TTIC 31210: Advanced Natural Language Processing
Assignment 3: Generative Modeling and Structured Prediction

Kevin Gimpel

Assigned: May 22, 2017
Due: 11:00 pm, June 1, 2017
Submission: email to `kgimpel@ttic.edu`

Submission Instructions
Package your report and code in a single zip file or tarball, name the file with your last name followed by "_hw3", and email the file to `kgimpel@ttic.edu` by 11:00 pm on the due date. For the report, use LaTeX and some standard style files, such as those prepared for ACL or ICLR. Your report should be a maximum of 6 pages.

Collaboration Policy
You are welcome to discuss assignments with others in the course, but solutions and code must be written individually. You may modify code you find online, but you must be sure you understand it!

Lateness Policy
We want you to do the assignments, even if that means turning them in late (whether partially or fully). There will likely be a penalty assessed if assignments are turned in after the due dates, but we will continue to accept late submissions of assignments until the end of the quarter.

Overview

In this assignment, you will do part-of-speech tagging using some of the methods you've learned from the generative modeling and structured prediction segments of the course.

Data

You should use the English Universal Dependencies dataset available here: `https://github.com/UniversalDependencies/UD_English`. Use the following two files:

- `en-ud-train.conllu`: training data; contains 12,543 annotated sentences.
- `en-ud-dev.conllu`: development data; contains 2,002 annotated sentences.

Lines beginning with # are comments and can be skipped. Each other non-blank line corresponds to a single word in a sentence. Blank lines separate sentences.

You only need to worry about the first few fields in each line for this assignment. In particular, the second field in each line is the word, the fourth field in the line is the universal part-of-speech tag for the word, and the fifth field is the fine-grained part-of-speech tag for the word. For this assignment, you should use the 17-tag universal tag set. So you can discard all fields from each non-blank line other than the word (second field) and its universal POS tag (fourth field).

Hidden Markov Models for Tagging

We will use a hidden Markov model (HMM) for part-of-speech tagging. We will use Y_t to denote the random variable corresponding to the label (tag) at position t. We will use X_t to denote the random variable corresponding to the symbol (word) at position t. A typical HMM uses the following conditional independence assumptions:

$$\text{for } k > 1 : Y_t \perp\!\!\!\perp Y_{t-k} \mid Y_{t-1} \tag{1}$$
$$\text{for } k > 1 : Y_t \perp\!\!\!\perp Y_{t+k} \mid Y_{t+1} \tag{2}$$
$$\text{for } k \neq 0 : X_t \perp\!\!\!\perp Y_{t+k} \mid Y_t \tag{3}$$

We will use $Y_{i:j}$ to denote the set of random variables $Y_i, Y_{i+1}, ..., Y_{j-1}, Y_j$. We will use lowercase letters to denote values of random variables.

The probability of a length-T sequence of labels and symbols is:

$$p(X_{1:T} = x_{1:T}, Y_{1:T} = y_{1:T}) = p(\langle EOS \rangle \mid Y_T = y_T) \prod_{t=1}^{T} p(Y_t = y_t \mid Y_{t-1} = y_{t-1}) p(X_t = x_t \mid Y_t = y_t)$$

where $\langle EOS \rangle$ is the end-of-sentence label and where we fix $Y_0 = \langle BOS \rangle$, where $\langle BOS \rangle$ is the start-of-sentence label.

The parameters of the $P(Y_t \mid Y_{t-1})$ distributions are called the **transition probabilities** and the parameters of the $P(X_t \mid Y_t)$ distributions are called the **emission probabilities**.

1. Your Tasks (50 points + up to 10 points extra credit)

1.1 Understanding HMMs (5 points)

Explain why an HMM must include the generation of the "stop" label $\langle EOS \rangle$. Hint: in what way would the model no longer be a probabilistic model over arbitrary-length sequences?

1.2 Supervised Learning for HMMs: Implementation (15 points)

- **(a) (10 points)** Implement a supervised HMM for POS tagging. Use maximum likelihood estimation for learning, i.e., "count and normalize":

  $$P(Y_t = y \mid Y_{t-1} = y') \leftarrow \frac{\text{count}(\langle y', y \rangle)}{\sum_{y''} \text{count}(\langle y', y'' \rangle)} \tag{4}$$
  $$P(X_t = x \mid Y_t = y) \leftarrow \frac{\text{count}(\langle y, x \rangle)}{\sum_{x'} \text{count}(\langle y, x' \rangle)} \tag{5}$$

  where $\text{count}(\langle y', y \rangle)$ is the number of times in the training set that the tag bigram "y' y" was observed, and $\text{count}(\langle y, x \rangle)$ is the number of times in the training set that word x was tagged with tag y.

