Assignment 2
CSEP 517: Natural Language Processing

University of Washington

Due: April 23, 2017

In this assignment, you will think about how we can evaluate the quality of word embeddings. Word embeddings are often described as a representation of a word's *semantics*, or meaning. When viewed this way, word embeddings are sometimes called **distributed semantic vectors** (or just "semantic vectors"). This problem is meant to get you thinking about how we might evaluate word meaning representations, both intrinsically (the first problem) and extrinsically on a text classification task (the second problem). The latter will also give you the chance to get a little acquainted with TensorFlow, a widely used library for neural networks.

The construction of word embeddings from text corpora is now a cottage industry. The main thing you need to know is that most methods are based on a simple idea: words that occur in similar contexts should have similar embeddings. Slightly more formally, you can think of word embedding construction methods as doing:

1. Decide on a representation of the distribution of **contexts** each word $v \in \mathcal{V}$ occurs in, as a vector. A common choice is a $|\mathcal{V}|$-length vector containing the counts of all words occuring just before or just after (say, within two word positions) of some token of the word v.[1]
2. Compress these long, sparse vectors into smaller (roughly 100-dimensional) vectors, usually "dense" (few or no 0s) vector.

While we don't have time to get into more details of standalone methods for constructing word embeddings, you are encouraged to read the cited papers if you want to learn more about that.

1 Analogy Task (40%)

Similar to language models, embeddings are trained directly from a large collection of natural language text without being tied to a specific NLP application or subtask. How can we measure the quality of learned embeddings? Some of the commonly accepted evaluation methods are based on word similarity tasks, word analogies (e.g., "man is to woman as king is to queen"), and semantic compatibilities between verbs and their semantic arguments (also known as "selectional preferences"; see Schnabel et al., 2015). In this assignment, you will explore an analogy task.

The analogy prediction task is defined as follows. Given a pair of words $\langle a, b\rangle$ and a third word c, choose a fourth word d so that the analogy "a is to b as c is to d" holds. In other words, the relationship between c and d should be as close as possible to that between a and b. Note: the system need not characterize this relationship or give it a name! The relationship is taken to be implicit in the pair $\langle a, b\rangle$.

[1]The idea that a word's meaning can be represented by the distribution of contexts it occurs in is known—somewhat confusingly—as *distributional semantics*. It dates back to the 1950s [Firth, 1957]. The required quotation from Firth is "you shall know a word by the company it keeps."

Mikolov et al. [2013a] proposed that simple algebraic operations could be applied to embeddings to find an analogy prediction. Let $\mathbf{v}_a$ be the vector for a, $\mathbf{v}_b$ the vector for b, and so on. For the d such that the analogy holds, we expect

$$\mathbf{v}_b - \mathbf{v}_a \approx \mathbf{v}_d - \mathbf{v}_c. \tag{1}$$

We therefore seek

$$d = \arg \max_{d \in \mathcal{V} \setminus \{a,b,c\}} \cos\left(\mathbf{v}_d, \mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c\right). \tag{2}$$

Note that the given words a, b, and c are excluded from consideration. The cosine similarity is often used as a similarity function between nonzero vectors; you can find details on Wikipedia.

1.1 Analogy Dataset

You will be given a subset of Mikolov's analogy dataset [Mikolov et al., 2013a], which includes four semantic relations and four syntactic[2] relations. In the test files, each line represents one analogy question, in the form of four words $\langle a, b, c, d \rangle$. For example:

Bangkok Thailand Cairo Egypt

A question is counted as correctly answered only if the predicted word is the same as the given word. For example, given the first three words "Bangkok Thailand Cairo", the task is to predict "Egypt".

The full set of analogy questions can be found in the file `word-test.v1.txt` in the assignment tarball.[3] The groups of relations are delimited by lines starting with a colon (:) and you should only work with these groups: capital-world, currency, city-in-state, family, gram1-adjective-to-adverb, gram2-opposite, gram3-comparative, and gram6-nationality-adjective.

1.2 Word Embeddings

"Pretrained" word embeddings are word embeddings that are already constructed in advance of your NLP project (whether your project is a neural language model, a text classifier, or a class assignment). The advantage of pretraining is that it simplifies learning your model, because the embedding parameters are fixed in advance. The disadvantage is that, if your embeddings happen to be bad for your task, you're stuck with them.

