Melissa Yu
melissayu@college.harvard.edu
CS281-F17

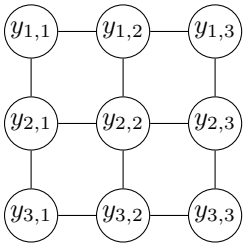
Assignment #4

Due: Monday 5:00pm, November 13, 2017

Collaborators: Alex Lin

Graphical Models for Denoising

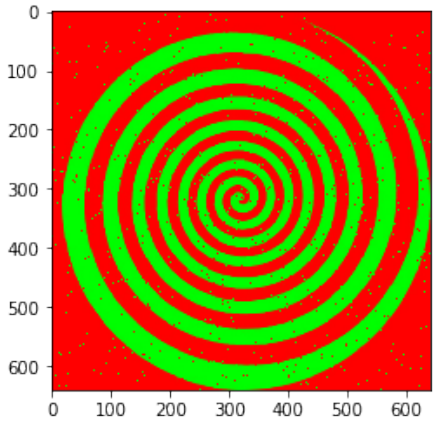
We have seen several variants of the grid-structured Ising model where we have a binary-valued variable at each position of a grid. Here we consider the grid Potts model which has the same graphical model structure, but instead with multiple labels K at each node of the undirected graph $y_{i,j} \in \{1, \ldots, K\}$.



In particular consider a conditional Potts model for image denoising. The input x will consist of a picture with pixels x_{ij}, where each pixel is one of K different colors and has been perturbed by random noise. Each random variable y_{ij} represents the color we think each pixel should take after denoising. Unary potentials represent the prior belief for that pixel based on its observed value. Neighbor potentials enforce smoothness of the labeling. Specifically, $\theta(y_{ij} = k) = 10 * \delta(x_{ij} = k)$, and for all neighboring pixels $n \in \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$,

$$\theta(y_{i,j}, y_n) = \begin{cases} 10 & y_{i,j} = y_n \\ 2 & |y_{i,j} - y_n| = 1 \\ 0 & o.w. \end{cases}$$

This is obviously a simplified and discretized view of the true denoising problem, but it gives us a reasonable place to start. As an example consider the problem with $K = 2$ and noise over the image of a spiral.

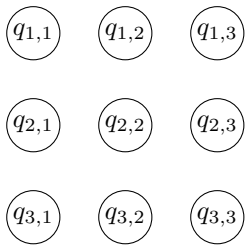


[Note: for the example problems we will show k=1 as red, k=2 as green, and k=3 as blue. We will represent this as the last dimension of the image in a one-hot encoding, so $x[i, j, 0] = 1$ for red, $x[i, j, 1] = 1$ for green, and $x[i, j, 2] = 1$ for blue. Here red is "close" to green and green is close to blue, but red is not close to blue. This is not supposed to be physically true, just part of the problem.]

Problem 1 (Variational Inference for Denoising, 30pts)

For the problems below we have provided a set of example images in the form of numpy arrays including a small sample, a flag, a bullseye, and the large spiral above.

1. First as a sanity check, consider the 3x3 image small with $K = 2$. Compute using brute-force the true posterior marginal probability $p(y_{i,j}|x)$ of any node.
2. Now consider a variational-inference based approach to this problem. Using mean-field factorization, with $q(y)$ factored to each node of the graph, derive local mean field updates.



3. Implement these mean-field updates with a synchronous schedule in PyTorch/numpy. (This means that all parameters are updated with expectations from the previous time step.). Run for 30 epochs starting with q set to a uniform distribution. Graph the results on the small images and compare to the brute-force approach. Compare the variational values to the exact marginals for the small example. Note: running on the spiral example will likely require a fast/matrix implementation.
4. Implement Loopy Belief Propagation with a synchronous or non-synchronous schedule in PyTorch/Numpy following the algorithm given in Murphy (Section 22.2.2). Run for 30 epochs using the starting conditions in in Algorithm 22.1. Compare to the mean field approach.
5. (Optional) Write out the Integer Linear Programming formulation for the maximum assignment problem. What is the advantage of mean field compared to the ILP approach?
6. (Optional) Install the PuLP toolkit in python. Implement the ILP formulation for this problem. Compare your solution for the smaller images.

