CMPT 413/825: Natural Language Processing **(Due: 10/14/20)**

Homework 2, Fall 2020

Name: Coulton Fraser
SFU Email: coultonf@sfu.ca*, ID:* 301405411 *Students discussed with:*

Course Policy: Read all instructions carefully before you start working on the assignment, and before you make a submission. The course assignment policy is available at https://angelxuanchang.github.io/nlp-class/.

Problem 1. Vector Semantics

(a) A word vector is a $|V| \times |V|$ matrix that stores the count of neighbouring words. Say +4 and -4 words around the target word.

(b) The distributional hypothesis is the idea of words that have similar context have similar meanings. This can be used to learn representation of words in a large corpus by using measures of counts for surrounding words.

(c) Dense vectors contain fewer parameters and therefore can prevent overfitting by reducing variance. They also do a better job of capturing synonymy between words that would otherwise be in different vectors.

(d) Word2Vec cant handle out of vocabulary words. Word2Vec also treats words as a single unit; however you can have words such as meaningless that cant be broken into two parts.

(e) You could average all the word vectors in the sentence.

Problem 2. Term Frequency Representations

(a) TF-idf weights

Words	**doc1**	**doc2**	**doc3**
amazing	2.72	3.1	2.53
perfect	3.95	2.34	0
nice	2.11	0	0
good	1.83	1.96	2.25

(b)

$$\begin{aligned} CosSim &= \frac{1448}{(46.615 + 45.22)} \\ &= 0.69 \end{aligned} \tag{0.1}$$

(c) The distribution of word counts is often skewed. There can also be very common words such as 'the', 'there', 'and', etc... The TF-IDF represent a more normal distribution of word frequency through documents than just the counts. Also if a word occurs too frequently in all documents then TF-IDF will ensure that words that are common in all documents have less impact on distribution than those which are more unique to certain documents.

(d) The TF-IDF uses raw counts that have been transformed essentially. This transformation when applied to large vocabularies and corpus could be very slow. Another limitation of TF-IDF is that it doesn't capture the sequence of words, its only a bag-of-words model.

Problem 3. PPMI

(a) PPMI Weights

PPMI	**toast**	**tasty**	**bone**	**muscle**
butter	2.17	1.24	NAN	NAN
cheese	1.76	2.1	NAN	NAN
human	NAN	NAN	1.82	2.53
skeleton	NAN	NAN	2.36	-1.55

(b) PPMI Weights Laplace

PPMI L.	**toast**	**tasty**	**bone**	**muscle**
butter	2.15	1.27	-2.6	-1.84
cheese	1.72	2.02	-3.58	-3.09
human	-3.33	-2.57	1.79	2.44
skeleton	-3.36	-2.6	2.32	1.52

(c) The zero values no longer produced a divide by zero error because laplace smoothing fixed the values to be 1. This caused the values of the PPMI to be negative where ever there was a 0.

Problem 4. Learning Word Vectors

(a)

$$\begin{aligned}
\vec{w}_c &= h = \boldsymbol{W}^\top \vec{x} = W_i^\top \\
\vec{w}_t &= \boldsymbol{W'}^\top \vec{h} = W_j'^\top \\
\theta &= \{w_c, w_t\} \\
L(\theta) &= \prod_V P(w_j|w_i;\theta) \\
L(\theta) &= \prod_V \frac{e^{\vec{w}_c^\top \vec{w}_t}}{\sum_V e^{\vec{w}_c^\top \vec{w}_t}} \\
l(\theta) &= \sum_V \left(\vec{w}_c^\top \vec{w}_t - log(\sum_V e^{\vec{w}_c^\top \vec{w}_t}) \right)
\end{aligned} \tag{0.2}$$

(b)

$$\begin{aligned}
w'_{ij} &= w_{ij} + \eta^t \times \frac{\partial l(\theta)}{\partial w_j} \\
&= w_c - \frac{1}{\sum_V e^{w_c^\top w_t}} \times e^{w_c^\top w_t} \times w_c \\
&= w_c \left(1 - \frac{e^{w_c^\top w_t}}{\sum_V e^{w_c^\top w_t}} \right)
\end{aligned} \tag{0.3}$$

(c) After training, you can get the word embedding through the **W** matrix. The **W** matrix can be found by deriving the above formula in Q4b but rather than deriving for w_{ij} you would derive the equation for w_{ki}. You would then be left with the word embedding. For a specific word embedding if you had a one-hot vector, the activated position in x_k would be the k-th row.

