CS688: Graphical Models - Spring 2014

Assignment 4

Assigned: Thursday, Mar 27th. Due: Thursday, April 10th 2:30pm

General Instructions: Submit a report with the answers to each question at the start of class on the date the assignment is due. You are encouraged to typeset your solutions. To help you get started, the full LaTeXsource of the assignment is included with the assignment materials. For your assignment to be considered "on time", you must upload a zip file containing all of your code to Moodle by the due date. Make sure the code is sufficiently well documented that it's easy to tell what it's doing. You may use any programming language you like. For this assignment, you may **not** use existing code libraries for inference and learning with CRFs or MRFs. If you think you've found a bug with the data or an error in any of the assignment materials, please post a question to the Moodle discussion forum. Make sure to list in your report any outside references you consulted (books, articles, web pages, etc.) and any students you collaborated with.

Academic Honesty Statement: Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

Introduction: In this assignment, you will experiment with Markov Chain Monte Carlo inference and stochastic maximum likelihood learning for restricted Boltzmann Machines (RBMs). We will use unsupervised feature extraction for handwritten digit recognition as an application domain.

Data Sets: We will use binary image data for this assignment taken from the MNIST handwritten digit data set. Examples from this data set are pictured below. Each image has size 28×28.

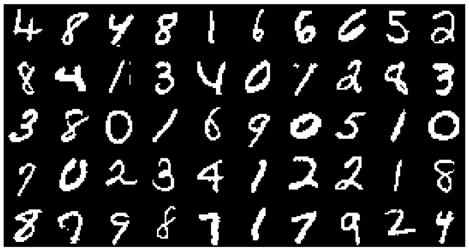

Figure 1: Example data from the MNIST data set.

A model trained on the MNIST data has been provided for use in inference experiments. The MNIST data set contains binary images stored as row vectors. The training image file *MNISTXtrain.txt* contains $60,000$ rows and the test image file *MNISTXtest.txt* contains $10,000$ rows. Each row of the file contains one 28×28 binary image stored as a 784 element vector in row-major format. The MNIST data set also contains labels indicating the class of each image. The class labels go from 1 to 10 and correspond to the digits 0 to 9. The $60,000$ training image labels are stored in the file *MNISTYtrain.txt*. The $10,000$ test labels are stored in the file *MNISTYtest.txt*.

Graphical Model: We will consider modeling this data using a binary restricted Boltzmann machine model, pictured below. The RBM is a Markov network model with two layers of variables $\mathbf{X}$ and $\mathbf{H}$. The $\mathbf{X}$ variables $X_1, ..., X_D$ constitute the first layer of the model and are referred to as the *visible variables* or *visible units*. The second layer of the model consists of the $\mathbf{H}$ variables $H_1, ..., H_K$ referred to as the *hidden variables* or *hidden units*. The Markov network structure underlying the RBM is a bipartite graph where there are edges between the visible units and the hidden units, but not within the hidden units or within the visible units.

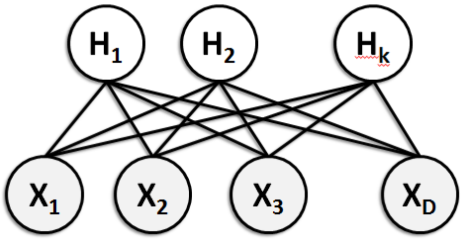

Figure 2: RBM Graphical Model

In the context of learning models for images, the $\mathbf{X}$ variables represent the binary pixels of the image. It is more convenient to consider each image as a length $D = 784$ vector for use with this model, which is how they are stored in the data files. The hidden units are also a binary vector of length K. When K is less than D, we can view the hidden units as a lower-dimensional encoding of the image. We can think of learning the RBM model parameters roughly as learning to extract features that can be used to encode the visible units with minimal loss of information. This encoding can be used for a variety of tasks including classification, which we will explore later in the assignment.