1. See code solution. The posterior marginal probability is given by

```
[[[  1.0000e+00   5.7991e-12   9.3612e-14]
[  1.0000e+00   1.9552e-15   4.2487e-18]
[  1.0000e+00   5.7991e-12   9.3612e-14]]

[[  1.0000e+00   1.5193e-08   9.3349e-14]
[  1.0000e+00   6.5694e-12   4.2410e-18]
[  1.0000e+00   1.5193e-08   9.3349e-14]]

[[  2.5238e-03   9.9748e-01   2.0611e-09]
[  2.4846e-03   9.9752e-01   7.4111e-13]
[  2.5238e-03   9.9748e-01   2.0611e-09]]]
```

2. We have the target distribution

$$\log p(y \mid x) = \sum_{s \sim t} \theta(y_s, y_t) + \sum_t \theta_t(y_t) + const$$

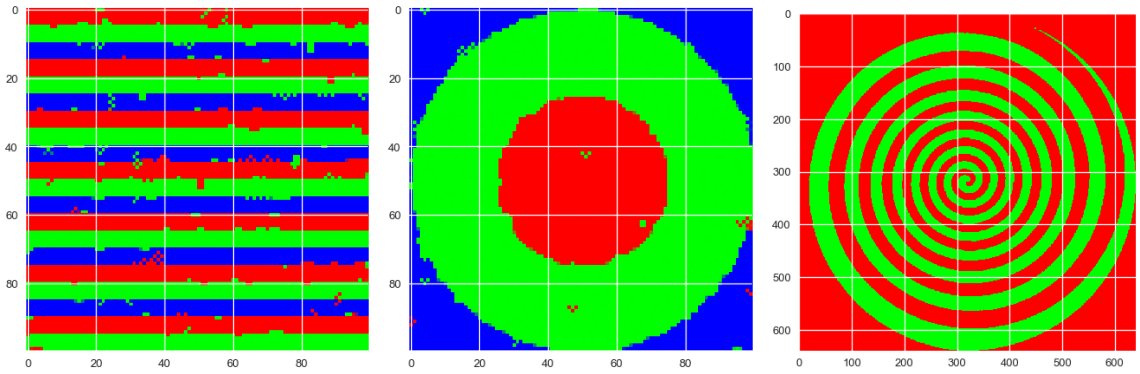


Figure 1: Denoised images for Mean Field

Dropping all terms without y_i and taking expectations w.r.t. $y_j : j \neq i$, the update is

$$\log q_i(y_i) = \theta_i(y_i) + \sum_{n \in ngh(i)} \sum_{y_n=1}^{K} \theta(y_i, y_n) q_n(y_n) + const$$

3. See code solution. See figure 1 for results on the images. On the small example, we obtain the following variational values:

```
[[[  1.0000e+00   5.1091e-12   9.3576e-14]
[  1.0000e+00   1.7139e-15   4.2484e-18]
[  1.0000e+00   5.1091e-12   9.3576e-14]]

[[  1.0000e+00   1.5219e-08   9.3534e-14]
[  1.0000e+00   5.1091e-12   4.2484e-18]
[  1.0000e+00   1.5219e-08   9.3534e-14]]

[[  4.5398e-05   9.9995e-01   2.0611e-09]
[  1.5252e-08   1.0000e+00   6.9182e-13]
[  4.5398e-05   9.9995e-01   2.0611e-09]]]
```

   The variational values are almost exactly the same as the exact marginals, with the largest difference in value (on the order of 10^{-3}) occurring at (2, 0) and (2, 1).

4. See code solution. See figure 2 for results on the images. The results for LBP appear to be better than those for MF; region boundaries are much less pixelated and fewer anomalies remain. This makes sense because (1) checkerboard patterns often occur when MF is used without damping and (2) LBP is generally more accurate than MF since it optimizes over both node and edge potentials, unlike MF, which only optimizes over node potentials.

5. We have the Integer LP:

$$\begin{aligned} \max_{y} \quad & \sum_{s \sim t} \theta(y_s, y_t) + \sum_{t} \theta_t(y_t) \\ \text{subject to} \quad & y_t \in \{1, \ldots, K\} \ \forall t \end{aligned}$$

   Unfortunately, the marginal polytope of y in general has a number of facets that is exponential in the number of nodes, making the ILP difficult to solve without relaxations.

