1 Introduction

- This report provides a summary of the work conducted on implementing and evaluating the Word2Vec model. The program will be developed in Python with the help of the pytorch module. All use will be allowed with the exception of it's torch.nn.embedding layer.
- Problem Description: tasked to program and train a simple word embedding (word2vec) model.
- Methods and metrics planned for evaluation: For initial evaluation of model the Cross Entropy Loss value will be used.
- Outline of solution: The Word2vec CBOW model will be used as CBOW is found to train faster than Skip-Gram, and can better represent more frequent words[1]

2 Data

- Examples/samples of dataset(s): Tiny.raw and Large.raw
- Key characteristics: Lines of raw text
- Necessary data preparation for further processing: pre-processing such as word segmentation, tokenization, normalization and lower-casing. Most of these are necessary, optional preparations are such as removing stop words.

For the Word2Vec CBOW implementation, a custom pytorch dataset was used to be later compatible with torch's dataloader and usable with other models if needed. The required methods of init,getitem and len were used so that it can be passed on later. We initialise the dataset with the selected file of tiny (large is optional, but tiny was chosen so that it's training time will be lower and the lack of GPU training on student's laptop). the dataclass has a few attributes such as vocabulary which houses all sets of tokens, word_idx which houses a dictionary to convert all words to an integer obtained using enumerate() method and idx_word which was used for testing purposes. The dataset will be elaborated more on the next section

3 Methods

The Word2Vec CBOW model is based on the context of words and target within a given corpus. It predicts a target word based on its surrounding context words. The model achieves this by training a shallow neural network with a single hidden layer. The network learns to optimize word embeddings, representing each word as a dense vector in a continuous space.

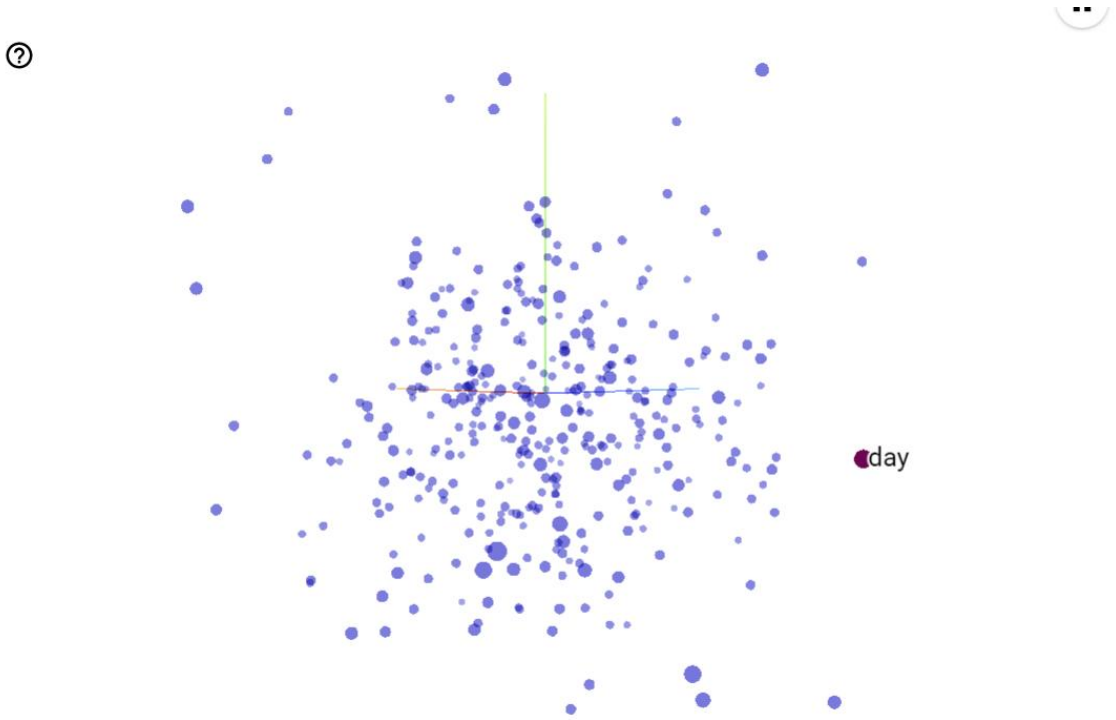
With the knowledge of what CBOW is, the dataset continues to be built expecting that the getitem method returns the target value and the context. We convert text to indices using word_idx. An example is [hi,i,am,a,boy] into [3,5,6,7,8] and then get a target and the context words (which is now actually an integer). This is also known as one hot encoding.

The target must have same amount of context words on both sides according to [2]. Using the previous example of [hi,i,am,a,boy] and a window size of 1. The available targets are only [I,am,a] and the context words are the 1(window_size parameter) word(s) to its left and right of the target.

