TTIC 31190: Natural Language Processing – Fall 2023
Assignment 2 (85 Points)

Instructor: Freda Shi and Jiawei (Joe) Zhou
Assigned: Thursday, October 19, 2023
Due: 11:59PM CDT/CST, Thursday, November 2, 2023

Submission Instructions

Submit your report (in pdf format) and code through Gradescope. There will be two separate assignments on Gradescope for report and code submission. You can find a link to Gradescope on the Canvas assignment page. Your report should contain all of the required results and analysis.

Lateness Policy

If you turn in an assignment late, a penalty will be assessed. The penalty will be 2% (of the entire point total) per hour late. You will have 4 late days to use as you wish during the quarter. Late days must be used in whole increments (i.e., if you turn in an assignment 6 hours late and want to use a late day to avoid penalty, it will cost an entire late day to do so). If you wish to use a late day for an assignment, indicate that on your report or in the comments along with your homework submission through Canvas.

Collaboration Policy

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

1. Neural Networks for Part-of-Speech Tagging (75 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 as well as word embeddings from training the skip-gram model of `word2vec` on a large corpus on unlabeled English tweets. 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, including using the word embedding for the word being labeled concatenated with the word embeddings of neighboring words and additional feature functions; details are below. As your evaluation metric, use tagging accuracy, i.e., the percentage of tokens that were tagged correctly.

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 two popular toolkits, but feel free to use any others that you may find:

- PyTorch: `https://pytorch.org/`
- TensorFlow: `https://www.tensorflow.org/`

If you don't have any experience with these toolkits, don't worry. It's expected that a nontrivial portion of the time spent on this assignment will involve becoming familiar with a standard neural toolkit, which will likely be useful for the final assignment (and potentially later in your career). You are welcome to search for tutorials and sample code on the Web. If you use or adapt someone else's code, credit them in the comments of your code.

Provided Data

The following data files are provided to you:

- `twpos-train.tsv`: training data (1173 tweets) (TRAIN)
- `twpos-dev.tsv`: development data (327 tweets) (DEV)
- `twpos-devtest.tsv`: development test data (327 tweets) (DEVTEST)
- `orig-train.tsv`: original unprocessed version of TRAIN
- `orig-dev.tsv`: original unprocessed version of DEV
- `orig-devtest.tsv`: original unprocessed version of DEVTEST
- `twitter-embeddings.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 found in the rest of the file. The file contains one word per line in the format "word[space]dimension_1[space]dimension_2[space]...[space]dimension_50".

The annotated tweet files (`twpos-*` and `orig-*`) 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 Table 1 for the list of tags and examples of each. When creating instances for training or computing accuracies, each token should be considered as a single instance, though the tokens in surrounding lines can be used for additional features. That is, there should be 4821 predictions to make in DEV and 4639 predictions to make in DEVTEST (i.e., those should be the denominators when computing accuracies).

For your primary experiments, you can just use the `twpos-*` files. The `orig-*` files are provided in case you want to use them to define feature functions or see what the original data looked like before the preprocessing. The same sort of preprocessing used to generate the `twpos-*` files was used to process the 56 million English tweets before `word2vec` was run on them, so the pretrained embeddings above are best used with the `twpos-*` files. We provide the `orig-*` files as well because some of the preprocessing removed information that may be useful for the task.

1.1 A Baseline Neural Network Tagger (40 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 example, if $w = 1$ and the word embedding dimensionality is d, the total dimensionality of the input will be $3d$. For words near the sentence boundaries, pad the sentence with beginning-of-sentence and end-of-sentence characters (<s> and </s>). The word embeddings should

Tag	Description	Examples
Nominal, Nominal + Verbal		
N	common noun	books someone cook/chef mvp
O	pronoun (personal/WH; not possessive)	it you u meeee
S	nominal + possessive	books' someone's pitchers
L	nominal + verbal	he's book'll iono (= *I don't know*)
^	proper noun	lebron usa iPad
Z	proper noun + possessive	America's lakers Frank's
M	proper noun + verbal	Mark'll momma's
Other open-class words		
V	verb including copula, auxiliaries	might gonna ought couldn't is eats
A	adjective	good fav lil earlyy
R	adverb	super really proli 2 (i.e., *too*)
!	interjection	lol haha FTW yea right
Other closed-class words		
D	determiner, possessive pronoun	the teh its it's his
P	pre- or postposition, or subordinating conjunction	while to for 2 (i.e., *to*) 4 (i.e., *for*)
&	coordinating conjunction	and n & + BUT
T	verb particle	out off Up UP
X	existential *there*, predeterminers	there both all
Y	X + verbal	there's all's
Twitter/online-specific		
#	hashtag (indicates topic/category for tweet)	#YouGotTheWrongDavid #NowPlaying
@	at-mention (indicates another user as a recipient of a tweet)	@BarackObama @Dropbox
~	discourse marker, indications of continuation of a message across multiple tweets	RT and : in retweet construction RT @user : hello
U	URL or email address	http://bit.ly/xyz
E	emoticon	:-) :b (: <3 o__O
Miscellaneous		
$	numeral	2010 four 9:30 $50ish
,	punctuation	!!! .... ?!? -
G	other abbreviations, foreign words, possessive endings, symbols, garbage	ily (*I love you*) wby (*what about you*) 's ♫ --> awesome...I'm

Table 1: The set of tags used to annotate tweets.

be randomly initialized and learned along with all other parameters in the model.

