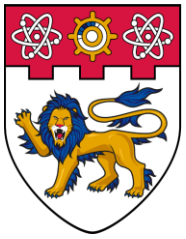


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4020 Project 1
Technical Review Report

Name	Matriculation No.
Lee Ying Ying, Pamela	U2121263A
Pearlina Tan Qinlin	U2221690F
Wong Yi Xian	U2121401C
Gwee Jia Xiang	U2122287H

Abstract

This report investigates the performance of two different similarity search methods - Sentence2Vec (Sentence2Vec) and Bidirectional Encoder Representations from Transformers (BERT) using two datasets taken from Kaggle. The models were trained on the training data, fine-tuned using validation data, and assessed on the test data to compare the performance between the two methods. Our findings highlight the strengths and limitations of both models, providing insights into their effectiveness in similarity search tasks. The results demonstrate that while BERT's contextual embeddings generally lead to better performance for complex datasets, Sent2Vec provides a computationally efficient alternative with comparable performance on simpler datasets.

1 Introduction

Similarity search is instrumental in applications like search engines, recommendation systems, and plagiarism detection. It works by determining how similar two pieces of text are, whether at the level of individual words, sentences, or entire documents.

Word-level representations allow models to understand linguistic patterns, like synonyms or analogies, at the word-level. However, it is insufficient for sentence or document level representations due to the lack of context. Sentence embeddings such as Sentence2Vec and transformer-based models like Bidirectional Encoder Representations from Transformers (BERT), are crucial in extending word-level representations to encompass sentence-level representations

In this report, we aim to analyse the effectiveness of Sentence2Vec and BERT in the task of similarity search. By comparing these two methods, we hope to understand how well each approach captures semantic similarities between sentences, and how their underlying architectures impact performance in various similarity search tasks. Through a series of experiments, we analyse the strengths and limitations of both models, providing insights into which method is better suited for real-world applications requiring sentence-level similarity analysis.

2 Literature Review

2.1 Sentence2Vec

Sentence2Vec is an extension to the concept of Word2Vec, a natural language processing technique for obtaining vector representations of sentences enabling computers to understand and process natural languages more effectively. Sentence2Vec uses pre-trained word embeddings from Word2Vec to capture semantic meaning of sentences by taking the words and their context into consideration and assigning numerical vertices to the entire sentence. Through the conversion of sentences into numerical vertices, Sentence2Vec makes tasks such as sentiment analysis, document retrieval and text classification possible (ActiveLoop, n.d).

To better understand Sentence2Vec, we have to first understand how Word2Vec works.

2.2 Word2Vec

Word2Vec generates a vector representation for each word in the vocabulary and effectively captures semantic relationships by positioning words that appear in similar contexts close together in the vector space. As a result, similar words have similar vector representations (Suri, 2022).

2.3 Architecture of Word2Vec

Word2Vec is a shallow neural network consisting of the input, hidden and output layers as seen in the figure below.

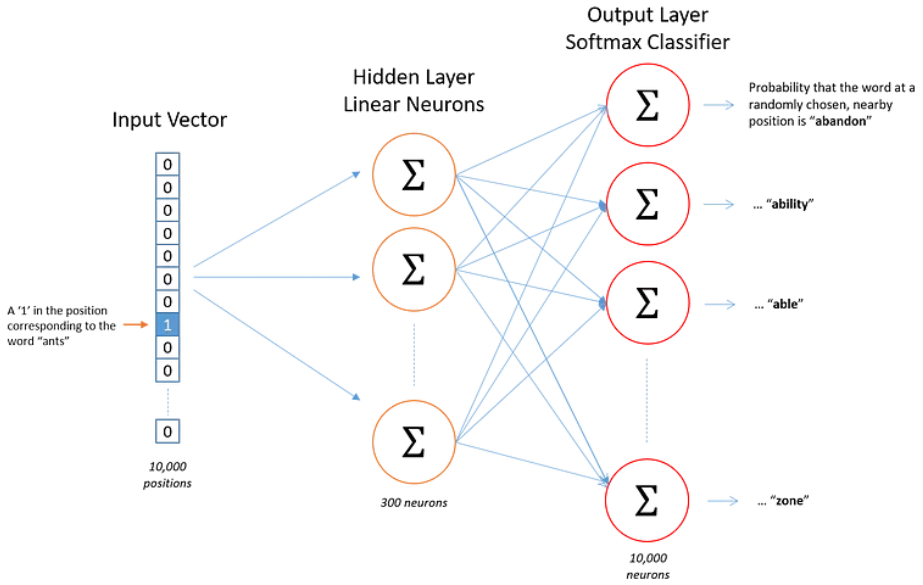


Figure 1. Neural Network Layers of Word2Vec (Suri, 2022)

Input Layer

The input layer consists of all the documents and texts from the training set, which are represented with one-hot encoding to enable the network to process these texts (Suri, 2022).

Hidden Layer
The size of the hidden layer corresponds to the dimensionality of the word embeddings being learned (Suri, 2022).

Output Layer
Each word has a set of *n* weights that represent its various characteristics which are treated as word embeddings. The output layer uses these embeddings to predict the context word or target word when given a particular input (Suri, 2022).