The dataset was later passed on to the dataloader. With the modification of original dataset finally done, we proceed on to the CBOW neural network method which can be done with pytorch. We first need to apply an embedding layer which is used as trainable "lookup" tables using the input as indices. So we just have to initialise a random matrix with a dimension of [vocab_size,embedding_dim]. And example would be now that the matrix's row index of 3, which is the word hi now has 10 random numbers. It is then sent to a linear layer then a softmax layer and later backpropogated to optimize the weights.

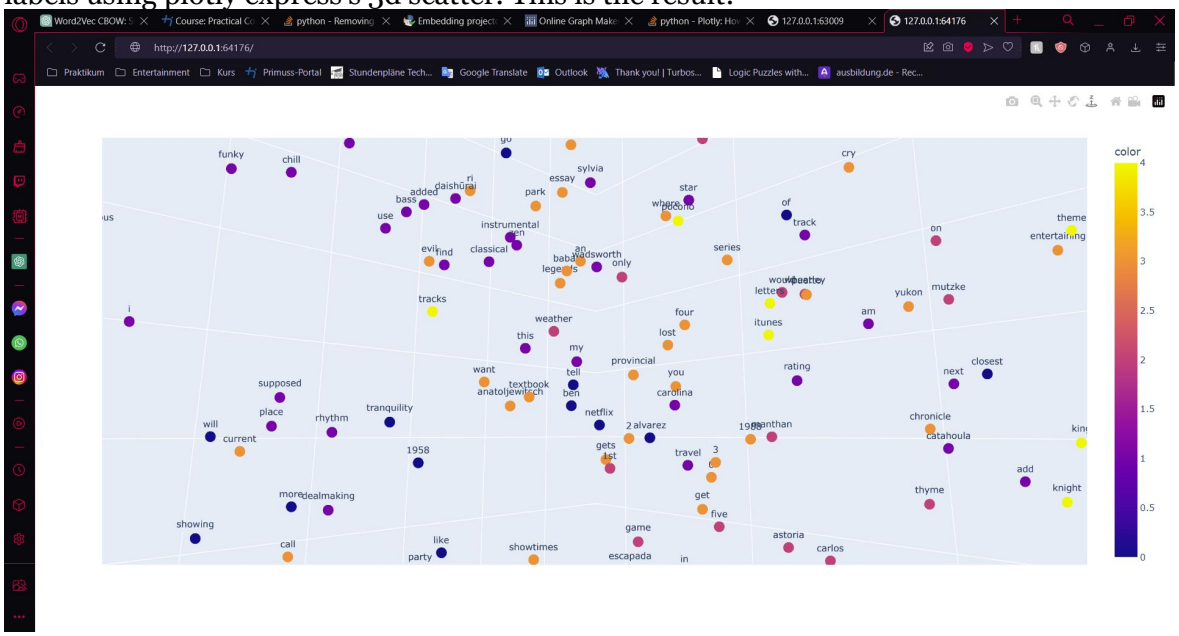
4 Experiments

To evaluate the Word2Vec CBOW model, a transparent experiment design was followed. Various hyperparameters, such as the embedding dimension, window size, and learning rate, were selected to find the optimal configuration. The model was then trained on the dataset. The embedding matrix was then used to make 2 tsv's to be put into an open source embedding projector on tensorflow "https://projector.tensorflow.org/". This is the following result:



The open source projector used TSNE to find that the 10 embedding dimension weights have different weights that affect the embedding of the words.

To cluster the words, I used sklearn's kmeans clustering into 5 clusters then did a TSNE using sklearn's TSNE method and fitted my embedding matrix's weight. I then plotted the labels using plotly express's 3d scatter. This is the result:



5 Evaluation and discussion

Using the results obtained from above, there are good clusters with words such as the 1st cluster being the words: zen,classical, instrument,bass, funky, chill, rhythm that are clustered correctly, but somehow also a word that is expected to be clustered together with the 1st cluster which arent such as tranquility being clustered together with the 2nd cluster : showing, netflix, alvarez, seats, patricks. Dying and die are also not clustered together.

I am quite puzzled as to why these clusters can be so right at times and wrong at times. I attribute these to the tiny dataset size, Polysemy and Word Ambiguity, Context window limitations and loss of rare word Information.

6 Conclusion

Overall, addressing potential problems can enhance the performance and robustness of the Word2Vec CBOW model for computing word embeddings, allowing it to capture a wider range of word relationships and handle various linguistic challenges

References

[1] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[2] Goldberg, Y., Levy, O. (2014). Word2Vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.

[3] Camacho-Collados, Jos e Pilevar, Mohammad Taher. (2018). From Word To Sense Embeddings: A Survey on Vector Representations of Meaning. Journal of Artificial Intelligence Research. 63. 743-788. 10.1613/jair.1.11259.