Synthetic Word Embedding Generation for Downstream NLP Task

Hoang Viet

School of Computer Science and Engineering

Supervisor: Assoc Prof Chng Eng Siong

Submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Bachelor of Computer Engineering

2021

Acknowledgements

This project was made possible by the support and guidance of the following people who helped me during the project. I would like to express my greatest gratitude and appreciation to them.

Firstly, I would like to express my sincere thanks and gratitude to Associate Professor Chng Eng Siong for his guidance and thoughtful advice throughout the duration of this project despite his busy schedule. His given directions and support have been vital to the success of the project and I am truly appreciative to have been given this opportunity to explore and conduct this study under him.

Prof. Chng have also kindly engaged his team to have weekly meetings with me to share my progress and get valuable advice that helps address the hurdles along the way. I would like to thank members of the team, specifically Damien Cheng, Ms Vu Thi Ly, Ms Ho Thi Nga and Mr Kyaw Zin Tun for providing their valuable feedback and keeping us connected to the rest of the team.

Lastly, I would like to express my gratitude to my family and friends who provided the much needed support throughout the project.

Abstract

Distributional word representation such as GloVe and BERT has garnered immense popularity and research interest in recent years due to their success in many downstream NLP applications. However, a major limitation of word embedding is its inability to handle unknown words. To make sense of words that were not present in training, current NLP models use sub-word embedding (obtained via sub-word segmentation algorithms). However, this approach often fails to capture the semantic sense of the word due to words being broken down in a syntactic manner. There have been other approaches to tackle embedding of unknown words using ConceptNet and Recursive Neural Network, but did not enjoy much usage due to their complexities in design. This report presents a novel solution to generate embedding for OOV using a neural rather than symbolic approach. This approach capitalizes on the existing semantics captured in known words' embeddings and trains a simple feed-forward neural network to capture the compositionality function of embeddings in their latent space. Linguistic studies have shown that the compositionality function is broad and varied, therefore this report introduces a preliminary study into the compositionality of noun, with focus on certain named entities. The trained network is able to generate an embedding for an unknown word based on its context words, which can be obtained via crawling of web data. This synthetic embedding can then be incorporated into the embedding matrix of existing application. From the experiments on GloVe and RoBERTa embeddings, it can be concluded that the synthetic embedding is a feasible light-weight option that can supplement the understanding of many NLP downstream applications due to its ease of synthesis, the quality of the embedding over subword tokens and the short amount of time it takes to generate embedding for unknown words.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Word embedding as a dense representation of word has received unparalleled popularity among NLP practitioner since its inception compared to other sparse representation such as Brown Cluster or LSA features. Mikolov et al first introduced the word2vec model in 2013 [4] and in 2014, GloVe embedding was introduced by Pennington et al [5], GloVe embedding was trained using an architecture similar to CBOW of word2vec, with a slight tweak in objective function, as shown in Figure Figure 1.1 below.

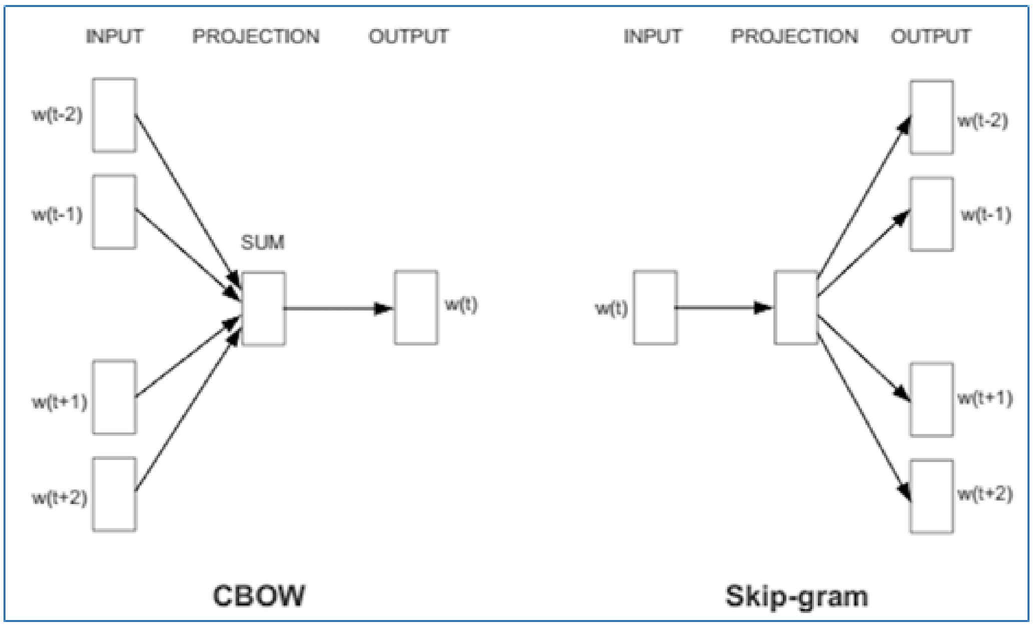


FIGURE 1.1: Architecture of the word2vec model

GloVe authors showed that the ratio of two words' co-occurrence probability (rather than their co-occurrence probabilities themselves) is what conveys information, and that this information is encoded as vector differences. GloVe acquired a lot of traction since they seemed to consistently and significantly outperform standard Distributional Semantic Models. GloVe embedding, however, does not deal well with the problem of polysemy, where the same word has different semantics in different context. This limitation inspires the creation of contextual embedding, with the introduction of transformer models in 2018. Introduced by Vaswani et al in the groundbreaking "Attention is All You Need" paper [3], the Bidirectional Encoder Representations from Transformers (BERT) is a language model that produce tokens which are hugely useful across many NLP tasks. It is distinguished from previous language models by the fact that its learnt representations include context from both sides of the sentences from the architecture that is shown in Figure Figure 1.2 below.

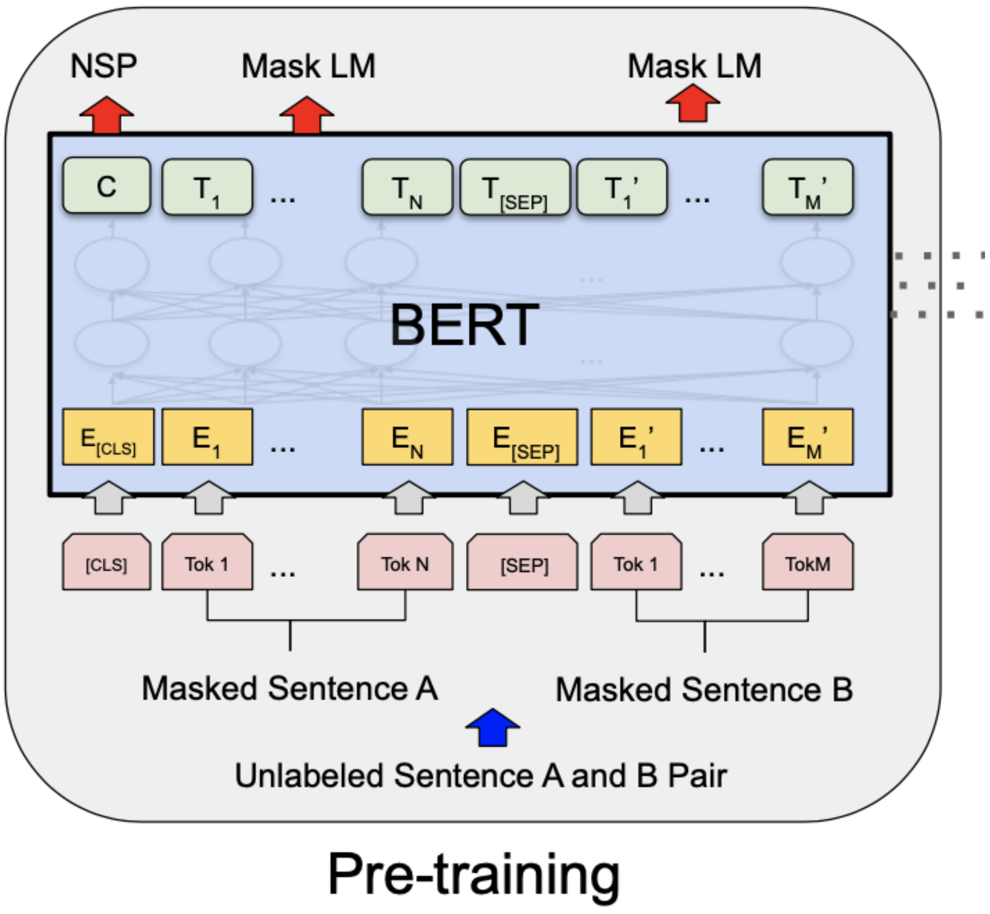


FIGURE 1.2: Bert Architecture

Despite the major advancement in word embedding research over the last decade, dealing with unknown words still remains as one of the most challenging problems for research and production NLP system. Every NLP model is limited by a fixed-size vocabulary and hence limit the amount of meanings the model can encapsulate about the world. GloVe has a vocabulary size of 40 000 words, and treat any unknown words as OOV while BERT boasts a vocabulary of 30 000 tokens that make uses of sub-words and characters to make sense of an unknown word. As both system are trained on huge corpus of data using tremendous computational power, it is inefficient to retrain the embedding to incorporate new words as NLP systems adapt to the ever-changing environment that they have to operate in. Therefore, in this report, we propose a simple method to synthesize a reasonable embedding for a given unknown word. The system is designed to assist mainly in production system, where domain-specific and new words are rare in training corpus but common in production due to high usage from users. An overview of the system supported by this report is shown in Figure Figure 1.3 below.

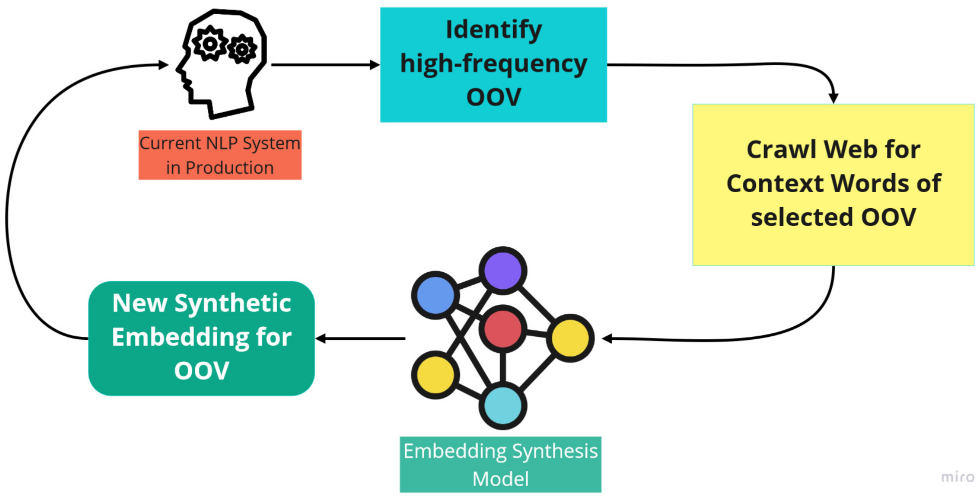

Current NLP System in Production
Identify high-frequency OOV
Crawl Web for Context Words of selected OOV
Embedding Synthesis Model
New Synthetic Embedding for OOV
miro

FIGURE 1.3: Workflow of Embedding Synthesis

The workflow of the system is as below:

1. Current system identify high frequency unknown words that are of high importance
2. For each word, crawl wiki and related news article about the word
3. Pre-process each word and feed it into the embedding synthesis network
4. Obtain the new embedding and incorporate it into the current system either via embedding matrix or replacement of embedding

The system is designed to be light-weight and fast to avoid disruption to NLP services in production. The synthetic embedding will help in downstream task to understand unknown words better, with negligible loss. Compared to sub-word segmentation, the synthetic embedding can capture much more semantics information of the word, especially for named entities such as Pfizer. In contrast to pretraining the whole network with OOV corpus, and other sophisticated methods such as ConceptNet ensemble on GloVe [3] or Morphology Recurrent Neural Net [6], the approach in this study is significantly faster and resource-efficient.

To evaluate the quality of embedding, we follow the suggestion in [7]. Relatedness is considered the primary evaluation metric, as analogy is inappropriate in the setting of OOV (which are often rare words). Categorization and Selectional Preference is a feasible metrics but prove to be more challenging to implement.

1.2 Objectives

Embedding synthesis is a challenging problem in the research community as there exist no objective metric nor method in creating perfect embedding. Many approaches have been implemented and each of them suffer from their unique limitations, be it resource-heavy or accuracy. In this report, the primary objective is to conduct a study on existing methods and understand their advantages as well as disadvantages. From these understanding, we can create a proof-of-concept model that benefit from the advantages of existing approaches while mitigate as many of the pitfalls as possible.

The aims of this report are:

1. Design a neural network architecture that are light-weight and is capable of synthesize word embedding
2. Conduct experiments with different data and hyper-parameters choices on the architecture
3. Generate synthetic embedding for a subset of unknown words for GloVe and RoBerta and evaluate these embedding
4. Provide a method to incorporate existing embedding into system that use GloVe embedding

By securing the above objectives, we will be able to effectively synthesize high-quality embedding in a semi-autonomous manner, thereby greatly reducing the cost and need to retrain the entire network to incorporate meanings of unknown words. As many downstream NLP application such as QA-chatbots and NER suffer from a lack of understanding of unknown words, especially domain-specific named entities, the approach presented in this report can assist NLP engine in production environment by enhancing their capabilities in understanding the text data presented to them by users.

1.3 Scope and Assumptions

The scope of the synthetic embedding generated by the approach in this report is limited to unknown nouns (for training) and named entities of special interest to production such as Pfizer and COVID-19 (for testing). All the related process and methods are developed and tested on online news related to selected keywords. However, it is possible to re-implement the discussed approach in this study, with minor modifications, to adapt it to other unknown words as well.

The assumptions of the report are:

1. Unknown words in NLP production system are typically nouns as they reflect the new trends and phenomena around the world, or cater to services and knowledge in a specific domain. An example of such trend is COVID-19, which has affected many ASR and QA system that are not trained on these words before. Therefore, having a good quality understanding of these words are of special interest as they have potential in enhancing business and services alike.
2. The system is designed to trade accuracy and quality reasonably in exchange for a light-weight and fast synthesis method. Cycle time is one of the most crucial factor in user-facing NLP systems, therefore having quick iteration time thanks to fast synthesis can indirectly improve the end-user experience with NLP applications significantly. Additionally, as embedding is used as an input to downstream tasks, a slight reduction in accuracy is acceptable for well-tuned system as they are able to make maximum use of the information captured inside the embedding.

1.4 Report Organization

This report is sectioned into five chapters as follows:

- Chapter 1 presents a brief introduction into the project and its background, followed by the project's objectives, scope and assumptions
- Chapter 2 focuses on the literature review of existing approaches to synthesize word embedding, followed by a brief discussion on metrics suited for the objective and inspiration to the current approach
- Chapter 3 proposes the system design, the methodology and specifications of the system
- Chapter 4 details the implementation of each individual component of the system, from data curation and ingestion to output
- Chapter 5 presents the experiments results and evaluate its efficacy
- Chapter 6 gives a conclusion to the report and explore possible future work

Chapter 2

Literature Review

This section presents a survey of existing approaches to deal with OOV as well as their advantages and disadvantages. The main approaches can be broadly divided into (1) sub-word segmentation, (2) fine tuning on better corpus, and (3) other novel methods. By examining the various different approaches to dealing with unknown words, we gain a better understanding of the properties of word-embedding in their latent space and the time and memory complexity of each method's implementation. From there, we hope to combine the insights and groundwork of these studies to come up with an implementation best suited for our specifications.

2.1 Sub-word Segmentation

Segmentation, in general, is a technique to separate an input text into useful components for analysis. Therefore, given any text sequence, the aim is to break it down into meaningful tokens that can be understood by the model. Before sub-word segmentation becomes popular, word segmentation was widely adopted in research and production field. However, it requires a huge vocabulary size (approximately 70000 words for English) and thus pose a huge constraints in training and storing models.Further more, many tokens overlap, such as 'look' and 'looks', which do not provide meaningful information to NLP models. Character-level segmentation and word-char hybrid models were introduced in [8], where the character model evaluated on the *newstest2015* NMT dataset achieved competitive BLEU-score with significantly smaller vocabulary size. However it is difficult to train and tune these character level models due to its long convergence (3 months

Input	Cooked
Word level	Cooked
Char level	C o o k e d
Sub-word level	Cook ed

Table 2.1: Word, Character and Sub-word segmentation

compared to 3 weeks for a similar word-level model [8]. To tackle these issues, [9] introduced the concept of segmenting words into sequences of sub-word units to provide a more meaningful representation within a reasonable vocabulary size. A brief overview of the aforementioned approaches is presented in table Table 2.1 for illustration.