  Smooth the emission distributions by including an extra special symbol "UNK" designated for unknown words. Assign a count of 1 to the emission of UNK from each tag.[1] When you encounter new words in the development set, convert them to the special UNK symbol. You do not have to

[1] In general, this is a bad way to smooth, but it's ok for the purposes of this assignment.

smooth the transition distributions.

Note: you may want to implement this in the log domain. That is, instead of storing probabilities, store log-probabilities. When you would ordinarily multiply together two probabilities, instead just add their log-probabilities. When you would ordinarily add together two probabilities, instead log-add their log-probabilities. The log-add function can be found in standard libraries or pseudocode can be found online. You probably do not need to do this until you start doing Gibbs sampling though.

Submit your code.

- **(b) (5 points)** After estimating the parameters, print the top 10 most probable words emitted by the adjective tag ("ADJ"), along with their probabilities. Also, print the 5 tags that are most likely to follow the proper noun ("PROPN") tag.

1.3 Gibbs Sampling for Decoding (30 points)

For supervised HMMs, we can use maximum likelihood estimation to estimate the transition and emission parameters of the HMM, as you did above. There is no inference required during learning.

However, to use the model to tag new data, we need to solve the following:

$$\hat{y}_{1:T} = \underset{y_{1:T}}{\operatorname{argmax}} \; p(Y_{1:T} = y_{1:T} \mid X_{1:T} = x_{1:T}) \tag{6}$$

Typically, the Viterbi algorithm is used to exactly solve this test-time inference ("decoding") problem.

Instead of the Viterbi algorithm, you will use Gibbs sampling for decoding. Gibbs sampling defines a way to draw samples from $p(Y_{1:T} = y_{1:T} \mid X_{1:T} = x_{1:T})$ by repeatedly sampling from $p(Y_t = y_t \mid Y_{-t} = y_{-t}, X_{1:T} = x_{1:T})$ where Y_{-t} represents all Y random variables other than Y_t and y_{-t} denotes their values.

- **(a) (5 points)** Show that

  $$p(Y_t = y_t \mid Y_{-t} = y_{-t}, X_{1:T} = x_{1:T}) \propto p(Y_t = y_t \mid Y_{t-1} = y_{t-1})p(X_t = x_t \mid Y_t = y_t)p(Y_{t+1} = y_{t+1} \mid Y_t = y_t)$$

- **(b) (2 points)** Write down the two special cases below:

  $$p(Y_1 = y_1 \mid Y_{-1} = y_{-1}, X_{1:T} = x_{1:T}) \propto \; ?$$

  $$p(Y_T = y_T \mid Y_{-T} = y_{-T}, X_{1:T} = x_{1:T}) \propto \; ?$$

- **(c) (10 points)** Implement Gibbs sampling for your HMM using the above formulas.

  You have to start your Gibbs sampler by initializing the **state** of the sampler. Here, the state consists of values for the Y_t variables. Initialize by sampling each tag uniformly at random from the set of tags (excluding the special start and end tags).

  Let's define an **iteration** of Gibbs sampling as the sampling of each Y_t in the sequence conditioned on all other Y_t values in the current state. So a single Gibbs sampling iteration for a sentence with T words will require T sampling steps, each time sampling from the conditional distribution of one Y_t conditioned on all others. Implement the ability to run the sampler for a given number of iterations. At the end of that number of iterations, evaluate the state by comparing its tags to the ground truth tags and recording the number that match. Run a completely new Gibbs sampler for k iterations on each sentence in the development set. Compute tagging accuracy by accumulating the counts of matching tags and total tags across all sentences and dividing.

During implementation and debugging, you can print the number of sampling steps in which a random variable actually changed its value. You should see this number decrease and then stabilize after a small number of iterations (though it should not actually shrink to zero assuming the posterior has nonzero probability mass on multiple taggings, which it typically will).

Submit your code.