Probabilistic Model: The probabilistic model underlying the RBM is essentially the same as that underlying the Ising model and the CRF model. The main difference is that in the RBM, the hidden variables are never observed. The model can be defined through a joint energy function on the hidden and visible variables as seen below. The model parameters W^P_{dk} encode the pairwise compatibility between x_d and h_k. The model parameters W^B_k and W^C_d provide a bias that either encourages or discourages h_k and x_d from turning on.

$$E_W(\mathbf{x}, \mathbf{h}) = -\sum_{d=1}^{D}\sum_{k=1}^{K} W^P_{dk} x_d h_k - \sum_{k=1}^{K} W^B_k h_k - \sum_{d=1}^{D} W^C_d x_d \tag{1}$$

Using the standard log-linear mapping between energies and probabilities, we obtain the joint distribution over the visible and hidden variables as follows. Like the Ising model and CRF, the partition function of the RBM is a sum over all configurations of all the variables in the model.

$$P_W(\mathbf{X} = \mathbf{x}, \mathbf{H} = \mathbf{h}) = \frac{\exp(-E_W(\mathbf{x}, \mathbf{h}))}{\sum_{\mathbf{x}'}\sum_{\mathbf{h}'} \exp(-E_W(\mathbf{x}', \mathbf{h}'))} \tag{2}$$

Since the hidden variables are never observed, when we learn the model parameters we must marginalize away the hidden variables and optimize the log likelihood of the parameters given the values of the visible variables only. The marginal probability of the visible variables is given below.

$$
\begin{aligned}
P_W(\mathbf{X}=\mathbf{x}) &= \sum_{\mathbf{h}} P_W(\mathbf{X}=\mathbf{x},\mathbf{H}=\mathbf{h}) && (3)\\
&= \frac{\sum_{\mathbf{h}} \exp(-E_W(\mathbf{x},\mathbf{h}))}{\sum_{\mathbf{x}'}\sum_{\mathbf{h}'} \exp(-E_W(\mathbf{x}',\mathbf{h}'))} && (4)
\end{aligned}
$$

Inference Algorithm: One of the main benefits of RBM models relative to other models is that inference for the visible variables given the hidden variables and inference for the hidden variables given the visible variables is very easy due to the structure of the graph. In particular, given all of the $\mathbf{X}$ variables, all of the $\mathbf{H}$ variables are independent of each other since there are no connections within the hidden layer. By symmetry, the $\mathbf{X}$ variables are all independent of each other given the $\mathbf{H}$ variables. This gives us a very simple block-Gibbs algorithm for drawing samples from an RBM. The algorithm alternates between sampling each visible unit independently of the other visible units conditioned on all of the hidden units and sampling each hidden unit independently of the other hidden units given the values for the visible units. The required conditional probabilities are given below.

$$
\begin{aligned}
P_W(X_d = x_d|\mathbf{h}) &= \frac{\exp(W_d^C x_d + \sum_{k=1}^K W_{dk}^P x_d h_k)}{1+\exp(W_d^C + \sum_{k=1}^K W_{dk}^P h_k)} && (5)\\
P_W(H_k = h_k|\mathbf{x}) &= \frac{\exp(W_k^B h_k + \sum_{d=1}^D W_{dk}^P x_d h_k)}{1+\exp(W_k^B + \sum_{d=1}^D W_{dk}^P x_d +)} && (6)
\end{aligned}
$$

The complete block Gibbs sampling algorithm for the RBM is listed below. Recall that to draw a sample z from a distribution $P(Z)$ over a binary random variable Z, we first draw a random number u from a uniform distribution on $[0,1]$. We set $z=1$ if $u<P(Z=1)$ and $z=0$ otherwise. Combined with the conditional distributions listed above, this is all you need to implement the block Gibbs sampler given RBM parameters W^P,W^B and W^C.

Algorithm 1 Block Gibbs Sampler for the RBM model

```
$RBMBlockGibbsSample(W^P, W^B, W^C, S)$
for $k$ from 1 to $K$ Initialize $h_k^0$ to a random binary value
for $s$ from 1 to $S$ do
    for $d$ from 1 to $D$ Sample $x_d^s \sim P_W(X_d = x_d|\mathbf{h}^{s-1})$
    for $k$ from 1 to $K$ Sample $h_k^s \sim P_W(H_k = h_k|\mathbf{x}^s)$
end for
Return $\mathbf{x}^1, ..., \mathbf{x}^S$ and $\mathbf{h}^1, ..., \mathbf{h}^S$
```

Learning Algorithm: To learn the model, we require the gradients of the average log marginal likelihood. We form the average log marginal likelihood below, and list it's gradients with respect to W_{ij}^P and W_j^B and W_i^C

$$\mathcal{L}(W|\mathbf{x}_{1:N}) = \frac{1}{N}\sum_{n=1}^{N}\log(P_W(\mathbf{X}=\mathbf{x}_n)) \tag{7}$$

