CS 688: Graphical Models - Spring 2016

Assignment 2: Part B

Assigned: Friday, Feb 26. Due: Monday, March 7, start of lecture

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.

Question 1. (*30 points*) **CRF Training**: Implement the average log conditional likelihood objective function and its gradient function (the computation of the partial derivatives of the objective function with respect to each parameter), as derived in Part A of the assignment. Use your implementation of the sum-product message passing algorithm from Part A as a subroutine to make your objective and gradient function implementations computationally tractable. Implement the learning algorithm for CRFs by using a numerical optimizer to **maximize** the log conditional likelihood function.

Train the CRF model on the training data. Evaluate the prediction error on the complete test set. As in Part A, predict the character with the highest marginal probability for each position of each test word. Report the error rate averaged over all predicted characters in all test words for each model.

Question 2. (*10 points*) The previous question asked for what is sometimes called the "minimum Bayes risk decoding", where for a single chain $y_1..y_T$, the prediction is the vector of the marginal maximum at each position,

$$\hat{y} = [\arg\max_k P(y_t = k \mid x)]_{t=1..T}$$

An alternative approach is the "Viterbi decoding" which predicts a single chain that jointly maximizes the posterior[1]

$$\hat{y} = \arg\max_{y_1..y_T} P(y_1..y_T \mid x)$$

[1]Technically, we should say the "predicted probability," not "posterior" since in a CRF this doesn't come from Bayes rule. But "posterior" is just... convenient terminology?

We haven't covered how to calculate Viterbi; it can be done with a variant of belief propagation (the "max-product" algorithm). But, assuming it is feasible to compute:

2.1 (*5 points*): *Give a reason you might prefer to use MBR over Viterbi.*

2.2 (*5 points*): *Give a reason you might prefer to use Viterbi over MBR.*

Question 3. (*30 points*) **Analyzing Transition Parameters**: Using the model trained over all 400 training instances, visualize the transition parameters.

3.1 (*15 points*): *Display the 10×10 transition parameters array W^T as a gray-scale image. Include a color bar. Label the rows and columns of the images using the character labels. Please be clear which refers to the left versus right character. We provide helper function (`plot_helper`) for visualization.*[2]

3.2 (*15 points*): *Visualization is somewhat easier if vowels and consonants occur together in the order "EAIOTNSHRD". Permute the 10×10 transition parameter array W^T to a new array $W^{T'}$ with this order, and display $W^{T'}$ as a gray-scale image. Which transitions have the largest parameter values? Why might this be?*

Question 4. (*30 points*) **Stochastic Gradient Descent for CRFs**: Train your CRFs with a vanilla version[3] of SGD. NOTE: g here is for the gradient of the *negative* loglikelihood. If you are are using the positive loglik, you'll add, not subtract it, from the current parameter.

Algorithm 1 Stochastic maximum likelihood for learning CRFs

```
**Initialize** weights $\theta = \vec{0}$
**Set** step size $\eta = 10^{-4}$, batch size $B = 1$, counter $k = 0$
**for** each epoch **do**
  randomly permute the training data
  **for** each minibatch of size $B$ from the training data **do**
    **for** each training instance $i$ in minibatch **do**
      compute gradient $g_i$
    **end for**
    $g_k = \frac{1}{B} \sum_{i \in B} g_i$
    $\theta_{k+1} = \theta_k - \eta g_k$     *(Subtract gradient of negative loglikelihood.)*
    $k = k + 1$
  **end for**
**end for**
```

[2]In Matlab, an array can displayed as an image using `imagesc`. Use `colormap gray` to set the colormap and `colorbar` to display the colorbar. In python you can use the `matplotlib` module's `imshow` function with the option `interpolation='nearest'` to display an array as an image. There are also functions to set the colormap and display the color bar. See the matplotlib documentation for more details.

[3]refer to Murphy 19.5.5 for a general version for MRFs.

4.1 (*30 points*): *Plug in your gradients computed from Assignment 2a into the vanilla SGD algorithm. At the end of each epoch, record your prediction error on the complete test set. Use the batch size* $B = 1$ *and a fixed step size* $\eta = 10^{-4}$*, and run this algorithm for 400 epochs. Summarize your results in a line graph showing prediction error versus training epoch. Make sure to label the axes of your plot.*

4.2 (*0 points*): *There is nothing to turn in for this question. This is just for fun. Tweak the step size, number of epochs, batch size, and etc. Can you get a better result?*

Implementation note: mathematically, the pseudocode talks about many different vectors. But computationally speaking, it is more efficient to implement sums with in-place updating when possible. When implementing for the first time, do whatever is easiest, then to make it fast think about (1) minimizing memory allocations, and (2) minimizing Python for loops (which are somewhat contradictory goals).