Assignment 3
CSE 447/547M: Natural Language Processing

University of Washington

Due: February 25, 2019, 11:59pm

In this assignment you will learn more about sequence models, both by implementing models to label sequences and by thinking more deeply about the theory behind them. First, you will use AllenNLP to train and analyze two different models to perform part-of-speech (POS) tagging, where, given a natural language sentence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_\ell \rangle$, the task is to find the most likely sequence of POS tags $\hat{\boldsymbol{y}} = \langle y_1, y_2, \ldots, y_\ell \rangle$. Second, you will adapt the Viterbi algorithm to decode a slightly different version of a HMM from what you saw in class.

Dataset

The data you need to tag is from Twitter, which introduces some added challenges (as discussed in lecture). The tag set $\mathcal{L}$ comes from Liu et al. [2018]; it's considerably smaller than the tag set used in the Penn Treebank (45), which is in turn a reduced tag set from the 87-tag tag set used for the Brown corpus. These are the files which can be obtained online:[1]

- `en-ud-tweet-train.conllu`: data for training (1,639 tweets).
- `en-ud-tweet-dev.conllu`: data to select and tune your models (710 tweets)
- `en-ud-tweet-test.conllu`: data to evaluate performance (1,201 tweets)

These files are in the CoNLL-U format.[2] It will be useful to understand both Universal Dependencies (UD) and the CoNLL-U format.

1 Simple Twitter POS Tagger (25%)

Your first task is to train AllenNLP's `simple_tagger` model[3] to predict POS tags for Twitter data. This model encodes the input sequence with a neural network, mapping the sequence of token embeddings to a new sequence, which is then transformed to the size of the label set. It then decodes that sequence by selecting the most likely label for each position in the sequence. Although the neural network used to score each label for each word depends on the entire input sequence and is shared across all words, each prediction is treated as an independent decision; the label prediction at token i does not depend on the label assigned to previous token $i-1$. In other words, this model finds the most likely label at every step individually, without concern for other labels. In the terminology from lecture, this is a "version 0" sequence model with no need for the Viterbi algorithm.[4]

[1] https://github.com/Oneplus/Tweebank

[2] https://universaldependencies.org/format.html

[3] https://github.com/allenai/allennlp/blob/master/allennlp/models/simple_tagger.py

[4] Somewhat confusingly, AllenNLP uses the label "`seq2seq`" to refer to some of the internals of this model. This label is most commonly used to refer to a neural "version ∞" approach that gets a lot of use in NLP (but that's not what it means in AllenNLP). We haven't discussed that model in lecture; it may come up later in the course.

Deliverables: In the writeup, be sure to fully describe your models and experimental procedure.

1. Implement an AllenNLP `DatasetReader` that can load Tweebank v2 and pass it to the `simple_tagger` AllenNLP `Model`.
2. Create an AllenNLP configuration file that uses your new `DatasetReader` and trains using the `simple_tagger` model.
3. Report the accuracy of your model on the training and development sets across the hyper-parameter values you tested (at most 5).
4. Take the best-performing model from (3) and report its accuracy on the test set.
5. Implement an AllenNLP `Predictor` that outputs a file containing the tweet and the model's predicted POS tags. The output should be in the tab-separated format as the provided `test-gold-labels.tsv`[5], with one instance per line. Run the `Predictor` to get your best model's predictions on the test set.

For ease of grading, register your `DatasetReader` as `twitter_tagger_read` and your `Predictor` as `twitter_tagger_pred`, and both should be in a folder called `twitter_tagger`. In addition to the folder, provide your configuration in `simple-twitter-tagger.jsonnet` and your predicted output in `simple-tagger-output.tsv`.

Hints and Resources: You should *not* be writing a significant amount of code to complete this first task. However, it may take some to time to figure out which pieces of the AllenNLP pipeline need to be changed and in what ways. In addition to the tutorials and resources provided for the previous assignments, here are some more resources that will be useful:

- https://github.com/allenai/allennlp/blob/master/tutorials/getting_started/walk_through_allennlp/training_and_evaluating.md
- https://universaldependencies.org/
- https://allenai.github.io/allennlp-docs/api/allennlp.commands.predict.html
- https://github.com/allenai/allennlp/blob/master/allennlp/predictors/sentence_tagger.py
- https://github.com/allenai/allennlp/blob/master/allennlp/predictors/predictor.py

2 Conditional Random Field (CRF) POS Tagging (25%)

You will now use AllenNLP's `crf_tagger`[6] model to predict POS tags for Twitter data.