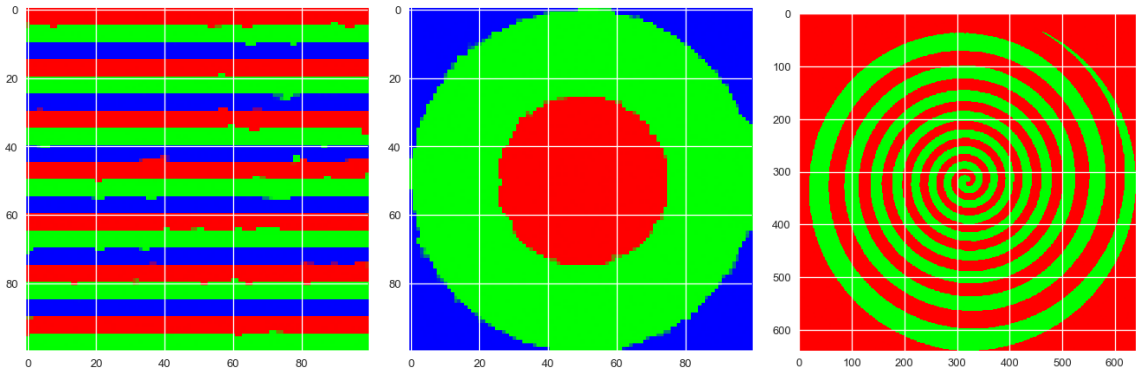


Figure 2: Denoised images for LBP

Modeling users and jokes with a Bayesian latent bilinear model

The next two questions will develop Bayesian inference methods for the simplest version of the latent bilinear model you used to model jokes ratings in HW3. The data set we'll use is the same as in HW3, a modified and preprocessed variant of the Jester data set. However, to make things easier (and to make being Bayesian more worthwhile) **we'll only use subsampling to 10% of the training data**. The other ratings will form your test set.

The model

The model is the same as in HW3, but with Gaussian priors on the latent parameter matrices U and V. Let $r_{i,j} \in \{1, 2, 3, 4, 5\}$ be the rating of user i on joke j. A latent linear model introduces a vector $u_i \in \mathbb{R}^K$ for each user and a vector $v_j \in \mathbb{R}^K$ for each joke. Then, each rating is modeled as a noisy version of the appropriate inner product. Specifically,

$$r_{i,j} \sim \mathcal{N}(u_i^T v_j, \sigma_\epsilon^2).$$

Fix σ_ϵ^2 to be 1.0, and start with $K = 2$. We put independent Gaussian priors on each element of U and V:

$$U_{i,k} \sim \mathcal{N}(0, \sigma_U^2 = 5)$$

$$V_{i,k} \sim \mathcal{N}(0, \sigma_V^2 = 5)$$

Problem 2 (Stochastic Variational Inference, 30pts)

Recall that variational inference optimizes a lower bound on the log marginal likelihood (integrating out parameters θ), like so:

$$\log p(x) = \log \int p(x,\theta)d\theta = \log \int p(x|\theta)p(\theta)d\theta \tag{1}$$

$$= \log \int \frac{q_\lambda(\theta)}{q_\lambda(\theta)} p(x|\theta)p(\theta)d\theta = \log \mathbb{E}_{q_\lambda} \frac{1}{q(\theta)} p(x|\theta)p(\theta)d\theta \tag{2}$$

$$\geq \mathbb{E}_{q_\lambda} \log \left[\frac{1}{q_\lambda(\theta)} p(x|\theta)p(\theta) \right] = \underbrace{-\mathbb{E}_{q_\lambda} \log q_\lambda(\theta)}_{\text{entropy}} + \underbrace{\mathbb{E}_{q_\lambda} \log p(\theta)}_{\text{prior}} + \underbrace{\mathbb{E}_{q_\lambda} \log p(x|\theta)}_{\text{likelihood}} = \mathcal{L}(\lambda) \tag{3}$$

In this case, $\theta = U, V$ and $x = R$:

$$\mathcal{L}(\lambda) = -\mathbb{E}_{q_\lambda} \log q_\lambda(U,V) + \mathbb{E}_{q_\lambda} \log p(U,V) + \sum_{n=1}^{N} \mathbb{E}_{q_\lambda} \log p(r_n|U,V) \tag{4}$$

This is a general formula that works for many different priors, likelihoods and variational approximations. Here we will keep things simple and choose $q(U,V)$ to be a Gaussian factorized over every single entry of each matrix for U and V, e.g. the same form as the prior. Thus our variational parameters will consist of a mean and variance for each entry in U and V: $\lambda_{ik}^{(\mu U)}$, $\lambda_{ik}^{(\sigma^2 U)}$, $\lambda_{jk}^{(\mu V)}$, and $\lambda_{jk}^{(\sigma^2 V)}$.