2.4 Types of Word2Vec
The 2 main architectures for training Word2Vec are Continuous Bag-Of-Words (CBOW) and Skip-gram.

2.4.1 Continuous Bag-Of-Words (CBOW)
CBOW takes into consideration the surrounding words around the target's position as context to predict the target word. The context words are typically within a window of words, and their order does not matter in CBOW's prediction process (Suri, 2022).

Input Layer
In CBOW, the input is the surrounding context words within the defined window. The defined window size would determine the number of context words. For example, if the defined window size is 2, the number of context words would be 2 infront and 2 behind for a total of 4.

Hidden Layer
In the hidden layer of CBOW, the one-hot encoded vectors of the context words are projected into a lower-dimensional continuous vector space, known as the embedding space. The size of the hidden layer corresponds to the number of dimensions in the word embeddings. The context word vectors are averaged and passed through the hidden layer, where the model learns to represent words in this reduced space during training. Since the vectors are averaged, the order of the context words does not affect the outcome.

Output Layer
The output layer uses a softmax function that outputs a probability for all possible words and the word with the highest probability is chosen as the predicted target word. The model is trained to improve the accuracy of these predictions over time by adjusting the word embeddings learned in the hidden layer.

2.4.2 Skip-gram
Skip-gram attempts to predict the surrounding context words given a target word (Suri, 2022). It aims to achieve this by learning word embeddings through maximising the probability of context words with the following equation

$$\prod_{-c \le j \le c, j \neq 0} P(w_{t+j}|w_t)$$

where:

- w_t is the target word
- c is the context window size
- w_{t+j} are the context words around w_t
- The product is taken over all positions j in the context window

Input Layer

The input layer takes in a one-hot encoded vector that represents the target word. This vector has a length equal to the vocabulary size, with the target word index set to 1 and all other indices set to 0.

Hidden Layer

In the hidden layer of skip-gram, the one-hot vector of the target word is projected into a lower-dimensional embedding space. Similar to CBOW, the size of the hidden layer corresponds to the dimensionality of the word embeddings. By using a weight matrix with rows representing the word embeddings, the hidden layer learns to map each one-hot vector to a dense embedding vector. During training, the weight matrix is constantly updated to minimise prediction error.

Output Layer

The output layer consists of a series of softmax functions that output probabilities for all words in the vocabulary, predicting multiple context words independently for each target word. The model aims to maximise the probability of predicting the correct context words for each target word.

2.5 Overview of Word2Vec

Word2Vec is able to help match words with similar semantic meaning. However, Word2Vec can only take 1 word each time, while a sentence consists of multiple words (Stanleyfok, 2017). To obtain sentence similarity, Sentence2Vec, a wrapper of Word2Vec, is used.

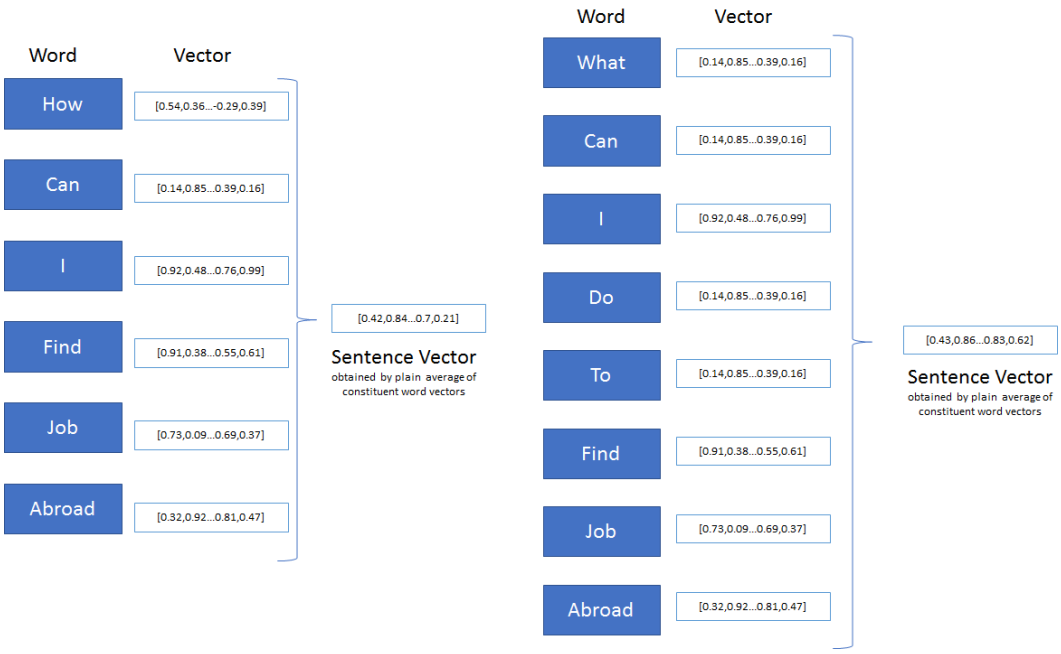


Figure 2. Illustration of Sentence2Vec (Stanleyfok, 2017)

2.6. BERT

While Sentence2Vec has shown great promise in the NLP domain, its inability to capture contexts beyond the sentence boundary is an important limiting factor, especially in tasks requiring crucial paragraph or document-level context. Therefore, we turn to BERT to address these limitations with its transformer architecture.

