MCA Assignment [HW-3] Text Representation and Retrieval

Jay Rawal 2017240

Question1

- **Algorithm**

  The first task is to **pre-process** the textual data, so after importing the “abc” corpus we first remove all the non-alphabetical characters (except full-stop) and replace them with spaces. We also separate them, into different lines.

  Then we perform **subsampling,** we first get all the unique words and their respective counts and then with certain formula remove the highly occurring words with a probabilistic method.

  We then form two **dictionary** which has the mapping of **unique words to unique indexes** and **vice-versa**. We further use indexes to train our model.

  Then we **make pairs**, with defined **window size**. This is done so that we can train our model on the **context words and target words**.

  We then **make batches of our pairs**, in any DL library. I have used **PyTorch** to make batches of size 1000 pairs. **2700+ batches** were created.

  We then define our neural network, in our case we had **three layers.**
  - The first layer converted our pairs into **embeddings** (of size 100)
  - The second layer is the **hidden linear** layer. It gives back vocab size output from embeddings.
  - Third and final layer is a **softmax** layer.

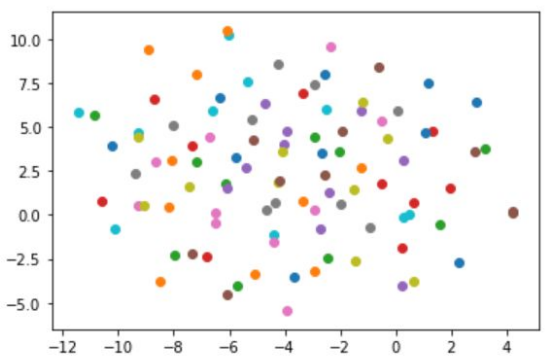
  We define an **optimizer** as well, we use **adam optimizer** to refine our network weights. We also define a function which takes in the embeddings from the first layer in our current network and uses the **TSNE model** to reduce our embedding size to make it into 2d plottable points while maintaining the distances between them.

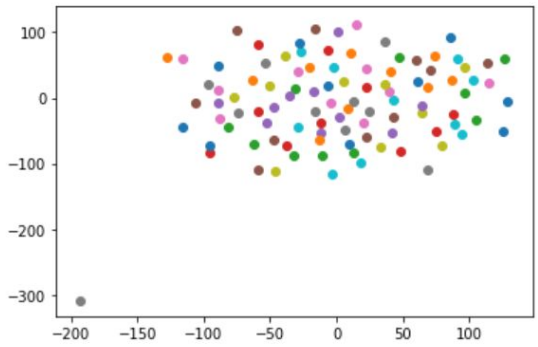
  We then **train our neural network** on our batches and perform analysis after each epoch. Results of which are shown further in this report.

- **Results**

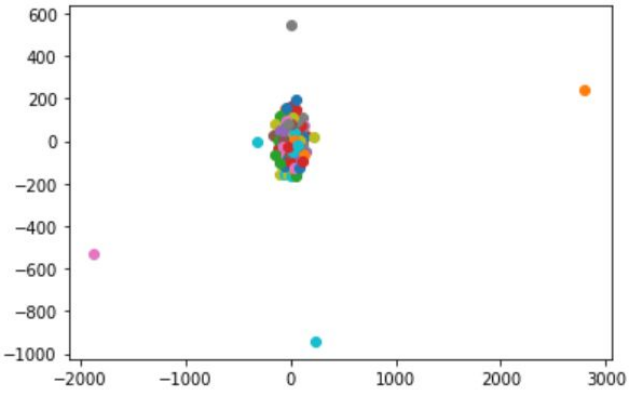
  After running the code and training our neural network, each epoch we generated the 2d representation of 100 unique words to observe the pattern. Transformation is a hefty process and thus a sub-sample was chosen. 25 epochs were calculated.

  We don't observe much variations as the words are at random and even if they would be forming clusters with their similar words, it would be hard to see.






```
Transforming number of word_embeddings  100 (100, 100)
```



  We did however observed that with some iterations, some words of the 100 random were being drastically separated from the whole bunch suggesting that their context was completely different from the others present in this small sample.

