MCA Assignment 3 Report

Vishaal Udandarao
2016119

April 30, 2020

1 Word2Vec

1.1 Pre-processing

I took the raw texts from the abc corpus provided with the nltk library. The abc corpus contains two raw files – 'rural.txt' and 'science.txt'. I extract the raw files and concatenate the texts to form the training corpus. For the pre-processing steps, I first remove any extra whitespaces and tabs. I then tokenize the entire text using a standard word tokenizer. No extra lemmatization/stemming or stop-word removal is done. This tokenized corpus is then ready to be converted into training data that can be fed into the word2vec models.

1.2 Creation of Training Data

For creating the training data, I follow the following steps:

- I create a vocabulary set out of the entire tokenized word set after pre-processing.
- Next, I create index mappings from words to indices and indices to words. These mappings can be used later on to pass into the embedding layer as well as computing similarities between the learned word representations.
- Next, I perform two different types of vocabulary refinement:
    1. No refinement: The vocabulary set is kept the same and passed on for the next component in the data creation pipeline.
    2. Refinement with subsampling: Following the subsampling procedure followed in the original Word2Vec paper, I discard words from the vocabulary set with a probability given by:

$$P_i = \sqrt{\frac{sN}{c_i}}(1 - \sqrt{\frac{sN}{c_i}})$$

where i denotes the index of the word being considered, s is the sampling rate (taken to be 0.001) and c_i is the count of the occurrences of the word (indexed by i) in the entire corpus.

- Next, I create separate training datasets for the CBOW and Skipgram models. The creation procedures are as follow:

    1. CBOW: For CBOW, I take a context size of 2 (on either side of the target window). Therefore, my training samples are of the form: [C, T] where C is an array of 4 word indices (2 from before the target and 2 from after the target) and T is the word index of the target word. I discard all training samples whose C length is less than 4.
    2. Skipgram: For Skipgram too, I take a context size of 2. However, here training is done in a pairwise fashion. Hence, the training samples are of the form: [T, C] where T is the index of the target word and C is the index of the context word. However, due to this, the number of training samples will blow up to approximately 4 times the number of samples of CBOW.

- The final vocabulary size is 34881. The total number of training samples for all 4 configurations is given in the table below:

Model Configuration	**Number of training samples**
CBOW without subsampling	750985
CBOW with subsampling	424065
Skipgram without subsampling	3003934
Skipgram with subsampling	1697090

1.3 Training Procedure

All the 4 models were trained for 20 epochs with 50 iterations in each epoch. The batch size for every model was taken to be 1000. Training was done in a random sampling fashion in every batch by keeping track of trained batches. The embedding layer dimensionality was taken to be 50. Both the Skipgram and CBOW models were trained with the negative log likelihood loss by taking a log softmax over the model logits. All the models were trained with an SGD optimizer with a static learning rate of 0.001.

1.4 Results

1.4.1 Word Similarities

For each of the 4 models, I compute the most (top 5) similar words based on cosine similarity to the following list of words –

1. music
2. dinosaur
3. cancer
4. flight

5. support

The top 5 most similar words corresponding to each model (along with cosine similarity scores) are:

Word	**Similar word**	**Cosine similarity value**
music	whole	0.562
	insecure	0.561
	leash	0.507
	Reverend	0.503
	short-circuits	0.502
dinosaur	tightened	0.593
	Dairyfarmers	0.538
	pruritus	0.501
	McVernon	0.501
	millions	0.479
cancer	electromagnetics	0.588
	firms	0.541
	magnetised	0.527
	Hertz	0.519
	Wine	0.501
flight	integrating	0.512
	reactor	0.495
	nanomaterials	0.488
	outside	0.482
	compartmentalise	0.476
support	encircling	0.545
	fleabane	0.509
	upset	0.491
	utilisation	0.479
	retire	0.474

Table 1: Most similar words for CBOW without subsampling

Word	Similar word	Cosine similarity value
music	federal-owned	0.511
	Coleoptera	0.492
	radius	0.486
	darker	0.483
	sceptic	0.475
dinosaur	energy	0.493
	Hume	0.482
	made-up	0.48
	shorter	0.476
	geometrical	0.470
cancer	dysfunction	0.508
	hats	0.501
	Johnson	0.499
	commons	0.491
	hype	0.481
flight	switching	0.621
	chardonnay	0.512
	traversed	0.503
	waste	0.464
	worrisome	0.459
support	panned	0.539
	modalities	0.497
	warn	0.497
	expanding	0.482
	payers	0.464