Based on the idea of segmenting text input into meaningful token within a reasonable vocabulary size, many sub-word approaches have emerged and are adopted across the field. In this report, we will focus on the main sub-word techniques (and its variants) that are most widely used and are most popular in NLP:

1. Byte-Pair-Encoding (BPE)
2. WordPiece
3. Unigram Language Model (ULM)

2.1.1 Byte-Pair-Encoding

[9] proposed this segmentation method based on the Byte Pair Encoding Algorithm compression algorithm, which is where the name is derived from. It is was proposed for the task of Neural Machine Translation, but is applicable to other tasks as well. Given a text corpus, the algorithm works as follows:

1. Initialize the vocabulary
2. Split each word into characters
3. In every round, get the most frequent pair of characters, combine them to make a new token
4. Repeat step 3 until the desired vocabulary size is achieved

The idea behind BPE is to find the minimally satisficing number of tokens that can maximize the representation of information in the training dataset, using compression algorithm. In BPE, character level tokens are merged to form more meaning tokens that are evaluated using frequency in training corpus, It is an effective approach to tackle rare and unknown words while keeping the vocabulary size to a pre-defined parameter. However, the tokens obtained by BPE has one major limitation: there are instances where there are more way of encoding a word due to token overlap, and there is no easy way to prioritize which encoding. This has a huge impact on the accuracy of downstream task. Additionally, due to its global convergence over the entire training corpus, BPE takes considerable amount of time to train, especially for large size corpus where merging is computationally expensive and memory-hungry. Nonetheless, the idea of using a compression algorithm in BPE has been the foundation for many other probabilistic algorithm such as WordPiece and SentencePiece.

2.1.2 WordPiece

WordPiece [10] replaces the frequency count evaluation in step (3) of BPE by a probabilistic approach, using a Language model. Given a text corpus, the algorithm works as follows:

1. Initialize the character tokens list from the text corpus
2. Build a Language Model (LM) using the list of tokens
3. Generate a new token by combining two existing tokens. Choose token that maximize the likelihood of the LM over the training corpus.
4. Repeat step 2 and 3 until the number of tokens is reached

As illustrated above, WordPiece is a greedy algorithm that tries to maximize the likelihood of the LM, and hence it suffers the same limitation of being computationally expensive as BPE. However, the author of [10] presents a few speed-up heuristic that can cut down the training time to a few hours on a single machine:

- Potential token to be added should exist in the training data, rather than greedy combination of existing tokens

- Potential tokens should have a high priors, or other heuristics features that can maximize the likelihood of the LM
- combine several clustering steps for orthogonal group of pairs
- modify existing LM in step 2 based on the change rather than rebuild the LM.

These speedups address the concern of computational resources and hence make WordPiece an extremely viable options for production environment. Due to it having the core idea of BPE but none of the practical limitations, WordPiece (and its variants) are widely used in modern NLP pipeline such as Bert and GPT-2, to tokenize input text.

2.1.3 Unigram Language Model

Introduced in [11], unigram leverages on the likelihood maximization idea to select the most probable candidate subword - token to be added into the vocabulary. Additionally, the vocabulary is initialized with all base characters and most common substrings. The algorithm then works as follows:

1. Heuristically initialize the vocabulary
2. Optimizing the subword occurrence probabilities using Expectation-Maximization (EM) algortihm by maximizing the likelihood on the LM, similar to BPE
3. Compute the loss for each subword, defined as the drop in likelihood if the subword is removed from vocabulary
4. Keep the top n% of the subwords sorted by the computed loss. Do not remove base characters.
5. Repeat until the desired vocabulary size is reached.

Unlike WordPiece, Unigram works by removing the tokens optimized over the log likelihood over the vocabulary at the training step. The major advantage of Unigram over BPE is that it output multiple sub-word segmentation with corresponding probabilities [11], which not only disambiguate the issues of BPE, but also make the downstream task more robust by introducing noise in sub-words in the form of multiple candidates [11].

However, it is difficult to implement the idea proposed due to step 1, as the heuristics in choosing the initial vocabulary is not mentioned in the paper. The way the vocabulary is chosen has a huge impact on the accuracy as well as convergence of the algorithm, and this poses a non-trivial constraints to Unigram algorithm.

The aforementioned methods of segmenting words leverage on the idea of maximizing most information content in tokens while minimizing the number of tokens used. While differing in the way the vocabulary is initialized and how tokens are added/subtracted from the list of tokens, the core idea remains that the tokens have to maximize the likelihood of the language model from which they are built from. Generally, this works well with unknown words that are not named entities, which are often syntactically constructed by its component sub-words. A trained token lists from the above approaches can then capture the information in the unknown words by extracting the information from known sub-words, thus directly improves the model understanding of input text.

Broadly speaking, sub-word segmentation works well when unknown words are syntactically constructed from known words (morphology), and hence it suffers when unknown words are semantically constructed. Named entities is a prime limitation of sub-word segmentation, as the name and its sub-words bear no information for the meaning of the word. Spoken language (where word takes on a different meaning based on context) and polysemy (words have multiple meanings) are other examples where sub-word segmentation fails to extract useful information out of text input. Despite this limitation, the benefits of understanding new unknown words in a morphological manner is still extremely significant and hence sub-word segmentation remains widely used for most tasks in NLP.

2.2 Fine-tuning on domain-specific corpus

Contrasting with segmenting words into their syntactic components to better understand the unknown words, this approach seeks to broaden the vocabulary to progressively include more words into the existing models. There are two popular approaches that this report will present, transfer learning and retraining

2.2.1 Transfer learning a general language model

Transfer learning is a general technique where model trained on a certain tasked is fine-tuned on data of a different task (for example, a general masked language model can be fine-tuned for sentiment classification). In this report's context, transfer learning is fine-tuning a general language model on a small corpus of unknown text to enable the general LM to learn the meaning of these words using what it has already knows. The process is illustrated in Figure 2.1 below.

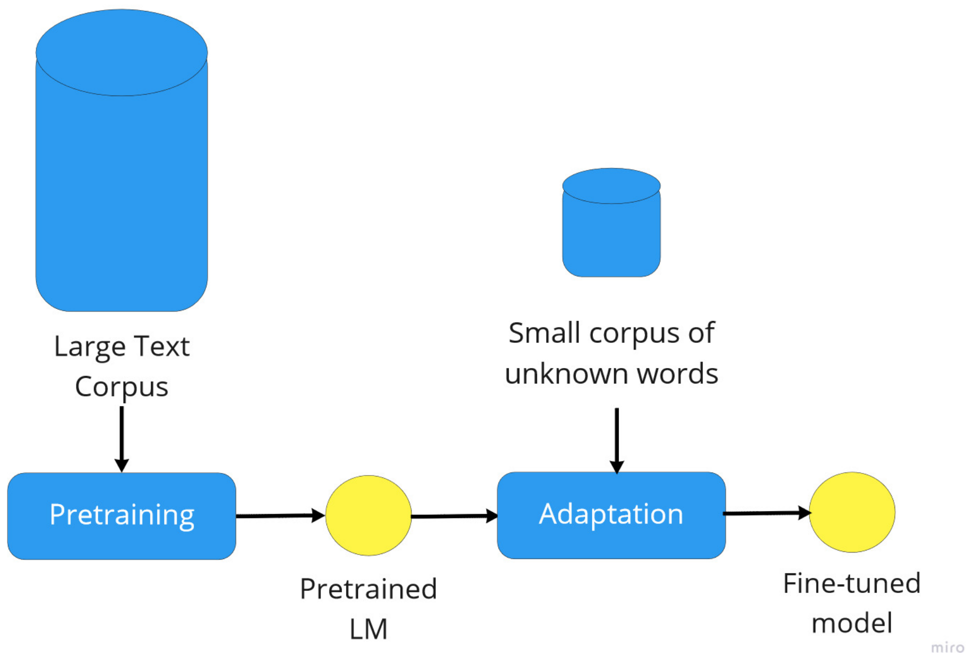


FIGURE 2.1: Transfer learning for OOV

The idea was first formally introduced in the ELMo paper [12] by Peters et al. from AllenNLP. ElMo presented pretraining and adoption as two separated tasks as follows:

1. Pretraining of the bi directional Language Model (biLM) is done on the 1 Billiion Word Benchmark, a dataset for LM from Google.
2. After the biLM is pretrained, the model can be adapted to tasks via feature extraction, with the weights of the biLM frozen.

The same idea is also introduced in the ULMFit paper [13] where the training procedure is almost identical (differ only in the pretraining corpus being the wikitext-103 corpus).

The ULMFit model is then adopted to various tasks, exceeding many cutting-edge models in text classification of the same year.

A major disadvantage of this method is observed by [14], where the Language Models may catastrophically forget the patterns learned from their source domains. This is possibly due to the training data corpus size being significantly larger than the domain specific task, causing the model is to be over-fitted on the domain that it was originally trained on, which has been noticed by [15]. The over-fitting has a tell-tale sign of having monotonically increasing evaluation loss but monotonically decreasing training loss during the training period. However, the general constraints of domain-specific text is that they are not as readily available as general-purpose text, which are easier to crawl automatically or have large corpora ready to be used. With the size of domain specific corpora being a major limitation, the effectiveness of transfer learning on new domain is constrained to the ratio of domain-specific text and the original corpora that it was trained on, to ensure a sufficiently accurate and precise model for the specific task.

2.2.2 Retraining from scratch

The issue of OOV is most astute in the area with highly domain-specific vocabulary such as Biomedical and Medicine. Much research efforts have been devoted to this area to tap on and utilize the recent innovation in NLP such as BERT to apply it to these fields. The original BERT model [16] was pre-trained in an unsupervised manner on a large corpus of unlabeled text data, which includes the entire Wikipedia text corpus (estimated 2.5 billions words) and Book Corpus (800 million words). As most of the text that is used on BERT are general domain text, the proportion of domain-specific text in the training data is skewed, forming part of what is known as the long-tail distribution learning problem. BERT is therefore well-tuned on general domain language, but suffers when it comes to domain-specific corpus. [17] has noted this issue with the long-tail distribution in web data as well. This problem of domain distribution in training corpus has also been observed in and studied by [15], which also suggested a method of adversarial training to combat such issue. However the approach was not widely adopted and instead the community gathered large unlabelled text data of a different domain to build their own corpus, such as [18].

There has been efforts to train BERT on a different set of corpus, also known as domain specific adaptation of BERT, with the most notable being the ClinicalBERT [19] and

BioBert[20]. As noted by [20], the previous approaching of fine-tuning a general language model such as BERT on unknown medical words proves to be unsatisfactory due to a massive distribution shift from general domain corpora (BERT was trained on the English wiki and Brown corpus) to the biomedical corpora. However, a major disadvantage of this method is it being computationally expensive and resource hungry, as training these large models require sufficient batch size and hence adequate GPU RAM is required. [20] used eight NVIDIA V100 (32GB) GPUs for pretraining, and it took 23 days for the BioBERT v1.1's pretraining to be complete.

Additionally, this huge effort in retraining a model on a large corpora of domain-specific text corpus also results in major carbon footprint [1] due to the huge amount of electricity consumption from GPU/TPU training required from clusters of distributed computers, shown by Figure 2.2 below.

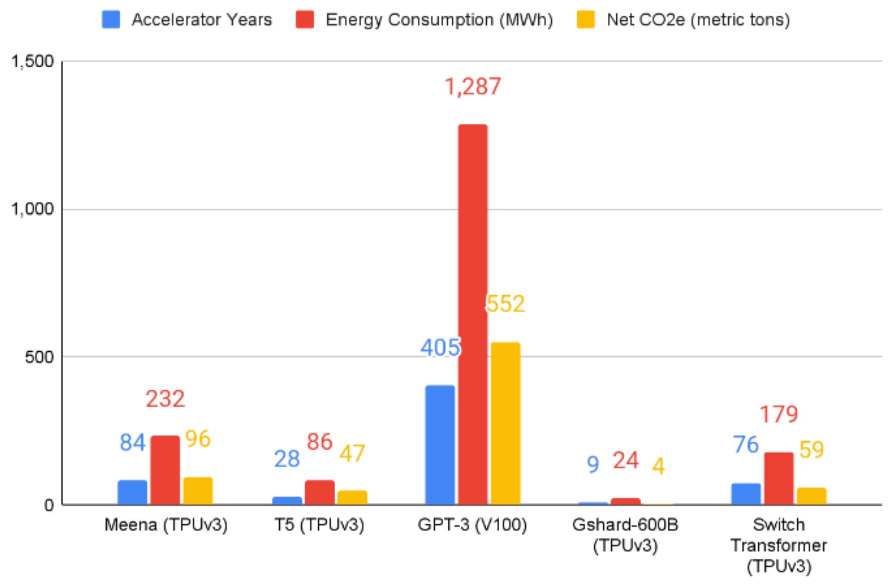


FIGURE 2.2: Accelerator years of computation, energy consumption, and CO2 for five large NLP models, taken from [1]

To summarize. the previous two approaches dealt with the problem of generating embedding for a word from a syntactic and deep-learning manner. With syntactical approach, the semantics sense of the word is lost almost entirely, and while deep-learning approach of retraining or transfer learning ensure a better quality embedding, it is extremely expensive in terms of computational power and is hardly a viable option for most projects. As such, there has been novel approaches to synthesize word embedding that can capture the compositionality of unknown words from existing, known words in a predictable and interpret-able manner. The next two sections will discuss ConceptNet and Recursive Neural Network (not to be confused with Recurrent Neural Network), that aims to capture this compositionality functions via Knowledge Graph and Morphology function respectively.

2.3 ConceptNet

Introduced in [21], ConceptNet is a knowledge graph with nodes being words and phrases of natural language and its edges are labelled relations between the words and phrases. The knowledge graph captured concepts from multiple sources including expert-created resources, crowd-sourcing, and games with a purpose. At first, it appears that ConceptNet makes use of general purposes text to make its embedding. However, the difference here is that the edges are labelled, and hence the relation are explicitly captured using these edges. As such, the data distribution in this case is not as important as the coverage of relations between words.

ConceptNet is designed to represent the general knowledge involved in understanding language and hence it can facilitate in natural language applications by explicitly define the understanding and relationship behind the words people use. When it was first introduced, the nodes were just represented by words. However, after the conception of word2vec [5] and it has been shown via analogy task that the word embedding display relationship in their latent space similar to the semantics we assigned to them, word embeddings were introduced into ConceptNet 5 [21]. When the traditional ConceptNet is combined with GloVe embeddings which has meanings represented in a distributional semantics manner, it provides applications with understanding that they would not acquire from distributional semantics alone as there are explicit, long-span dependencies that were not captured during the training process of using neighboring words. It also performs better than other knowledge graph such as WordNet or DBPedia.

It is noted in [21] that knowledge-informed word embeddings are capable of solving analogies challenges; they perform the same as or slightly better than non-word- embedding systems on the same evaluation. The author also noted that with different word embeddings incorporated (word2vec, GloVe, and LexVec), the performance can be slightly better or worse. This means that the semantics presented in difference word embeddings are not always compatible with the symbolic relations defined in the concept net. This is expected due to the fact that word embeddings are synthesized with an objective of reducing the loss on predicting words from their context words, and as such the neural network can optimize in such a way that synthesize semantics which are not human-understandable, but facilitate in this optimization objective. Additionally, as stated in [21], ConceptNet's role is to provide a sufficiently large, free knowledge graph that focuses on the common-sense meanings of words and not named entities. As such it still suffers greatly on detecting meaning of named entities. Nonetheless, this focus on words as they are used in natural language makes it easier to compile and of greater usage for different general purposes application.

[21] also studied the performance of purely distributional semantics (word2vec, GloVe, and LexVec), word embeddings that represent only relational knowledge (ConceptNet PPMI), and the combination of the two (ConceptNet Numberbatch). It is shown that ConceptNet NumberBatch has been marginally more successful that the others, but the results is inconclusive and it is not certain if ConceptNet Numberbatch can be easily implemented in other system, as the generation process is not trivial.

There has been other efforts to make use of ConceptNet and other relations databases and combine it with embeddings to produce high quality embeddings for unknown words, introduced by [2]. The approach combines the symbolic relations in Paraphrase database (PPDB) and ConceptNet 5, together with existing distributional semantics embeddings in the workflow shown in Figure 2.3 below to create embedding for new unknown words.