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^P_{ij}} = \left(\frac{1}{N}\sum_{n=1}^{N} x_{ni}P_W(H_j=1|\mathbf{X}=\mathbf{x}_n)\right) - P_W(X_i=1,H_j=1) \tag{8}$$

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^B_{j}} = \left(\frac{1}{N}\sum_{n=1}^{N} P_W(H_j=1|\mathbf{X}=\mathbf{x}_n)\right) - P_W(H_j=1) \tag{9}$$

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^C_{i}} = \left(\frac{1}{N}\sum_{n=1}^{N} x_{ni}\right) - P_W(X_i=1) \tag{10}$$

The gradient for each parameter has the form of a difference of two terms. We call the first term, which depends on the data, the positive term, and the second term, which is independent of the data, the negative term. Due to the partition function, the average log marginal likelihood itself can not be tractably computed. While the positive term in each of the gradients can be computed exactly using the inference formula for hidden variables, the negative term in the gradients involves a marginal probability that can not be tractably computed and requires approximation.

Given a collection of samples $\tilde{\mathbf{x}}_{1:S}$ drawn from $P_W(\mathbf{X},\mathbf{H})$, we can form a stochastic approximation to the gradients as seen below. Note that we could also use samples for H_j to approximate the gradient, but this would unnecessarily introduce additional noise since we can exactly represent the conditional distribution $P_W(H_j=1|\mathbf{X}=\tilde{\mathbf{x}}_s)$.

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^P_{ij}} \approx \left(\frac{1}{N}\sum_{n=1}^{N} x_{ni}P_W(H_j=1|\mathbf{X}=\mathbf{x}_n)\right) - \left(\frac{1}{S}\sum_{s=1}^{S} \tilde{x}_{si}P_W(H_j=1|\mathbf{X}=\tilde{\mathbf{x}}_s)\right) \tag{11}$$

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^B_{j}} \approx \left(\frac{1}{N}\sum_{n=1}^{N} P_W(H_j=1|\mathbf{X}=\mathbf{x}_n)\right) - \left(\frac{1}{S}\sum_{s=1}^{S} P_W(H_j=1|\mathbf{X}=\tilde{\mathbf{x}}_s)\right) \tag{12}$$

$$\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^C_{i}} \approx \left(\frac{1}{N}\sum_{n=1}^{N} x_{ni}\right) - \left(\frac{1}{S}\sum_{s=1}^{S} \tilde{x}_{si}\right) \tag{13}$$

Given a stochastic approximation to the gradients, we can then take a step in the gradient direction to update the parameters. This requires the selection of a step size α.

$$W^P_{ij} \leftarrow W^P_{ij} + \alpha\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^P_{ij}} \tag{14}$$

$$W^B_{j} \leftarrow W^B_{j} + \alpha\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^B_{j}} \tag{15}$$

$$W^C_{i} \leftarrow W^C_{i} + \alpha\frac{\partial\mathcal{L}(W|\mathbf{x}_{1:N})}{\partial W^C_{i}} \tag{16}$$

We now have all the components in place that are needed to learn the model. We will apply the stochastic maximum likelihood algorithm. We interleave the block Gibbs updates needed to compute the negative contribution to the gradient with the parameter updates. We run C parallel chains during learning. On each iteration of the algorithm, we compute the positive contribution to the gradients, compute the negative contribution to the gradients using the current sample from each of the C chains, update the current samples from each of the C chains using one step of block Gibbs, and then take a step in the gradient direction.

There are two other modifications that help the learning algorithm to converge faster and give better results. The first modification is to break the data up into a set of B batches and to update the model parameters after processing the data in each batch. This is called mini-batch stochastic descent. The second modification is to add regularization to the parameters to prevent over-fitting during training. This requires the introduction of a regularization parameter λ.

Detailed pseudo code for the complete mini-batch stochastic ascent learning algorithm for binary RBMs is given below. We use T to indicate the number of learning iterations, B to indicate the number of batches, N_B to indicate the number of data cases in each batch, and C to indicate the number of chains. We use the vectors $\tilde{\mathbf{x}}_c$ and $\tilde{\mathbf{h}}_c$ to store the states of each of the C chains. We use α to indicate the step size for the gradient descent procedure and λ to indicate the strength of the regularizer.

1. (*20 points*) **Derivations:** Perform the following derivations. Show your work.