2.6.1 Transformer Architecture

A transformer architecture consists of an encoder-decoder structure where the encoder takes in an input sequence and outputs a vector for every element in that sequence. The decoder uses the encoder's features and other inputs to generate a target sequence. It then uses the predicted sequence as input to generate the next sequence, making it an auto-regressive model. The decoder is usually optimised for generating outputs and the encoder-decoder structure can be used independently depending on the NLP task.

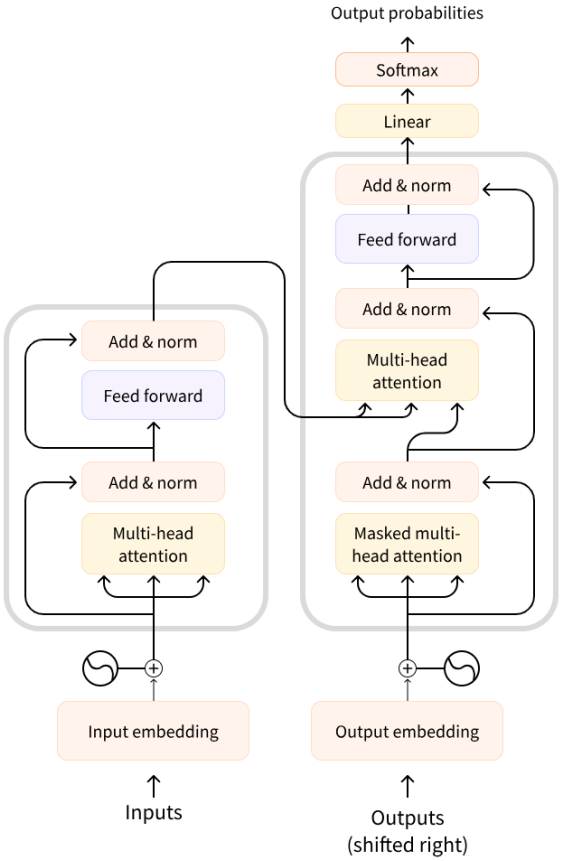


Figure 3. Standard Transformer Architecture

2.6.2 BERT Model

The BERT model only uses the encoder portion of the transformer architecture. It consists of n-identical layers with each n-layer consisting of two components: the multi-head self-attention mechanism and a fully connected feed-forward neural network.

The multi-head self-attention mechanism allows BERT to record relationships between all words in an input sequence simultaneously, thereby creating contextual embeddings that takes into account words appearing before and after the target word.

BERT's approach consists of 2 steps: Pre-training and Fine Tuning. BERT is trained on two training objectives: Mask Language Model (MLM) and Next Sentence Prediction (NSP).

Pre-Training

BERT is trained on large amounts of unlabeled text (Wikipedia and BooksCorpus) to understand the overall structure and semantics of various languages. Additionally, the primary objective is to enable BERT to capture contextual relationships between words and sentences.

Masked Language Model

The first training objective is on MLM where a word in a sentence is masked using the [MASK] token, and the model is forced to analyse its context and remaining words to predict the [MASK] token.

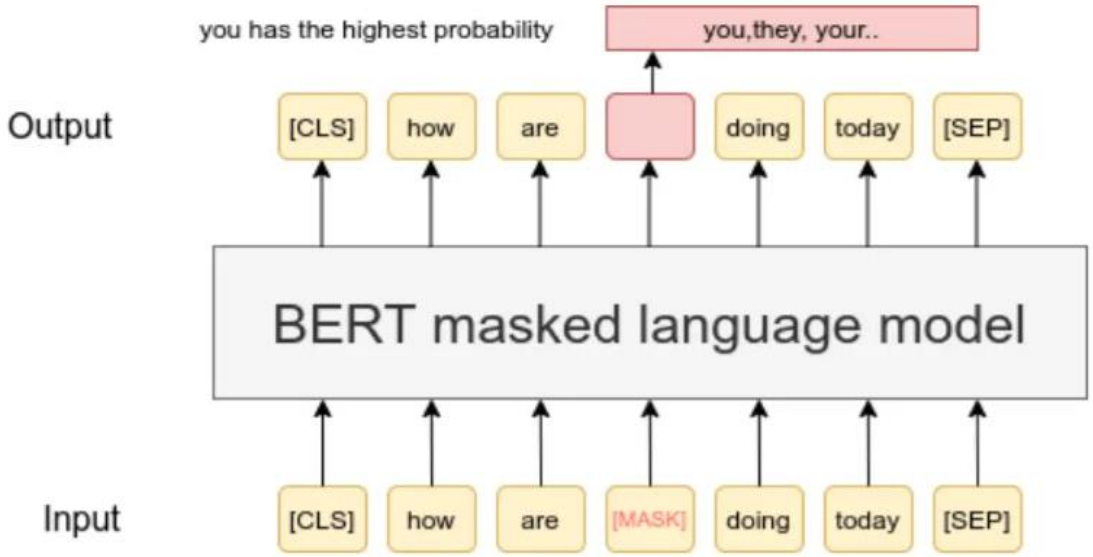


Figure 4. Example of MLM Prediction Task (Yadav, 2023)

During this training objective, the model first receives masked input from the input and learns to predict the [MASK] token by minimising prediction error. The model is thus forced to use the remaining words and its context to predict, therefore creating a bidirectional manner where the model uses both left-to-right and right-to-left contexts instead of a unidirectional context (Yadav, 2023).

