CMSC 25025 / STAT 37601

Machine Learning and Large Scale Data Analysis

Assignment 5 Due: Tuesday, May 22, 2018 at 1:30 pm.

1. *Demystifying word2vec* (20 points)

   In this problem, we aim to understand the inner workings of word2vec, one of the most popular word embedding methods.

   Here we will focus on what's called the skip-gram method. See the word2vec Wikipedia page for more information.

   The specifics of the skip-gram method depend on the notion of a *context*. Given a sequence of words $w_1, w_2, \ldots, w_T$, the contexts of the t-th word w_t are the words in a symmetric window around w_t. For example, with a context window of size two, the contexts of the word "embeddings" in the sentence

   $$\text{I } \underbrace{\text{love word}} \text{ embeddings } \underbrace{\text{so much}}.$$

   are *love, word, so,* and *much.* The word-context pairs (w, c) associated with the word "embeddings" are therefore (embeddings, love), (embeddings, word), (embeddings, so), and (embeddings, much).

   Under this definition of context, both words and contexts are drawn from the same vocabulary $\mathcal{V}$. Let $\mathcal{S}^p$ denote the set of observed word-context pairs in a given corpus. Let $N^p(w, c)$ denote the number of times the pair (w, c) appears in $\mathcal{S}^p$, and let $N^p(w) = \sum_{c' \in \mathcal{V}} N^p(w, c')$ denote the number of times w appears in $\mathcal{S}^p$. Similarly, $N^p(c)$ is the number of times the context c occurs in $\mathcal{S}^p$.

   Consider a word-context pair (w, c) and let $\mathcal{O}$ denote a random variable whose value is one if the pair (w, c) was observed in the corpus, and zero otherwise. In the skip-gram approach, this distribution is modeled as:

   $$\mathbb{P}(\mathcal{O} = 1 \,|\, w, c) = \sigma(v_w^T v_c) = \frac{1}{1 + e^{-v_w^T v_c}}$$

   where σ denotes the logistic function, and $v_w, v_c \in \mathbb{R}^d$ are d-dimensional word embeddings for the word w and the context word c, respectively.

   The skip-gram method aims to learn embedding vectors for all words and contexts so as to maximize $\mathbb{P}(\mathcal{O} = 1 \,|\, w, c)$ for pairs (w, c) found in the corpus, while maximizing $\mathbb{P}(\mathcal{O} = 0 \,|\, w, c)$ for randomly sampled negative examples, assuming that randomly sampling a context word for a given word from $N^p(c)/N(\mathcal{S}^p)$ is likely to result in an unobserved (w, c) pair. It has been shown empirically that this approach gives rise to similar word embeddings for words that share similar contexts (e.g. republican and democrat).

In class we showed that for a given embedding ϕ:

$$\ell(\phi) = \sum_{w \in \mathcal{V}} N^p(w) \left[\sum_{c \in \mathcal{S}_w^p} N^p(c|w) \log[\sigma(v_w^t v_c)] + \sum_{c \in \mathcal{S}_w^n} N^n(c|w) \log[\sigma(-v_w^t v_c)] \right]$$

$N^p(c|w)$ is the number of times c appeared as a center for w in the positive sample and $N^n(c|w)$ for the negative sample. On average $N_n(c|w) = kN^p(c)/N(\mathcal{S}^p)$ and doesn't depend on w, so from now on we redefine $N_n(c|w) = kN(c)/N(\mathcal{V})$.

(a) Show that

$$\ell(\phi) = \sum_{c \in \mathcal{V}, w \in \mathcal{V}} N^p(c,w) \log[\sigma(v_w^t v_c)] + kN^p(w) \frac{N^p(c)}{N(\mathcal{S}^p)} \log[\sigma(-v_w^t v_c)] \tag{1}$$

(b) Now focus on an individual term in the above sum. Define $x = v_w^T v_c$ and substitute x into Equation (1). Take the derivative $\frac{\partial \ell}{\partial x}$, set it equal to zero, and *show all steps* needed to get the following equivalence:

$$\frac{\partial \ell}{\partial x} = 0 \tag{2}$$

$$\Rightarrow \; e^{2x} - \left(\frac{N^p(w,c)}{k \cdot N^p(w) \cdot \frac{N^p(c)}{N(\mathcal{S}^p)}} - 1 \right) e^x - \frac{N^p(w,c)}{k \cdot N^p(w) \cdot \frac{N^p(c)}{N(\mathcal{S}^p)}} = 0. \tag{3}$$

(c) Define $y = e^x$, substitute y into Equation (3), and solve for the roots of the resulting quadratic equation. Finally, by substituting y with e^x and x with $v_w^T v_c$, show that one of the roots occurs at

$$v_w^T v_c = \log \left(\frac{N^p(w,c) \cdot N(\mathcal{S}^p)}{N^p(w) \cdot N^p(c)} \right) - \log k.$$

2. *Word Embedding Experiments* (40 points)

The following experiments should be done with the Wikipedia corpus here:

```
/project/cmsc25025/wikipedia/wiki-text.txt
```

The number of unique words in the Wikipedia corpus is too large for our purposes. Before proceeding, you should come up with a smaller vocabulary $\mathcal{V}$ that you will use for the remainder of the word embedding experiments. You can filter all of the unique words in the Wikipedia corpus in a number of ways. Here are some examples:

