ABBOUDI Mohammed Amine
mohammed-amine.abboudi@polytechnique.edu

Lab session # 2
ALTEGRAD 2019

10/28/19

1 Question 1

$$\mathcal{L} = \sum_{c \in C_t^+} \log\left(1 + \exp(-w_{c^+}.w_t)\right) + \sum_{c \in C_t^-} \log\left(1 + \exp(w_{c^-}.w_t)\right) \tag{1}$$

1.1 Deriving the partial derivatives of the loss w.r.t w_{c^+} :

$$\frac{\partial \mathcal{L}}{\partial w_{c^+}} = \frac{\partial \log\left(1 + \exp(-w_{c^+}.w_t)\right)}{\partial w_{c^+}}$$
$$\implies \frac{\partial \mathcal{L}}{\partial w_{c^+}} = \frac{-exp(-w_t.w_{c^+})}{1 + exp(-w_t.w_{c^+})}.w_t$$

2 Question 2

2.1 Deriving the partial derivatives of the loss w.r.t w_{c^-} :

$$\frac{\partial \mathcal{L}}{\partial w_{c^-}} = \frac{\partial \log\left(1 + \exp(w_{c^-}.w_t)\right)}{\partial w_{c^+}}$$
$$\implies \frac{\partial \mathcal{L}}{\partial w_{c^+}} = \frac{exp(w_t.w_{c^-})}{1 + exp(w_t.w_{c^-})}.w_t$$

3 Question 3

Before plotting the similarity plot of the most frequent word, we handpick words whose we would like to compute. For instance we will calculate the similarity between `banana` and `film`. It is equal to 44.79%, which is to be expected since banana and film rarely coincide in a five word-wide window. On the other hand, the words `drama` and `thriller` have a very high similarity (93.05%) since they co-occur quite frequently together given our movie biased dataset. Moreover, since the embedding matrix gives a way to calculate the "semantic" distance between any given words present in the vocabulary, it inevitably carries with it the inherent bias present in the training dataset, that is evermore present in the similarity between the words `good` and `movie` which is 90.84%. Objectively speaking, these two words have nothing semantically in common apart from the fact they they occur very closely to each other in a movie reviewing website.
Biases' intrinsic presence in any type of dataset should be lent more attention since that the objective of such exercise is to extract the underlying semantic similarity and not a co-occurrence scoring algorithm. Diversifying the sources of the document is an option to mitigate this risk. However if the aim is to create a topic oriented similarity metric then a dataset biased in that direction is preferable.

Below are the plots of the resulting embedding matrix sampled to the 500 most frequent words in the dataset before and after training the shallow neural network. Given that the embedding space is of dimension 30, we applied dimensionality reduction techniques, specifically ***Principal Component Analysis*** to reduce the dimension from 30 to 10, then we used ***t-Distributed Stochastic Neighbor Embedding*** which is well suited for the visualization of high-dimensional datasets and has helped reduce the dimension further to 2 which is much more informative than just projecting onto a 2 dimensional space.

We can see that before the training, `movie` is very far away from `film`, almost at opposite sides along each axis (similarity score is 9.83%). While after training they are much closer together, which is proof that the model is learning. Since we cannot see clearly the different distinctions between a good portion of the words, one might say that the size of the embedding space is smaller than necessary, so an augmentation in it could give more poignant results with the drawback that it would need more computing time.

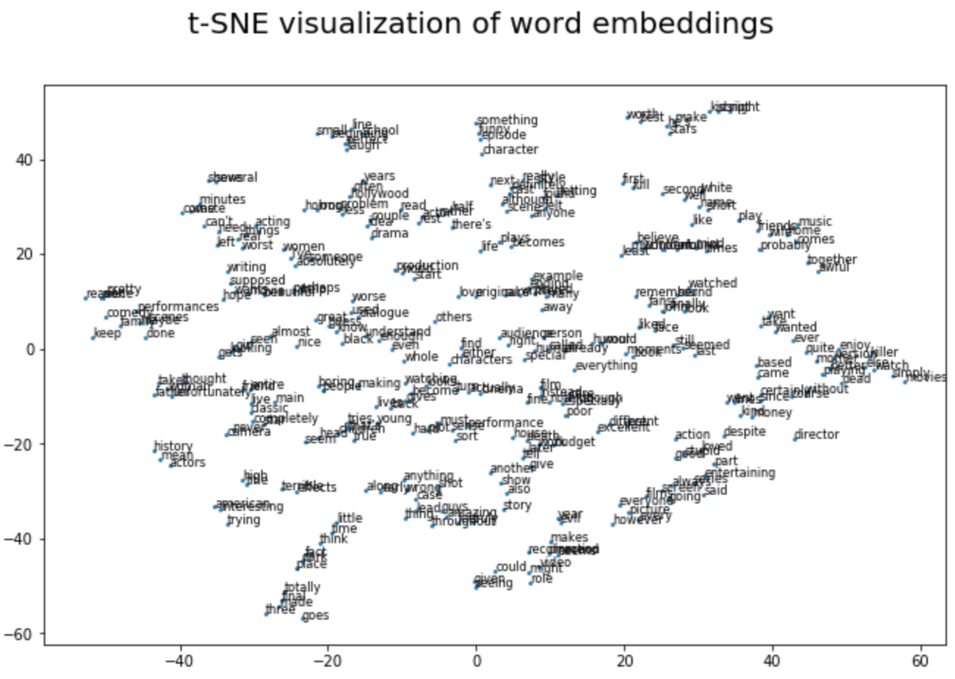

Figure 1: Word embedding of the 500 most frequent words (Loss = 15.9).

