**Text

Description automatically generated**

**Word Embeddings**

**Coursework 2024/2025**Steve Galea (0015902H)

Study-unit: **Topics in Applied Data Science**

Code: **CIS5231**

Lecturer: Prof John Abela

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

Declaration Plagiarism is defined as “the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the assignment submitted is my work, except where acknowledged and referenced.

I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

A close up of a signature

AI-generated content may be incorrect.Work submitted without this signed declaration will not be corrected, and will be given zero marks.

STEVE GALEA \_\_\_\_\_\_\_\_\_\_

Student Name Signature

Word Embeddings Coursework 2024/2025

CIS5231

Course Code Title of work submitted

18/07/2025

Date

# Table of Contents

# Introduction

Applications of word embedding techniques are wide-ranging — from text classification, text generation (such as summarisation), and question answering, to speech recognition and machine translation. To achieve the above, word embedding techniques map words, along with their surrounding context at times, into continuous vector spaces [1].

Word Embeddings are a type of Vector Space Model used in Natural Language Processing (NLP) that are learnt from large data corpora in an unsupervised manner. These models identify patterns from language data sources and represent them as dense and continuous vectors. Unlike one-hot encoded data, which is a sparse representation, using such vectors for representations offers various benefits [1]. Within the domain of NLP, word embeddings can be learned using prediction-based models that rely on local context or through count-based models that leverage global word occurrences [11].

Dense vectors are computationally less expensive and do not require human curation or extensive manual feature engineering to represent a word numerically. The word vector is determined effectively through its context, and such vector embeddings enable models to handle noise or missing data better, make generalisations and inferences from unseen data. This is because word embeddings capture the semantic similarity and cluster words with similar meaning together in the embedding space. This is very unlike rule-based symbolic methods, relying on direct human labelling and offering limited scalability and vocabulary. In dense vector word embeddings[1-3]. Dense vectors produced by earlier models like Word2Vec are static representations, meaning that the vector does not account for revision of the vector representation, even if the context is different. This limitation of polysemy and homonymy is addressed by later models like BERT and ELMo [3]. These vector embeddings have become foundational components of modern neural architectures and deep sequence models [1,3, 8-11]. Notably, Vaswani et al. [10] incorporated them into the multi-layered Transformer architecture, which ultimately paved the way for models like ChatGPT.

# Literature Review Of Word Embedding Techniques

Mikolov et al. [4] and Pennington et al. [5] produced Word2Vec and GloVe models, respectively, to come up with highly accurate semantic word encodings. Other models, such as FastText, created by Joulin et al. [6] and Bojanowski et al. [7], BERT by Devlin et al. [8], or ELMo by Peters et al. [9], aimed to address limitations in the vectors encountered by the previous two models.

Mikolov et al [4] created the Continuous Bag Of Words (CBOW) and Skip-gram architectures to compress words from very large datasets into a dense set of vectors via a log-linear classifier. CBOW and Skip-gram both make use of feed-forward functions, aiming to learn word representations. CBOW employs a feed-forward network function which accepts as input a context of many words, with a missing middle target word. CBOW hence tries to predict the central target word from the surrounding words. The order of the words does not matter. Skip-gram is the opposite, where with a target word as an input, it tries to predict the surrounding context. It is more complex computationally than CBOW when the context window range is increased to consider distant word vectors; however, the quality of the overall word vectors is increased. These two architectures were tested with the Google News 6B token corpus, and the vocabulary tested was limited to the most frequent 1 million, and later 30 thousand words, due to time constraints. The experiments carried out involved experimenting with 1 or 3 epochs; varying (e.g. 300 or 600) embedding dimensionality; 783M or 1.6B words, and of course CBOW or Skip-gram. The model was trained with mini-batches, backpropagation and Adagrad stochastic gradient descent (learning rates 0.025 to 0) on a single CPU. Large-scale tests were carried out as well with multiple CPU instances using the full 6B dataset, and 1000-sized embedding dimensions for both architectures. For word similarity and analogy evaluation tasks, various semantic and syntactic questions were carried out to evaluate the accuracy, with the authors coming up with various relationships (e.g. France – Paris, USA - pizza, cold - colder). n. With larger datasets and larger embedding dimensions, the authors indicate that the performance increases, and high-quality embeddings can be achieved with a simple architecture.

A diagram of a diagram

AI-generated content may be incorrect.

Figure .: The above figure was replicated from Mikolov et al. [4], showing the two distinct architectures.

Pennington et al. [5] combine the benefits of the global matrix factorisation methods (e.g. latent semantic analysis (LSA)) and the local context window models achieved by Mikolov et al [4] (Skip-gram). GloVe counts the word occurrences in the provided corpus as a word-context co-occurrence matrix, extracts probabilities and factorises them. This effectively captures the global probabilities and relationships within the corpus, unlike Word2Vec, which focuses on a local window context. The same Semantic and Syntactical tests as Mikolov et al. [4] were carried out over different models and hyperparameters. The datasets used were Wikipedia dumps from 2010 and 2014 and Gigaword5, to obtain 6 billion tokens. The top 400,000 most frequent words were used, 300+ vector embeddings (trained with 50 epochs for <300, but 100 epochs for >300), and Adagrad with an initial learning rate of 0.05 was used. An investigation into the left and right sentence contexts was made. The GloVe model outperformed the other baselines, providing superior and faster word vectors and robustly supporting larger datasets. Word similarity, word analogy, and named entity recognition are the NLP tasks for which the GloVe model appears to outperform.