- **(d) (5 points)** Run k iterations of Gibbs sampling for each sentence in the development set, for $k \in \{1, 2, 5, 10, 100, 500, 1000, 2000\}$, then take the final sample and record its counts of correct and total tags in order to compute the tagging accuracy of the entire development set. Report the tagging accuracies for all k values. Note that for each k value, you should run a completely new sampler for each development set sentence.

- **(e) (4 points)** Since we are doing test-time inference, we are interested in getting a setting of the variables Y_t that maximizes the posterior, rather than just generate samples from the posterior. One way to approximate this is to sharpen each conditional distribution in the sampler by raising every probability to the power β and then renormalizing to get a distribution. For $\beta < 1$, this will flatten the distribution, and for $\beta > 1$, this will sharpen the distribution.

  The experiments in (d) used $\beta = 1$. Repeat the experiments from (d) (for all k values) but now vary β. Experiment with $\beta \in \{0.5, 2, 5\}$ and report your tagging accuracies for all k values and all β values.

- **(f) (4 points)** Repeat (e) but this time begin with $\beta = 0.1$ for each sentence and then increase β by adding 0.1 after each sampling iteration for that sentence. For the next sentence's sampler, reset β again to 0.1 and repeat. Report your tagging accuracies for all k values considered above. Experiment with a few other schedules of varying β across iterations to see if you can get better resulting tagging accuracies.

  This practice is often described as "annealing" as it is similar to methods like simulated annealing and deterministic annealing. Starting with $\beta < 1$ helps the sampler to mix faster (in order to converge to generating samples from the true posterior), then increasing β over time causes the sampler to generate samples from the sharpened posterior. As β increases, sampling from the sharpened posterior should get closer and closer to the argmax operation that the Viterbi algorithm performs.

1.4 Extra Credit: Gibbs Sampling for Minimum Bayes Risk Decoding (up to 10 points extra)

The Viterbi algorithm solves the argmax decoding problem, which is often called *maximum a posteriori* or MAP inference. However, there are other decoding criteria that can work better in practice, especially for structured prediction.

One popular generalizing decoding framework is that of minimum Bayes risk (MBR) decoding. For HMMs, MBR decoding has the following form:

$$\hat{y}_{1:T} = \operatorname*{argmin}_{y_{1:T}} \sum_{y'_{1:T}} p(Y_{1:T} = y'_{1:T} \mid X_{1:T} = x_{1:T}) \text{cost}(y_{1:T}, y'_{1:T}) \tag{7}$$

We'll consider two cost functions. First let's define the **0-1 cost**:

$$\text{cost}_{0-1}(y_{1:T}, y'_{1:T}) = \mathbb{I}[y_{1:T} \neq y'_{1:T}]$$

where $\mathbb{I}[f]$ returns 1 if f is true and 0 otherwise. The second is the **Hamming cost**:

$$\text{cost}_{\text{Hamming}}(y_{1:T}, y'_{1:T}) = \sum_{t=1}^{T} \mathbb{I}[y_t \neq y'_t]$$

This cost function counts the number of tags that don't match between the two tag sequences.

- **(a) (3 points)** Show that MBR decoding (solving Eq. 7) reduces to MAP inference (solving Eq. 6) when using 0-1 cost.

- **(b) (7 points)** It can be shown that MBR decoding with Hamming cost reduces to

  $$\text{for all } t,\ \hat{y}_t = \operatorname*{argmax}_{y} p(Y_t = y \mid X_{1:T} = x_{1:T})$$

  You do not need to prove this.
  Note that your Gibbs sampler gives you samples from

  $$p(Y_{1:T} \mid X_{1:T} = x_{1:T})$$

  For MBR decoding, we need to calculate

  $$p(Y_t = y \mid X_{1:T} = x_{1:T}) = \sum_{y_{-t}} p(Y_t = y, Y_{-t} = y_{-t} \mid X_{1:T} = x_{1:T})$$

  We can use our samples from our Gibbs sampler to approximate this summation!

  Use your Gibbs sampler's samples to estimate the posteriors of individual variables Y_t in order to do MBR decoding. The decoding rule for a given position t ends up having a very simple form given a set of samples. Compare tagging accuracies across different numbers of sampling iterations, in particular when using $k \in \{1, 2, 5, 10, 100, 500, 1000, 2000\}$ and fixing $\beta = 1$. Then experiment with different fixed values of β and report how the results vary with different β values.