(a) Starting from the definition of the joint probability on $\mathbf{X}$ and $\mathbf{H}$ given in Equation 2, derive the conditional probability $P_W(X_d = x_d|\mathbf{h})$ shown in Equation 5. **Hint:** You can do this from first principles, but it will be easier if you use the factor reduction algorithm.

(b) Starting from the definition of the average marginal log likelihood in Equation 7, derive the partial derivative of the average marginal log likelihood with respect to W^P_{ij} shown in Equation 8. **Hint:** This derivation is very similar to both the Ising model and CRF model maximum likelihood parameter estimation derivations.

2. (*20 points*) **Inference, Burn-In and Autocorrelation:** Implement the block Gibbs sampler described in Algorithm 1. In the models directory, you will find the files *MNISTWP.txt*, *MNISTWB.txt* and *MNISTWC.txt*. These files contain the parameters of a binary RBM model trained on the MNIST data set using $K = 100$ hidden units. *MNISTWP.txt* contains the learned W^P parameters as a 100×784 matrix (space-separated). *MNISTWB.txt* contains the learned W^B parameters as a 1×100 vector (space separated). *MNISTWC.txt* contains the learned W^C parameters as a 1×784 vector (space separated). Use your implementation of the Block Gibbs sampler and these trained parameters to answer the following questions.

(a) Run your sampling implementation on the trained MNIST model for 500 iterations starting from a random binary hidden vector. Display every 5th sample of the visible variables as a 28×28 image and include these images in your report (there is Matlab/Octave helper code for assembling the individual images into a single large image in the Code/Images directory). How similar do the samples look compared to the example training data shown in Figure ? Describe any differences. How similar are the samples to each other?

(b) Use your implementation to run 100 separate chains (initialized to different random binary hidden vectors) for 500 iterations each. Display the final sample of the visible variables from each of the 100 chains as a 28×28 image and include the images in your report. How similar are the samples from the different chains?

Algorithm 2 Mini-batch stochastic gradient ascent for the RBM model

```
$RBMLearn(\mathbf{x}_{1:N}, T, B, C, K, \alpha, \lambda)$
#Initialize the Gibbs chains
for $c$ from 1 to $C$ do
  for $k$ from 1 to $K$ do Initialize $\tilde{h}_{ck}$ to a random binary value end
end for
#Initialize the parameters
for $k$ from 1 to $K$ do Initialize $W^B_k \sim \mathcal{N}(0, 0.1^2)$ end
for $d$ from 1 to $D$ do Initialize $W^C_d \sim \mathcal{N}(0, 0.1^2)$ end
for $k$ from 1 to $K$ do
  for $d$ from 1 to $D$ do Initialize $W^P_{dk} \sim \mathcal{N}(0, 0.1^2)$ end
end for
for $t$ from 1 to $T$ do
  for $b$ from 1 to $B$ do
    #Compute positive gradient contribution from each data case in batch b
    $gWC^+ \leftarrow 0, gWB^+ \leftarrow 0, gWP^+ \leftarrow 0$
    for $n$ from $1 + (b-1)N_B$ to $bN_B$ do
      for $d$ from 1 to $D$ do $gWC^+_d \leftarrow gWC^+_d + x_{nd}$ end
      for $k$ from 1 to $K$ do
        $p_k \leftarrow P_W(H_k = 1|\mathbf{X} = \mathbf{x}_n)$
        $gWB^+_k \leftarrow gWB^+_k + p_k$
        for $d$ from 1 to $D$ do $gWP^+_{dk} \leftarrow gWP^+_{dk} + x_{nd}p_k$ end
      end for
    end for
    #Compute negative gradient contribution from each chain and sample states
    $gWC^- \leftarrow 0, gWB^- \leftarrow 0, gWP^- \leftarrow 0$
    for $c$ from 1 to $C$ do
      for $d$ from 1 to $D$ do $\tilde{x}_{cd} \sim P_W(X_d|\mathbf{H} = \tilde{\mathbf{h}}_c)$ end
      for $k$ from 1 to $K$ do $\tilde{h}_{ck} \sim P_W(H_k|\mathbf{X} = \tilde{\mathbf{x}}_c)$ end
      for $d$ from 1 to $D$ do $gWC^-_d \leftarrow gWC^-_d + \tilde{x}_{cd}$ end
      for $k$ from 1 to $K$ do
        $p_k \leftarrow P_W(H_k = 1|\mathbf{X} = \tilde{\mathbf{x}}_c)$
        $gWB^-_k \leftarrow gWB^-_k + p_k$
        for $d$ from 1 to $D$ do $gWP^-_{dk} \leftarrow gWP^-_{dk} + \tilde{x}_{cd}p_k$ end
      end for
    end for
    #Take a gradient step for each parameter in the model
    for $d$ from 1 to $D$ do $W^C_d \leftarrow W^C_d + \alpha\left(\frac{gWC^+_d}{N_B} - \frac{gWC^-_d}{C} - \lambda W^C_d\right)$ end
    for $k$ from 1 to $K$ do
      $W^B_k \leftarrow W^B_k + \alpha\left(\frac{gWB^+_k}{N_B} - \frac{gWB^-_k}{C} - \lambda W^B_k\right)$
      for $d$ from 1 to $D$ do $W^P_{dk} \leftarrow W^P_{dk} + \alpha\left(\frac{gWP^+_{dk}}{N_B} - \frac{gWP^-_{dk}}{C} - \lambda W^P_{dk}\right)$ end
    end for
  end for
end for
Return $W^P, W^B, W^C$
```