Table 2: Most similar words for CBOW with subsampling

Word	Similar word	Cosine similarity value
music	luring	0.582
	Clack	0.500
	Added	0.479
	financing	0.470
	confessed	0.463
dinosaur	ferret	0.501
	human-sized	0.488
	pinpointing	0.486
	undo	0.479
	kingdom	0.470
cancer	negotiations	0.555
	tradespeople	0.528
	consolidating	0.510
	nuclear	0.506
	illness	0.497
flight	insulating	0.573
	achieved	0.505
	flaked	0.495
	solicitor	0.480
	tips	0.479
support	Americanisation	0.514
	tear	0.510
	keyboards	0.491
	admit	0.487
	forgotten	0.460

Table 3: Most similar words for Skipgram without subsampling

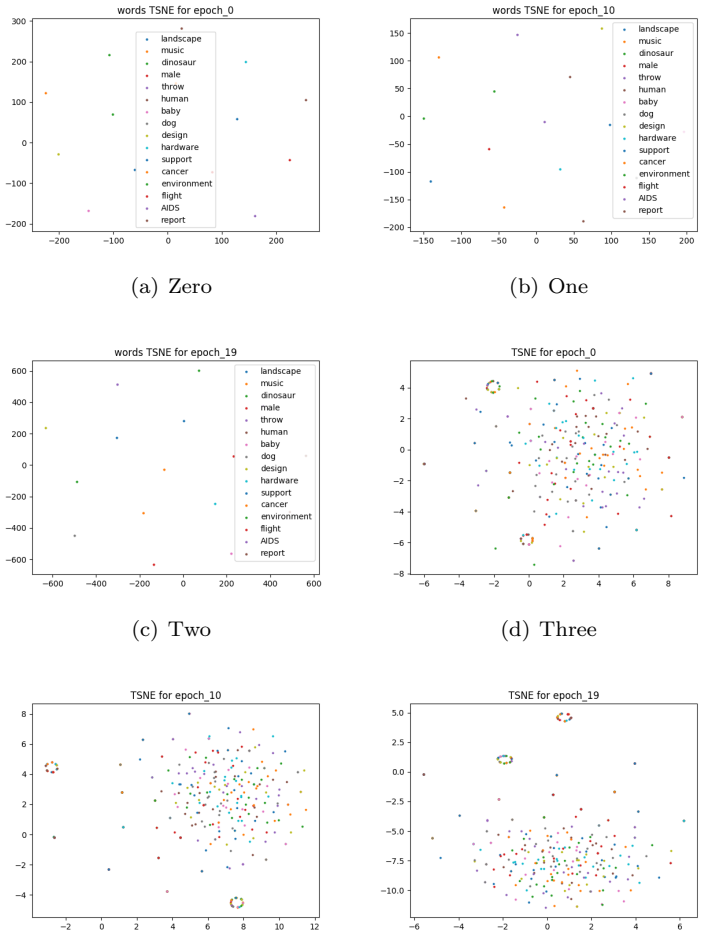
Word	Similar word	Cosine similarity value
music	Name	0.502
	orgasm	0.500
	winegrape	0.499
	in-built	0.490
	crystalline	0.475
dinosaur	principles	0.587
	Dupont	0.506
	unveiling	0.491
	arrangements	0.489
	diagnose	0.478
cancer	reviews	0.505
	inferotemporal	0.499
	rebound	0.471
	degree	0.471
	remained	0.469
flight	starch	0.538
	ultra-clean	0.514
	sliced	0.504
	severe	0.500
	unaffected	0.494
support	downsizing	0.525
	battlefields	0.483
	remoteness	0.471
	Amazon	0.467
	colt	0.463

Table 4: Most similar words for Skipgram with subsampling

1.4.2 TSNE plots

For each of the 4 model configurations, I have plotted TSNE plots across epochs to show the model convergence.

The following TSNE plots depict the representations of 20 random words chosen from the corpus.The words are labelled in the plots. I have also plotted 400 randomly sampled words across epochs.