1. Derive the expression for the KL divergence between two univariate Gaussians.

2. Exploiting the conditional independence of the model, we can write the variational objective (which we want to maximize) as:

$$\mathcal{L}(\lambda) = -KL(q_\lambda(U) \mid\mid p(U)) - KL(q_\lambda(V) \mid\mid p(V)) + \sum_{n=1}^{N} \mathbb{E}_{q_\lambda} \log p(r_n|U,V)$$

   Simplify the first two terms of this model to get a closed form expression.

3. The third term is the likelihood of the data under an expectation wrt the variational parameters. Assume that we approximate this term using a single sample of rows $\tilde{u}_i$ and $\tilde{v}_j$ for each rating $r_{i,j}$. Write out the full objective with this approximation for the last term.

4. Unfortunately this is difficult to optimize, since the sampled variables depend on the variational parameters λ. An alternative method, known as *reparameterization*, replaces expectation of the form $\mathbb{E}_{X\sim\mathcal{N}(\mu,\sigma)}[f(X)]$, in terms of $\mathbb{E}_{Z\sim\mathcal{N}(0,1)}[f(Z\sigma+\mu)]$. Rewrite the objective in this form using sampled dummy variables $\tilde{z}$ (and no $\tilde{u}_i$ or $\tilde{v}_j$).

5. Using PyTorch, set up this model using `nn.Embedding` for the variational parameters. For numerical stability, store the log of the variance in the embedding table, and also initialize this table with very low values, e.g. `logvar.weight.data = -1000`. For $K = 2$, optimize the variational parameters for 10 epochs over the sampled data. Use Adam with learning rate 0.001.

   Plot the training and test-set log-likelihood as a function of the number of epochs, as well as the marginal likelihood lower bound. That is to say: at the end of each epoch, evaluate the log of the average predictive probability of all ratings in the training and test sets using 100 samples from q(U,V). The lower bound is the sum of entropy, prior and likelihood terms, while the training-set and test-set likelihoods only use the likelihood term.

6. Fit your variational model for $K = 1$ to $K = 10$, and plot the training-set log-likelihood, test-set log-likelihood, and lower bound for each value of K. How do the shapes of these curves differ?

1. Let $p(x) = \mathcal{N}(x \,|\, \mu_1, \sigma_1^2)$ and $q(x) = \mathcal{N}(x \,|\, \mu_2, \sigma_2^2)$. We have the following:

$$\begin{aligned} KL(p||q) &= \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \\ &= -\frac{1}{2}(1 + \log 2\pi\sigma_1^2) + \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} \\ &= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \end{aligned}$$

2. We use the priors described in the problem specs. Noting that the KL-divergence of products decomposes into the sums of KL-divergences of its parts, we can write

$$\begin{aligned} & KL(q_\lambda(U) \,||\, p(U)) + KL(q_\lambda(V) \,||\, p(V)) \\ =& \sum_{ik} KL(q_\lambda(U_{ik}) \,||\, p(U_{ik})) + \sum_{jk} KL(q_\lambda(V_{jk}) \,||\, p(V_{jk})) \\ =& \left[\sum_{ik} \log\frac{\sigma_U}{\lambda_{ik}^{(\sigma U)}} + \frac{(\lambda_{ik}^{(\sigma U)})^2 + (\lambda_{ik}^{(\mu U)})^2}{2\sigma_U^2} - \frac{1}{2}\right] + \left[\sum_{jk} \log\frac{\sigma_V}{\lambda_{jk}^{(\sigma V)}} + \frac{(\lambda_{jk}^{(\sigma V)})^2 + (\lambda_{jk}^{(\mu V)})^2}{2\sigma_V^2} - \frac{1}{2}\right] \\ =& \left[\sum_{ik} \log\frac{\sigma_U}{\lambda_{ik}^{(\sigma U)}} + \frac{(\lambda_{ik}^{(\sigma U)})^2 + (\lambda_{ik}^{(\mu U)})^2}{2\sigma_U^2}\right] + \left[\sum_{jk} \log\frac{\sigma_V}{\lambda_{jk}^{(\sigma V)}} + \frac{(\lambda_{jk}^{(\sigma V)})^2 + (\lambda_{jk}^{(\mu V)})^2}{2\sigma_V^2}\right] + const \end{aligned}$$