Word2Vec and GloVe, however, both suffer from out-of-vocabulary words as a top X filter is done to limit the vocabulary [3-5]. With this in mind, authors [6-7] created FastText, an extension of the skip-gram model, and in it, it represents each word as an n-gram of characters. The authors made use of Czech, French, Spanish, German, English and Russian Wikipedia dumps for training. The model used was an RNN with 650 LSTM units and trained by Adagrad. It was found that morphologically richer languages like German tend to separate compound nouns into morphemes. Besides that, the FastText model can generate word vectors for out-of-vocabulary words by averaging the n-grams [6-7]. Unfortunately, the n-grams model may not scale, and it can grow rapidly in size, possibly posing a notable limitation.

The limitation of static embedded vectors is a pertinent issue across the simpler models created by [4-7]. BERT by Devlin et al. [8] is a multi-stacked transformer-based bidirectional encoder-decoder architecture that directly generates context-dependent embeddings in the feedforward loop. Two token embeddings are created within BERT to create a dynamic vector representation for the same word, but different contexts through attention mechanisms; the first is WordPiece tokenisation to represent the syntax and semantics, and the second is tokenisation for the position. These two embedded encodings properly handle the limitation of polysemy and homonymy, and the use of bi-directional context, unlike the word embeddings of [4-7] can consider the entire context (instead of a window in Word2Vec Skipgram[4] and FastText [6-7]) from both directions. ELMo by Peters et al. [9] similarly makes use of a bidirectional LSTM architecture to generate dynamic context-dependent embeddings.

# Design Decisions:

## Acquisition and Pre-Processing (1\_data\_pre\_processing.py)

The News Articles[[1]](#footnote-1) dataset was acquired from Kaggle. It contains 2,692 scraped news articles and headlines related to business and sports from 2015 to 2017. For this assignment, since computations will be carried out on a CPU and a domain-specific dataset is needed, the **sports** articles (1408 records) will be focused on.

A screenshot of a computer

AI-generated content may be incorrect.

Figure .: Initial raw data set, with business and sports news articles

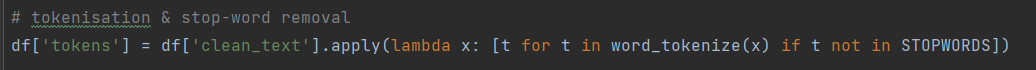
Pre-Processing Steps:

1. Text Normalisation: Convert all text to lowercase to maintain consistency.

2. Punctuation Removal: Remove punctuation using regular expressions to clean the text, as per the following code snippet.

A screen shot of a computer program

AI-generated content may be incorrect.

3. Tokenisation: Split text into individual tokens (words) & Stop-Word Removal: Filter out common stop words to reduce noise and focus on semantically rich words..

5. Vocabulary Limiting/Filtering: Construct a vocabulary of the top 10,000 most frequent words, replacing infrequent words with an <UNK> token. Lemmatisation was carried out using **spacy**. Indexing was also carried out, where I converted tokens into numerical indices based on the constructed vocabulary

A computer screen shot of a program

AI-generated content may be incorrect.

7. Dumping Data created -> the data created was dumped into the Data folder.

A screen shot of a computer program

AI-generated content may be incorrect.

Sample script run:

A screenshot of a computer

AI-generated content may be incorrect.

## Models in models/cbow.py and models/skipgram.py

* The models were kept simple, with no error backpropagation. Also, the models created follow using ChatGPT (prompt in the ChatGPT log attached), and adapted to match 2 source code links to create the CBOW and Skipgram Code and training loop:
  1. Prompt in the appendix
  2. [pytorch-continuous-bag-of-words/cbow.py at master · FraLotito/pytorch-continuous-bag-of-words · GitHub](https://github.com/FraLotito/pytorch-continuous-bag-of-words/blob/master/cbow.py)
  3. [Skip-Gram Word2Vec Algorithm Explained | by Ido Leshem | Medium](https://leshem-ido.medium.com/skip-gram-word2vec-algorithm-explained-85cd67a45ffa)
* One-hot encoding was not used, as dense vector representations are preferred. Instead, training was performed using mini-batches and computing the CrossEntropyLoss & optimisers upon was preferred and put in **2\_and\_3\_pytorch\_model\_training\_and\_optimisations.py** handle the training and stochastic descent with Adam.
* Softmax was implemented in both CBOW and Skipgram models to get log-probabilities.