Next Sentence Prediction

The second training objective is on NSP which identifies connected and disconnected sentences by predicting whether the second sentence is a continuation of the first sentence.

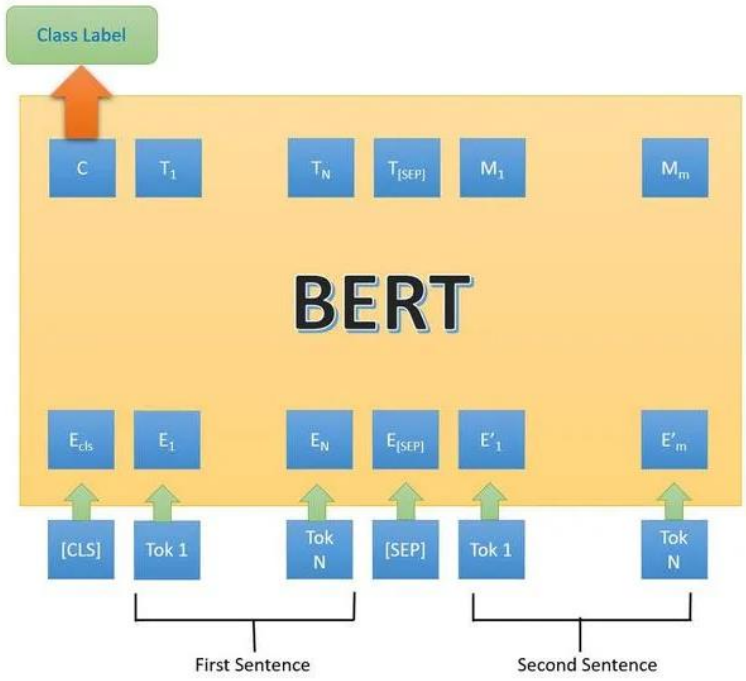


Figure 5. NSP Example in training (Yadav, 2023)

The objective of this training is to help BERT capture context and build a deeper understanding of the semantics and nuances of the language. This is done by determining a given pair of sentences: 1) whether the second sentence is a continuation of the first or 2) the second sentence is randomly sampled from a corpus through a binary classification layer added on top of the BERT model. The classification layer takes the final hidden state representation from the [CLS] token and outputs a probability using a SoftMax activation function (Yadav, 2023).

Sentence 1: "Studying is fun."

Sentence 2: "I am going to eat at North Spine."

[CLS] Studying is fun. [SEP] I am going to eat at North Spine. [SEP]

Figure 6. Example of an NSP input

Figure 6 shows an example input sequence into the BERT model for classification. The [CLS] token is a special token that represents the entire input sequence while the [SEP] token separates the 2 sentences. This input sentence is then passed through many encoder models as shown in Figure 5 to capture the relationships between tokens and learn its contextual representations (GeeksforGeeks, 2024).

During pre-training, MLM and NSP are trained together. The model aims to minimise the combined loss function of MLM and NSP to produce a robust model capable of understanding context within sentences and relationships between sentences.

Fine Tuning

After pre-training, BERT is then fine tuned on the downstream task using labelled data provided. The fine-tuning process involves adjusting the model parameters to suit the downstream task. Modifications include adding task specific layers or adjusting to the task specific labelled data. It is then further trained to minimise the task specific loss function.

2.6.3 BERT Architectures

There are many variations of BERT architectures such as the bert-base and bert-large models with various parameters. The model used in this project is the BERT-base model which contains 12 encoder layers, 12 attention heads per layer and a hidden size of 768 to give us contextual embeddings of dimensions 768.

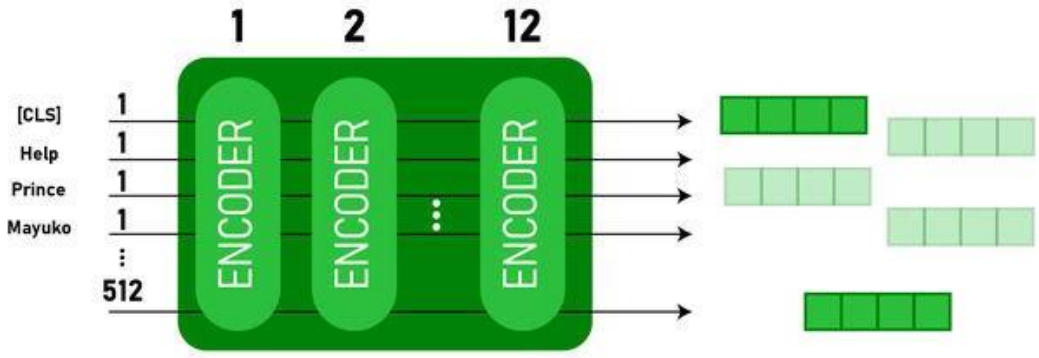


Figure 7. Bert-Base with 12 encoder layers (GeeksforGeeks, 2024)