3. We approximate the last term by:

$$\mathbb{E}_{q_\lambda} \log p(r_n \,|\, U, V) \approx \log \mathcal{N}(r_n \,|\, \tilde{u_i}^\top \tilde{v_j}, \sigma_\epsilon^2),$$

where $\tilde{u_{ik}}$ and $\tilde{v_{jk}}$ are sampled so that each $\tilde{u_{ik}} \sim \mathcal{N}(\lambda_{ik}^{(\mu U)}, \lambda_{ik}^{(\sigma^2 U)})$ and each $\tilde{v_{jk}} \sim \mathcal{N}(\lambda_{jk}^{(\mu V)}, \lambda_{jk}^{(\sigma^2 V)})$.
Our objective is now

$$\begin{aligned} \mathcal{L}(\lambda) = &- \left[\sum_{ik} \log\frac{\sigma_U}{\lambda_{ik}^{(\sigma U)}} + \frac{(\lambda_{ik}^{(\sigma U)})^2 + (\lambda_{ik}^{(\mu U)})^2}{2\sigma_U^2}\right] - \left[\sum_{jk} \log\frac{\sigma_V}{\lambda_{jk}^{(\sigma V)}} + \frac{(\lambda_{jk}^{(\sigma V)})^2 + (\lambda_{jk}^{(\mu V)})^2}{2\sigma_V^2}\right] \\ &- \sum_{n=1}^{N} \frac{(r_n - \tilde{u_i}^\top \tilde{v_j})^2}{2\sigma_\epsilon^2} + const \end{aligned}$$

4. We rewrite the last term as:

$$\mathbb{E}_{q_\lambda} \log p(r_n \,|\, U, V) \approx \log \mathcal{N}(r_n \,|\, (\tilde{z}_U \odot \lambda_i^{(\sigma U)} + \lambda_i^{(\mu U)})^\top (\tilde{z}_V \odot \lambda_j^{(\sigma V)} + \lambda_j^{(\mu V)}), \sigma_\epsilon^2),$$

where $\tilde{z}_U$ and $\tilde{z}_V$ are sampled from $\mathcal{N}(0, I_K)$.
Our objective is now

$$\begin{aligned} \mathcal{L}(\lambda) = &- \left[\sum_{ik} \log\frac{\sigma_U}{\lambda_{ik}^{(\sigma U)}} + \frac{(\lambda_{ik}^{(\sigma U)})^2 + (\lambda_{ik}^{(\mu U)})^2}{2\sigma_U^2}\right] - \left[\sum_{jk} \log\frac{\sigma_V}{\lambda_{jk}^{(\sigma V)}} + \frac{(\lambda_{jk}^{(\sigma V)})^2 + (\lambda_{jk}^{(\mu V)})^2}{2\sigma_V^2}\right] \\ &- \sum_{n=1}^{N} \frac{\left(r_n - (\tilde{z}_U \odot \lambda_i^{(\sigma U)} + \lambda_i^{(\mu U)})^\top (\tilde{z}_V \odot \lambda_j^{(\sigma V)} + \lambda_j^{(\mu V)})\right)^2}{2\sigma_\epsilon^2} + const \end{aligned}$$

5. See code solution. See figure 3 for graphs of the results for learning rate 0.04. Note that all values are up to a constant additive factor.

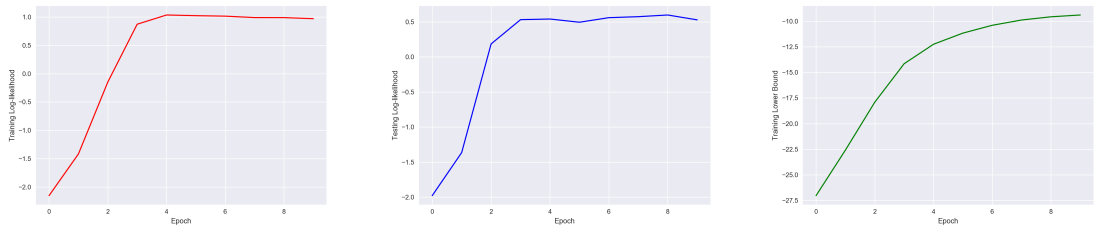


Figure 3: $K = 2$. Training log-likelihood, testing log-likelihood, and training lower bound for 10 epochs.