- Remove words that appear less than n times (e.g. try $n = 500$). You may use any existing python packages to compute word counts, for example `nltk.FreqDist` or `collections.Counter`.
- Remove all words that appear in the `stopwords` list of `nltk` package.

```
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

You should aim to have roughly $15,000$ words in your vocabulary. In what follows, you may simply ignore words that do not appear in your final filtered vocabulary V. Whatever code you run try it first on a small section of the wiki data. The final run may take a while.

PMI Embeddings

Following the results in (1), let $M \in \mathbb{R}^{N(\mathcal{V})\times N(\mathcal{V})}$, and using $k = 1$, denote a matrix such that the (i, j)-th entry is the dot product between the word embedding vectors for the i-th and j-th words:

$$M_{ij} = v_{w_i}^T v_{w_j} = \log\left(\frac{N^p(w_i, w_j)\cdot N(\mathcal{S}^p)}{N^p(w_i)\cdot N^p(w_j)}\right)$$

This is known as the pointwise mutual information (PMI) of (w_i, w_j).

If we stack the word embedding vectors as rows of a matrix $V \in \mathbb{R}^{N(\mathcal{V})\times d}$, then we have that $M = VV^T$. In (1), we have therefore shown (after sweeping some assumptions under the rug) that skip-gram with negative sampling is equivalent to factorizing the symmetric matrix M whose entries consist of the pointwise mutual information.

Note: the matrix M below will be quite large. For this part of the problem set, you might need to request an RCC node with more memory. 15GB should suffice.

(a) Using the Wikipedia data with a symmetric context window size of 5, compute the PMI matrix $M \in \mathbb{R}^{N(\mathcal{V})\times N(\mathcal{V})}$ whose (i, j)-th entry is the PMI of (w_i, w_j). To avoid degeneracy, add 1 to the cooccurence statistics $N^p(w_i, w_j)$ so that

$$M_{ij} = \log\left(\frac{(N^p(w_i, w_j) + 1) \cdot N(\mathcal{S}^p)}{N^p(w_i) \cdot N^p(w_j)}\right)$$

where $\mathcal{S}^p$ is the set of all word-context pairs observed in the Wikipedia corpus. Note that $N^p(w_i, w_j)$ is simply the number of times w_i and w_j cooccur within 5 words of each other across the entire corpus.

(b) Now we will factorize the matrix M to obtain word embeddings. Take the k-SVD of M with $k = 50$ to obtain

$$M = U\Sigma V^T = \underbrace{U\Sigma^{\frac{1}{2}}}_{W}\underbrace{\Sigma^{\frac{1}{2}}V^T}_{C},$$

where $U \in \mathbb{R}^{|V|\times 50}, \Sigma \in \mathbb{R}^{50\times 50}$, and $V^T \in \mathbb{R}^{50\times|V|}$. You may use the `scipy` package to compute the k-SVD:

```
U, s, V =
  scipy.sparse.linalg.svds(scipy.sparse.csr(M), k=50)
```

(c) Let the rows of $W = U\Sigma^{\frac{1}{2}}$ be the learned PMI word embeddings. You will use these embeddings in several tasks below. We recommend that you serialize these embeddings to disk for later use (e.g., by using the `cPickle` package).

(d) For each of the following words, find the 5 closest words in the embedding space: `physics, republican, einstein, algebra, fish`.

(e) A surprising consequence of some word embedding methods is the resulting linear substructure. This structure can be used to solve analogies like

```
france : paris :: england : ?
```

by computing the nearest embedding vector to v where v is

$$v = v_{\text{paris}} - v_{\text{france}} + v_{\text{england}}$$

Define 3 analogies `X:Y=Z:W`.

```
republican:democrat=conservative:?

...
```

and report the top 5 words you get.

3. *RNN for predicting characters* (40 points)

The following online post is a very good presentation of RNN's and LSTM's http://colah.github.io/posts/2015-08-Understanding-LSTMs/

In this problem you will be experimenting with different RNN architectures to predict characters. The data set is `/project/cmsc25025/Shakespear/tinyshakespeare.txt`. The code is `/project/cmsc25025/Shakespear/LSTM_EXP.ipnyb`. The gpu code is `/project/cmsc25025/Shakespear/LSTM_EXP_GPU.ipnyb`

In this code there is a generic network that takes as input a 'cell' (an RNN, an LSTM etc.) There is a generic training function, a generic testing function that produces a test loss, and a generic simulation function. Each of these functions gets input a 'cell' that is defined in some function.

(a) Run the basic LSTM architecture given in the notebook. Plot the error on training as a function of epoch. Try a few priming sentences and see what the trained architecture produces. Load the test set and estimate the error on prediction in the test set.

(b) Run the same experiment with the basic RNN architecture. This has 1/4 of the parameters of the LSTM. Compare the training error plot with the original LSTM. Compare the test set error.

(c) Try 5 alternatives. For example an RNN with the same number of parameters as the LSTM. Or an RNN with more than one internal layer. Explain each of your models, plot the training error rate functions as a function of epoch for all your models and show the error rates on the test set as well.

(d) For each model, run a simulation starting with the same initialization text of your choice.