TTIC 31190: Natural Language Processing
Assignment 3: Neural Networks for Part-of-Speech Tagging

Kevin Gimpel
Assigned: Feb. 8, 2016
Due: 11:59 pm, Feb. 29, 2016
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 11:59 pm on Feb. 29, 2016. To limit file size, you should NOT include any data files that we provided to you (we already have them!). Please include an estimate of how many hours you spent on this assignment. This will help us better calibrate assignments in future years. It will not affect your grade.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code must be written individually.

Neural Networks for Part-of-Speech Tagging

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 neural network to predict the POS tag of a word token given its context. So, the input to your neural 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. We also talked about concatenating additional feature functions, such as a binary feature that returns 1 if the word begins with a capital letter, a feature that returns the number of characters in the word, etc.

For more details on the tag set and annotation, please see Gimpel et al. (2011) and/or Owoputi et al. (2013). Also, documentation and the original downloads are available from `http://www.cs.cmu.edu/~ark/TweetNLP/`. As your evaluation metric, use tagging accuracy, i.e., the percentage of tokens that were tagged correctly.

Neural Network Toolkits

You are strongly 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:

- PENNE (Python): `https://bitbucket.org/ndnlp/penne`
- Theano (Python): `http://deeplearning.net/software/theano/`
- TensorFlow (C++): `https://www.tensorflow.org/`
- CNN (C++): `https://github.com/clab/cnn`
- Netlab (MatLab): `http://www.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/`
- Torch: `http://torch.ch/`

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; 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".

1 Required:

The following 2 steps are required:

1. Train a 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: use a single hidden layer of width 128 and use ReLU nonlinearities. While terminology varies in the community, we will refer to this as a "2-transformation" neural network or a "1-layer" neural network. (See slides from Feb. 9 for more details.)

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.

2. Experiment with different choices for representing the input. For example, you could vary w (in addition to $w = 1$, try $w = 0$ and $w = 2$ and see which works best). You could also add additional features like those we discussed in class, e.g., a feature that returns 1 if the center word begins with a capital letter and 0 otherwise, a feature that returns 1 if the center word consists entirely of punctuation symbols and 0 otherwise, a feature that returns the number of characters in the center word, additional versions of those features for each of the context words, etc. Another option is to compare updating the word embeddings during training and keeping them fixed.
These suggestions are deliberately open-ended. You do not need to do all of the above, but you do need to do interesting experimentation and draw conclusions based on the results.

2 Your Choice:

After doing the above, do **one (1)** of the following three components:

1. **architecture engineering**: Experiment with different functional architectures. For each comparison below, use DEV to choose the best setting and report results on DEVTEST.
    - 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.
2. **regularization**: Implement and experiment with ways of doing regularization.
    - Perform L2 regularization and report results on DEV for different values of the regularization strength (λ). Report the DEVTEST accuracy corresponding to the λ that performs best on DEV.
    - Use dropout for the single hidden layer and report results on DEV for different dropout rates (e.g., {0.0, 0.2, 0.5, 0.9}). Report the DEVTEST accuracy corresponding to the dropout rate that performs best on DEV.
3. **error analysis**: Analyze the tagging errors that your system makes on DEV. Print a sample of the errors and manually go through 25 of them. Categorize the errors by answering the following questions for each error (note: multiple questions could apply to a single error):

(a) Is the error due to a rare word/wordform?

(b) Is the error due to a rare sense of a (relatively) frequent wordform?

(c) Could the error be addressed by adding simple feature functions based on surface forms? (e.g., regular expressions to identify emoticons, punctuation, and URLs)

(d) Would the error go away if we used more contextual information from the tweet?

(e) Would the error go away if we used structured prediction (e.g., sequence labeling) instead of token classification? (These would be cases in which making a joint prediction of multiple tags might be likely to fix the error.)

(f) Is the error due to something else?

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. [1, 2]

Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390, Atlanta, Georgia. Association for Computational Linguistics. [1]