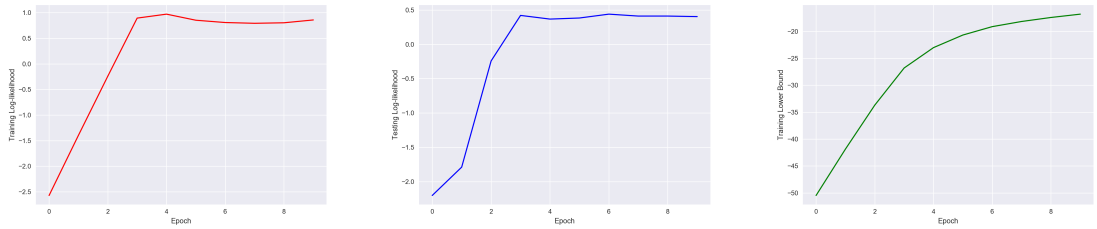


Figure 4: $K = 4$. Training log-likelihood, testing log-likelihood, and training lower bound for 10 epochs.

6. See code solution. Inspecting the graphs in 5, we can see that the log-likelihood curves plateau after around 4 epochs, indicating convergence. All curves are roughly increasing in epochs. Note that while training log-likelihood curves are roughly comparable across K's, higher K values achieve worse performance on the testing set, which makes sense since there are more parameters to infer and we may be overfitting. Also, as K increases, the lower bound decreases because the penalty on the weights imposed by the prior increases.

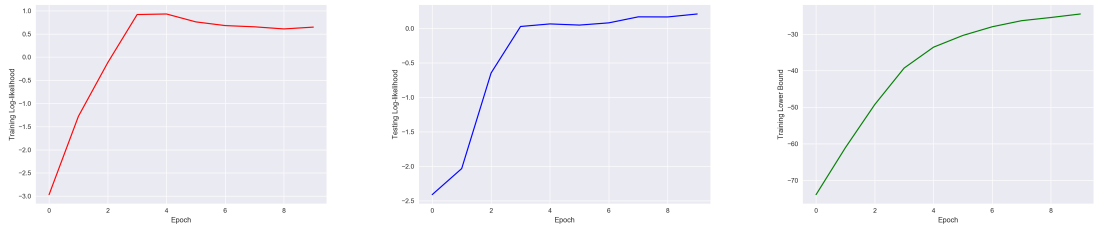


Figure 5: $K = 6$. Training log-likelihood, testing log-likelihood, and training lower bound for 10 epochs.

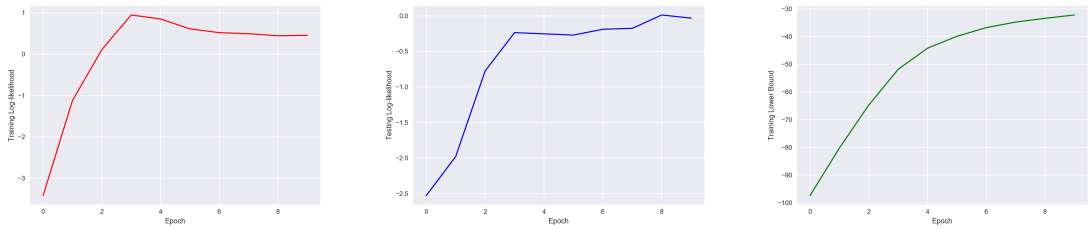


Figure 6: $K = 8$. Training log-likelihood, testing log-likelihood, and training lower bound for 10 epochs.

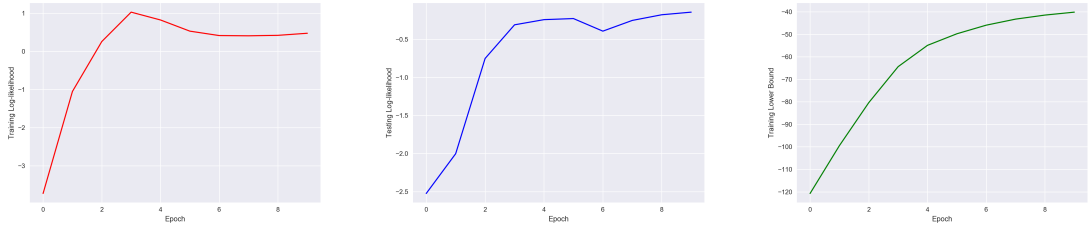


Figure 7: $K = 10$. Training log-likelihood, testing log-likelihood, and training lower bound for 10 epochs.