BERT requires the input to be in a specific format: all inputs are to be tokenized and special tokens are added to form the input representation. Similarly with Figure [], words are broken up into tokens using a tokenizer and special tokens are added depending on the downstream task at hand. Each word would generate a contextual embedding size of 768 dimensions and these numeric representations are generated after taking in the context of its left and right. The generated embeddings can be used for further analysis or as inputs to another model (GeeksforGeeks, 2024).

3 Methodology

3.1 Datasets

We obtained our datasets from Kaggle to evaluate the performance of Sentence2Vec and BERT in similarity search tasks. These datasets vary in structure and purpose, allowing for a comprehensive comparison of the 2 models.

3.1.1 Dataset 1: Semantic Textual Similarity Benchmark (STS-Benchmark) Dataset

The STS Benchmark dataset provides pairs of sentences in JSON Lines (jsonl) format, where each entry contains several fields relevant to the pair of sentences being compared. The following figure shows an example entry.


```
"root" : { 8 items
  "split" : string "test"
  "genre" : string "main-captions"
  "dataset" : string "MSRvid"
  "year" : string "2012test"
  "sid" : string "0024"
  "score" : float 2.5
  "sentence1" : string "A girl is styling her hair."
  "sentence2" : string "A girl is brushing her hair."
}
```


Figure 8. Example Entry

The dataset consists of pairs of sentences that is accompanied by a numerical score representing the degree of semantic similarity between the two sentences, ranging from 0 (completely dissimilar) to 5 (completely equivalent in meaning). This value is human-annotated, and provides a benchmark for assessing how well the model can capture and represent the semantic relationships between sentences. We normalise this score provided and use it as the ground truth when evaluating our models.

3.1.2 Dataset 2: Text Document Classification Dataset

This dataset contains 2,225 rows and 2 columns. Each row contains a document, which could be an article or a short piece of text from one of five categories (Politics, Sport, Technology, Entertainment, Business). The labels range from 1 to 5, indicating the different categories the documents belong to.

```
df = pd.read_csv('df_file.csv')
df['Text'] = df['Text'].astype(str)
df['Label'] = df['Label'].astype(str)
df.head()
```

	Text	Label
0	Budget to set scene for election\n \n Gordon B...	0
1	Army chiefs in regiments decision\n \n Militar...	0
2	Howard denies split over ID cards\n \n Michael...	0
3	Observers to monitor UK election\n \n Minister...	0
4	Kilroy names election seat target\n \n Ex-chat...	0

Figure 9. Example of Dataset 2

3.1 Sentence2Vec Model Implementation

3.1.1 Sentence2Vec Model on STS Dataset

3.1.2.1 Text Tokenisation

The model implementation begins with preprocessing the textual data by converting the sentences in the dataset into a structured format that the Word2Vec model can understand. Each sentence in the dataset is tokenised and then trained on a Word2Vec model.

3.1.2.2 Training, Validation and Test Data

The training, validation and test data were provided on Kaggle.

3.1.2.3 Training the Sentence2Vec Model

The Word2Vec model is trained on the tokenised training data. After that, the word vectors for each sentence are averaged to create sentence embeddings. This process is applied to both the training and validation sets, resulting in sentence vectors that capture the overall meaning of each sentence.

3.1.2.4 Validation Dataset & Evaluation Metrics

The model's performance is initially assessed on the validation dataset. Key evaluation metrics—such as F1-Score, Precision, Recall and Accuracy—were used to measure how well the model generalises to unseen data during this phase. These metrics ensure the model strikes the right balance between accuracy and misclassification.

3.1.2.5 Hyperparameter Tuning

First, we experimented with a range of epoch values—[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]—using both the Skip-Gram (SG) and Continuous Bag of Words (CBOW) architectures. Importantly, instead of using standard classification metrics, we utilised Mean Squared Error (MSE) to determine the best epoch value. This approach was appropriate as the dataset provides ground truth similarity scores, allowing us to compute a regression-based loss (MSE) to measure the difference between predicted and actual similarity scores.

We further tested a range of vector sizes—[50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300]—to identify the best vector dimensionality for the embeddings. The optimal vector size and epoch value were chosen based on the lowest MSE and their corresponding impact on classification metrics like F1-Score.

3.1.2.6 Testing on Unseen Data

After hyperparameter tuning, the final model was evaluated on the unseen test dataset. F1-Score, Precision, and Recall were again used as performance metrics to determine the model's effectiveness.

