Lab 2: Source code can be found at https://github.com/MSIA/mas7522_msia414_2019/tree/hw2. This includes a python script that can be run to reproduce the results, a jupyter notebook, and PDF version of the jupyter notebook.

1. Comparison of Word2Vec embeddings:

I altered 2 different hyperparameters to evaluate the effectiveness of Word2Vec embeddings. The two hyperparameters chosen were size and model type. I tried sizes 10, 100, and 200, and both CBOW and Skip-Gram model types. To evaluate the performance of the different models, I calculated cosine similarity for these 5 words: ‘happy,’ ‘truth,’ ‘schedule,’ ‘time,’ and ‘friend.’

The most similar word for each of these words and models is detailed in the below table:

Model	Size	Word	Similar Word
CBOW	10	happy	afraid
CBOW	100	happy	pleased
CBOW	200	happy	pleased
SkipGram	10	happy	jeez
SkipGram	100	happy	willing
SkipGram	200	happy	pleased
CBOW	10	truth	what
CBOW	100	truth	bible
CBOW	200	truth	meaning
SkipGram	10	truth	absolute
SkipGram	100	truth	falsehood
SkipGram	200	truth	ture
CBOW	10	schedule	patches
CBOW	100	schedule	mission
CBOW	200	schedule	conference
SkipGram	10	schedule	chevrolet

SkipGram	100	schedule	hosting
SkipGram	200	schedule	devilsislander
CBOW	10	time	place
CBOW	100	time	moment
CBOW	200	time	moment
SkipGram	10	time	place
SkipGram	100	time	consuming
SkipGram	200	time	fiddling
CBOW	10	friend	thread
CBOW	100	friend	wife
CBOW	200	friend	colleague
SkipGram	10	friend	girlfriend
SkipGram	100	friend	girlfriend
SkipGram	200	friend	girlfriend

For the most part, CBOW seems to find words that make the most sense, while SkipGram sometimes struggles, especially at higher size. While CBOW seems to get better with higher size, SkipGram sometimes creates illogical matches when a higher sized model is used. For example, as size increases, CBOW does better at matching the world schedule (patches at 10, mission at 100, conference at 200), while SkipGram struggles at higher size (chevrolet at 10, hosting at 100 - which makes sense, and then devilsislander at 200, which makes no sense).

2. Comparison of Scientific Papers

Aspect	Word2Vec	BERT
Year of Publication	2013	2018
Number of	15286	1831

Citations		
Software Implementation	Python, using gensim package	Python, using keras_bert package
Corpus Size	1 billion words (Google)	800 million words (BooksCorpus), also another 2,500 million words (English Wikipedia)
High-Level Reason for Algorithm Development	Interested in gleaning meaning of word from relevant surrounding words in a sentence, creating interpretations of words that do not directly relate to their original definitions. This means that phrases, not words, are given as individual tokens in a sentence. For example, sports team names combined with a city name has a different meaning than the name of the sports team and the city name in isolate (i.e. maple leaf and Toronto Maple Leafs have explicitly different representations).	The paper's authors believe that previous approaches for language representation are neither exhaustive enough or generalizable enough due to their structure. Previous approaches either tune too many parameters that affect prediction, or only work in one direction (left-to-right), gleaning context from the sentence holistically. They wish to create a language representation model that is generalizable and exhaustive, while also computationally efficient, that works bi-directionally with data that is completely unlabeled, thus reducing the need for a large quantity of fine-tuned parameters in pre-training.
Technical Approach	The overarching goal was to find representations of words that fit them to the context of the sentence, allowing for prediction of surrounding words in a sentence or document. Hierarchical softmax is used to calculate the probability of a word given the "center word." The hierarchical aspect of softmax greatly speeds up computation by utilizing a binary tree representation of probabilities, which allows for O(log2(#of words)) computational speed. There exists an inherent dataset imbalance due to rare vs. frequent words, so a sampling technique is also used to reduce this bias.	To test this algorithm, a random quantity of tokens are masked within a corpus and the remaining tokens are utilized to predict the masked ones. The second task involved "Next Sentence Prediction" (NSP), which tested whether or not a given sentence in a corpus could be predicted by the previous one. Additionally, parallelization is used so a token can be evaluated from both directions, increasing both computational efficiency and accuracy.