For this assignment, you are free to use any pretrained word embeddings you can find, as long as you cite their papers in the writeup. Here are some popular choices:

- Word2vec [Mikolov et al., 2013a,b]: `https://code.google.com/archive/p/word2vec/`
- GloVe [Pennington et al., 2014]: `http://nlp.stanford.edu/projects/glove/`
- Dependency-based embeddings [Levy and Goldberg, 2014]: `https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/`

Note that embeddings vary not only in the method of construction from data, but also in dimensionality, amount and type of text used, and more. When you report on your choices, be precise!

Additional notes. It is okay to use existing tools to efficiently find the most similar vector; as always, cite the tools you used. Please do not download massive files onto the class server; you may have to settle for lower-dimensional word embeddings or files that contain smaller vocabularies of word embeddings.

[2]Not to split hairs, but your instructor thinks these are better described as morpho-syntactic relations.

[3]This was downloaded from: `http://www.fit.vutbr.cz/~imikolov/rnnlm/word-test.v1.txt`

1.3 Deliverables

- Run the analogy test. Pick any two sets of pretrained embeddings, implement the analogy prediction method described in Equation 2, and compare their accuracies[4] on the eight analogy tasks listed above. Make sure to mention the details of your selection in writing.
- One known problem with word embeddings is that antonyms (words with meanings considered to be opposites) often have similar embeddings. You can verify this by searching for the top 10 most similar words to a few verbs like *increase* or *enter* that have clear antonyms (e.g., *decrease* and *exit*, respectively) using the cosine similarity. Discuss why embeddings might have this tendency.
- Design two new types of analogy tests that are not part of Mikolov's analogy dataset. You will create your own test questions (3 questions for each type, so in total 6 new questions). Report how well the two sets of embeddings perform on your test questions. You're encouraged to be adversarial so that the embeddings might get an accuracy of zero! Discuss any interesting observations you have made in the process.

2 Sentiment Classification (60%)

A cell phone company is considering making a touchscreen version of its most popular phone. While they haven't decided whether they'll partner with Aaple or Goggle, they do know that their new smartphone is going to have unprecedented Twwiter integration. Based on some initial research, they suspect that users of different platforms twweet differently. The company's CSO[5] decides to investigate by looking at how twweets coming from aPhone and Adenoid devices differ in sentiment. They choose to hire you as their VP for NLP to perform sentiment analysis on a set of twweets they have already collected (coming from aPhone and Adenoid users).

As training and development data, you have a set of social media messages in English that have been categorized into two classes: *positive* and *negative* sentiment. As test data, you have a separate set of messages.

Log-linear model. You're provided with a simple log-linear model (also known as "multinomial logistic regression") code base (see `sentiment.py`) using TensorFlow. By making minor modifications to this code, you will be able to create a feedforward neural network classifier. More concretely, the code you're given creates a *computation graph* for the log-linear classifier:

$$\mathbf{h} = \mathbf{x}^\top \mathbf{W} \tag{3}$$

$$\mathbf{p} = \mathrm{softmax}(\mathbf{h}) \tag{4}$$

$$\hat{y} = \arg\max_i p_i \tag{5}$$

where $\mathbf{x} \in \mathbb{R}^{\text{\#input\_features}}$, $\mathbf{W} \in \mathbb{R}^{\text{\#input\_features} \times \text{\#labels}}$. Here, #labels $= 2$ and #input_features is the number of *input* features (therefore, each input-output feature combines one of the input features with one of the two labels). The input feature vector $\mathbf{x}$ represents a single message (input only), and in this assignment you will experiment with two options:

1. the average of the embedding vectors for the word tokens in the message
2. the average of the *one-hot* vectors for the word tokens in the message

Finally, $\hat{y}$ is the predicted class (positive or negative).

[4]Accuracy is $\frac{\text{number of correctly answered questions}}{\text{number of questions attempted}}$.

[5]Chief sentiment officer, obviously.

Feedforward neural network. Next, create a feedforward neural network. The recipe in equations is:

$$\mathbf{h}_1 = \text{nonlinearity}\left(\mathbf{x}^\top \mathbf{W}_1 + \mathbf{b}_1\right)^\top \tag{6}$$

$$\mathbf{h}_2 = \text{nonlinearity}\left(\mathbf{h}_1^\top \mathbf{W}_2 + \mathbf{b}_2\right)^\top \tag{7}$$

$$\mathbf{p} = \text{softmax}(\mathbf{h}_2) \tag{8}$$

$$\hat{y} = \arg\max_i p_i \tag{9}$$

Compared to the log-linear model, you must incorporate the bias terms $\mathbf{b}_1$ and $\mathbf{b}_2$ as additional parameters to learn, add a "hidden layer," and add nonlinearities (refer to the lectures for some possibilities).