3.1.2 Sentence2Vec Model for Text Classification Document Dataset

3.1.2.1 Text Tokenisation

The model implementation begins with preprocessing the textual data. This involves converting the raw text into a structured format that the Word2Vec model can understand. Each sentence in the dataset is tokenised, meaning it is split into individual words (tokens), with common preprocessing steps such as lowercasing, removing punctuation, and eliminating stop words. After the text is cleaned, a Word2Vec model is applied. The model learns word embeddings, which are dense vector representations of words that capture their semantic relationships based on their surrounding context in the text.

3.1.2.2 Training, Validation and Test Data

The dataset is split into training (80%), validation (10%), and test (10%) sets, with the training set used for learning the word embeddings, the validation set used to fine-tune the hyperparameters, and the test set reserved for evaluating the final model's performance.

3.1.2.3 Training the Sentence2Vec Model

The Word2Vec model is trained on the tokenised training set. We then create sentence embeddings for both the training and validation sets by averaging the word vectors.

3.1.3.4 Nearest Neighbour Classification

Instead of using traditional classifiers, we implemented a Nearest Neighbour approach to classify sentences based on their vector representations. For each sentence in the validation dataset, its vector representation (computed using Word2Vec) is compared with the vector representations of all sentences in the training dataset. The sentence is classified based on the label of the nearest neighbour (the training sentence with the closest vector, based on cosine similarity or Euclidean distance).

3.1.2.5 Validation Dataset & Evaluation Metrics

The trained model is then tested on the validation dataset. The model’s performance on the validation data is then evaluated using the following evaluation metrics: F1-Score, Precision and Recall. A confusion matrix is also created.

3.1.2.6 Hyperparameter Tuning

First, we tested on a range of epoch values [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]on CBOW. We then used the F1-score to determine the best epoch value.

Then, we tested on a range of vector sizes [50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300] and used the F1-score to determine the best vector size value.

3.1.2.7 Test on Unseen Data

The tuned training model was then tested on the unseen test dataset and the following evaluation metrics were used to determine the model's efficacy: F1-Score, Precision and Recall. A confusion matrix is also created.

3.2 BERT Model Implementation

3.2.1.1 BERT Model for STS dataset

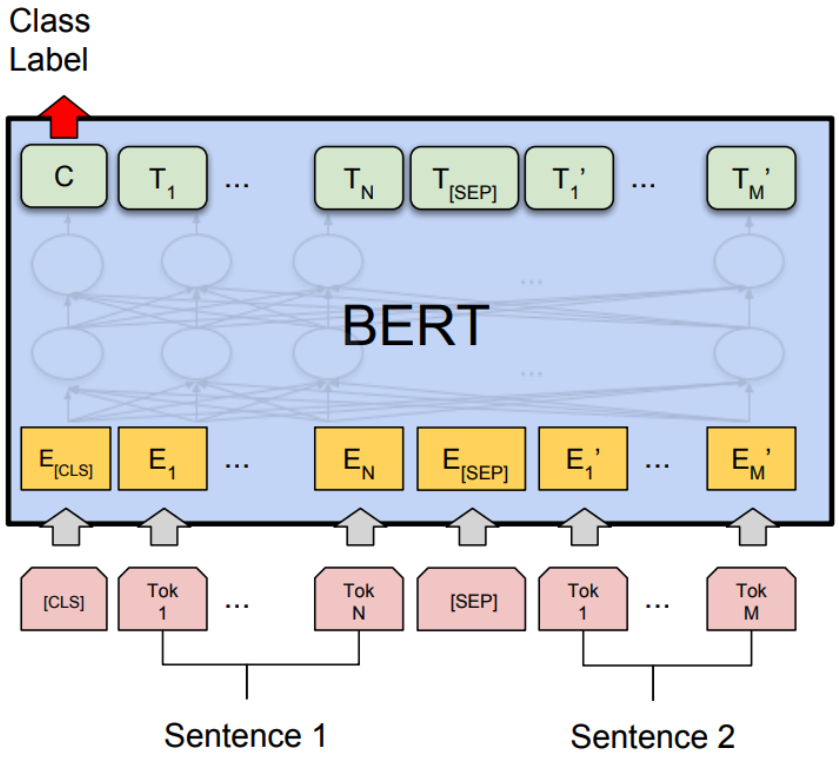


Figure 10. The BERT architecture (Yadav, 2023)

For the first STS dataset, we specifically use BertForSequenceClassification, designed for classification tasks by appending a classification layer on top of the BERT architecture. This model has been pre-trained on a vast corpus of text data and can be fine-tuned for various downstream tasks, including sequence classification.

3.2.1.2 Retrieving Data

The implementation first includes a function to read the JSONL files, extracting sentence pairs for training and testing. Each line in the file corresponds to a JSON object, from which the relevant sentence pairs are retrieved and stored in a list.

3.1.1.3 Tokenize Inputs to Reduce Max Length

The input sentences are tokenized using the BERT tokenizer. The maximum sequence length is reduced to **128** tokens to speed up processing and reduce training time. This is crucial for ensuring that the input size remains manageable, particularly when training with large datasets. The tokenizer prepares the input data for the model by converting sentences into input IDs and attention masks.

3.1.1.4 Create Dataloader

The tokenized inputs are wrapped in a PyTorch DataLoader, which facilitates efficient batch processing and shuffling of data during training. The DataLoader is critical for managing the training loop and ensuring that the model can handle large datasets effectively.