## Training Process (2\_and\_3\_pytorch\_model\_training\_and\_optimisations.py)

## Training Loop Implementation

## Hyper-Parameter Tuning

The below experiments were run. Not every combination of mode

* Batch sizes: 16 (CBOW only), 32, 64, 128, 256
* Epochs (including checkpoints): 10, 20, 30, 40, 50 or 100
* Window sizes: 2, 3, 4, 5, 10
* Learning rates: 0.0001, 0.001, 0.01, 0.05
* Models: cbow, skipgram

## Performance Monitoring

The models’ performance was measured through training loss. Models with the lowest training loss and at a converged state were considered to have finished training. All obtained Loss Plots are listed in the appendix.

## Adaptation

To complete within a 2-3 hour training window on a CPU, one should lower the epoch count. Note, these were estimated, but should keep to a 2-3 hour window.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Epochs | Learning Rate | Window size | Batch size |
| CBOW | 100 | 0.01 | 2 | 128 |
| SkipGram | 15 | 0.01 | 5 | 256 |

For CBOW, ~50 epochs were observed to fit in a 2-3 hour window, but for SkipGram, this took longer due to the more complex processing.

## Visualisation Techniques Used

A training loss against Epochs curve was used to determine convergence (e.g. **loss\_plot\_skipgram\_model\_256b\_0.05l\_10e\_10w.pth.png**).

A computer screen with text

AI-generated content may be incorrect.

Also, for each model, all TSNE was used to reduce the high-dimensional embeddings of 10,000 words to a 2D space to visualise and plot any clusters (e.g. file **top\_10000\_tsne\_word\_embeddings\_skipgram\_model\_256b\_0.05l\_10e\_10w.pth.png**). To supplement this, a dedicated csv file with all coordinate and word data was created to describe points and clusters (e.g. **tsne\_embeddings\_skipgram\_model\_256b\_0.05l\_10e\_10w.pth.csv**)

In addition, the following categories were manually created to have a more manageable visualisation to interpret. The conditions for the categories and the words were chosen intuitively and were derived from the dataset.A computer screen shot of text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

# Discussion of results

# Challenges Encountered

* Hyperparameter optimisation was carried out with both CPUs and GPUs offered by Google Colab and Kaggle. Despite that, due to time constraints, further hyperparameter optimisation was not possible.

# Possible Enhancements

* Make use of a different dataset
* Feed larger datasets (6 Billion) and decrease epoch count
* Make use of error back propagation
* Make use of a different optimiser, like Adagrad, used by Mikolov et al. [4]
* Make use of a different loss function, instead of CrossEntropyLoss
* Make more use of GPUs
* Further hyperparameter experimentation:
  + Mini-batches can be made smaller
  + Further window sizes
  + Further experiments with epoch counts

# References

[1] J. Abela, ‘JA117 NLP Transformers & Large Language Models (LLMs) Part 1 – Introduction and Concepts’, University Of Malta, 2025.

[2] J. Abela, ‘JA117 NLP Transformers & Large Language Models (LLMs) Part 2 – Preliminaries’, University Of Malta, 2025.

[3] J. Abela, ‘JA117 NLP Transformers & Large Language Models (LLMs) Part 3 – Word Embeddings’, University Of Malta, 2025.

[4] G. C. J. D. Tomas Mikolov Kai Chen, “Efficient Estimation of Word Representations in Vector Space.” 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[5] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532–1543, 2014

[6] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, “FastText.zip: Compressing text classification models.” 2016. [Online]. Available: https://arxiv.org/abs/1612.03651

[7] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information.” 2017. [Online]. Available: https://arxiv.org/abs/1607.04606

[8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[9] M. E. Peters et al., “Deep contextualized word representations.” 2018. [Online]. Available: https://arxiv.org/abs/1802.05365

[10] A. Vaswani et al., “Attention is All you Need,” in Advances in Neural Information Processing Systems, I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[11] F. Almeida and G. Xexéo, “Word Embeddings: A Survey.” 2023. [Online]. Available: https://arxiv.org/abs/1901.09069

# ChatGPT usage

Search terms for the literature review were suggested

# A screenshot of a computer AI-generated content may be incorrect.A screenshot of a computer AI-generated content may be incorrect.

ChatGPT also aided in the highlighting of results from Pennington et al. [5], suggesting what to focus on in the literature review.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

ChatGPT provided a starting point in establishing a boilerplate model to train. Refinements were done by referring to other sources like:

* 1. [pytorch-continuous-bag-of-words/cbow.py at master · FraLotito/pytorch-continuous-bag-of-words · GitHub](https://github.com/FraLotito/pytorch-continuous-bag-of-words/blob/master/cbow.py)
  2. [Skip-Gram Word2Vec Algorithm Explained | by Ido Leshem | Medium](https://leshem-ido.medium.com/skip-gram-word2vec-algorithm-explained-85cd67a45ffa)

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

Code-wise, ChatGPT helped as well in the design choices: A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a black screen

AI-generated content may be incorrect.A black and white text

AI-generated content may be incorrect.

ChatGPT also helped to provide Pytorch lines of code that help with acceleration: A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

# Source Code Appendix

1. https://www.kaggle.com/datasets/asad1m9a9h6mood/news-articles [↑](#footnote-ref-1)