{h}^{(t)})$, you can compute the marginal probability of a hidden variable (the probability that the variable $\mathbf{h}_j$ takes value 1) as follows:

$$p(\mathbf{h}_j = 1) = \frac{\text{Number of samples } (\mathbf{x}^{(t)}, \mathbf{h}^{(t)}) \text{ such that } \mathbf{h}_j^{(t)} = 1}{\text{Total number of samples } (T)} \tag{10}$$

As T goes to infinity, this converges to the true value for $p(\mathbf{h}_j = 1)$. Note that in this case, we are counting the number of times $\mathbf{h}_j = 1$ shows up in the samples.

1.2.1 Sampling Procedure

Each position of the vector $\mathbf{x}$ and $\mathbf{h}$ is a binary random variable. We can sample a binary random variable y based on the probabilities $P(y = 1) = p$ and $P(y = 0) = 1 - p$. If $u \sim$Uniform$([0, 1])$, then,

$$y = \begin{cases} 1 & \text{if } u < p \\ 0 & \text{if } p \leq u \end{cases} \tag{11}$$

In order to generate the value of one node, you will use either (6) or (7) and the value u sampled from the *Uniform* distribution.

To ensure that everyone gets the same samples, you must use the following procedure *exactly*. Create a `java.util.Random` object with a seed of 0. Only use this object for sampling and use the same object for all sampling operations. You will start by sampling the values for the vector $\mathbf{h}$. Sample positions of the vector in increasing order, starting from the 1th

position to the nth position. Each position should be sampled with a single call to the `nextDouble()` method on the `Random` object. You will then repeat this sampling procedure for vector $\mathbf{x}$. Each iteration should proceed in this fashion.

1.3 Initialization

We need to randomly initialize $\mathbf{x}$ as $\mathbf{x}^{(0)}$. To ensure that everyone gets the same values of $\mathbf{x}$, you must use the following procedure *exactly*.

$$\mathbf{x}_i^{(0)} = \begin{cases} 1 & \text{if } i \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where i is starting from 1.

1.4 Implementation

Note: all the indices are starting from 1.
We have provided you 3 samples of the data file, `RBM_data.txt`. The format is `"m n"` on the first line, m bias parameters for the visible nodes in the next m lines and n bias parameters for the hidden nodes in the subsequent lines. At the very end of the file, there is the $m \times n$ weight matrix $\mathbf{W}$. Feel free to try out different configurations and parameters to get different probability distributions, just make sure it contains all the needed parameter values in the right format.

You will be able to get the values for m and n by calling the following functions in `cs475.RBM2.RBMParameters`:

```
public int numVisibleNodes()   // returns m
public int numHiddenNodes()  // returns n
```

We have also provided functions in `cs475.RBM2.RBMParameters` that give you the bias parameters and the weights,

```
public double visibleBias(int i)  // returns the bias for node x_i
public double hiddenBias(int i)  // returns the bias for node h_i
public double weight(int i, int j) // returns the weight(i,j)
```

1.4.1 What You Need to Implement

We have provided you with the class `cs475.RBM2.RBMEnergy` with one method left blank that you will need to implement:

```
public class RBMEnergy {
     public double computeMarginal(int j) {

     // TODO: Add code here

     }
}
```

This function should return the estimate of $p(\mathbf{h}_j = 1)$.

1.5 How We Will Run Your Code

We will run your code using the data file and the optional `num_samples` argument:

```
java cs475.RBM2.RBMTester -data RBMData.txt -num_samples T
```

Note that we may use data files with different values of m and n, so make sure your code works for any reasonable input.

Your output should just be the results of the print statements in the code given. **Do not print anything else in the version you hand in.**

2 Analytical (20 points)

1. (10 points) Latent Dirichlet Analysis We have discussed the Latent Dirichlet Analysis (LDA) model in class. LDA has been widely used for modeling a document collection by topics. Consider LDA with conditional probabilities:

$$\boldsymbol{\phi}_k \sim \text{Dirichlet}(\boldsymbol{\beta}) \tag{13}$$

$$\boldsymbol{\theta}_i \sim \text{Dirichlet}(\boldsymbol{\alpha}) \tag{14}$$

$$z_{ji} \mid \theta_i \sim \text{Multinomial}(\boldsymbol{\theta}_i) \tag{15}$$

$$d_{ji} \mid z_{ji}, \boldsymbol{\phi}_{z_{ji}} \sim \text{Multinomial}(\boldsymbol{\phi}_{z_{ji}}) \tag{16}$$

where j, i and k are index for words, documents and topics respectively. Hence, $\boldsymbol{d}_i = \{d_{1i}, \ldots, d_{Ni}\}$ indicates the words inside document i. We also know that:

$$P(\boldsymbol{d} \mid \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{\boldsymbol{z}} P(\boldsymbol{d} \mid \boldsymbol{z}, \boldsymbol{\beta}) P(\boldsymbol{z} \mid \boldsymbol{\alpha}) \tag{17}$$

Let N_{wki} be the total number of times we assign word w to topic k, so we have: $N_{wki} = |\{j : d_{ji} = w, z_{ji} = k\}|$.

(a) Write down $P(\boldsymbol{d} \mid \boldsymbol{z}, \boldsymbol{\beta})$ and $P(\boldsymbol{z} \mid \boldsymbol{\alpha})$ using their conditional probabilities in terms of the count variables N defined above.

(b) Explain why exact probabilistic inference on $P(\boldsymbol{z} \mid \boldsymbol{d}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is infeasible.

(c) Suppose we replace the multinomial distribution in (16) with a Gaussian. What type of distribution should we use in (13)? Explain the difference between the resulting model as compared to a Gaussian mixture model.

2. (10 points) Principal Component Analysis Consider a matrix $\boldsymbol{X}$ of size $d \times n$, whose columns $\{\boldsymbol{x}_i\}_{i=1}^n$ are data points and $\boldsymbol{x}_i \in \mathbb{R}^d$. For now, assume that the data points are centered, say $\boldsymbol{X}\mathbf{1} = \mathbf{0}$ where $\mathbf{0}, \mathbf{1} \in \mathbb{R}^n$ are column vectors. Suppose that we project all data points onto a unit vector $w \in \mathbb{R}^d$.

(a) Let $\hat{\boldsymbol{X}}$ be the matrix of the projected data. Express $\hat{\boldsymbol{X}}$ in terms of $\boldsymbol{X}$ and w, and explain how to reconstruct $\boldsymbol{X}$ using $\hat{\boldsymbol{X}}$ and w.

(b) Give a constrained optimization problem over w for maximizing the sample variance. Solve this optimization problem to find the first Principal Component (*Hint*: First find the sample mean and variance after this projection. Then form the Lagrangian, take the derivatives and set it equal to zero).

(c) Given the first Principal Component w_1, setup and solve the constrained optimization program for the second Principal Component. (*Hint*: same as previous part, with one more constraint.)

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `library.zip`. **You must submit source code (.java files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (library.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

4 Questions?

Remember to submit questions about the assignment to the appropriate group on the class discussion board: http://bb.cs475.org.