3.1.1.5 Training Configuration

We utilised the **AdamW optimize**r with its default parameters. It is an extension of the original Adam optimizer (Kingma, D. P., & Ba, J., 2017) that uses decoupled weight decay regularisation that helps prevent overfitting. This is particularly useful in complex models with large datasets where the optimizer has the ability to adapt the learning rate for each parameter individually, leading to a more efficient training.

The training phase involves a series of steps:
Forward Pass: The model processes the input data and generates predictions.
Dummy Loss Function: A Mean Squared Error (MSE) loss function is employed to measure the model's performance
Backward Pass: Gradients are computed, and the model's weights are updated using the optimizer.

3.1.1.6 Hyperparameter Tuning

We experimented with a range of epoch values—[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150], then we utilised Mean Squared Error (MSE) to determine the best epoch value. This approach was appropriate as the dataset provides ground truth similarity scores, allowing us to compute a regression-based loss (MSE) to measure the difference between predicted and actual similarity scores.

We further tested with different learning rates, as we researched, we found out that learning rates for BERT models are usually small, ranging from (1e-5 to 5e-5). Hence we tested it out, which was important to see which learning rate had a lower MSE.

3.1.1.7 Evaluation Metrics

To evaluate the performance of our models, we tracked a suite of metrics:

1. **Accuracy**: The proportion of correct predictions over the total number of cases evaluated, a measure of the model's overall performance
2. **Precision**: The ratio of true positive predictions to total number of cases evaluated. High precision indicates a low rate of false positives.
3. **Recall**: Measures the proportion of actual positives that were correctly identified
4. **F1 score**: The harmonic mean of precision and recall, providing a balance between them and is useful when we seek a model that maintains a balance between precision and recall.
5. **Mean Squared Error (MSE)**: This metric quantifies the average squared difference between predicted scores and actual labels. A lower MSE indicates better model performance.

3.1.2.1 BERT model on Text Document Classification Dataset

3.1.2.2 Text Tokenization

After loading the dataset, we use BERT tokenizer to preprocess the text data. This includes converting the text into token IDs and creating attention masks for padding and truncation.

3.1.2.3 Training the BERT Model

With the data prepared, we initialised the BERT model specifically designed for sequence classification. The BertForSequenceClassification class from the transformers library was used for this purpose. We set the number of output labels according to the classification task.

The training process involved creating a DataLoader for the training set, which helps efficiently manage batch processing. The model was trained using the Adam optimizer, which is well-suited for handling the complexities of neural network training. The training loop iterated through the training data for a specified number of epochs, where the model was updated using backpropagation based on the loss computed from the predicted and true labels.

3.1.2.4 Generate Sentence Embeddings

After training, we aimed to generate sentence embeddings for both the training and validation sets. These embeddings represent the input texts in a way that captures their contextual meaning. The model was set to evaluation mode, and the tokenized texts were passed through the model to obtain the output logits, which serve as the sentence embeddings.

3.1.2.5 Classify Validation Data

Using the embeddings generated in the previous step, we classified the validation data. A nearest neighbour approach was implemented, where cosine similarity was used to measure the similarity between the validation embeddings and training embeddings. The class with the highest similarity score was assigned as the predicted label for each validation sample.

3.1.1.6 Evaluation Metrics

To assess the performance of the model on the validation set, we generated a confusion matrix. This matrix provides insights into the classification performance by showing the counts of true positives, true negatives, false positives, and false negatives for each class.

Finally, we calculated and reported the evaluation metrics for the test set predictions. The performance metrics included accuracy, precision, recall, and F1-score. These metrics provided a clear understanding of how well the model generalised to unseen data, indicating its effectiveness for the intended classification task.

4 Experiments

4.1 Dataset 1: STS-Benchmark Dataset

4.1.1 Sentence2Vec

The key parameters that we will be hypertuning in the CBOW and Skip-gram include:

- **Epoch:** One epoch includes one forward pass and one backpropagation of the training phase.
- **Vector Size:** Dimensionality of the word vectors

Base Model Parameters:

- **Epoch:** 10
- **Vector Size:** 100

Firstly, we fine-tuned the epoch of the model by going up in increments of 10 from 10 to 150.