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 an affine transformation which will produce scores for all possible POS tags. Use a softmax transformation on the scores to produce a probability distribution over tags.

learning: Use log loss as the objective function (log loss is often called "cross entropy" or "negative log-likelihood" when training neural networks, so those terms may be useful when searching for the right loss function in toolkits). Use SGD or any other optimizer you wish. Toolkits typically have many

optimizers already implemented.

initialization: Randomly initialize all parameters, including word embeddings, and train them. Note that embeddings for words that only appear in DEV/DEVTEST will not be trained at all. So, you need to be careful about how those embeddings are set in order to get good results. We suggest an initialization range of -0.01 to 0.01 for all word embedding parameters. (You could alternatively try setting embeddings for unknown words to all zeros or try learning an unknown word embedding during training.)
Train on TRAIN, perform early stopping and preliminary testing on DEV, and report your final tagging accuracy on DEVTEST. Report results with both $w = 0$ and $w = 1$. Submit your code.

Notes: With $w = 0$, I was seeing a best DEV accuracy of 77-78% and with $w = 1$ it improved to 80-81%. I set the size (dimensionality) of word embeddings to 50, and used SGD with a fixed step size of 0.02 and each mini-batch contained one word to be tagged. I trained for 10 epochs and evaluated on DEV once per epoch. It took approximately 10 seconds per epoch using PyTorch on a 3.3 GHz Intel Core i5.

1.2 Feature Engineering (15 points)

Add features to the model by concatenating your own feature function outputs to the word embedding concatenation used above. Define feature functions based on looking at the training data, based on looking at the errors your tagger makes on DEV, or simply based on your intuitions about the task. For example, you could add binary features if the center word contains certain special characters or capitalization patterns, a feature that returns the number of characters in the center word, features for particular prefixes, suffixes, and other character patterns in the center word, etc. These sorts of features could also be defined for context words. You may find it helpful to use the `orig-*` files when computing features. (You will probably still want to use the `twpos-*` files for the word embeddings, though.)

Develop and experiment with features and describe your results. You should be able to improve upon the accuracies you were seeing in Section 1.1.

1.3 Pretrained Embeddings (10 points)

Initialize your word embeddings using the pretrained embeddings from `twitter-embeddings.txt`. For words in the tagging datasets that are not in the pretrained embeddings, use the unknown word embedding (i.e., the embedding for the word "UUUNKKK"). The pretrained embeddings contain an embedding for the sentence end symbol `</s>`, but not the sentence start symbol `<s>`. You can either randomly initialize and train an embedding for `<s>` from scratch or you can use the embedding for `</s>` for both `<s>` and `</s>`. Document what you did in your write-up.

Conduct the following experiments:

- Experiment with updating (fine-tuning) the pretrained embeddings for both $w = 0$ and $w = 1$ and report your results. You should see improvements over the randomly-initialized word embedding experiments from Section 1.1.
- With $w = 1$, empirically compare updating the pretrained word embeddings during training and keeping them fixed.
- Combine your features from Section 1.2 with the use of pretrained embeddings. Do the features still help?

1.4 Architecture Engineering (10 points)

Using the best configuration from above, explore the space of neural architectures to see if you can improve your tagger further. Some suggestions are below:

- 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 nonlinearities, e.g., identity ($g(a) = a$), $\tanh$, ReLU, and logistic sigmoid.
- Experiment with $w = 2$ and compare the results to $w = 0$ and 1.

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

Implement and experiment with recurrent neural network (RNN) taggers. Compare a standard RNN to an LSTM or GRU, and also experiment with a bidirectional RNN (or bidirectional LSTM/GRU). Compare final tagging accuracies on DEVTEST. Submit your code.

2. Language Modeling (10 points)

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 $\mathcal{V}$ (assuming the sequence starts with <s> and ends with </s>), while Model U will assign zero probability to word sequences that contain a bigram that is not in D.

Smoothing is sometimes thought of as moving 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 does. However, this is not necessarily the case!

Provide a dataset D of sentences such that Model S assigns higher probability than Model U to some sentence in D. In addition to providing us with your dataset, show that Model S assigns higher probability to some sentence than Model U by writing out all bigram probabilities for both models as well as the probabilities of the key sentence in D for which Model S assigns higher probability.

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 $x \in \mathcal{V} \cup \{\texttt{</s>}\}$.

You can use computers to help you with this, but it's not necessary. No code needs to be turned in. Hint: Start with extremely small datasets and small vocabulary sizes. There is a solution that only uses two word types (in addition to <s> and </s>).