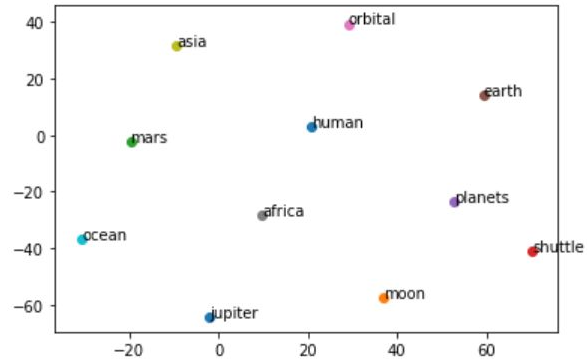
  We trained our neural net and achieved cost of **3.26**, bringing it down from the original **10.34**.

  We again ran a new model, with limited number of training vectors and 10 epochs.

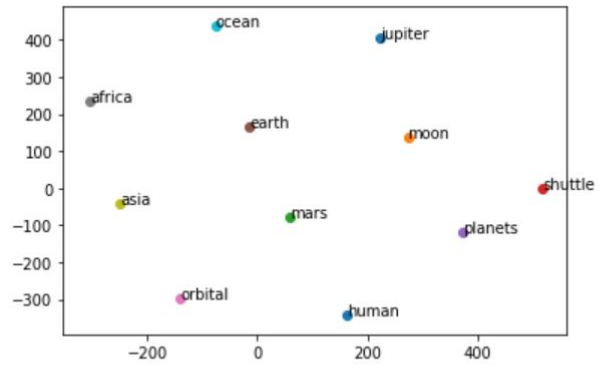
- To observe some type of clustering, a list of words were taken to see if the same meaning words come closer and different meaning drift away. Following was the result of initialized embeddings vs final embeddings.

We can observe the slight clustering (or coming closer of Asia, Africa on one side and shuttle, planets on another side).

```
['jupiter', 'moon', 'mars', 'shuttle', 'planets', 'earth', 'orbital', 'africa', 'asia', 'ocean', 'human']
```




```
iter: 1
```




```
iter: 10
```

NOTE: Training with the window size of 3, with effective batches being around 2700 making the total pairs of 23lakhs. It was a really computationally heavy task and thus running more epochs was not possible with limited personal machines.

More epochs would have resulted in better performance and even lesser cost and accurate results.

Question2

- **Algorithm**

  Relevance Feedback (vector adjustment)
  In relevance feedback we refine our query vectors with help of user feedback. We add the relevant documents vectors to our query vector (multiplied by a constant factor alpha) and subtract the non-relevant ones (multiplied by beta).

  Here we didn't have any user feedback thus we had two options:

  - **Psuedo Relevance Feedback**
    In this technique, the system takes the top-N results by itself and assume them to be relevant and similarly bottom-N as non-relevant and then perform the relevance feedback operation.

  - Taking the top-N results and matching from ground truth to see which are relevant and which are not.

  Reported below are the results of 1st case as it was mentioned in the assignment. However, case two is also present in the comments.

  Relevance Feedback (with vector adjustment and query expansion)

  Post vector adjustment in the previous part, we take the top-N terms from our document and add them to our query thus making our query even more accurate.

- **Results**
  alpha = 0.75 , beta = 0.1, iterations = 3

```
Baseline Retrieval
MAP: 0.49176786896815833

Retrieval with Relevance Feedback
C:\Users\Jay\AppData\Local\Programs\Python\Python37-32\lib\site-packa
ning: Changing the sparsity structure of a csr_matrix is expensive.
  self._set_arrayXarray_sparse(i, j, x)
MAP: 0.5796518966614829

Retrieval with Relevance Feedback and query expansion
MAP: 0.5823458089796806

C:\Users\Jay\Documents\College\sem6\mca\HW-3\src\Problem_2>
```

We observe that the **baseline algorithm** gives us a result of 0.49. This result would now help us to see the increase resultant of our relevance feedback algorithms.

For (pseudo) **relevance feedback with only vector adjustment**, we obtain the result of 0.57 which is significant increase thus showing that our algorithm is helping in achieving better results. This was on expected lines, as we are getting closer to the accurate query meaning by adding relevant docs weights.

Then further when we added the terms for **query expansion**, we got a minor increase with the result now being 0.58. I expected a more significant increase than what was obtained but this could also be due to the fact that the top-N terms aren't accurately defining the purpose of the query.

Also, another **possible factor** could be that the user feedback is not the actual one and this is just pseudo relevance feedback because of which accuracy is lower.

EXTRA:
If we take ground truth to see assumed user feedback for top-N documents, we get a higher accuracy of 73% with the same alpha, beta and number of iterations for vector adjustment relevance feedback.