CRF stands for "conditional random field" [Lafferty et al., 2001], which refers to a method for *training* the parameters of the scoring function in a sequence model. We won't have time to discuss it in detail, but you can think of it as a generalization of the **log loss**, now adapted for structured models. A CRF aims to maximize:

$$\sum_j \log p(\boldsymbol{y}_j \mid \boldsymbol{x}_j) \tag{1}$$

[5] https://courses.cs.washington.edu/courses/cse447/19wi/assignments/test-gold-labels.tsv

[6] https://github.com/allenai/allennlp/blob/master/allennlp/models/crf_tagger.py

where j ranges over training examples (labeled sequences) and $p(\boldsymbol{y} \mid \boldsymbol{x})$ is defined through

$$\text{softmax}\left(\sum_{i=0}^{\ell} s(\boldsymbol{x}, i, y_i, y_{i+1})\right)$$

(This should be a little surprising; how do we apply a softmax function to $|\mathcal{L}|^\ell$ sequences? The answer is to recognize that we never actually need to enumerate all $|\mathcal{L}|^\ell$ probabilities that come out of the softmax. To calculate the log loss (the normalized probability of the *correct* sequence), we can use a dynamic programming algorithm that exploits the decomposition of the sequence scores, the same way that the Viterbi algorithm does. Amazingly, this same dynamic programming algorithm also helps us backpropagate through the log-loss to update the scoring function's parameters. For this assignment, you don't need to worry about CRFs or the techniques required to train them, because they are already implemented in AllenNLP.)

Deliverables: As in part 1, be sure to fully describe your models and experimental procedure.

1. Using your `DatasetReader` from part 1, create an AllenNLP configuration file that trains using the `crf_tagger` model.
2. Report the accuracy of your model on the training and development sets across the hyper-parameter values you tested (at most 5).
3. Take the best-performing model from (2) and report its accuracy on the test set.
4. Use your `Predictor` from part 1 to get your best model's predictions on the test set.

Provide your configuration in `crf-twitter-tagger.jsonnet` and your model's predictions in `crf-tagger-output.tsv`.

3 Analyzing the Automatic POS Tagging (25%)

Now that you have two different models that label a tweet with its POS tags, you can better understand the strengths and weaknesses of the models through error analysis. Looking at the types of mistakes the models are making and examples of their predictions can help direct future research efforts to improve the model.

To do this, you will compare your results against the "gold" (i.e., human-labeled) results, which are provided for you in the file `test-gold-labels.tsv`[5].

Deliverables:

1. Write a script that takes your `Predictor` output file and the gold labels file and outputs a "confusion matrix" of the model's mistakes.[7] Present the confusion matrices for your best model from parts 1 and 2.
2. Discuss the similarities and differences you see between the two matrices. Where do the models do well? Where are they making mistakes? Present examples to support your claims.

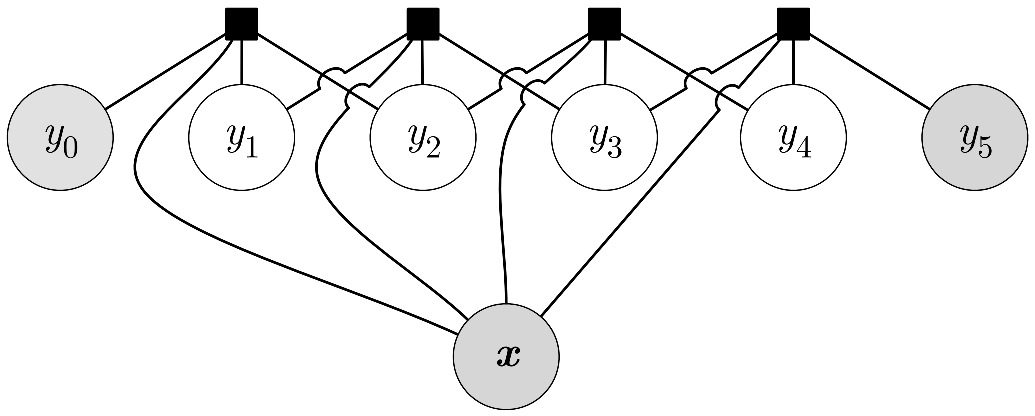
4 Viterbi for Second-Order Sequence Models (25%)