Hyperparameters and model variations. There are a number of hyperparameters and design choices you can vary:

- the length of the hidden layer $\mathbf{h}_1$;
- the mini-batch size for training, i.e., how many messages you process at once for each training pass, which will affect both the speed and the accuracy of learning;
- the choice of $\mathbf{x}$ (two options given above);
- number of epochs, i.e., the number of passes through the training dataset;
- the vocabulary size for the one-hot encoding of the twweets, which follows from your procedure for assigning words to `UNK`; and
- which nonlinearity you use.

2.1 Data & Code

TensorFlow is extremely convenient for trying out different models (as expressed as computation graphs), and you don't need to worry about the low-level details with optimization and learning (e.g., computing the derivatives, different optimization strategies) so long as the loss function of your model is fully differentiable with respect to the model parameters. Getting familair with the TensorFlow API can be a bit of hassle, especially if you are not already familar with Python. To ease your learning experience, we have provided you with most of the code you need for this assignment. Here are the steps to getting started.

1. Log in to `umnak.cs.washington.edu`.
2. Copy `/homes/iws/nasmith/A2.tar.gz` and unpack it. `cd` into directory `A2`.
3. Run `./setup.sh`, which will take a few minutes. This script installs TensorFlow and some other packages, locally.
4. Here are the files in the distribution:
   - The file `sentiment_2017.py` is the Python program that you will use to train and test your model. To see where to modify it, look for lines surrounded by `<CSEP517>` and `</CSEP517>`. Invoke it by running (for example):

     ```
     ./tf8/bin/python3 sentiment_2017.py one_hot
     ```

     Replacing `one_hot` with the name of a file containing word embeddings (the ones we give you end in `.pkl`) will replace one-hot embeddings with those in the file.
     After printing out training progress, the program prints out classification accuracy and other statistics.
   - `sampleSent.pkl` and `sourceSent.pkl` contain the training and testing twweets. You do not need to worry about loading them, the code is already written for that.
   - The word vectors are contained in these files:

  - `glove.6B.XXd.pkl`: GloVe trained on English Wikipedia and the English Gigaword corpus. Vocabulary size 400,000, overlap with training vocabulary is around 17,000.
  - `glove.twitter.27B.XXd.pkl`: GloVe trained on 27 billion tweets (all languages). Vocabulary size: 1.2 million, overlap with training vocabulary is around 2,000.
  - `GoogleNews-vectors-negativeXX.pkl`: Word2vec trained on the Google News corpus. Vocabulary size: 3 million, overlap with training vocabulary is around 15,000.

  `XX` is the dimension of the vectors. Note that we only provide you with the word vectors for the words in the training dataset, to reduce the file size.

2.2 Deliverables

- Complete the TensorFlow code to create a feedforward neural network classifier by creating the computation graph given by Equations 6–9. As you might have noticed, the equations for the neural network are not so different from those for the log-linear model. Thus, you can start by copying over the log-linear model code and then add just a few lines to it.
- Explore a few different embeddings (varying their dimensions or varying different sources of the pre-trained embeddings) to use with the classification model. Comment on how this affects accuracy, and offer explanations. Think about the different natural language corpora used to create these embeddings.
- Try using the one-hot representation (run with argument `one_hot`). Comment on its performance compared to the performance of the best embeddings.
- How does the neural network compare to the log-linear classifier?
- **Bonus**: Try different computation graphs by adding hidden layers, or by converting the input representation into parameters so that you can learn the task-specific embeddings (which can be initialized by the pre-trained embeddings, or not). If you do this, also provide the equations corresponding to your model.
- Given your best model, report the sentiment breakdown for aPhone and Adenoid users.

Submission Instructions

Submit a single gzipped tarfile (`A2.tgz`) on Canvas.

- **Code**: You will submit your code together with a neatly written README file to instruct how to run your code with different settings. 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 `A2.pdf` and include in the tarfile): Your writeup should be three pages long, or less, in pdf (one-inch margins, reasonable font sizes, preferably 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 R. Firth. A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*, pages 1–32. 1957.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proc. of ACL*, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-2050.pdf`.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a. URL https://arxiv.org/pdf/1301.3781.pdf. arXiv:1301.3781.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proc. of HLT-NAACL*, 2013b. URL http://www.aclweb.org/anthology/N/N13/N13-1090.pdf.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proc. of EMNLP*, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proc. of EMNLP*, 2015. URL http://www.aclweb.org/anthology/D15-1036.