The technique used in [2] is a variant of a technique called "retrofitting", which combines embed- dings learned from the distributional semantics of unstructured text with a source of structured connections between words that was introduced by [22]. Retrofitting refines vector space representations by optimizing related words embeddings to have comparable vector representations utilizing relational information from semantic lexicons, and it makes no assumptions about how the input vectors were created. The retrofitted embedding outperforms either source individually in terms of word similarity ratings. In [2] the author has combined a myriad of different studied techniques while adding some new techniques and new sources of knowledge. The novelty of this new retrofit technique lies in the ensemble procedure described in the paper as followed:

1. ConceptNet is used as the source of relations between words
2. align English terms from different sources using a lemmatizer

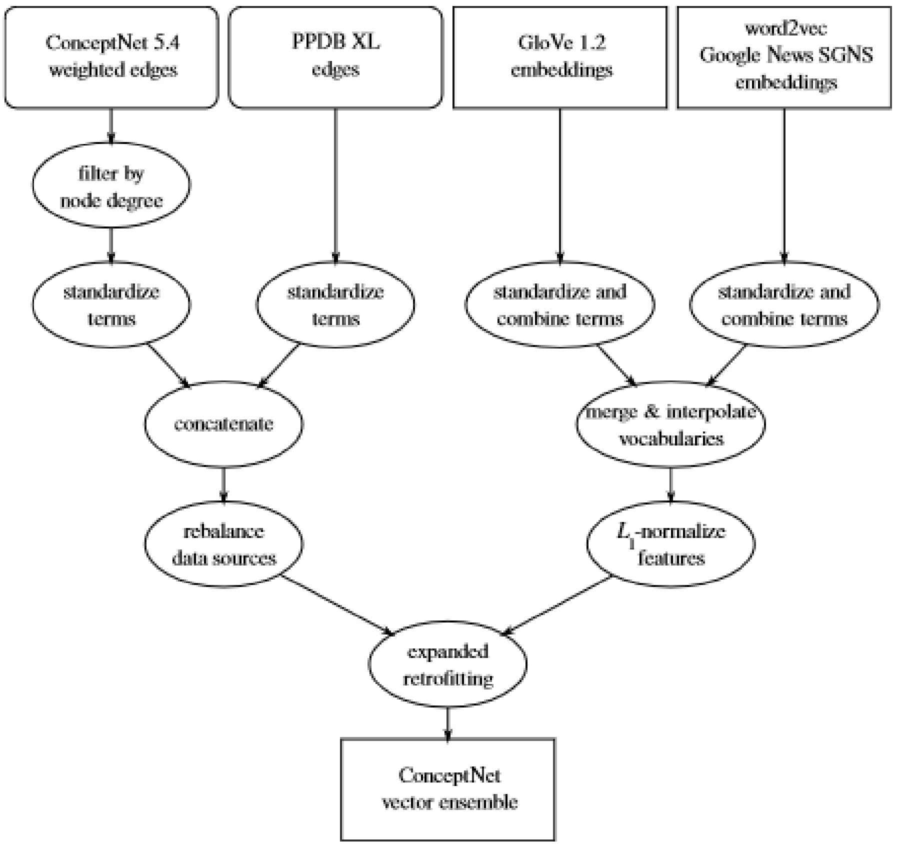

ConceptNet 5.4 weighted edges
PPDB XL edges
GloVe 1.2 embeddings
word2vec Google News SGNS embeddings
filter by node degree
standardize terms
standardize terms
standardize and combine terms
standardize and combine terms
concatenate
merge & interpolate vocabularies
rebalance data sources
L_1-normalize features
expanded retrofitting
ConceptNet vector ensemble

Figure 2.3: The flow of data in building the ConceptNet vector ensemble from its data sources, taken from [2]

3. a heuristic for merging together multiple term vectors is used.
4. local interpolation is used to fill gaps when aligning the two different distributional-semantics sources
5. re-scale the distributional-semantics features using L1 normalization.

This approach combines word2vec, GloVe, PPDB, and ConceptNet to produce the "ConceptNet vector ensemble" a space of multilingual term embeddings that delivers state-of-the-art performance on word similarity evaluations1 over both common and rare terms. This demonstrates the vector ability to synthesize good embeddings from its knowledge sources.

However, as [23] found, high-level choices about how to use a system can significantly affect its performance. While Levy found settings of hyper-parameters that made word2vec outperform GloVe,[2] made GloVe outperform Levy's tuned word2vec by pre-processing the words in GloVe with case-folding and lemmatization, and re-weighting the features using L1 normalization. Additionally, the architecture is highly complicated due to the various source of knowledge and the ensemble method, making it non-trivial to use it in production system. Furthermore, a well-known issue with ensemble method on large dataset is that inference time has non-negligible latency, which impacts the efficacy of the NLP application.

2.4 Recursive Neural Net

As discussed, words are viewed as independent entities in most existing work, with no explicit relationship between morphologically related words being modeled. As a result, unusual and difficult words are frequently underestimated, and all unknown words are represented with only one or a few vectors. [6] proposed a novel approach to build representations for morphologically complex words from their morphemes by combining recursive neural networks (RNNs), where each morpheme is a basic unit, with neural language models (NLMs) to consider contextual information in learning morphologically-aware word representations. The novelty of the approach lies in the context-sensitive structure built into the model shown in Figure 2.4 below.

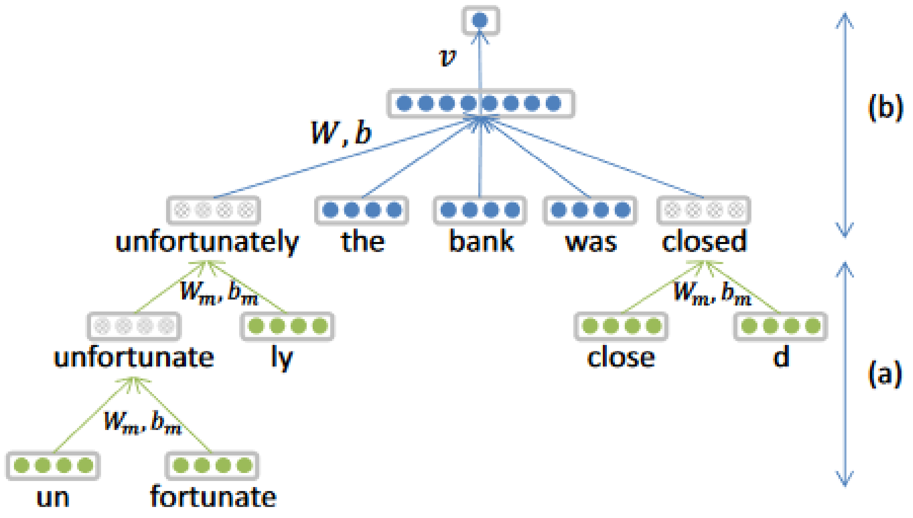


FIGURE 2.4: Context-sensitive morphological RNN in [2]

[6] observed that syntactically, the proposed RNN model could well enforce structural agreement among related words. For example, it returns V-ing as nearest neighbors for

"comment- ing" and similarly, JJ-ness for "heartlessness", an unknown word that previous distributional representation cannot handle. However, the author also noticed that for those cases mentioned above, the nearest neighbors are badly unrelated. On the semantic side, it is noticed that when structural agreement is not enforced, the RNN model tends to cluster words sharing the same stem together, showing that it is able to capture some aspects of the semantics of words. However, this might be undesirable when we want to differentiate semantics of words sharing the same stem, e.g. "affected" and "unaffected". Such semantic aspects are still better captured by the original distributional semantics.

The RNN explicitly models the morphological structures of words to learn morphemic compositionality and this allows for better estimation of rare and complex words and a more principled way of handling unseen words, whose representations could be constructed from vectors of known morphemes. By making use of the surrounding word contexts to provide further semantics to the learned morphemic representations, the author noticed that the embedding are context-sensitive, which could significantly outperform existing embeddings on word similarity tasks.

In conclusion, this report has given a overview of the four current approaches that production system and NLP in academia has proposed to partially tackle the OOV problem. Each of the methods have their own novelties, and seek out to tackle a certain challenges faced in their own respective tasks, to achieve high accuracy at the cost of computation or architecture complexity, or to trade off some accuracy for computational complexity.

Chapter 3

Proposed Approach

The literature review conducted in this report has shown that current research focus on accuracy and quality of the embedding at the cost of computational resources and long training time. While this approach guarantees accuracy, it might not be relevant and practical for production NLP system whose unknown words forms a long-tail distribution – their proportion is not significant but non-negligible. This presents a gap for NLP production system that wish to takle the problem of OOV in a light-weight manner. A system that can benefit sufficiently from the accuracy of aforementioned architecture, while taking less than a day to train is highly valuable for production as it will shorten the downtime of service upgrade tremendously. Therefore, this report aims to develop a low-cost, light weight neural network that can produce decent quality embeddings for production system to tackle the problem of unknown words in a cost-effective manner.

This chapter outlines the research hypothesis of the report, suggests a pipeline to address the hypothesis and list the possible options for each part of the pipeline. The analysis and selection of the most viable component for each part will be done in the next chapter.

3.1 Research Question

This report has been inspired by the architecture of the previous system mentioned in the literature reviews. From the literature examined, the most effective way of generating embedding is utilizing their neighboring words [24]. Although ConceptNet and other knowledge-relation database can provide more semantic explanation and richness to the embedding, it came at a stiff cost in implementation, as the relational knowledge are difficult to manipulate and takes up a large amount of storage. As such to fulfill the requirement of a light-weight system, it is practical to go with skip-gram style of architecture. However, for skip-gram to directly generate an embedding as an output and not a by product, there needs to be certain modification to the objective function. In particular, the objective function must allow the neural network to capture compositionality functions between the target word and its context words.

As studied by [25] and [24], skip-gram word vectors exhibit linear-additive compositionality. The paper suggested that skip-gram vector by their design can capture linear additivity among word vectors, under certain assumptions. [25] also further proposes methods to combat against non-linear additivity. As suggested by [26], to capture such non-linearity function, a non-linear activation function should be used, with ReLU and ELU being good candidates. Additionally, although the embedding that is to be generated are by-products of highly complex model, the compositionality function does not need to be equally complex as the semantic is already distilled in the embedding [27]. As such, model simplicity should work adequately well, especially when the aim is to build a light-weight system.

Therefore, a skip-gram architecture with the objective to capture word compositionality is the most ideal candidate for the task. It would ingest context words as input and output the respective embedding of the target word. For example, in order to generate an embedding for the word "COVID", we can supply the model with "infectious disease 2019 virus respiratory illness global pandemic". The objective of the model is slightly different from a language modelling task in the sense that context words might not appear in a natural sentence with the target word, but they are words with embedding which can provide explanatory features to generate the target word embedding.

3.2 Methodology

The end-to-end system would take in context words as input, and output an embedding for the target word. For this objective, there are several consideration for the implementation of the network architecture:

1. The size of the embedding will differ based on the downstream application. For example, GloVe embedding has 100, 200 and 300 dimension versions. BERT Embedding and its variants (not to be confused with the output of BERT's encoder) has a hidden size of 768. As such, to remain practical in production setting, the model should be able to adapt quickly with these settings – such as via modifying a hyperparameter.
2. There are many possible options for network architecture: Convolutional Neural Network, Feed-Forward Neural Network and seq2seq model.

Therefore, it is important that we can compare the performance of these possible variations of models and have a reasonable test metrics to be able to select a good model. It is also important to conduct separate experiments for GloVe and Roberta embedding, as they are of special research interests. For the architecture choice, there should also be preliminary experiments to help decide which architecture is most suitable for the task.

The model would be trained to generate embedding of already known words, and during inference it would be used to generate embedding of an unknown words using known context words.

3.3 System Architecture Overview

The entire end-to-end (E2E) pipeline consists of two part: training and deployment. Training pipeline is shown in Figure 3.1 below.

Referring to Figure 3.1, the dataset (section 3.5) and target words (section 3.9) are input, represented by the green boxes. There can be different choices affecting the architecture as shown by the blue blox component and the options available for that component, illustrated by the black box. One of such component is the embedding (section 3.4) and the other is the architecture (section 3.6). The training would output a model that is fine-tuned to capture the compositionality of the target chosen words or word class (depending on the choice of the words chosen).

Once a synthesis model is obtained, the generation of embedding for unknown words can be done in a manner outlined by Figure 3.2 below.

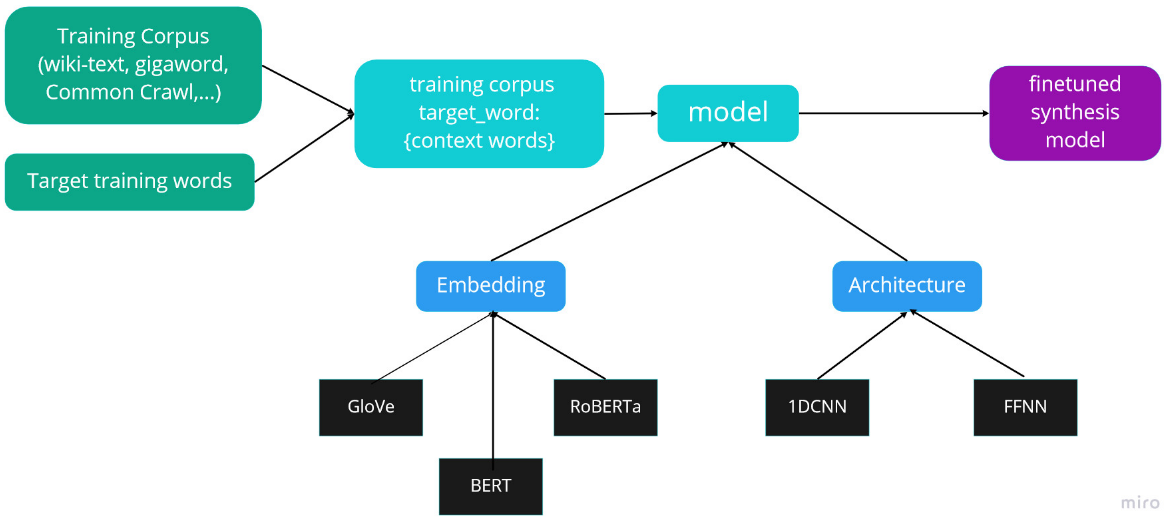

Training Corpus (wiki-text, gigaword, Common Crawl,...)
Target training words
training corpus target_word: {context words}
model
finetuned synthesis model
Embedding
Architecture
GloVe
RoBERTa
BERT
1DCNN
FFNN
miro

FIGURE 3.1: Training Pipeline

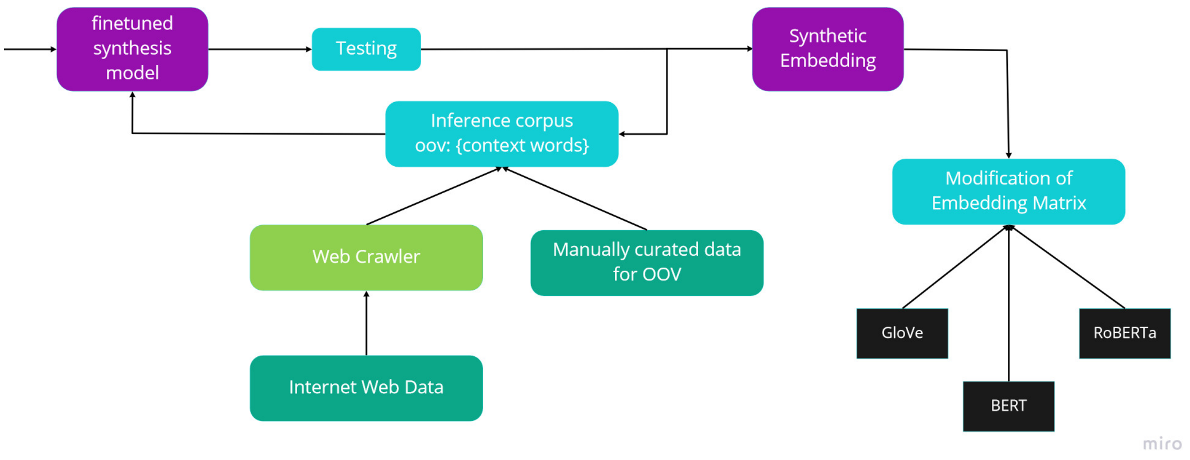

finetuned synthesis model
Testing
Synthetic Embedding
Inference corpus oov: {context words}
Modification of Embedding Matrix
Web Crawler
Manually curated data for OOV
Internet Web Data
GloVe
RoBERTa
BERT
miro

FIGURE 3.2: Deployment Pipeline

The fine-tuned model will require text input in a similar format as its training corpus. The target word is now defined (the unknown word) and hence it necessitates that need of a toolkit that can crawl data for this word (section 3.8), or this data has to be curated manually. The generated embedding will go through an iterative process (section 3.10 to ensure the necessary senses of the words has been incorporated, otherwise higher quality data would be needed for the inference corpus. Once the embedding is sufficiently tested, it will need to be integrated back into the original embedding matrix, depending on the downstream applications' embedding of choice.

3.4 Embedding Choice

For this report, GloVe [5] and Roberta [28] will both be evaluated for their respective strengths:

1. GloVe are easier to handle due to their size and ease of manipulation. The semantics contained within GloVe has been shown to be useful for most NLP applications [29] [30] [31].
2. Roberta is chosen for its usefulness in dealing with multiple languages and code-switching task [32] [33]. This is an useful capability that we would like to incorporate in the model as they would be extremely helpful for downstream applications.

