Natural Language Processing
Assignment 3

1. Language Modeling (10 points)

This section involves a pen-and-paper exercise (no programming is involved).

Using maximum likelihood estimation ("count and normalize"), Pat estimated two bigram language models on the same dataset D, one unsmoothed ("Model U") and one smoothed with add-one smoothing ("Model S"). Add-one smoothing simply adds 1 to all counts prior to normalization (subject to constraints on <s> and </s> described below). Both models are defined on the same vocabulary $\mathcal{V}$. So, Model S will assign nonzero probability to any sequence of words from $\mathcal{V}$, while Model U will assign zero probability to word sequences that contain a bigram that is not in D.

Typically, smoothing moves probability mass from observed to unobserved events. So we would generally expect that when Pat uses both models to compute the probability of a sentence $\boldsymbol{x}$ that is drawn from D, Model U will assign higher probability to $\boldsymbol{x}$ than Model S.

However, this is not always the case! **Construct a dataset D of sentences such that Model S will assign higher probability than Model U to some sentence in D.** (Hint: think small.)

Each sentence in D must start with <s> and end with </s>. You should also assume the following constraints on D and on all models estimated from D, whether smoothed or unsmoothed:

- It is illegal for </s> to immediately follow <s>, both in D and in any model estimated from D. So, in Model U and Model S, $P(\texttt{</s>} \mid \texttt{<s>}) = 0$.
- It is illegal for <s> to follow any other symbol. So, in Models U and S, $P(\texttt{<s>} \mid x) = 0$ for all $x \in \mathcal{V}$.

2. Neural Networks for Part-of-Speech Tagging (40 points)

You will implement and experiment with ways of using neural networks for part-of-speech (POS) tagging of English tweets. We provide small annotated train/dev/devtest sets. We also provide word embeddings obtained by training the skip-gram model of `word2vec` on a large corpus on unlabeled English tweets. You can use these word embeddings if you like, or you can use other off-the-shelf word embeddings found online, or you can simply randomly initialize the word embeddings and learn them while training your POS tagger.

You should build a feed-forward neural network to predict the POS tag of a word token given its context. So, the input to your network should be a single token with its context, and the output should be a predicted POS tag.

In class, we talked about how to represent the input $\boldsymbol{x}$ for this problem. The simplest way to start is to use the word embedding for the word being labeled, possibly concatenated with the word embeddings of neighboring words. You can also concatenate additional feature functions; details are below. As your evaluation metric, use tagging accuracy, i.e., the percentage of tokens that were tagged correctly.

For more details on the tag set and annotation, see Gimpel et al. (2011). Documentation and the original file downloads are available from `http://www.cs.cmu.edu/~ark/TweetNLP/`.

Neural Network Toolkits

You are encouraged to use a toolkit for this assignment (though you are welcome to implement the model and backpropagation from scratch if you prefer). Below are some toolkits, but feel free to use any others that you may find:

- PyTorch: `https://pytorch.org/`
- TensorFlow: `https://www.tensorflow.org/`
- DyNet: `http://dynet.io/`

Provided Data

The following data files are provided to you for these experiments. The annotated tweet files use a file format in which each line corresponds to a single word token in a tweet and a blank line separates tweets. Each non-blank line has the format "word[tab]POS", where "POS" is a single character representing the annotated POS tag. For example, "O" is pronoun, "N" is noun, "A" is adjective, etc. See Gimpel et al. (2011) for details on the tag set. When creating instances for training or computing accuracies, each word should be considered as a single instance, though the words in surrounding lines can be used for additional features.

- `tweets-train.txt`: training data (1000 tweets) (TRAIN)
- `tweets-dev.txt`: development data (327 tweets) (DEV)
- `tweets-devtest.txt`: development test data (500 tweets) (DEVTEST)
- `embeddings-twitter.txt`: 50-dimensional skip-gram word embeddings trained on a dataset of 56 million English tweets. Only vectors for the most frequent 30,000 words are provided. The final entry in the file is for the unknown word ("UUUNKKK") and should be used for words that are not among the 30,000 most frequent. The file contains one word per line in the format "word[tab]dimension_1[space]dimension_2[space]...[space]dimension_50".

2.1 A Baseline Neural Network Tagger (30 points)

Train a feed-forward neural network classifier to predict the POS tag of a word in its context. The input should be the word embedding for the center word concatenated with the word embeddings for words in a **context window**. We'll define a context window as the sequence of words containing w words to either side of the center word and including the center word itself, so the context window contains $1 + 2w$ words in total. For now, use $w = 1$, so if the word embeddings have dimensionality d, the total dimensionality of the input is $3d$. For words near the sentence boundaries, pad the sentence with beginning-of-sentence and end-of-sentence characters (<s> and </s>).

functional architecture: The input is the concatenation of word embeddings in the context window, with the word to be tagged in the center. Use a single hidden layer of width 128 with a $\tanh$ nonlinearity. The hidden layer should then be fed to a final affine transformation which will produce scores for all possible POS tags. That is, the output dimensionality will be equal to the number of possible tags. This final transformation, which usually does not use a nonlinearity, is often combined with a softmax function to transform the vector of scores into a probability distribution (in which case the combination of the affine transformation and the softmax is sometimes called a "softmax layer").

learning: Use log loss as the objective function (log loss is often called "cross entropy" when training neural networks). Use SGD or any other optimizer you wish. Toolkits typically have log loss (cross entropy) and many optimizers already implemented. Train on TRAIN, perform early stopping and preliminary testing on DEV, and **report your final tagging accuracy on DEVTEST**. Submit your code.

2.2 Architecture and Feature Engineering (10 points)

Explore the space of possible architectures and features to improve your tagger. Some suggestions are below:

- Add additional features to the input, whether based on looking at the training data, looking at the errors your tagger makes on development data, or simply based on your intuitions about the task. For example, you could add a binary feature that has a value of 1 if the center word contains certain special characters, a feature that returns the number of characters in the center word, features for particular prefixes/suffixes in the center word, etc.
- Vary w: in addition to $w = 1$, try $w = 0$ and $w = 2$ and report an empirical comparison.
- Compare updating the word embeddings during training with keeping them fixed.
- Compare the use of 0, 1, and 2 hidden layers. For each number of hidden layers, try two different layer widths that differ by a factor of 2 (e.g., 256 and 512).
- Keeping the number of layers and layer sizes fixed, experiment with different activation functions for the nonlinearity. Compare identity ($g(a) = a$), $\tanh$, ReLU, and logistic sigmoid.

These suggestions are deliberately open-ended. You do not need to do all of the above, but you do need to do some experimentation and draw conclusions based on the results. Submit your code.

2.3 Extra Credit: RNN Taggers (up to 5 points extra credit)

Implement and experiment with both an LSTM tagger and a bidirectional LSTM tagger. Compare final tagging accuracies on DEVTEST. Submit your code.

References

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., and Smith, N. A. (2011). Part-of-speech tagging for Twitter: annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, Portland, Oregon, USA. Association for Computational Linguistics. [2]