Problem 3 (Gibbs Sampling, 25pts)

.

In this problem we will consider a different sampling-based approach for estimating the posterior.

1. Write down the conditional equations for U and V. That is to say, write their conditional distributions, conditioned on all the other variables as well as the training data:

$$p(U_i \mid V, R)$$

$$p(V_j \mid U, R)$$

   Because the model is bi-linear, these updates should have fairly simple forms. Here, we mean U_i to mean the latent parameters corresponding to the ith user, and V_j to mean those for the jth joke.

2. A Gibbs sampler is an alternative model for computing the posteriors of intractable models. The method works by repeatedly alternating between drawing samples of U conditioned on V, and then samples of V conditioned on U. (We will derive in greater detail in coming classes).

   Give the pseudocode for running this algorithm using the posterior equations from above.

3. Run the Gibbs sampler for 100 steps (i.e. update both U and V 100 times). Plot the training and test-set log-likelihood as a function of the number of steps through your training set. That is, use all previous samples of U, V to evaluate the predictive probability of all ratings.

4. One reason to be Bayesian is that you don't have to worry about overfitting. Run your Gibbs sampler for $K = 1$ to $K = 10$, and plot the training and test-set log-likelihood for each value of K. How do the shapes of these curves differ from the curves you saw when doing maximum likelihood estimation in HW3?

1. Let $J(i)$ be the set of jokes for which user i has supplied a rating. We have

$$\begin{aligned}
p(U_i \mid V, R) &\propto p(R_i \mid U_i, V_{J(i)}) p(U_i) \\
&= \prod_{j \in J(i)} p(r_{ij} \mid U_i, V_j) \prod_{k=1}^{K} p(u_{ik}) \\
&= \prod_{j \in J(i)} \mathcal{N}(r_{ij} \mid u_i^\top v_j, 1) \prod_{k=1}^{K} \mathcal{N}(u_{ik} \mid 0, 5) \\
&= \exp\left(-\frac{1}{2} \sum_{j \in J(i)} (r_{ij} - u_i^\top v_j)^2 - \frac{1}{10} u_i^\top u_i \right) \\
&= \exp\left(-\frac{1}{2} \sum_{j \in J(i)} (u_i - \frac{r_{ij}}{v_j^\top v_j} v_j)^\top v_j v_j^\top (u_i - \frac{r_{ij}}{v_j^\top v_j} v_j) - \frac{1}{10} u_i^\top u_i \right) \\
&= \mathcal{N}(\mu_{iU}, \sigma_{iU}^2),
\end{aligned}$$

where

$$\mu_{iU} = \sigma_i^2 \sum_{j \in J(i)} (v_j v_j^\top)^{-1} \frac{r_{ij}}{v_j^\top v_j} v_j$$

$$\sigma_{iU}^2 = \frac{1}{5} I_K + \sum_{j \in J(i)} (v_j v_j^\top)^{-1}$$

Similarly, we have the following conditional for V:

$$\begin{aligned}
p(V_j \mid U, R) &\propto p(R_{:,j} \mid U_{I(j)}, V_j) p(V_j) \\
&= \exp\left(-\frac{1}{2} \sum_{i \in I(j)} (r_{ij} - u_i^\top v_j)^2 - \frac{1}{10} v_j^\top v_j \right) \\
&= \mathcal{N}(\mu_{jV}, \sigma_{jV}^2),
\end{aligned}$$

where

$$\mu_{jV} = \sigma_j^2 \sum_{i \in I(j)} (u_i u_i^\top)^{-1} \frac{r_{ij}}{u_i^\top u_i} u_i$$

$$\sigma_{jV}^2 = \frac{1}{5} I_K + \sum_{i \in I(j)} (u_i u_i^\top)^{-1}$$

2. Initialize the values of U_i and V_j. While not converged, for each variable whose posterior we wish to infer, we sample the value of that variable according to its conditional posterior distribution, given the most recent values of all other variables. In this problem, we would sample the variables $\{U_1, \ldots, U_N, V_1, \ldots, V_J\}$ in that order every epoch.