Both of these embeddings have their own merits. For GloVe, it is especially useful for developing Proof-Of-Concept model, which is exactly what this report set out to do. Roberta, on the other hand, is a strong candidate for downstream application that is used in production due to its robustness and flexibility.

3.5 Dataset

Choosing a suitable dataset is vital to the performance of the network. The dataset size should be substantial enough that the model can learn the compositionality of different words. It should also have sufficient variance so that the model can generalize well. For this report, there are also consideration of domain specific text, as discussed in the literature review section. A dataset that is balanced on various domains, with no obvious bias in certain domains, would be the most suitable training candidate for the network.

There are many corpora available for training large corpus: Common Crawl, Wikitext, Gigawords, and so on. With the constraints of domain in mind, it is sensible that wikitext is the most ideal candidate, as its text are explanatory in nature, so the data contain sufficient instances where target words that are defined by their context words. One drawback, however, is Wikitext is a relatively small corpus (16GB) among the language modelling corpora. Gigaword is another possible option, with size of 28GB. However, Gigaword do not possess as strong a quality as wikitext, due to it being a dataset of news. However, it should still contain a reasonable amount of word dependencies that are useful to our word compositionality function that the neural network should learn.

3.6 Network Architecture

To synthesize word embedding, possible architectures are 1D-Convolutional Neural Network and Feed Forward Neural Network. Seq2seq model is not considered because embedding is generated as a by-product and not as an output. The input embedding would come in batch of (batch size,seqlen, embedding size) and the model would need to output (batch size, embedding size) vectors.

A Feed-Forward Neural Network (FFNN) connects all the dimension of the input vectors to generate the dimension of the output vectors, through the use of some hidden layers as shown in Figure 3.3 below. It has the benefits of making full use of possible predictive features from the inputs, and optimize for the best function that generate the output as the network converge towards optimum point of loss.

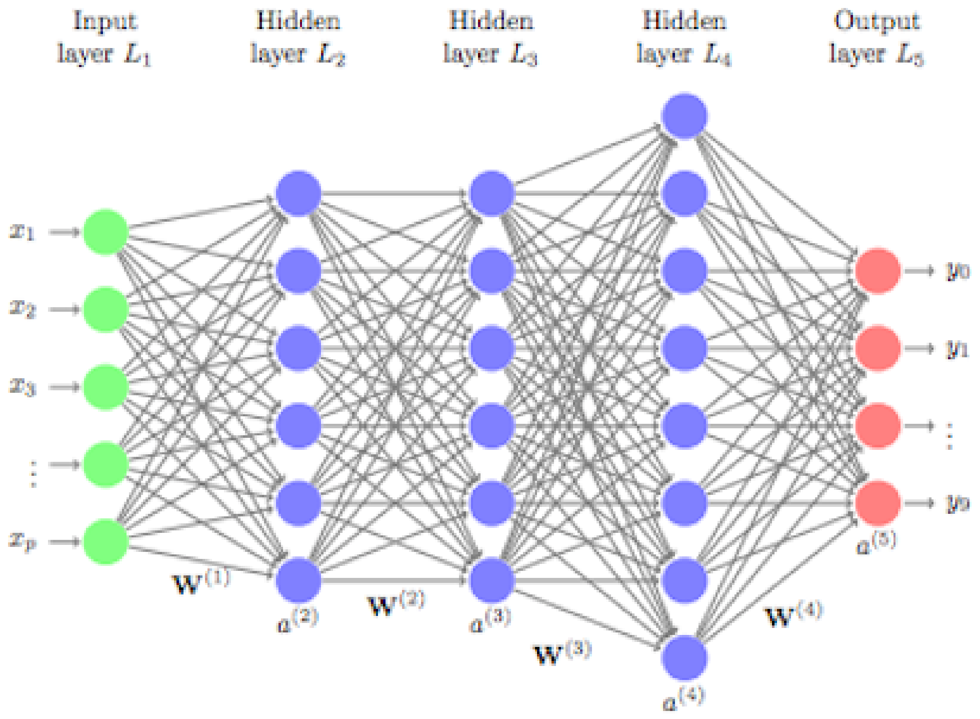


FIGURE 3.3: Feed Forward Neural Network Architecture

A constraint with this architecture is that it cannot handle variable length sequence, therefore a fixed-length context must be used. In the case text corpus do not have sufficient words on both sides of the target words, we would have to supplement it with pad tokens.

1D-Convolutional Neural Network (1D-CNN) is a promising option. Introduced by the Inception Net paper [34], 1D-CNN factorizes the need for excessive connections in the FFNN. It optimize this via only consider vector element of the same positions as shown in Figure 3.4 below.

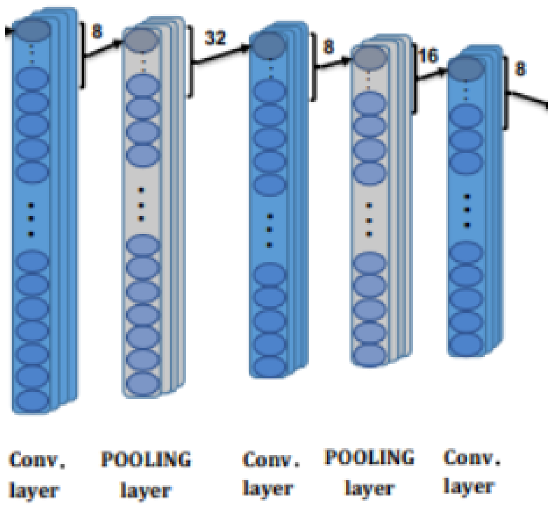


Figure 3.4: 1D-Convolutional Neural Network Architecture

This architecture act as a FFNN for each element in the input vector, thus reducing the amount of computation needs to be done.

For these architecture choice, there will be preliminary experiments to help decide which architecture is most suitable for the task, which is detailed in Chapter 4.

3.7 Data Preprocessing

As suggested in the hypothesis, words with high explanatory power would be the most useful input to our model. To achieve this aim, the preprocessing would need to include:

1. Remove punctuation and symbols
2. Normalize whitespace
3. Tokenize text corpora into words
4. Remove stopwords

There could be more complicated methods to distil a higher quality input for our model such as POS tagging and NER detection. Due to time and resource constrain, as well as the existing complexity of the current POC pipeline, these adjustments would be left to be considered in a future study.

3.8 Auxiliary tools

As indicated in the deployment pipeline earlier, there needs to be an automated tool to crawl web data to facilitate and augment the volume of text for the unknown words. This can be done via crawling news article on google, extracting as much information from the web news as possible. However, there are bound to be plenty irrelevant text presents from such approach, due to the organization of the news in a website, which would also contain advertisement and redundant metadata text. However, with our design of using only neighboring words as context words, the problem of having irrelevant words is partially minimized by design.

It is also desirable that in the modification of language model such as RoBERTa, we have a perplexity score evaluation toolkit to see the changes that have been made to the language models, to make sure that our modification does not affect its performance in other areas.

3.9 Training

For training of model, there are consideration for these factors:

1. Framework
2. Hyperparameter
3. Metrics

Due to our model being custom, it is desirable that the framework is able to support the customization. Pytorch is therefore the most suitable candidate. Hyperparameter and metrics is chosen based on literature values of relevant studies [35] [36]. However, there need to be a tool to monitor the value and adjust for an optimal configuration. Therefore, wandb is chosen to log the results and configuration for all the experiments.

The model will be trained on known words, and then used to infer unknown words. It is advised that the known words belong to the same class as the unknown words, and have similar semantics or word sense as the unknown words. The design of training is as follow:

- Choose a target word of interest (preferably a word in the same word class as the oov).
- Sieve through the text corpora and gather the context words for every instance of that target word.
- Get the embedding of the context words as inputs, the embedding of the target word as label. Since the training is done on known words, the label belongs in the embedding matrix.
- Perform gradient descent on the neural network to minimize the loss between output synthetic embedding and label embedding vector.

The loss chosen is L1 Loss, which is defined as

$$L1Loss = \sum_{i=1}^{n} |y_{true} - y_{predicted}| \tag{3.1}$$

Although there is regularization in the network, it is important that the training data also have enough variance for the model to generalize well. Inspired by the approach in [24], negative samplings has been implemented. For every word or word class that the model is trained on, 5 random words would be sampled. Their context words - target word embedding pair would be mixed into the training data to help the model generalize better.

3.10 Evaluation

There are several evaluation methods stated in [6] and [37], however they might not be the best candidate for this experiment because the unknown words belong to a class of long-tail distribution. Therefore, analogy test is chosen for this experiment for its simplicity and flexibility, inspired by [38] and [39]. This test provides a simple proxy to how similar the synthetic embedding are to the list of known words, which then provide a sense of meaning that is captured in the synthetic word embedding. The similarity metric would be cosine similarity, which is defined as:

$$similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} \tag{3.2}$$

3.11 Tools and Technologies

Table 3.1 below provides a minimum requirement setup for this project.

Hardware Requirement	At least 4GB RAM and 50GB of hard disk for the datasets (Wikitext is 16GB and Gigaword is 28GB). Optionally CUDA GPU to speed up training time
Operating System	Agnostic, as long as anaconda can be installed
Conda packages	The project is built using Python 3.9 and Conda is one of the easiest way to ensure environment isolation and handle packages. See Table 3.2 for the dependencies required
Other softwares	Prefereably Firefox/Chrome webbrowser or any other selenium-compatible browser

Table 3.1: Environment setup

For the python dependencies of the project, Table 3.2 below summarizes the necessary packages and the importance of each package.

There are also other packages that is pre-installed in a conda environment such as json, sys, argparse, numpy, random, time that is used throughout the project for managing data and monitoring process.

Pytorch	The framework to build the network, chosen for its flexibility and customizability
Huggingface Transformer	For the ready-to-use transformer pytorch model (specifically RoBERTa)
Selenium	For Web crawling
NLTK	For their word preprocessing tools and stopwords
BeautifulSoup	For web parsing
Wandb	For monitoring and tracking of experiments
tqdm	For monitoring of process duration and completion
Jupyter Lab	For development environment and running of code

Table 3.2: Conda dependencies

3.12 Summary of System Workflow

A summary of the full pipeline (training and deployment) is shown in Figure 3.5 below. All data input is shown in caleston green components, auxiliary tools in lime green, cyan components denote code component, output is purple and blue component are variable component, with black boxes denote the options available for that variable.

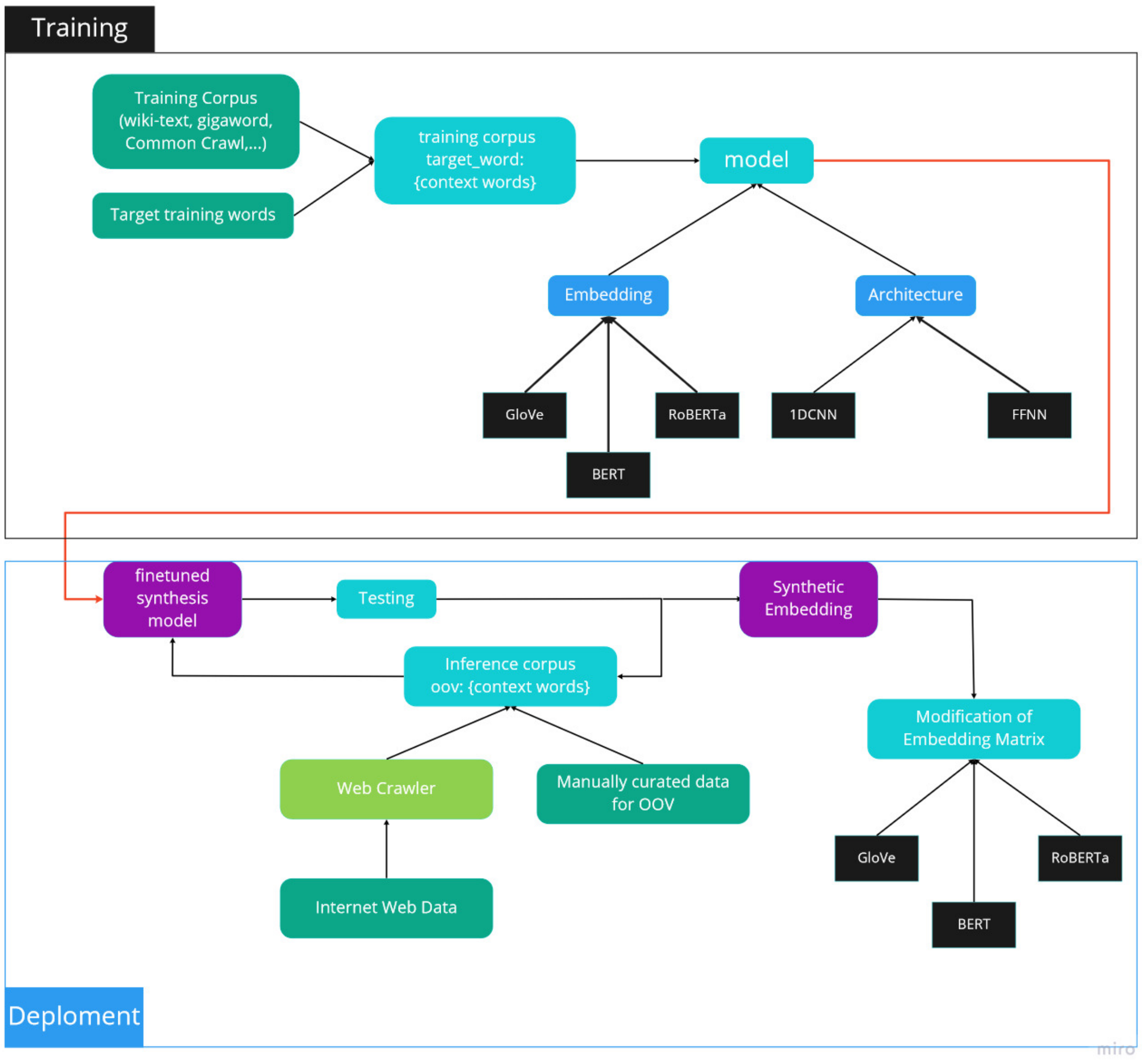

Training
Training Corpus (wiki-text, gigaword, Common Crawl,...)
Target training words
training corpus target_word: {context words}
model
Embedding
Architecture
GloVe
BERT
RoBERTa
1DCNN
FFNN
finetuned synthesis model
Testing
Synthetic Embedding
Inference corpus oov: {context words}
Web Crawler
Manually curated data for OOV
Internet Web Data
Modification of Embedding Matrix
GloVe
BERT
RoBERTa
Deploment
miro

FIGURE 3.5: System Pipeline

Chapter 4

Implementation

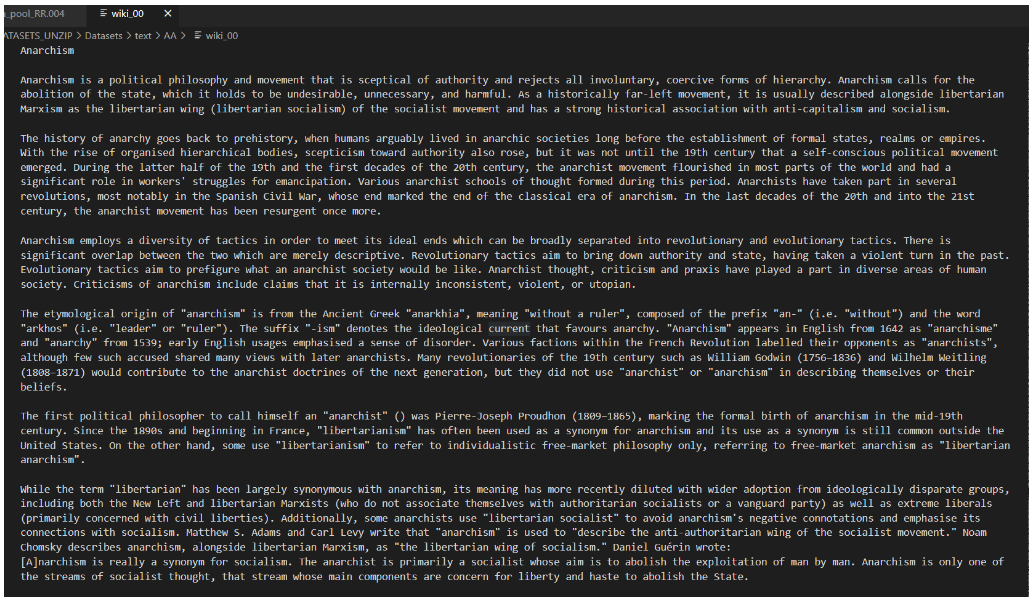
In this chapter, a detailed illustration of each component in the pipeline is elaborated and demonstrated. This chapter will also include experiments that support certain decisions in the pipeline component selection, to illustrate how the author has arrived at choosing certain architecture or implementation for each component of the pipeline. The different treatments of GloVe and RoBERTa embedding would also be detailed in section 4.5 of this chapter.

