ASSIGNMENT 2: DOMAIN ADAPTATION OF WORD VECTORS

Motivation

The motivation of this assignment is to get you started with thinking of words as embeddings and get you to reuse the existing resources for the specific domain that you may be interested in.

Problem Statement

The goal of the assignment is to build a domain adaptation system for word vectors. The input of the code will be a set of pre-trained word embeddings and a corpus of text, the domain of the textual given corpus is different from the domain of text that was used to train the word embeddings. You are expected to learn word embeddings given this data and show improvements for the task of masked language modelling (explained later).

For this assignment, you are expected to use non-sequential models. Hence, **you are not allowed to use RNNs or LSTMs or their extensions.** But you are allowed to use character-based CNN models, as they are not exactly sequential models.

Resources

All the resources will be available in the following github repository:
https://github.com/SaiKeshav/wv-da.git

Training Data

We are sharing a training dataset of text along with a link to a set of pre-trained word embeddings.

The Task

You need to write a model that constructs embeddings for the in-domain vocabulary. Also, given a sentence with one word masked out, your model should rank a given set of words in the order of how well the words fit the context in the place of the masked word. Note here that the masked sentences would belong to the same domain as that of the training corpus given to you.

Before you begin, try to carefully think about the problem and try to identify the minute details of the task which could better help you model the problem. To improve your system performance over the baseline, a few ideas to try are listed below.

1. Retrain the given word embeddings of an expanded vocabulary by using word2vec on the given text corpora.
2. Retrain the word embeddings using character CNN.

3. Try to handle the OOVs to be encountered due to the change of domain. To handle OOV words, you can try out various ideas involving the POS tags of words, learning character embeddings, etc.
4. Your idea here…

Input Files

You will be provided with two files - *eval_data.txt* and *eval_data.txt.td*

eval_data.txt: Every line contains a data-point which is a sentence with the masked word replaced with the following token: <<target>> , and the actual token will be provided at the end of the line. In the dev file, The ground truth word will be separated from the rest by a special delimiter '::::' (4 colons). The test_data file will have the same format but the actual word will be replaced by 'dummy'.

eval_data.txt.td: This file is line-aligned with the previous file. Every line contains a space-separated list of words, which form the target dictionary for each of the data-points in the previous file. Only one of the words will be the correct answer but you need to rank this set of words and output the respective ranks.

Output Files

output.txt: You are expected to produce a file where each line contains the output for every data point in eval_data.txt. (Hence, it must be line-aligned with eval_data.txt and eval_data.txt.td). Every line contains a space-separated list of ranks for each word mentioned in eval_data.txt.td (in the same order). The word with the highest probability (according to your model) must be assigned a rank of 1.

Metrics

Mean Rank: The rank of the correct word will be taken for each data point and averaged across all the data points. You must minimize this measure.
Mean Reciprocal Rank: The reciprocal of the rank of the correct word will be taken for each data point and averaged across data points. You must maximize this measure.

We will use results on both metrics to assign the final grade in your assignment.

Additional Details

- The test sentences will be tokenized using the nlkt punkt-word tokenizer and hence you are encouraged to use the same for your training data to ensure lesser variation between your training samples and test samples.
- In some-cases the target-word may be the proper nouns (name of a person, company, etc). Since predicting them based on a context of a single sentence is infeasible, we replace such targets (which are proper nouns) with a standard token: '-pro-'. To identify them, we use the

nltk pos_tagger and filter those target words with the tag as ‘NNP’ or ‘NNPS’. You are encouraged to use the above mechanism for identifying proper-nouns.

- Since the dev and test data are automatically generated and not manually curated, it is possible that some of the sentences may be ill-formed, or in some-cases more than one word may fit naturally into the sentence. But these will be in the minority, so please don’t complain about them. But if you do find a issue prominent in many of the dev examples, raise it on Piazza, and we will look into it.
- Every word must be assigned a unique rank. This will be checked in the evaluation scripts.
- The target word will never be a number or a punctuation-mark (however, it may contain a punctuation mark).

Submission Instructions

You will need to provide a *train.sh* and a *test.sh*, which are scripts to train your model from scratch and test it. You will also need to submit your trained model (named *model.pt*), in-case we decide not to train the model from scratch and directly test it (More details on submission of trained model will follow later. Stay tuned on Piazza…).

bash train.sh *input_data.txt*
It should generate a model file - *model.pt,* in the same directory.

bash test.sh *eval_data.txt eval_data.txt.td*
It should generate the output file (format mentioned above) named as *output.txt,* in the same directory

Try to take the assignment in the proper spirit and refrain from trying out funny businesses with the submissions (i.e., do not search for test set or use test set in any form while training). Most importantly, as you work on improving your baseline system, document its performance. Perform (statistical) error analysis on a subset of data and think about why the model is making these mistakes and what additional knowledge could help the classifier the most. That will guide you in picking the next feature (or model component) to add.

Testing your code

We will compute the MRR and MR values using a script named mrr.py It will take the generated output.txt, the evaluation data (eval_data.txt) and the target dictionary (eval_data.txt.gt) to compute the values.

python mrr.py *eval_data.txt eval_data.txt.td output.txt*

Make sure this works on your generated *output.txt* as we will use the same script during our final evaluation.

What to submit?

1. The first deadline will be on 12th March 11:55 PM for the submission of your system.
   Submit your code in a .zip file named in the format **<EntryNo>.zip.** Make sure that when we run "unzip yourfile.zip" a new directory is created with your entry number (in all caps). In that directory, the following files should be present.
   compile.sh
   train.sh
   test.sh
   writeup.txt

   You will be penalized if your submission does not conform to this requirement.

Your code will be run as :

1. bash compile.sh
2. bash train.sh *input_data.txt*
3. bash test.sh *eval_data.txt eval_data.txt.td*

If your code doesn't conform to the requirements you will lose 20% of the credit.

Your code should work on a single HPC node with 1 cpu and 1 gpu. You will be penalized for any submissions that do not conform to this requirement. The training script will be given maximum of 6 hrs.

2. The writeup.txt should have first line that mentions names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.

After this first line you are welcome to write something about your code, though this is not necessary.

Evaluation Criteria:

The quality of your trained word embeddings would be measured through a word-prediction task. You will be given a sentence with one word masked, and your task will be to predict the masked word from a dictionary of possible words (also referred to as masked-language modelling).

(1) This assignment is worth 80 points.
(2) Bonus will be given to outstanding performers.

What is allowed? What is not?

1. The assignment is to be done individually.
2. **You may use only Python for this assignment.**
3. **Pytorch is the only allowed deep learning framework for this assignment**

4. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
5. Feel free to search the Web for papers or other websites describing how to build a sentiment classifier. However, you should not use (or read) other people's sentiment mining code.
6. You cannot use any pre-existing ML softwares for your code.
7. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.
8. Your code will be automatically evaluated. You get significant penalty if it is does not conform to given guidelines.