(d) To obtain the CBOW model you would need to apply multiple contexts on the left side of the model as input. So you would have x_{Ck} and W having a matrix for each context of x fed into the hidden layer.

For word2vec skip gram, you have to compute C-multinomial distributions as output rather than just one and only 1 word vector input. Essentially the flip of the CBOW model.

These methods are computationally unrealistic unless using either hierarchical Softmax or negative sampling. Hierarchical Softmax utilizes a binary tree path for calculating output vector of words, reducing the need to sum all probabilities unnecessarily. Negative sampling is the process of only updating weights for the true vector output, and then a few samples of negative vector word outputs chosen from an arbitrary distribution.

Problem 5. Error Backpropagation

(a)

$$\begin{aligned}
&\frac{\partial}{\partial z_1^4} = \sigma(x)[1-\sigma(x)] = a_1^{(4)}(1-a_1^{(4)}) \\
&\delta_1^{(4)} = -\frac{y_n}{a_1^{(4)}}\left(a_1^{(4)}(1-a_1^{(4)})\right) - \frac{1}{1-a_1^{(4)}}\left(-a_1^{(4)}(1-a_1^{(4)})\right) + \frac{y_n}{a_1^{(4)}}\left(a_1^{(4)}(1-a_1^{(4)})\right) \\
&\delta_1^{(4)} = -\frac{a_1^{(4)}(1-a_1^{(4)})}{1-a_1^{(4)}}
\end{aligned} \tag{0.4}$$

(b)

$$\frac{\partial E_n}{\partial w_{12}^{(3)}} = argmax(a_2^{(3)})\delta_1^4 \tag{0.5}$$

(c)

$$\frac{\partial E_n}{\partial w_{12}^{(3)}} = g'(a_1^{(3)})w_{11}^{(3)}\delta_1^{(4)} \tag{0.6}$$

(d)

$$\frac{\partial E_n}{\partial w_{11}^{(2)}} = argmax(a_1^{(2)})\delta_1^3 \tag{0.7}$$

(e)

$$\frac{\partial E_n}{\partial w_{12}^{(3)}} = g'(a_1^{(2)})\sum_{k=1}^{3} w_{k1}^{(2)}\delta_k^{(3)}(g'(a_1^{(2)}) \tag{0.8}$$

(f)

$$\frac{\partial E_n}{\partial w_{11}^{(1)}} = argmax(a_1^{(1)})\delta_1^2 \tag{0.9}$$

Problem 6. Feed-Forward Neural Networks

(a)

$$a_2 = C\mathbf{w}_n^\top \mathbf{x} + \mathbf{w}_{h2}^\top \mathbf{x} \tag{0.10}$$

Since all units are linear, the two units can be added together along with the constant C and w5 added into Wn.

(b) The output of a linear unit is just a sum of the input multiplied by constant weights, so no matter how many times it is transformed it will only ever be a linear transformation of itself.

(c)

$$\begin{array}{lll} w_{11}^{(1)} = 1 & w_{21}^{(1)} = 1 & w_{1}^{(2)} = -1 \\ w_{12}^{(1)} = -1 & w_{22}^{(1)} = 1 & w_{2}^{(2)} = 1 \\ w_{13}^{(1)} = -1 & w_{23}^{(1)} = -1 & w_{3}^{(2)} = -1 \\ w_{14}^{(1)} = 1 & w_{24}^{(1)} = -1 & w_{4}^{(2)} = 1 \end{array} \tag{0.11}$$

(d) Easy to compute as it is linear above zero, no complex derivative. This will allow for faster training and inference times.
No vanishing gradient when doing gradient descent as the slope is consistent throughout the entire function.

(e) Both units in H should be linear activation functions, while the O layer will be a Sigmoid activation function.