4.1 Dataset Collection

This section discusses the methods to obtain the three most common datasets: Wiki, Gigaword and Common Crawl, discuss the limitations and advantages of each and corpus in the objective of the report.

4.1.1 Extraction

For wiki text, the most updated version is to use wikimedia dumps. The English wikidump can be found here. The files are compressed in bz2 format, and not easily extractable. There are multiple ways to work with bz2 compressed files, and this report use the WikiExtractor Tool by Giuseppe Attardi [40]. This tools presents an one-stop, lightweight method that can easily de-compress the files. Under each directory, there are text files for each wiki page, as shown in Figure 4.1 below. These are ready for the next processing stage.



```
Anarchism

Anarchism is a political philosophy and movement that is sceptical of authority and rejects all involuntary, coercive forms of hierarchy. Anarchism calls for the
abolition of the state, which it holds to be undesirable, unnecessary, and harmful. As a historically far-left movement, it is usually described alongside libertarian
Marxism as the libertarian wing (libertarian socialism) of the socialist movement and has a strong historical association with anti-capitalism and socialism.

The history of anarchy goes back to prehistory, when humans arguably lived in anarchic societies long before the establishment of formal states, realms or empires.
With the rise of organised hierarchical bodies, scepticism toward authority also rose, but it was not until the 19th century that a self-conscious political movement
emerged. During the latter half of the 19th and the first decades of the 20th century, the anarchist movement flourished in most parts of the world and had a
significant role in workers' struggles for emancipation. Various anarchist schools of thought formed during this period. Anarchists have taken part in several
revolutions, most notably in the Spanish Civil War, whose end marked the end of the classical era of anarchism. In the last decades of the 20th and into the 21st
century, the anarchist movement has been resurgent once more.

Anarchism employs a diversity of tactics in order to meet its ideal ends which can be broadly separated into revolutionary and evolutionary tactics. There is
significant overlap between the two which are merely descriptive. Revolutionary tactics aim to bring down authority and state, having taken a violent turn in the past.
Evolutionary tactics aim to prefigure what an anarchist society would be like. Anarchist thought, criticism and praxis have played a part in diverse areas of human
society. Criticisms of anarchism include claims that it is internally inconsistent, violent, or utopian.

The etymological origin of "anarchism" is from the Ancient Greek "anarkhia", meaning "without a ruler", composed of the prefix "an-" (i.e. "without") and the word
"arkhos" (i.e. "leader" or "ruler"). The suffix "-ism" denotes the ideological current that favours anarchy. "Anarchism" appears in English from 1642 as "anarchisme"
and "anarchy" from 1539; early English usages emphasised a sense of disorder. Various factions within the French Revolution labelled their opponents as "anarchists",
although few such accused shared many views with later anarchists. Many revolutionaries of the 19th century such as William Godwin (1756–1836) and Wilhelm Weitling
(1808–1871) would contribute to the anarchist doctrines of the next generation, but they did not use "anarchist" or "anarchism" in describing themselves or their
beliefs.

The first political philosopher to call himself an "anarchist" () was Pierre-Joseph Proudhon (1809–1865), marking the formal birth of anarchism in the mid-19th
century. Since the 1890s and beginning in France, "libertarianism" has often been used as a synonym for anarchism and its use as a synonym is still common outside the
United States. On the other hand, some use "libertarianism" to refer to individualistic free-market philosophy only, referring to free-market anarchism as "libertarian
anarchism".

While the term "libertarian" has been largely synonymous with anarchism, its meaning has more recently diluted with wider adoption from ideologically disparate groups,
including both the New Left and libertarian Marxists (who do not associate themselves with authoritarian socialists or a vanguard party) as well as extreme liberals
(primarily concerned with civil liberties). Additionally, some anarchists use "libertarian socialist" to avoid anarchism's negative connotations and emphasise its
connections with socialism. Matthew S. Adams and Carl Levy write that "anarchism" is used to "describe the anti-authoritarian wing of the socialist movement." Noam
Chomsky describes anarchism, alongside libertarian Marxism, as "the libertarian wing of socialism." Daniel Guérin wrote:
[A]narchism is really a synonym for socialism. The anarchist is primarily a socialist whose aim is to abolish the exploitation of man by man. Anarchism is only one of
the streams of socialist thought, that stream whose main components are concern for liberty and haste to abolish the State.
```

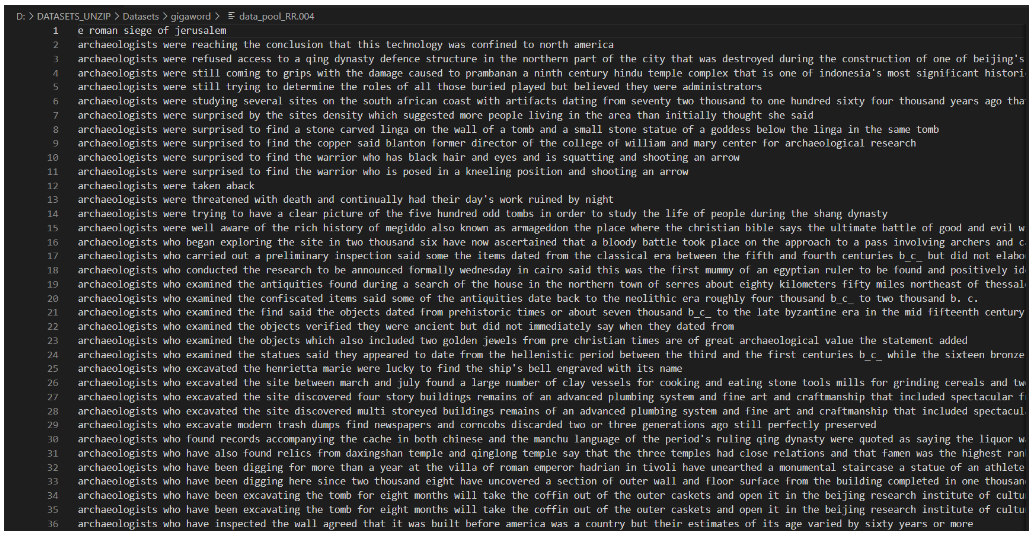
FIGURE 4.1: Wikitext Samples

For Gigaword corpus, it can be purchased from Language Data Consortium (LDC). All text data are presented in SGML form, using a very simple, minimal markup structure; all text consists of printable ASCII and whitespace. Hence extracting Gigaword is simply a matter of purchase. However, the corpus is large (28GB) and hence considerable effort need to be spent on how to process it via streaming method in order not to incur huge memory cost during processing. For this project, the gigaword dataset is divided into 50 different files, each of 500MB. This allows for the whole file to be loaded into RAM without causing a crash. A sample could be seen in Figure 4.2 below.

The Common Crawl (CC) dataset resides on Amazon S3 as part of the Amazon Public Datasets program which can be downloaded free using HTTP or S3 using this url. Common Crawl currently stores the crawl data using the Web ARChive (WARC) format. The WARC format allows for more efficient storage and processing of Common Crawl's free multi-billion page web archives, which can be hundreds of terabytes in size.

4.1.2 Limitations

This report decided not to go with CC corpus due to the lack of resources and storage available. However, between the Wikitext and Gigaword corpus, it is hypothesized that


D: > DATASETS_UNZIP > Datasets > gigaword > data_pool_RR.004

```
e roman siege of jerusalem
archaeologists were reaching the conclusion that this technology was confined to north america
archaeologists were refused access to a qing dynasty defence structure in the northern part of the city that was destroyed during the construction of one of beijing's
archaeologists were still coming to grips with the damage caused to prambanan a ninth century hindu temple complex that is one of indonesia's most significant histori
archaeologists were still trying to determine the roles of all those buried played but believed they were administrators
archaeologists were studying several sites on the south african coast with artifacts dating from seventy two thousand to one hundred sixty four thousand years ago tha
archaeologists were surprised by the sites density which suggested more people living in the area than initially thought she said
archaeologists were surprised to find a stone carved linga on the wall of a tomb and a small stone statue of a goddess below the linga in the same tomb
archaeologists were surprised to find the copper said blanton former director of the college of william and mary center for archaeological research
archaeologists were surprised to find the warrior who has black hair and eyes and is squatting and shooting an arrow
archaeologists were surprised to find the warrior who is posed in a kneeling position and shooting an arrow
archaeologists were taken aback
archaeologists were threatened with death and continually had their day's work ruined by night
archaeologists were trying to have a clear picture of the five hundred odd tombs in order to study the life of people during the shang dynasty
archaeologists were well aware of the rich history of megiddo also known as armageddon the place where the christian bible says the ultimate battle of good and evil w
archaeologists who began exploring the site in two thousand six have now ascertained that a bloody battle took place on the approach to a pass involving archers and c
archaeologists who carried out a preliminary inspection said some the items dated from the classical era between the fifth and fourth centuries b_c_ but did not elabo
archaeologists who conducted the research to be announced formally wednesday in cairo said this was the first mummy of an egyptian ruler to be found and positively id
archaeologists who examined the antiquities found during a search of the house in the northern town of serres about eighty kilometers fifty miles northeast of thessal
archaeologists who examined the confiscated items said some of the antiquities date back to the neolithic era roughly four thousand b_c_ to two thousand b. c.
archaeologists who examined the find said the objects dated from prehistoric times or about seven thousand b_c_ to the late byzantine era in the mid fifteenth century
archaeologists who examined the objects verified they were ancient but did not immediately say when they dated from
archaeologists who examined the objects which also included two golden jewels from pre christian times are of great archaeological value the statement added
archaeologists who examined the statues said they appeared to date from the hellenistic period between the third and the first centuries b_c_ while the sixteen bronze
archaeologists who excavated the henrietta marie were lucky to find the ship's bell engraved with its name
archaeologists who excavated the site between march and july found a large number of clay vessels for cooking and eating stone tools mills for grinding cereals and tw
archaeologists who excavated the site discovered four story buildings remains of an advanced plumbing system and fine art and craftmanship that included spectacular f
archaeologists who excavated the site discovered multi storeyed buildings remains of an advanced plumbing system and fine art and craftmanship that included spectacul
archaeologists who excavate modern trash dumps find newspapers and corncobs discarded two or three generations ago still perfectly preserved
archaeologists who found records accompanying the cache in both chinese and the manchu language of the period's ruling qing dynasty were quoted as saying the liquor w
archaeologists who have also found relics from daxingshan temple and qinglong temple say that the three temples had close relations and that famen was the highest ran
archaeologists who have been digging for more than a year at the villa of roman emperor hadrian in tivoli have unearthed a monumental staircase a statue of an athlete
archaeologists who have been digging here since two thousand eight have uncovered a section of outer wall and floor surface from the building completed in one thousan
archaeologists who have been excavating the tomb for eight months will take the coffin out of the outer caskets and open it in the beijing research institute of cultu
archaeologists who have been excavating the tomb for eight months will take the coffin out of the outer caskets and open it in the beijing research institute of cultu
archaeologists who have inspected the wall agreed that it was built before america was a country but their estimates of its age varied by sixty years or more
```

Figure 4.2: gigaword Samples

wikitext would be a better input for the model, due to it having more explanatory words surrounding one another. Gigaword on the other hand are more narrative, and the dependency among words that explain one another can be long due to the long verb and noun phrases commonly present in news. This would introduce data imbalance, where words that are frequently present in news would be easier to train on the network due to more data available, illustrated by the experiments on different words vaccine, virus, capital, location in figure Figure 4.3 below

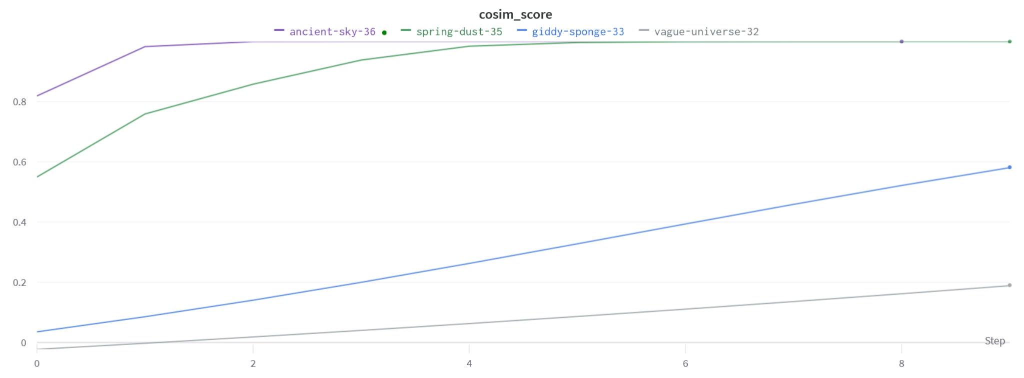


Figure 4.3: Difference in ease of training in best score

4.2 Training Corpus Preparation

For each of the word or word class that needs to be trained, the code will walk through the text corpora and sieve out any instances of that word, with its context words. If the word happens to be at the start of the document or paragraph, padding will be added accordingly. The resultant file would need to be stored in hard disk for training.

This approach alone would lead to low-quality context words due to presence of filler words. Therefore, NLTK was used to remove stopword and filler word, to make sure the input to the model has as much predictive features as possible. The pipeline can be further improved with the use of POS or NER tagger.

For this section of the pipeline, a context word window length of 10 per side was chosen after preliminary experiments. Therefore for each training corpus, each target word would have 20 context words, 10 on the left and 10 on the right. Its format is (target word):(context word 1) (context word 2) ... (context word 20).

4.3 Architecture Selection

There are two promising architecture suitable for the objective: 1D-CNN and FFNN. The results of the experiments comparing the losses on the different architectures is shown in Figure 4.4 below.

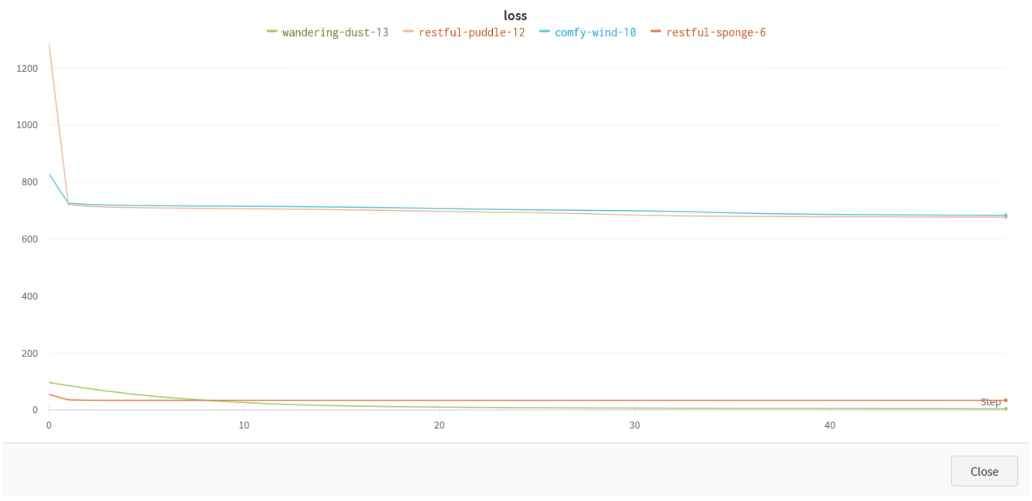


FIGURE 4.4: Comparison between different architectures

Figure 4.4 shows that the 1D-CNN architecture (comfy-wind-10 and restful-puddle-12) have higher loss values than the FFNN by a wide margin. This shows that our hypothesis about correlation among dimensions of the word embedding vector has certain validity:

there are some degree of correlations between different elements of the vector that enables the FFNN architecture to optimize the objective much better than 1DCNN, which consider the vector's elements to be orthogonal.

4.4 Loss, Hyperparameters Decision

4.4.1 Loss Selection

There are many choices of losses for the architecture. Below are the available options:

1. Mean Absolute Error (MAE) Loss (L1 Loss)

$$L1Loss = \sum_{i=1}^{n} |y_{true} - y_{predicted}| \tag{4.1}$$

2. Mean Squared Error (MSE) Loss (L2 Loss)

$$L1Loss = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2 \tag{4.2}$$

3. Cosine Embedding Loss

$$CosineEmbeddingLoss = \begin{cases} 1 - cos(x_1, x_2) & \text{if } y = 1 \\ max(0, cos(x_1 - x_2) - margin) & \text{if y=-1} \end{cases} \tag{4.3}$$

   It measures the loss given inputs x1, x2, and a label tensor y containing values (1 or -1). It is used for measuring whether two inputs are similar or dissimilar.