In lecture, we proposed "version 3," a second-order sequence model that defines scores of *triplets* of labels, $s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2})$. The problem to be solved when we decode with this model is:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^\ell} \sum_{i=0}^{\ell-1} s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2}) \tag{2}$$

[7] https://en.wikipedia.org/wiki/Confusion_matrix

To understand why there are ℓ scores in the summation (rather than $\ell + 1$, as in the first-order "version 2" case), consider the factor graph:[8]



The Viterbi algorithm can be adapted for this model. The form of the algorithm is identical to what we saw in lecture. First, prefix scores are computed "left to right," each using a formula with a local max, and along with each prefix score we also store a backpointer that stores the "argmax." Then the backpointers are followed "right to left" to select a label for each word in turn. Here is most of the algorithm:
Input: scores $s(\boldsymbol{x}, i, y'', y', y), \forall i \in \langle 0, \ldots, \ell \rangle, \forall y'' \in \mathcal{L}, \forall y' \in \mathcal{L}, \forall y \in \mathcal{L}$
Output: $\hat{\boldsymbol{y}}$

1. Base case: $\heartsuit_2(y', y) = s(\boldsymbol{x}, 0, \text{START}, y', y)$
   (Note that the subscript in "$\heartsuit_2$" refers to the last position in the triple, here, the word labeled "y," while the "0" argument to s refers to the first position in the triple.)
2. For $i \in \langle 3, \ldots, \ell - 1 \rangle$:
   - $\boxed{\text{Solve for } \heartsuit_i(*, *) \text{ and } b_i(*, *).}$
3. Special case for the end:

$$\heartsuit_\ell(y', y) = s(\boldsymbol{x}, \ell - 1, y', y, \text{STOP}) + \underbrace{\max_{y'' \in \mathcal{L}} s(\boldsymbol{x}, \ell - 2, y'', y', y) + \heartsuit_{\ell-1}(y'', y')}_{b_\ell(y', y) \text{ is the "argmax"}}$$

4. $(\hat{y}_{\ell-1}, \hat{y}_\ell) \leftarrow \operatorname{argmax}_{y', y \in \mathcal{L}^2} \heartsuit_\ell(y', y)$
5. For $i \in \langle \ell - 2, \ldots, 1 \rangle$:
   - $\hat{y}_i \leftarrow b_{i+2}(\hat{y}_{i+1}, \hat{y}_{i+2})$

Deliverables:

1. Give the recurrence for $\heartsuit_i(y', y)$ and for $b_i(y', y)$ (the boxed bit of the algorithm above). We've given you the base case and the special case for the end of the sequence.
2. Analyze the runtime and space requirements of this algorithm as functions of $|\mathcal{L}|$ (the number of labels) and ℓ (the length of the input sequence $\boldsymbol{x}$ and the output sequence $\hat{\boldsymbol{y}}$).

[8] https://en.wikipedia.org/wiki/Factor_graph

Submission Instructions

Submit a single gzipped tarfile (`A3.tgz`) on Canvas.

- **Code**: You will submit your config file and model prediction file for the best models from parts 1 and 2, along with a folder containing your `DatasetReader` and `Predictor`. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade.
- **Report** (use the filename `A3.pdf` and include it in the tarfile): Your writeup should be no longer than four pages, in pdf (one-inch margins, reasonable font sizes, LaTeX-typeset). Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

References

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, 2001.

Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A Smith. Parsing tweets into universal dependencies. In *Proceedings of NAACL*, 2018. URL `https://arxiv.org/pdf/1804.08228.pdf`.