(c) For the first 5 of the 100 chains run in part (b), compute the energy of each sample of the visible and hidden variables $(\mathbf{x}_s, \mathbf{h}_s)$ using Equation 1. Plot a single graph with 5 curves showing the energy trace for each chain versus the sampling iteration. Do the energies of the samples from the different chains converge to similar values? What can we conclude about the burn-in and autocorrelation times for the Gibbs sampler applied to this model?

3. (*30 points*) **Learning:** Implement the mini batch stochastic gradient ascent algorithm for RBM learning shown in Algorithm 2 and use your implementation to answer the following questions.

(a) Run your learning algorithm on the MNIST training data. Use $T = 50$ training iterations, $K = 400$ hidden units, $B = 100$ batches of $N_B = 600$ data cases at a time, $C = 100$ Gibbs chains, a step size of $\alpha = 0.1$ and regularization parameter $\lambda = 0.0001$. When training is complete, display the final sample from each of the 100 Gibbs chains as a 28×28 image. Include the image of the samples in your report. How similar do the samples look compared to the example MNIST training images shown in Figure 1? How do they compare to the samples produced by the $K = 100$ model used in Question 2?.

(b) The W^P parameters associated with each hidden unit k also form a 784-long vector which can be visualized as a 28×28 image. This image can be interpreted as the “receptive field" of the hidden unit. It indicates how much the different pixels in the input image contribute to the probability of activating the hidden unit. Display the receptive field images for each of the 400 hidden units. Comment on any structure (or lack of structure) you see in the receptive fields.

4. (*30 points*) **Feature Extraction and Classification:** Using your model trained on MNIST, convert each data case $\mathbf{x}_n$ into the lower dimensional vector $\mathbf{z}_n$ given by $\mathbf{z}_{nk} = P_W(H_k = 1|\mathbf{X} = \mathbf{x}_n)$. We will use these lower dimensional vectors as feature vectors for digit classification with a multi-class support vector machine classifier (`http://svmlight.joachims.org/svm_multiclass.html`). SVMmulticlass uses a simple, sparse ascii file format called the SVMlight format. The SVMlight format contains one data case per line. The format of line n is: y_n $1 : z_{n1}$ $2 : z_{n2}$ ... $K : z_{nK}$. If a particular feature z_{nk} is zero (or sufficiently close to zero) you can skip over the $k : z_{nk}$ entry in the file (there is helper Matlab/Octave code for writing SVMlight format files in the Code/SVM directory). Use your extracted features and the SVMmulticlass software to answer the following questions.

(a) Write out a classification training file in SVMlight format using the labels for the training digits and the lower-dimensional feature vectors $\mathbf{z}_n$ for the training digits. Also write out a classification test file in SVM-light format using the labels for the test digits and the lower-dimensional feature vectors $\mathbf{z}_n$ for the test digits. Run the SVMmulticlass classification code included in the Code/SVM directory on your training data file and use the trained model to classify the test cases. Report the error rate your model achieves as a percentage. The commands needed to run the SVMmulticlass code for training and testing are:

```
svm_multiclass_learn -c 1000000 -v 2 -e 1 <train_data_file> <model_file>
svm_multiclass_classify -v 2 <test_data_file> <model_file> <predictions>
```

(b) Re-run the svm training experiment from part (a) using the raw pixel values instead of the lower-dimensional representation extracted using the RBM. Report the error rate achieved using the raw pixel values as a percentage. Which features work better for classification?