4. Pairwise Inner Product (PIP) Loss

$$PIP = \sqrt{\sum_{i,j} (\langle v_i, v_j \rangle - \langle \hat{v}_i, \hat{v}_j \rangle)^2} \tag{4.4}$$

5. DICE Loss

$$DL = \frac{1}{N} \sum_{i} [1 - \frac{2p_{i1}y_{i1} + \gamma}{p_{i1}^2 + y_{i1}^2 + \gamma}] \tag{4.5}$$

MAE Loss and MSE Loss are the standard choice for most problems, whereas Cosine Embedding Loss (a variant of triplet loss) is more specific to comparing vector input's similarity, which is especially useful in Siamese Network, which was introduced by Bromley and LeCun to solve signature verification as an image matching problem [41]. PIP Loss was introduced by Ziyin and Shen in [42] as a novel metric on the dissimilarity between word embeddings. DICE Loss was introduced as a novel metric to combat data imbalance issue, and is a popular choice for long-tail problems. Cross-entropy is not considered as the objective is embedding synthesis rather than classification.

The results of the experiments comparing the losses on the same architecture is shown in Figure 4.5 below.

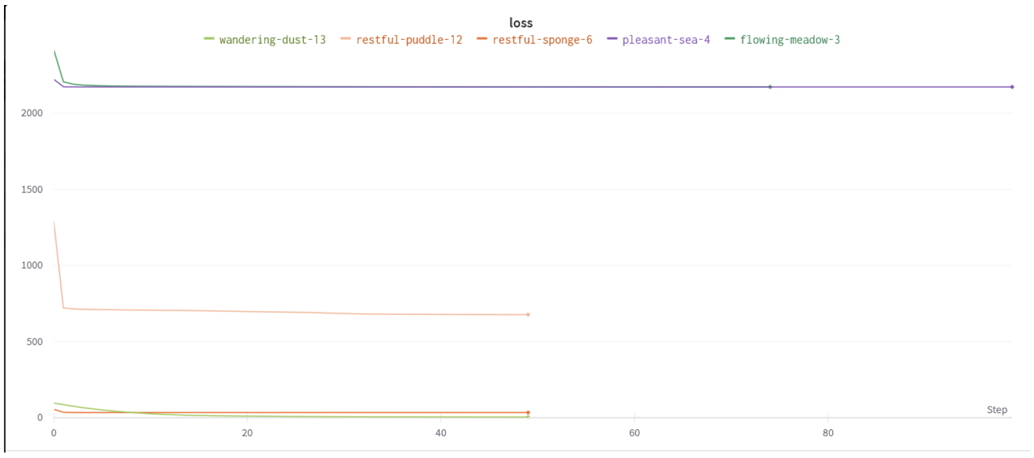


FIGURE 4.5: Comparision between different losses

Figure 4.5 shows that:

1. PIP and DICE loss results in very high loss value (flowing-meadow-3 and pleasant-sea-4 experiments respectively).
2. Cosine Embedding Loss (restful-puddle-12) also results in a high loss value, but significantly lower than PIP and DICE
3. L1 loss (wandering-dust-13) and L2 loss (restful-sponge-6) are significantly more numerical stable

A high loss value will make the network very unstable, due to the effect of Loss value in the Gradient Descent Equation (4.6):

$$\theta_1 = \theta_0 - \alpha \times \frac{\partial Loss}{\partial \theta_0} \tag{4.6}$$

A numerically high loss value will make the descend step extremely huge, hence causing the network to be unable to converge. This is complicated by the implementation of batch normalization [43] in most network, making the matrix value normalized (and therefore easier to train). A huge descent step on a normalized input would be catastrophic. Therefore, a suitable option is to choose a loss that has smaller numerical value or to normalize the loss. For this report, L1 loss was chosen for its numerical stability.

4.4.2 Hyperparameter

Table 4.1 below summarizes the hyperparameter used for this experiment.

Architecture	Fully Connected
Number of layers	2 hidden layers of size 2048, 512
Input and Output layers	Adjustable based on the embedding size of choice (100/200/300/768)
Negative Samples	5, as presented in [24]
Optimizer	Stochastic Gradient Descent (SGD)
Learning Rate	0.0005
Momentum	0.005
Weight decay	0.01
Criterion	L1 Loss
Epoch	50

Table 4.1: Hyperparamter Choices

There are several consideration for the choice:

1. Stochastic Gradient Descent (SGD) was chosen because from it tends to generalize better than Adam optimizer from the experiments. This is also noted by [44].
2. Batch size was chosen to be 64, but this parameter is highly dependent on the system that the model is running on. Increasing it can lead to much faster training time, at the cost of higher memory usage, especially for CUDA-enabled system.

4.5 Training

4.5.1 GloVe embedding

GloVe embedding are easier to manipulate as the raw file is a simple line-by-line text file. Hence, to use GloVe embedding, it is just a matter of extracting these into a matrix W, with its corresponding vocabulary. For each context word, the embedding can be looked up via an id in the vocabulary and retrieved as the corresponding column in the matrix W.

4.5.2 RoBERTa embedding

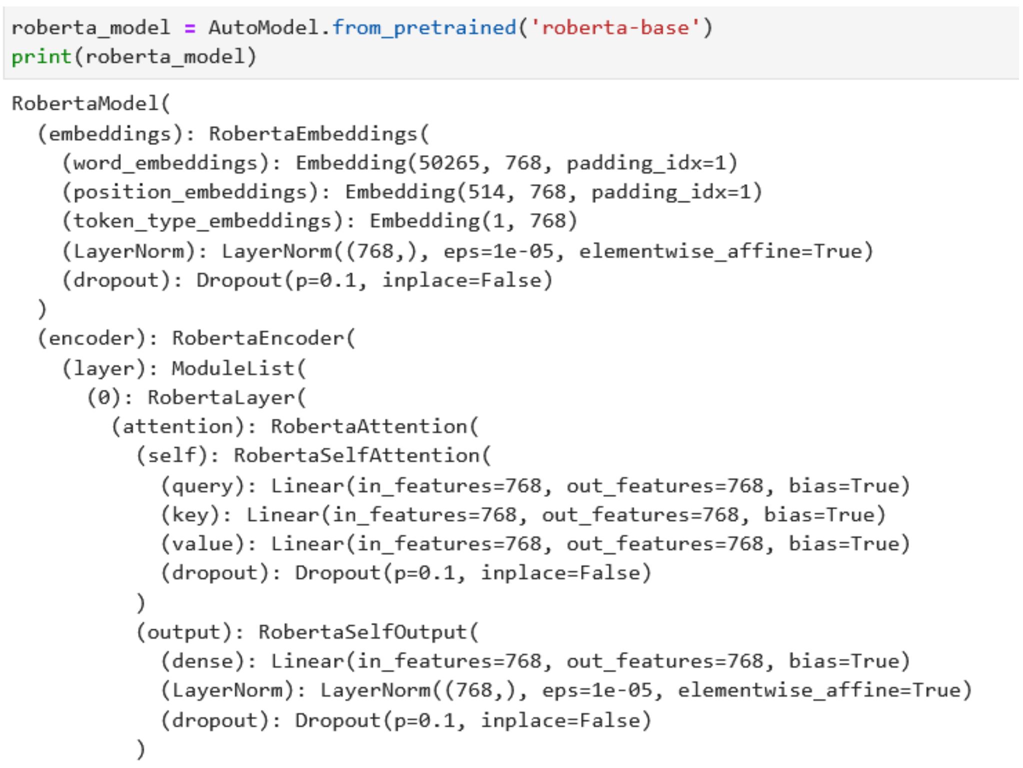
For RoBERTa, the process of extracting the embedding is considerably more intricate. Firstly, there is no single source for just the embedding, rather, the whole model's weights is stored in Amazon S3 file. Therefore, in order to extract this data, we would first use the Huggingface Library to retrieve the model's pretrained weights. The layers of HuggingFace RoBERTa is shown in Figure 4.6 below. This is consistent with the transformer architecture in the "Attention is All you need" paper [3], shown in Figure 4.7.

The embedding can then be extracted by accessing the weight of the word_embedding class of the embeddings class of the model. However, as this is built on PyTorch, the embedding layer and its components are not just tensor, but rather PyTorch Parameter, which is a special Tensor that has gradient built-in to them. Purely extracting these tensors and use them directly on our network as normal tensor will cause the gradient to be propagated back into the input tensors, causing catastrophic failures. It is important to detach the embedding layer into a numpy matrix before further usage.

4.5.3 Embedding-agnostic Training

The network is designed to adjust to different embedding size by tweaking the embed_size parameter. The whole training loop is as followed:

1. Note the system RAM and CPU / GPU available
2. Batch the training input into suitable size for the system



```
roberta_model = AutoModel.from_pretrained('roberta-base')
print(roberta_model)
```

```
RobertaModel(
  (embeddings): RobertaEmbeddings(
    (word_embeddings): Embedding(50265, 768, padding_idx=1)
    (position_embeddings): Embedding(514, 768, padding_idx=1)
    (token_type_embeddings): Embedding(1, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): RobertaEncoder(
    (layer): ModuleList(
      (0): RobertaLayer(
        (attention): RobertaAttention(
          (self): RobertaSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): RobertaSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
```

FIGURE 4.6: RoBERTa layers

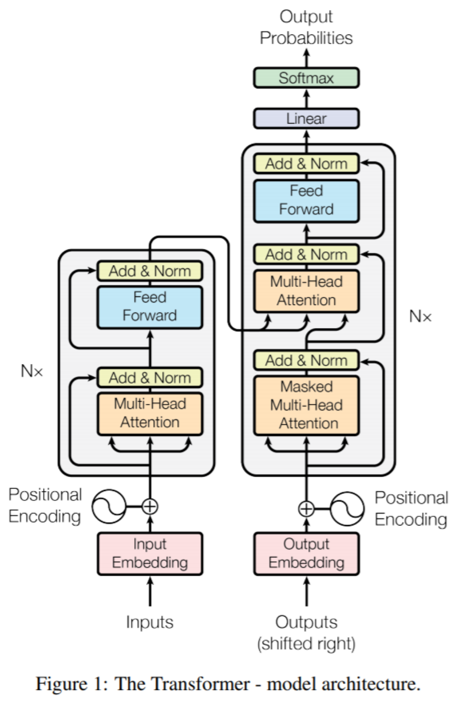


FIGURE 4.7: The original Transformers Architecture in Vaswani paper [3]

3. Preprocess the word into token id if RoBERTa is used
4. Fetch one batch by one batch data into the network, then extract the embedding.
5. Feed the embedding into the network and monitor the loss via wandb

It is vital that the training follows this sequence, especially step 4. This is because the embedding are stored as matrix, which if it was to be loaded for the whole training dataset would cause memory error as each word is now represented by a 100 size vector (GloVe) or 768 size vector (RoBERTa). For batch size of 64, and 1000 batches, RoBERTa would easily use up 50GB of RAM, which is not available for most system.

4.6 Evaluation

For training, the metrics that are used to monitor are cosine similarity score and L1 Loss. The latter is to ensure that no failure has happened during training, as well as to observe the degree of over-fitting during training. The more important end-metric is cosine similarity, as we seek to generate embedding closed to semantically-related words.

For inference, there should be varied degree of test to see how well the model has been trained. The test used would be the analogy test proposed by [38] and [39]. The synthetic embedding generated will be compared to all other embeddings in the vocabulary and see which word is closest to its semantics via cosine distance. This report will use the analogy test method in a 3-way test: (1) Testing on a single sample in the test set, (2) Testing on the average embeddings of a batch (64 examples), (3) Custom sentence testing.

4.7 Auxiliary Tools Implementation

4.7.1 Web Crawler Implementation

The Web Crawler is built using selenium, and it extracts the data for the unknown word as followed:

1. Open a browser and key in a Google search for that word

2. Crawl any new website that is not advertisement using heuristics url templates
3. Further filter to retain sites with more credibility such as news, wikipedia, etc.
4. Gather the text from paragraphs and title elements
5. Run them through the pre-processing pipeline that is used in the training stage to ensure data consistency.
6. Output the file in the correct format

Some of the heuristics used are inspired by the methods discussed in [45].

4.7.2 Perplexity Scorer

A perplexity scorer is useful for many language modelling task. For traditional n-gram model, perplexity is calculated as:

$$PP(W) = P(w_1 w_2 ... w_n)^{\frac{-1}{N}} \tag{4.7}$$

However it is noted that perplexity is not well defined for masked language model like BERT or RoBERTa. Nonetheless, the model's perplexity can be calculated by autoregressively factorizing a sequence and conditioning on the entire preceding subsequence at each step. However, most transformer models are fixed-length sequence, with RoBERTa max_length defined as 512. Therefore, the sequence is typically broken into subsequences equal to the model's maximum input size. The likelihood of a token is then approximated by conditioning only on the max_length-1 tokens that precede it, rather than the entire context.

One possible approach is to break the sequence into disjoint chunks and add up the decomposed log-likelihoods of each segment independently. This can be parallelly computed since the perplexity of each segment can be computed in one forward pass, but it serves as a poor approximation of the fully-factorized perplexity and will typically yield a higher (worse) perplexity because the model will have less context at most of the prediction steps. To alleviate the lack of context length in this approach, we can modify the subsequence to become a sliding-window strategy. This involves repeatedly sliding the context window so that the model has more context when making each prediction. This is a closer approximation to the true decomposition of the sequence probability and will typically yield a more favorable score but at the cost of heavy computation. It requires a separate forward pass for each token in the corpus.

4.8 Incorporation of Synthetic Embedding into applications

4.8.1 GloVe Embedding

Modifying GloVe embedding is a a simple task of writing a new line in the file with the format of (target word) (embedding sequence of float). However to conform with the existing GloVe file, it is important to always check if the word exists first in the vocabulary before adding in, as GloVe has a massive 400 000 tokens vocabulary.

If the word already exists, there are several possible options:

- Overwrite the existing embedding so the word is now replaced with its new semantics
- Take average of the two embeddings
- Sum the embeddings, then normalize the resultant embedding

For this report, we simply go with the second option as it would hopefully capture both existing and new senses of the word. However, downstream applications using glove might adjust accordingly.

4.8.2 RoBERTa

As aformentioned, the modification to RoBERTa is non-trivial due to its complexity. The change to the embedding layer of RoBERTa has to be done in a manner that it is unaffected by the rest of the application. The suggested method used in this report would be as followed:

1. Store the synthetic embedding(s) in a numpy matrix of hidden size of RoBERTa (768)

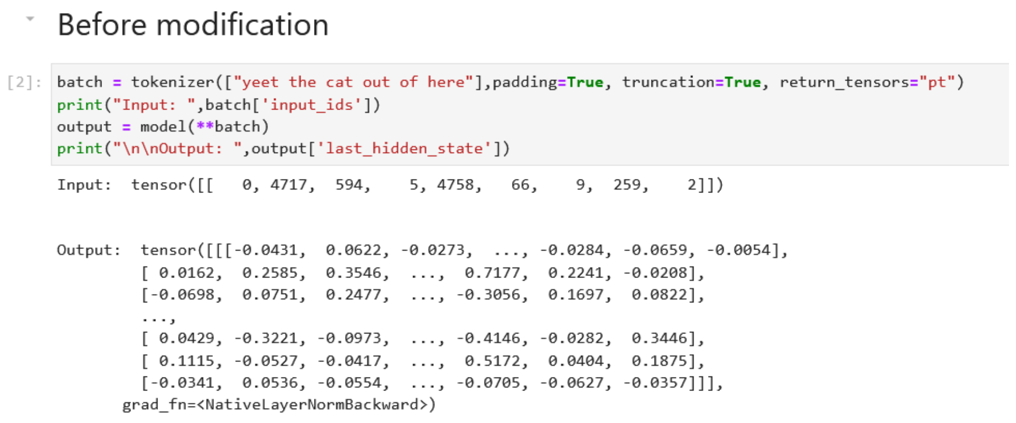
2. When the RoBERTa model is instantiated, add new word into the tokenizer via add_token function. This will change the vocab size of the model, and adjust the index accordingly. However the embedding layer has not been changed. Doing a forward pass now will cause an out-of-index error

3. Extract the existing embedding layer weight, detach it and concatenate it with the synthetic embeddding matrix by column. Parameterize this new matrix

4. Invoke the resizing operation of the model to the new length of the tokenizer. The embedding matrix is now correct but the last column is randomly initialized.

5. Overwrite the reference of the weight matrix with the new matrix that has the weights of the synthetic model

Figure 4.8 and Figure 4.9 shows the before and after modification.