(a) Zero

(b) One

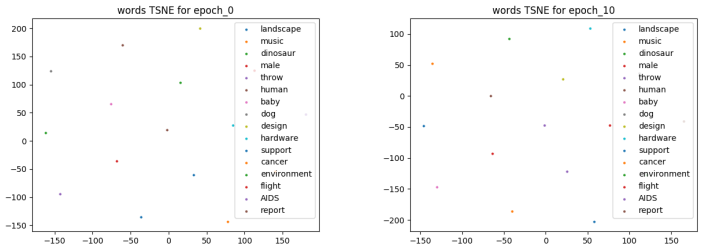
(c) Two

(d) Three

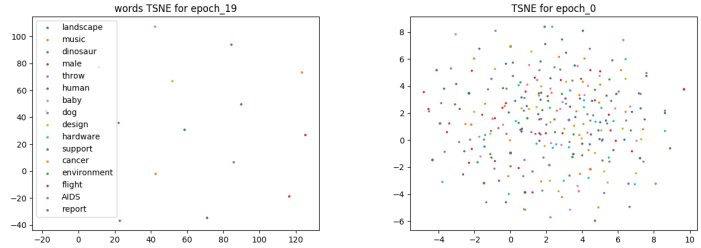
(e) Four

(f) Five

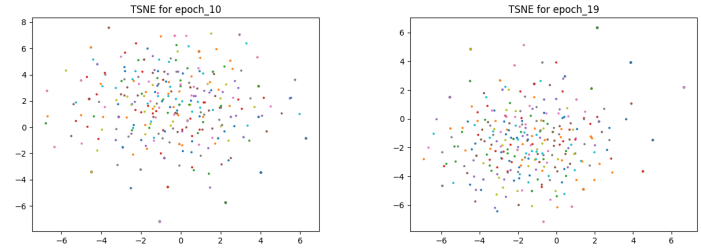
Figure 1: TSNE plots for CBOW with subsampling



