In this assignment you will train a Boltzmann machine with visible units only (i.e., an Ising model). The model distribution (at temperature = 1) is of the form

\[ P(\mathbf{s}) \propto e^{\sum_{i,j \neq i} T_{ij} s_i s_j + \sum_i b_i s_i} \]  

(1)

The weights \( T_{ij} \) represent the strength of coupling between pairs of units, and the biases \( b_i \) influence (but do not alone determine) the mean activity of each unit. In the unclamped state the probability of setting any given unit to the +1 state conditioned on the other units is

\[ P(s_i = 1 | \{s_{\bar{i}}\}) = \sigma(\sum_{j \neq i} T_{ij} s_j + b_i) \]  

(2)

The function `draw.m` uses this formula to draw samples by Gibbs sampling (called by `sample.m`). In addition to the learning rule for \( T_{ij} \) discussed in class, you will need implement the learning rule for the biases, \( b_i \):

\[ \Delta b_i \propto \langle s_i \rangle - \langle s_i \rangle_{P(\mathbf{s})} \]  

(3)

where as before the first term averages under the data distribution (training samples), and the second term averages under the model distribution.

1. **Training on test data with ground truth.** First we will train on a test dataset where the solution is known. To do this, create a reference model with symmetric weight matrix \( T_r \) and biases \( b_r \). You may want to choose \( T_r \) and \( b_r \) to have a specific structure, or you can just use random numbers, e.g.,

   \begin{verbatim}
   Tr=randn(N);
   Tr=(Tr+Tr')/2;   % makes matrix symmetric
   br=randn(N,1);
   \end{verbatim}

   I would recommend using something like N=10. Then, generate a data matrix with 100,000 samples from this model using the function `sample.m`:

   \begin{verbatim}
   X=sample(Tr,br,100000,10);
   \end{verbatim}

   (The last argument sets the number of initial Gibbs sweeps before taking samples.)

   Now train a Boltzmann machine on this data to see if you can recover the model it was generated from. Compare the learned weights \( T \) and \( b \) to those used to generate the data, \( T_r \) and \( b_r \). You should also verify that the means and covariances of samples generated from your learned model match those of the data. The script `boltz.m` has most of what you will need. You will need to fill in the parts for the learning rule and calculating the averages that go into the learning rule.

2. **Data with higher-order structure.** Here you will train on data with higher-order (i.e., more than pairwise) correlations. These data are 3x3 pixel image patches extracted from a larger image of scribbled lines, as shown in figure 1.

   a) Load the image `scribble.mat` and create a data matrix of 100,000 3 × 3 pixel patches extracted from this image using...
X=extract.patches(im,3,100000);
Then calculate the probability distribution over each of the 512 ($2^9$) possible patterns
using
P=prob(X)
and sort the distribution using
[Ps ind] = sort(P,'descend');
If you plot this in log-log coordinates you should see that the probability of these
patterns falls off roughly according to a power law (why this is so is an interesting
question in itself). You can also show the patches rank ordered according to this
distribution using
show.patches(ind)

b) Now try to train a Boltzmann machine to emulate this distribution using your
program from above. Once the model has been trained, verify that the means and
pairwise correlations of patterns generated from the model match those of the data.
You can also show the patches generated from the model, rank-ordered to probability,
using the same procedure as above (now using the matrix of generated patterns S
instead). You will see that they do not match those of the data very well—i.e., it
does not generate lines with high probability, because lines require higher-order (more
than pairwise) statistics to characterize.

c) (optional) Now modify the script boltz.m to allow for hidden units. You may
find it helpful to establish two sets of indices - one for the visible units and one for
the hidden units. In the clamped phase you will need to sample from the posterior
distribution $P(s^h|s^v)$, so it will be a bit more time consuming. Note that you can use
the function sample to do this, since we have

$$ P(s^h|s^v) \propto e^{\sum_{i,j\neq i} T_{ij}^h s_i^h s_j^h + 2\sum_{i,j} T_{ij}^{hv} s_i^h s_j^v + \sum_i b_i^h s_i^h} = e^{\sum_{i,j\neq i} T_{ij}^h s_i^h s_j^h + \sum_i \tilde{b}_i^h s_i^h} $$ (4)

where $T^{hh}$ denotes the connections among the hidden units, $T^{hv}$ denotes the connections between the hidden units and visible units, and $\tilde{b}_i^h = 2 \sum_j T_{ij}^{hv} s_j^v + b_i^h$. Note the
equivalence between equation 5 and equation 1 above. Thus, you can sample from the posterior just as you did for the prior in part (b) above by substituting $T^{hh}$ and $\tilde{b}^h$ as the arguments for the weight matrix and bias when you call `sample`. Once the model has converged, look at the patches that are generated to see if their distribution more closely matches that of the data.