Before modification

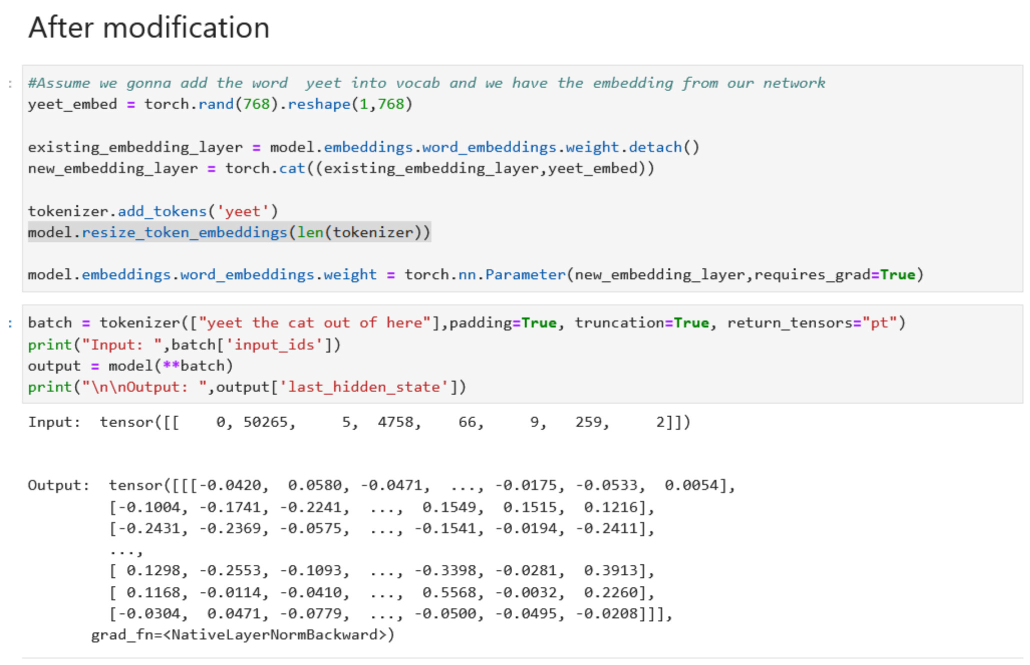
```
[2]: batch = tokenizer(["yeet the cat out of here"],padding=True, truncation=True, return_tensors="pt")
     print("Input: ",batch['input_ids'])
     output = model(**batch)
     print("\n\nOutput: ",output['last_hidden_state'])

     Input:  tensor([[    0, 4717,  594,     5, 4758,    66,     9,   259,     2]])


     Output:  tensor([[[-0.0431,  0.0622, -0.0273,  ..., -0.0284, -0.0659, -0.0054],
              [ 0.0162,  0.2585,  0.3546,  ...,  0.7177,  0.2241, -0.0208],
              [-0.0698,  0.0751,  0.2477,  ..., -0.3056,  0.1697,  0.0822],
              ...,
              [ 0.0429, -0.3221, -0.0973,  ..., -0.4146, -0.0282,  0.3446],
              [ 0.1115, -0.0527, -0.0417,  ...,  0.5172,  0.0404,  0.1875],
              [-0.0341,  0.0536, -0.0554,  ..., -0.0705, -0.0627, -0.0357]]],
            grad_fn=<NativeLayerNormBackward>)
```

FIGURE 4.8: RoBERTa model before modification, unable to tokenize the word 'yeet'

As can be seen before the modification, the word 'yeet' is broken into two separate token, id 4717 and id 594, which is 'ye' and 'et'. These two tokens provide little value into the verb sense of the word 'yeet'.

After the modification, the word 'yeet' is now a new word with id 50265, with its embedding contain much richer information than the combination of 'ye' and 'et'. The output of the model is hence also changed slightly.

Other approach would include modifying the tokenizer files (which would include the vocab file, special token files) as well as the config.json file due to the change in size of the embedding layer. The whole amazon s3 file could also be modified directly, and


After modification

```
#Assume we gonna add the word  yeet into vocab and we have the embedding from our network
yeet_embed = torch.rand(768).reshape(1,768)

existing_embedding_layer = model.embeddings.word_embeddings.weight.detach()
new_embedding_layer = torch.cat((existing_embedding_layer,yeet_embed))

tokenizer.add_tokens('yeet')
model.resize_token_embeddings(len(tokenizer))

model.embeddings.word_embeddings.weight = torch.nn.Parameter(new_embedding_layer,requires_grad=True)
```

```
batch = tokenizer(["yeet the cat out of here"],padding=True, truncation=True, return_tensors="pt")
print("Input: ",batch['input_ids'])
output = model(**batch)
print("\n\nOutput: ",output['last_hidden_state'])
```

```
Input:  tensor([[    0, 50265,     5,  4758,    66,     9,   259,     2]])


Output:  tensor([[[-0.0420,  0.0580, -0.0471,  ..., -0.0175, -0.0533,  0.0054],
         [-0.1004, -0.1741, -0.2241,  ...,  0.1549,  0.1515,  0.1216],
         [-0.2431, -0.2369, -0.0575,  ..., -0.1541, -0.0194, -0.2411],
         ...,
         [ 0.1298, -0.2553, -0.1093,  ..., -0.3398, -0.0281,  0.3913],
         [ 0.1168, -0.0114, -0.0410,  ...,  0.5568, -0.0032,  0.2260],
         [-0.0304,  0.0471, -0.0779,  ..., -0.0500, -0.0495, -0.0208]]],
       grad_fn=<NativeLayerNormBackward>)
```

FIGURE 4.9: RoBERTa model after the modification

then loaded by the from_pretrained(s3_url_to_modified_model). However, the modification suggested in this report is much cleaner, involve only a few more lines of codes and very little storage space (for the synthetic embedding matrix).

Chapter 5

Experiments and Results

5.1 Experiments Setup

For each training corpus, 20% of the data is hold out as validation set. The cosine similarity and loss on this data is monitored. After training is done, the models are tested against web crawled data on unknown words and see which words are most closely related to the synthetic embeddings.

5.2 Validation Results And Analysis

Synthetic embeddings validation results for GloVe embeddings' best experiments are shown in Table 5.1.

known words	cosim score	loss
vaccine	0.966	5.17
sick	0.9799	4.986
poison	0.9483	6.148
pneumonia	0.9072	6.119
pandemic	0.9959	6.443

Table 5.1: Results on validation set for GloVe embeddings

In Table 5.1, the model is able to converge and achieve very high cosine similarity score (best result at 0.991). It also took approximately 25 epoch to each the elbow point on average, and the loss is shown to be monotonically decreasing. This shows that the model is able to learn new information about synthesis process at every step.

The runtime is shown in Table 5.2

known words	cosim score	run time
vaccine	0.966	5min 23s
sick	0.9799	12m 59s
poison	0.9483	6m 48s
pneumonia	0.9072	5m 43s
pandemic	0.9959	28m 27s

TABLE 5.2: Training runtime for GloVe Experiments

Synthetic embeddings validation results for RoBERTa embeddings best experiments are shown in Table 5.3.

known words	cosim score	loss
president	0.9997	138.28
capital	0.9969	84.613
authority	0.891	192.077
governor	0.556	2001.31

TABLE 5.3: Results on validation set for Roberta Embeddings

The runtime is shown in Table 5.4.

known words	cosim score	runtime
president	0.9997	13h 53m 43s
capital	0.9969	5h 20m 6s
authority	0.891	14h 55m 6s
governor	0.556	1h 10m 47s

TABLE 5.4: Training runtime for Roberta Embeddings

There are a few differences between GloVe and RoBERTa synthesis process:

1. RoBERTa takes substantially longer to converge (approximately 14 hours compared to 15 minutes for GloVE), but require much lower epoch to converge (approximately 10 compared to 30 for GloVe)

2. The best cosine similarity score is also not as close to GloVe

3. The L1 loss is the only metric that differ non-substantially between the two embeddings types.

For observation 1, this is expected due to the substantial increase in embeddings size from 100 to 768. With the architecure being fully connected, this leads to an exponential increase in amount of trainable parameters, causing substantial increase in runtime. However, the lower epoch to train suggest that RoBERTa embeddings are much richer in information than GloVe, which is again exepected due to the amount of data RoBERTa was trained on.

Observation 2 may be a corollary of the over-parameterization of the network due high-dimension input vector, as suggested in [46]. For higher dimension vector, it is more challenging for the network to generalize well on all the various senses captured in different element of the embeddings vector.

5.3 Inference Results

The model is able to synthesize the embeddings for 'covid' (Figure 5.1), 'pfizer' (Figure 5.2).

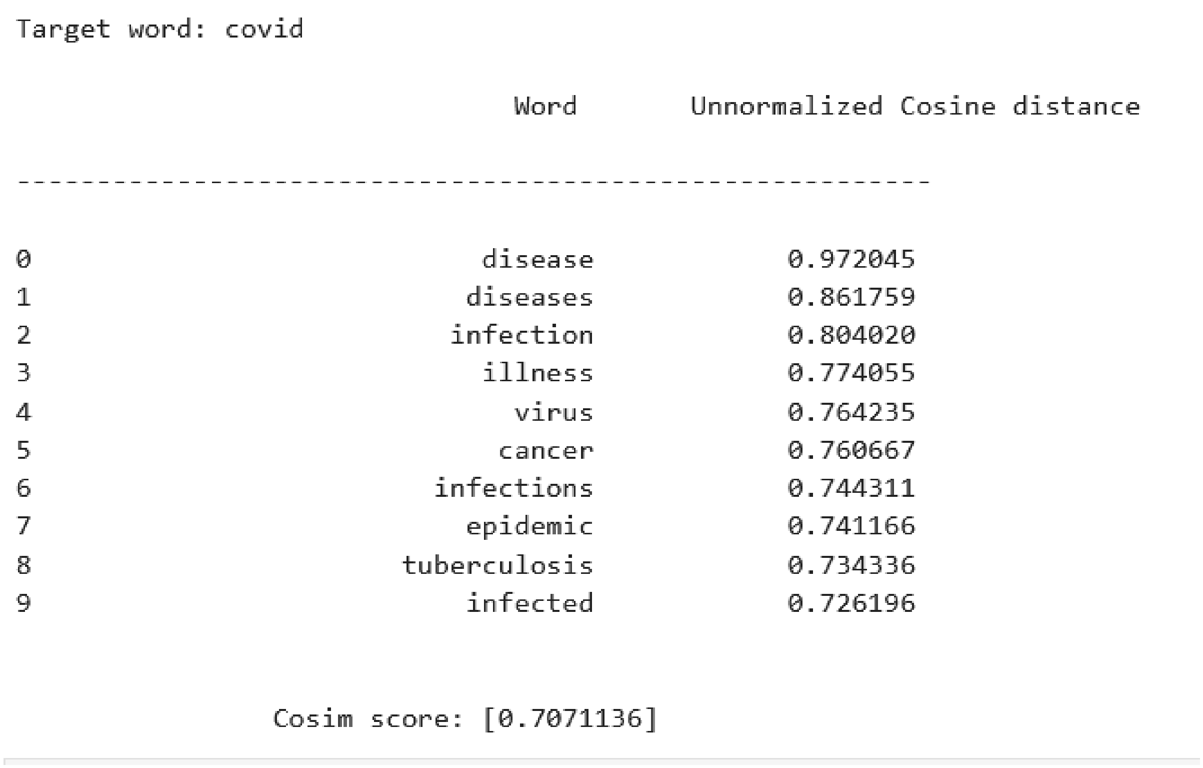
The model was trained on different words with negative examples, and is able to synthesize a new embeddings for a new word that is different from the word it was trained on (the cosine similarity score in training was 0.99, but during inference the synthetic embeddings similarity is 0.8).

Similar result was observed for RoBERTa embeddings in Figure 5.3

However, when faced with Singapore named entities, the model fails to generate a sensible embeddings that correlate well with the entity of interest. For example, in Figure 5.4, for the word tekong which is a location in Singapore, the model does not learn the semantic that well.

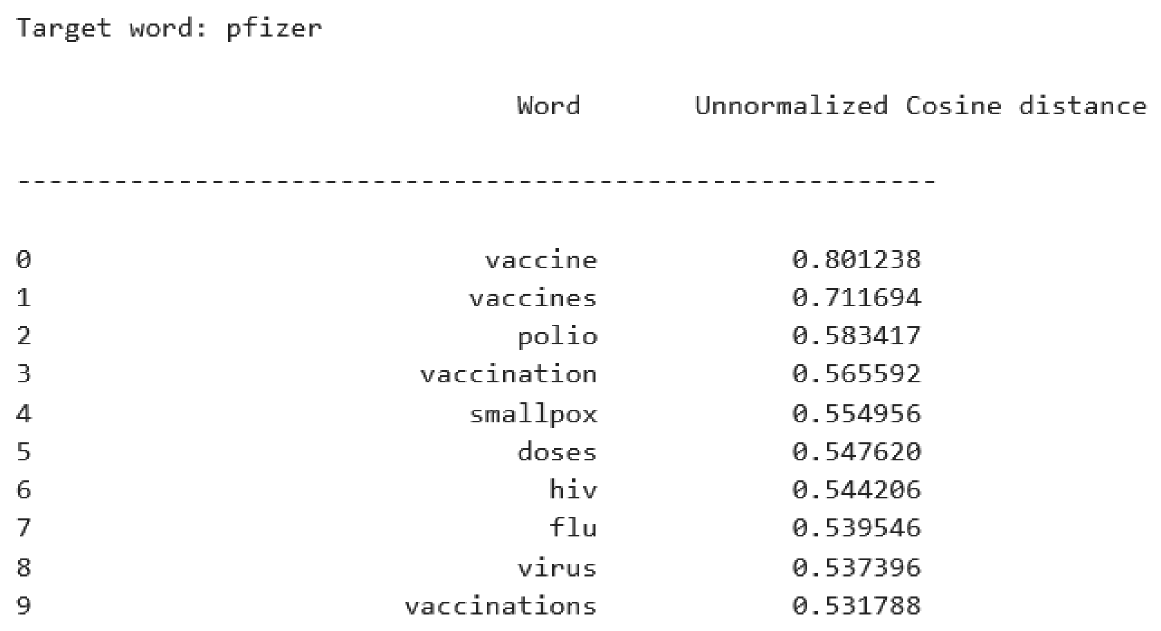
5.4 Effect of Modification

We measure the perplexity of the RoBERTa model before and after the modification. For this task we use the wiki-text-2 test set for perplexity computation.



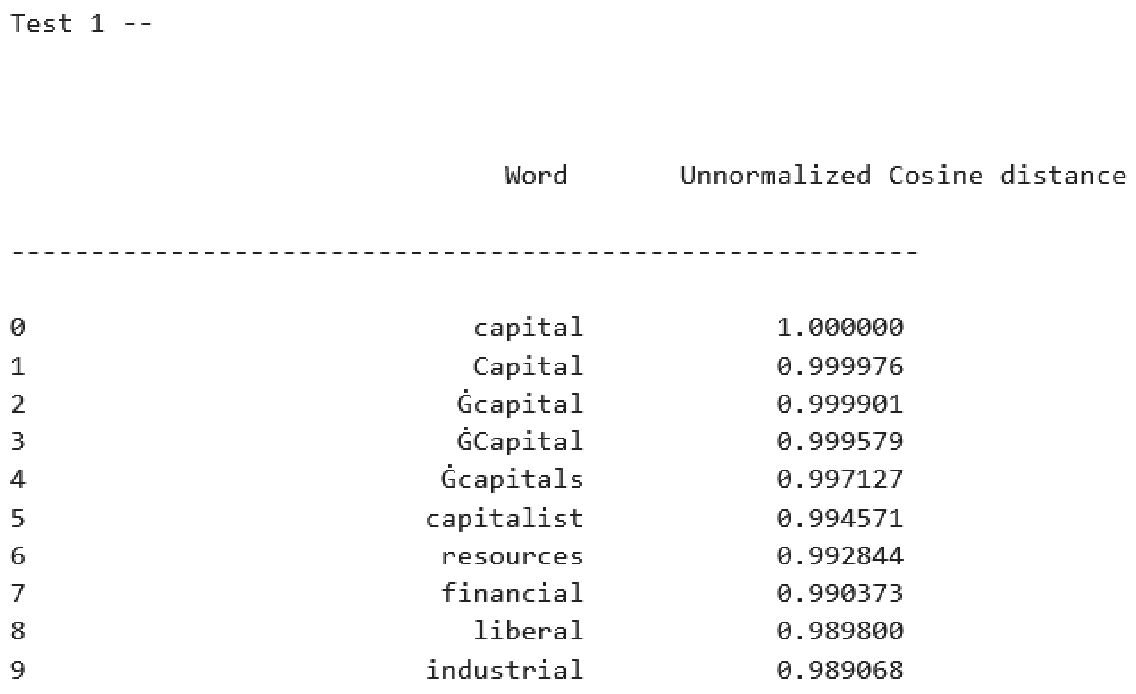
```
Target word: covid

                               Word       Unnormalized Cosine distance

---------------------------------------------------------

0                              disease          0.972045
1                             diseases          0.861759
2                            infection          0.804020
3                              illness          0.774055
4                                virus          0.764235
5                               cancer          0.760667
6                           infections          0.744311
7                             epidemic          0.741166
8                         tuberculosis          0.734336
9                             infected          0.726196


                Cosim score: [0.7071136]
```