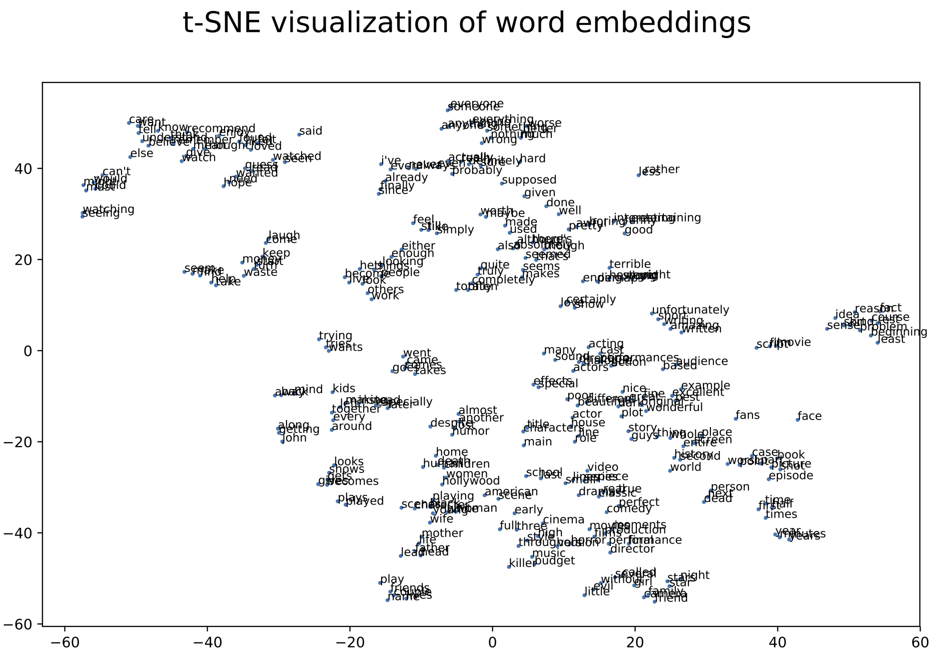

Figure 2: Word embedding of the 500 most frequent words (Loss = 2.82).

4 Question 4

We propose two ways of implementing a document embedding matrix. The first method, which is proposed in [2], aims at generating a representative vector for each document.

The point is to add another matrix to train, dubbed the **Paragraph Matrix** which lives in $\mathcal{R}^{\mathcal{N}\times\mathcal{D}}$ where N is the number of documents and D is the embedding size or dimension. Each row in the matrix is only used when looking at the corresponding document, while word vectors (rows of W_t and W_c) are shared across all documents. The task is therefore to predict the missing word in a text window given its context and the

corresponding paragraph vector. The training of both sets of vectors is done by stochastic gradient descent and the gradient is obtained via backpropagation.
In order to implement it in our pipeline, we must declare the paragraph matrix D and initialize it at random, we must also change the way we sample the data, more explicitly we should return a tuple when sample a text window. The first element of this tuple is the `id` of the document, and the second is the list of words indices. We also should alter the loss function, since we must take into account the loss induced by the paragraph vector, as well as the updated gradients of the loss where the partial derivative w.r.t the target word should be updated and the partial derivative w.r.t the paragraph vector added. Finally we must update the Paragraph matrix after each SGD step accordingly.

The second technique, which is more modular in its implementation was introduced in [1]. It comes after training the word embedding matrix, where we create a new document made out of random words (which may be non existent in the dictionary), calculating its weight using a standard weighting scheme and comparing its weights with the document in question (to be encoded). The WME (Word Mover's Embedding) 2 is computed across all documents to create the document embedding matrix.

$$\Phi_\omega(x) := \exp\left(-\gamma WMD(x, \omega)\right) \tag{2}$$

The algorithm is as follows:

Algorithm 1 Word Mover's Embedding: An Unsupervised Feature Representation for Documents

```
   Input: Texts $\{x_i\}_{i=1}^N$, $D_{\max}$, $R$.
   Output: Matrix $Z_{N\times R}$, with rows corresponding to text embeddings.
1: Compute $v_{\max}$ and $v_{\min}$ as the maximum and minimum values, over all coordinates of the word vectors $\boldsymbol{v}$ of $\{x_i\}_{i=1}^N$, from any pretrained word embeddings (e.g. Word2Vec, GloVe or PSL999).
2: for $j = 1, \ldots, R$ do
3:     Draw $D_j \sim$ Uniform$[1, D_{\max}]$.
4:     Generate a random document $\omega_j$ consisting of $D_j$ number of random words drawn as $\omega_{j\ell} \sim$ Uniform$[v_{\min}, v_{\max}]^d$, $\ell = 1, \ldots, D_j$.
5:     Compute $\boldsymbol{f}_{x_i}$ and $\boldsymbol{f}_{\omega_j}$ using a popular weighting scheme (e.g. NBOW or TF-IDF).
6:     Compute the WME feature vector $Z_j = \phi_{\omega_j}(\{x_i\}_{i=1}^N)$ using WMD in Equation (2).
7: end for
8: Return $Z(\{x_i\}_{i=1}^N) = \frac{1}{\sqrt{R}}[Z_1\ Z_2\ \ldots\ Z_R]$
```

References

[1] Lingfei et al. **IBM Research**. Word mover's embedding: From word2vec to document embedding. 2018.

[2] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, 2014.