(a) Zero (b) One

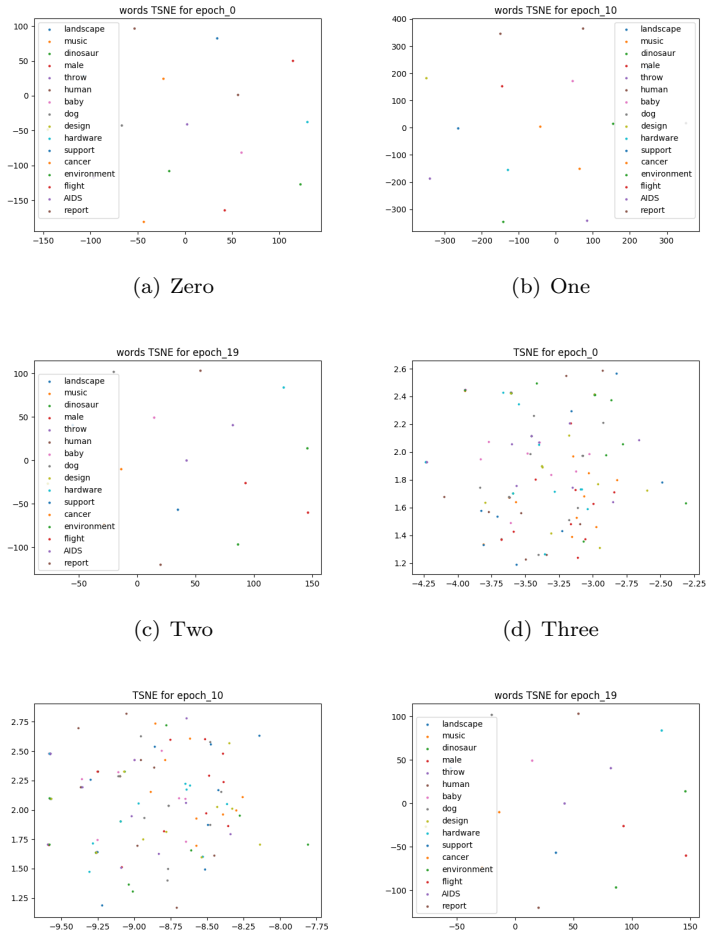


(c) Two (d) Three



(e) Four (f) Five

Figure 2: TSNE plots for CBOW without subsampling



(a) Zero

(b) One

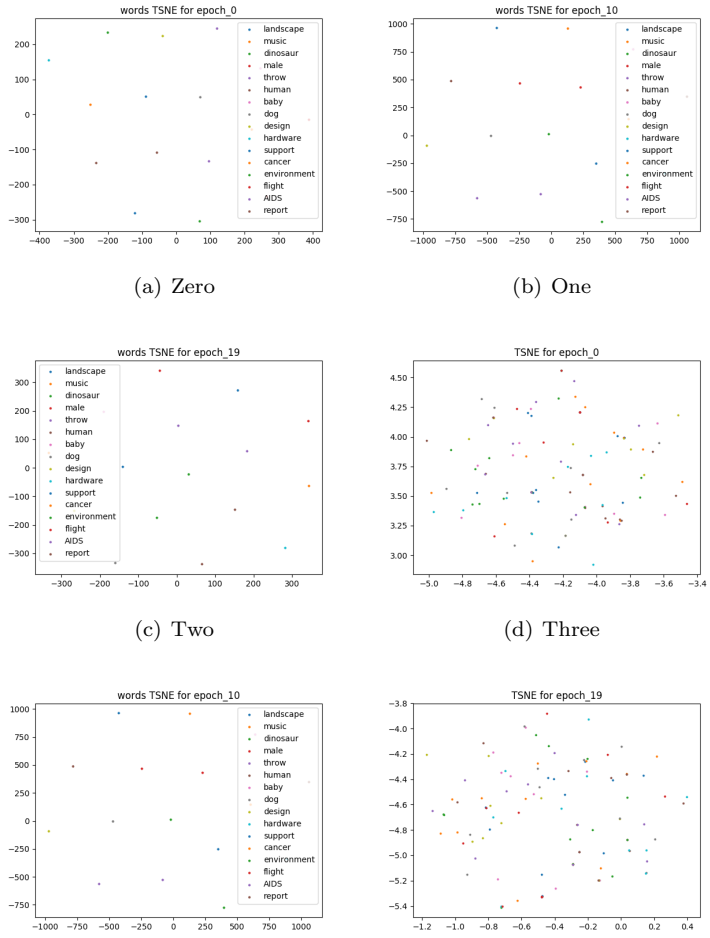
(c) Two

(d) Three

(e) Four

(f) Five

Figure 3: TSNE plots for Skipgram with subsampling



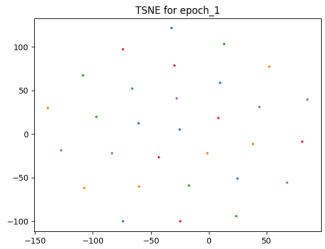
(a) Zero (b) One

(c) Two (d) Three

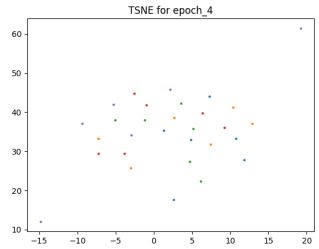
(e) Four (f) Five

Figure 4: TSNE plots for Skipgram without subsampling

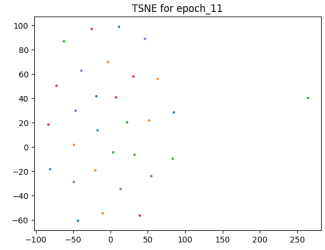
I have also plotted a visualization of 5 random words (landscape, throw, music, dinosaur and male) along with their top 5 most similar words (computed by measuring cosine similarity of terms), for the CBOW with subsampling model. The visualization is as follows:



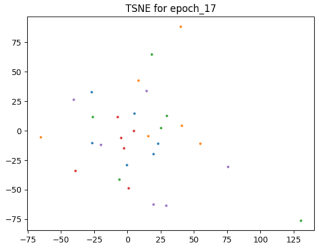
(a) Zero



(b) One



(c) Two



(d) Three

From the visualizations, we can clearly see that with increased training, more coherent clusters of similar words are being formed. This implies that although the model is still not perfectly able to distinguish between the contextual similarities of words, it is slowing learning about the semantic similarities of words. Therefore, the close clusters make sense as the number of epochs increase. The TSNE plots therefore provide further validation about the model convergence.

2 Relevance Feedback

In this question, we had to implement two relevance feedback methods. I describe both briefly:

- Pseduo Relevance Feedback: This is a simple implementation of the Rocchio query optimization method. Here, after the top documents are returned for a particular query, we take the top n documents returned and mark them as relevant and the bottom n documents are marked as non relevant. We then optimize the query based on the following equation:

$$Q_{new} = Q_{old} + \alpha \frac{1}{N_r} \sum_{i=0}^{N_r} D_i - \beta \frac{1}{N_{nr}} \sum_{i=0}^{N_{nr}} D_i$$

  This is done for three iterations. We do not need to use ground truth in

this scenario since we can directly simulate the user feedback by using a proxy relevancy and non-relevancy list as stated in this link.

- Query Expansion: This method involves modifying the initial query with some added term weights. This is done by using a global thesaurus to find similar words and then expand the query with those words. This method also helps to increase the recall of the retrieved results. In each relevance feedback iteration, I take the most similar word to the one that has the highest tf-idf value in the given query vector. I then update the tf-idf value of the most similar term in the query vector to that of the original term. The intuition behind this is that we are increasing the term weightage given to the most similar terms that are already very important to the query (due to their high tf-idf values). This is done in conjunction with the pesudo relevance feedback mechanism.

  The results obtained with both the methods (over three RF iterations) are shown below:

```
Baseline Retrieval
MAP: 0.5183859040856561

Retrieval with Relevance Feedback
MAP: 0.6104622452890981

Retrieval with Relevance Feedback and query expansion
MAP: 0.6219271897743666
```

  The changes in performance are pretty much inline with our expectations. The query expansion method is expected to perform better as it incorporates both local and global changes to the query vector. However, the query optimization method takes into account only local changes with respect to relevant and non-relevant documents for that particular query iteration. Further, the relevance feedback mechanism operates at a per-document level feedback where it optimizes the query vector based only on the relevancy and non-relevancy of documents. However, the query expansion method leverages both document-level as well as term-level feedback due to which it has a much broader optimization that it can perform on the query. Therefore, the results are as expected.