FIGURE 5.1: Analogy result for ’COVID’



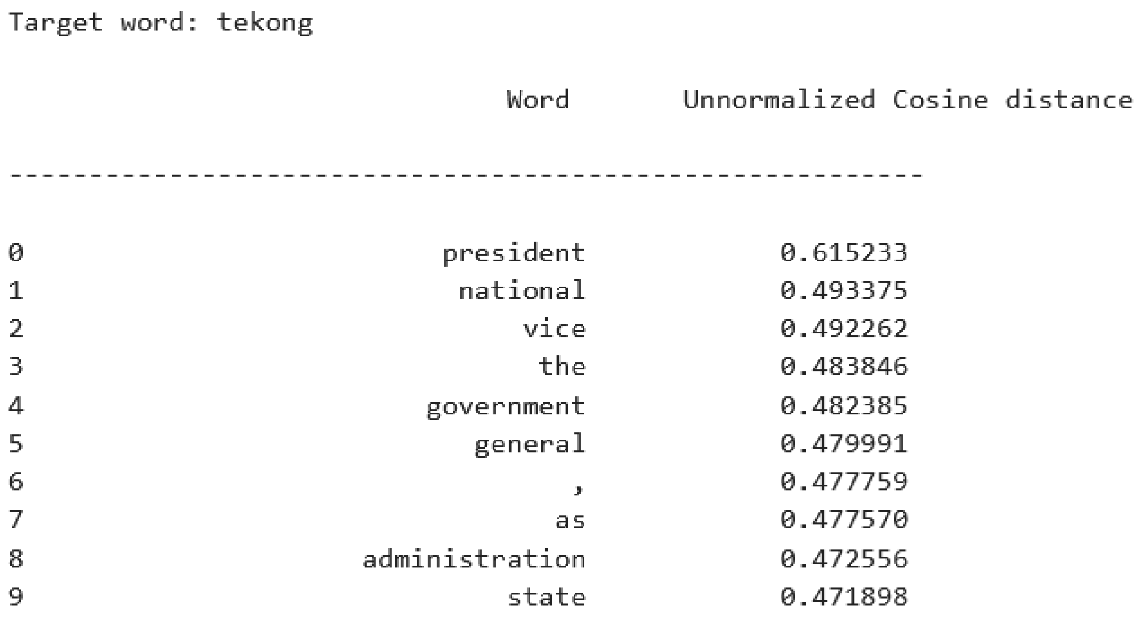
```
Target word: pfizer

                               Word       Unnormalized Cosine distance

---------------------------------------------------------

0                              vaccine          0.801238
1                             vaccines          0.711694
2                                polio          0.583417
3                          vaccination          0.565592
4                             smallpox          0.554956
5                                doses          0.547620
6                                  hiv          0.544206
7                                  flu          0.539546
8                                virus          0.537396
9                         vaccinations          0.531788
```

FIGURE 5.2: Analogy result for ’Pfizer’



```
Test 1 --

                                   Word      Unnormalized Cosine distance

--------------------------------------------------------------

0                                capital           1.000000
1                                Capital           0.999976
2                               Ġcapital           0.999901
3                               ĠCapital           0.999579
4                              Ġcapitals           0.997127
5                             capitalist           0.994571
6                              resources           0.992844
7                              financial           0.990373
8                                liberal           0.989800
9                             industrial           0.989068
```

Figure 5.3: Analogy result for 'Washington'



```
Target word: tekong

                                   Word      Unnormalized Cosine distance

--------------------------------------------------------------

0                              president           0.615233
1                               national           0.493375
2                                   vice           0.492262
3                                    the           0.483846
4                             government           0.482385
5                                general           0.479991
6                                      ,           0.477759
7                                     as           0.477570
8                         administration           0.472556
9                                  state           0.471898
```

Figure 5.4: Analogy result for 'tekong'

Number of words added	Before	After
1	1.8113005	1.8113005
5	1.8113005	1.8113005
10	1.8113005	1.8113012

TABLE 5.5: Changes in perplexity

As can be seen in Table 5.5, there is only a slight perturbation in the overall perplexity. This shows that the addition of the new synthetic embeddings does not affect the overall quality of the embeddings.

5.5 Limitations and Future Improvements

During the course of experiments, there are interesting observation made on the failed experiments. For gigaword corpus, it has been noted that the domain distribution are highly skewed to political and media-related events. This translates to different target word would have vastly different amount of data, and some words would suffer catastrophic failure if trained using solely gigaword as the text does not appear frequently enough to be trained on. Take for example, the word 'cat' and 'authority'. 'cat' rarely makes it into the news, but 'authority' appear in many different articles. Figure 5.5 shows the difference in the two words' training.

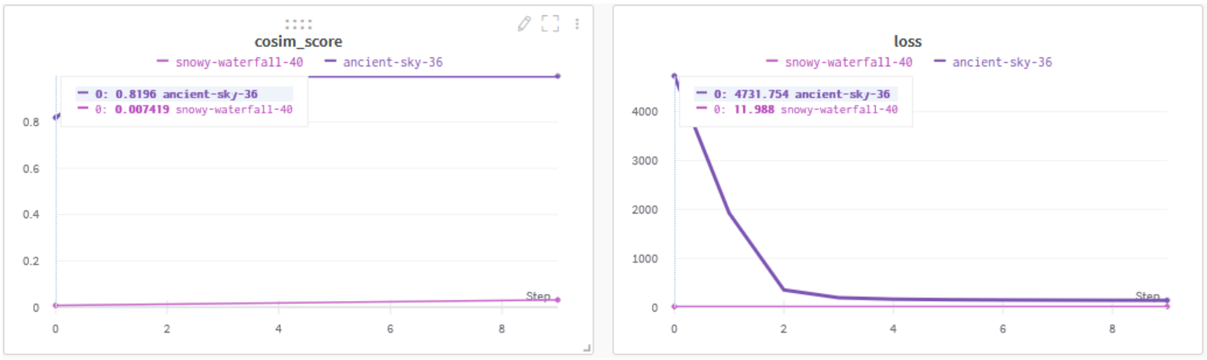


FIGURE 5.5: Catastrophic Failure in Gigaword

Although the loss is low for 'cat', the cosine similarity score increase very slowly due to the small amount of data it can operate on. This poses a substantial hindrance for training of corpus using gigaword.

Due to time constraints and computing resource, as well as scope of the project being a proof of concept, the number of target word that was used to train the model is limited. This severely hinder the ability of the model to generalize. However, from our experiments results with negative samples, the model has the potential to improve significantly with more training data.

From the experiments, it has been noted that there is a positive correlation for the required number of examples used in training with the embeddings size. For GloVe, which is 100 size vector, we need minimally 1000 examples for the model to train sufficiently and generalize, but for RoBERTa, at least 20000 examples are required. A rule of thumb is that the higher the embeddings size, there should be a 10x increase in amount of training data to ensure convergence.

To increase the model capability to generalize to more class of words, a simple way is to train the model with more target words (more negative examples, which is currently kept at 5). According to [24], the maximum number of negative samples should be kept to 25, from which point the marginal benefits drop substantially. Therefore, the model would benefit from increasing the negative samples number more than 5, to increase generalization capability. With such modification, there would be a need for more data as well as longer epoch training to converge.

For cross-language capability, this report's literature review and experiments have shown that data class imbalance pose a huge problem to training, and thus it is important that the data of each language conforms to the suggestion proposed earlier in this section. This might be challenging if the language is not common, but there are possible solutions as proposed by [47] [48].

5.6 Feasibility Study Evaluation

5.6.1 Computing Resource

All of the experiments were run on CPU-only personal computers. It has been demonstrated that even at low resource level, it is possible to run the experiments due to the compact size of the network and batching method to allow efficient computation.

The training time for the network, even with high-dimension embeddings, does not take long to reach convergence.

5.6.2 Portability

The whole setup is contained within a conda environment and thus any conda compatible platform would be able to rerun this experiments, For this report, the experiments were run on 2 separate machine – one Windows and one Linux, both of which send the log results to wandb for centralized monitoring.

The model itself weighs 20MB if fine tuned for GloVe embeddings, and weighs 120MB if fine tuned for RoBERTa. This further ensure that the model would not take up too much additional space for downstream deployment.

Overall, it is feasible for the system proposed to be deployed in a production environment due to its light-weight and fast inference time.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this report, an end-to-end embedding synthesis pipeline was designed to generate synthetic embedding for unknown words regardless of the embedding dimension. The pipeline started with gathering large text corpus, highlighted some strength and weaknesses of each corpus in light of the synthesis objective. A pre-processing pipeline is then designed to capture context words for each desired target word, which was then fed into the feed-forward neural network to optimize on the L1 Loss. After training, the model is tested on the validation set and inference data on the web before the synthetic embedding generated can be incorporated into the existing system.

As this is a proof-of-concept for the task, and there was constraints on the computing resources available, the focus was put into designing a workable solution for the embedding synthesis objective. From the experiments done, it is shown that the model is capable of synthesizing embedding for unknown word that has the semantics desired on specific group of words. With more data and computational resources, the model can improve substantially on its ability to generalize.

The system designed is light-weight and fast in both training and deployment, subjecting to the size of the word vector. The report also suggested methods to incorporate the synthetic embedding into existing system, for both GloVe and RoBERTa embedding. The perplexity test has shown that the modification does not affect the overall performance of the existing embeddings, while complimenting its performance by providing more meaningful embedding to unknown words. THe embedding can be used for multiple downstream NLP tasks, be it NER or POS tagging.

6.2 Future Work

The system proposed in this report is only a proof-of-concept and have a lot of potential to improve further. In this section, we would be making suggestions of future works that can benefit from this existing report.

6.2.1 Pruning

As observed with high dimensional vector, the network trainable parameters get increasingly large for RoBERTa. Therefore, adaptation of the existing fully connected network is extremely desirable and beneficial. One of the possible method would be to prune the network to significantly downsize the fully connected layers. Dropout and 1D-CNN can also be incorporated into the network to reduce the number of connections among neurons. The im would be to keep the model as light-weight as possible, making it practical for deployment purposes.

6.2.2 Improvement in Input Data

Experiments in this report has shown that more quality input data would be beneficial for the network to generalize. In order to synthesize for larger class of words, the preprocessing pipeline can be improved much more with NER or POS tagger to ensure we distil words with highest predictive and explanatory features for the objective of synthesizing embedding. Additionally, the use of Common Crawl and other large corpora of text would supplement the usage of only wiki-text and gigaword, in order to combat the issue of catastrophic failure observed in some of the gigaword experiments. With more data, the domain coverage would improve and effectively allow the model to train much better. A rule of thumb suggested by this report is that each word class should have at least 1000 examples for effective training.

6.2.3 Cross Lingual Embedding

As downstream applications increasingly deal with a variety of languages apart from English, the capability to generate embeddings that encapsulate semantics in multiple languages are desirable. In order to incorporate this capability into our network, there are two possible approaches: train the network on existing cross lingual embedding such as bert-base-multitlingual case or align the trained embedding in English with other languages and capture the translation function. Of the two methods, this report suggests that the former is a more viable option due to its ability to robustly capture the semantics presence in each language. However, it would demand a much larger text corpus (prefereably larger than the wiki text and gigaword text combined), in which dataset imbalance may pose a big issues. If such approach is to be perused, it is essential to collect a balanced multilingual corpus to train the model.

Bibliography

[1] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021. vi, 15

[2] Robyn Speer and Joshua Chin. An ensemble method to produce high-quality word embeddings (2016), 2019. vi, 17, 18, 19

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`. vi, 2, 4, 40, 41

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. 1

[5] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://aclanthology.org/D14-1162`. 1, 16, 25

[6] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://aclanthology.org/W13-3512`. 4, 19, 30

[7] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1036. URL `https://aclanthology.org/D15-1036`. 4

[8] Minh-Thang Luong and Christopher D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. *CoRR*, abs/1604.00788, 2016. URL `http://arxiv.org/abs/1604.00788`. 8, 9

[9] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL `https://aclanthology.org/P16-1162`. 9

[10] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, 2012. doi: 10.1109/ICASSP.2012.6289079. 10

[11] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates, 2018. 11

[12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. 13

[13] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018. 13

[14] Subendhu Rongali, Abhyuday Jagannatha, Bhanu Pratap Singh Rawat, and Hong Yu. Continual domain-tuning for pretrained language models, 2021. 14

[15] Yitong Li, Timothy Baldwin, and Trevor Cohn. What's in a domain? learning domain-robust text representations using adversarial training, 2018. 14

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`. 14

[17] Serge Sharoff. Know thy corpus! robust methods for digital curation of web corpora, 2020. 14

[18] Leon Strømberg-Derczynski, Manuel R. Ciosici, Rebekah Baglini, Morten H. Christiansen, Jacob Aarup Dalsgaard, Riccardo Fusaroli, Peter Juel Henrichsen, Rasmus Hvingelby, Andreas Kirkedal, Alex Speed Kjeldsen, Claus Ladefoged, Finn Årup Nielsen, Malte Lau Petersen, Jonathan Hvithamar Rystrøm, and Daniel Varab. The danish gigaword project, 2021. 14

[19] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission, 2020. 14

[20] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, Sep 2019. ISSN 1460-2059. doi: 10.1093/bioinformatics/btz682. URL `http://dx.doi.org/10.1093/bioinformatics/btz682`. 15

[21] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge, 2018. 16, 17

[22] Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons, 2015. 17

[23] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. doi: 10.1162/tacl_a_00134. URL `https://aclanthology.org/Q15-1016`. 18

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013. 21, 22, 29, 39, 53

[25] Alex Gittens, Dimitris Achlioptas, and Michael W. Mahoney. Skip-gram Zipf + uniform = vector additivity. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 69–76, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1007. URL `https://aclanthology.org/P17-1007`. 22

[26] Giovanni Alcantara. Empirical analysis of non-linear activation functions for deep neural networks in classification tasks. *CoRR*, abs/1710.11272, 2017. URL `http://arxiv.org/abs/1710.11272`. 22

[27] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to pmi-based word embeddings, 2019. 22

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. 25

[29] Rui Mao, Chenghua Lin, and Frank Guerin. Combining pre-trained word embeddings and linguistic features for sequential metaphor identification, 2021. 25

[30] Eda Okur, Shachi H Kumar, Saurav Sahay, and Lama Nachman. Towards multimodal understanding of passenger-vehicle interactions in autonomous vehicles: Intent/slot recognition utilizing audio-visual data, 2019. 25

[31] Li Lucy and Jon Gauthier. Are distributional representations ready for the real world? evaluating word vectors for grounded perceptual meaning. In *Proceedings of the First Workshop on Language Grounding for Robotics*, pages 76–85, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2810. URL `https://aclanthology.org/W17-2810`. 25

[32] Tanvi Dadu and Kartikey Pant. Towards code-switched classification exploiting constituent language resources, 2020. 25

[33] Genta Indra Winata, Samuel Cahyawijaya, Zihan Liu, Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Are multilingual models effective in code-switching?, 2021. 25

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. 27

[35] Hugo Caselles-Dupré, Florian Lesaint, and Jimena Royo-Letelier. Word2vec applied to recommendation: Hyperparameters matter, 2018. 28

[36] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications, 2020. 28

[37] Robyn Speer and Joshua Chin. An ensemble method to produce high-quality word embeddings. *CoRR*, abs/1604.01692, 2016. URL `http://arxiv.org/abs/1604.01692`. 30

[38] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C.-C. Jay Kuo. Evaluating word embedding models: methods and experimental results. *APSIPA Transactions on Signal and Information Processing*, 8, 2019. ISSN 2048-7703. doi: 10.1017/atsip.2019.12. URL `http://dx.doi.org/10.1017/ATSIP.2019.12`. 30, 42

[39] Asahi Ushio, Luis Espinosa-Anke, Steven Schockaert, and Jose Camacho-Collados. Bert is to nlp what alexnet is to cv: Can pre-trained language models identify analogies?, 2021. 30, 42

[40] Giuseppe Attardi. Wikiextractor: A tool for extracting plain text from wikipedia dumps, Dec 2020. URL `https://github.com/attardi/wikiextractor`. 33

[41] Bentz James W Bottou L eon Guyon Isabelle LeCun Yann Moore Cliff Sackinger Eduard Bromley, Jane and Roopak Shah. Signature verification using a siamese time delay neural network, 1993. 38

[42] Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding, 2018. 38

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 39

[44] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven HOI, and Weinan E. Towards theoretically understanding why sgd generalizes better than adam in deep learning, 2020. 39

[45] Ali Tourani and Amir Seyed Danesh. Using exclusive web crawlers to store better results in search engines' database, 2013. 43

[46] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study, 2018. 49

[47] Ajay Kulkarni, Deri Chong, and Feras A. Batarseh. Foundations of data imbalance and solutions for a data democracy, 2021. 53

[48] Lian Yu and Nengfeng Zhou. Survey of imbalanced data methodologies, 2021. 53