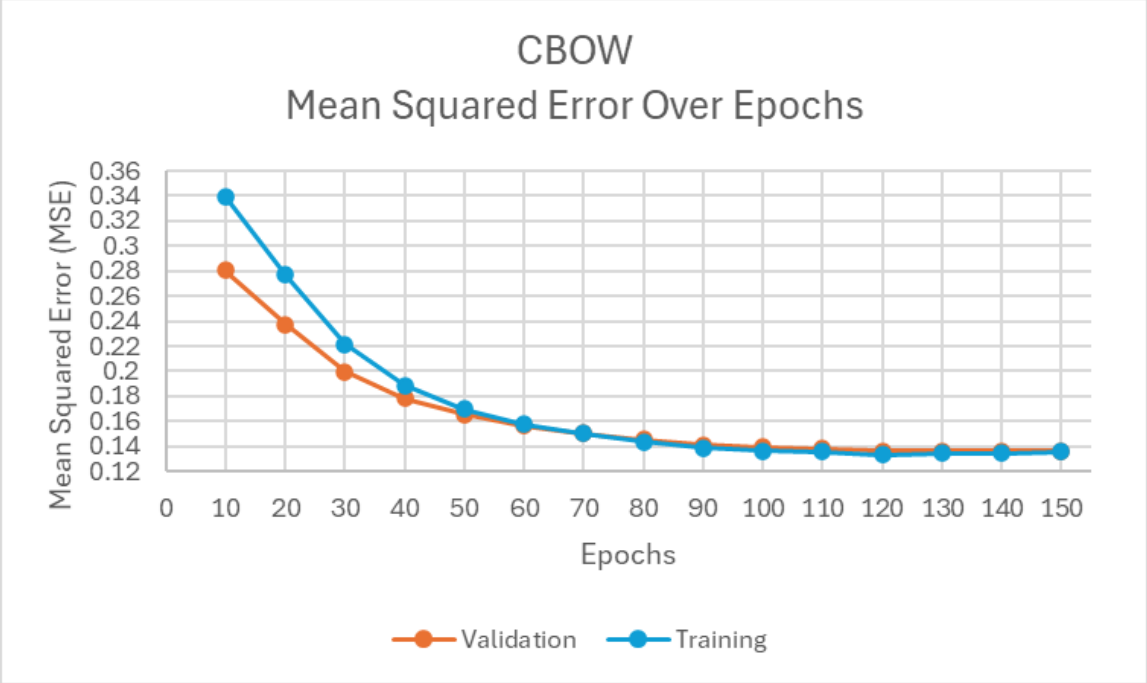


Figure 11. CBOW Mean Squared Error vs Epochs

The CBOW model's MSE decreases as we increase the number of epochs for both the training and validation data as seen from the graph above. Around 110 epoch, the decrease in the MSE is no longer significant as seen by the relatively straight line in the graph after 110 epoch. This indicates that the model has reached its optimal performance at 110 epoch with an MSE of 0.1358 and thus we will use 110 epoch as the fine-tuned epoch parameter for CBOW going forward.

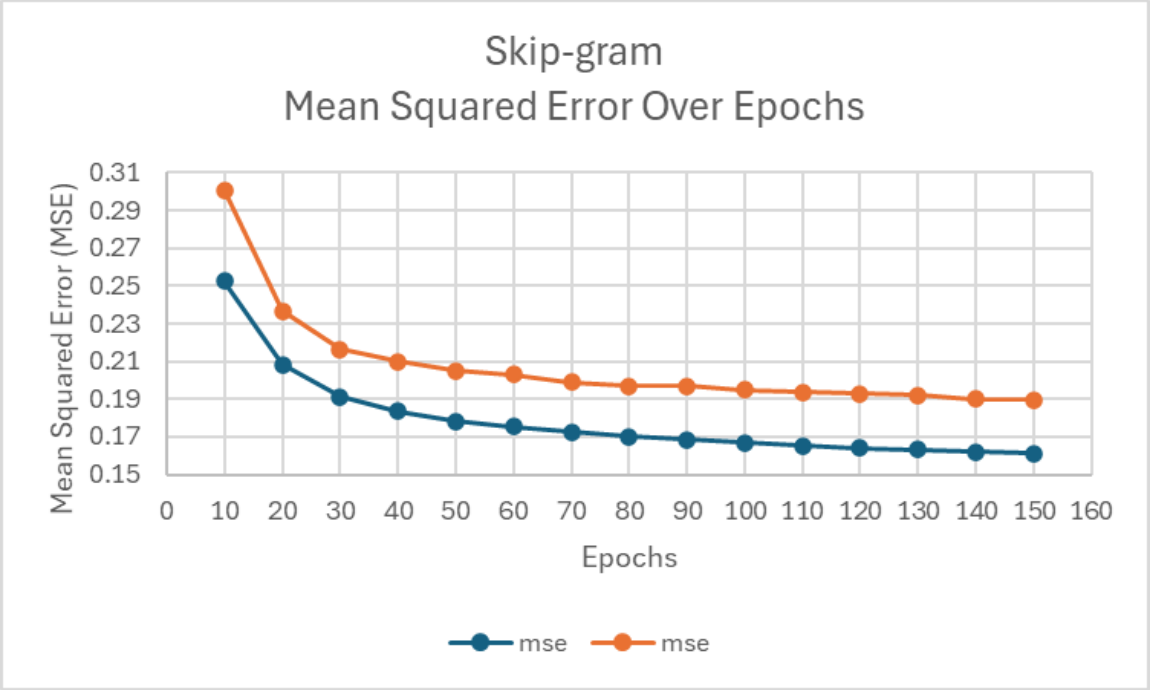


Figure 12. Skip-gram Mean Squared Error vs Epochs

The Skip-gram model's MSE decreases as the epoch values increase, showing a similar pattern to that of the CBOW model. At around 80 epoch, the model has reached a stable MSE score of around 0.19. Thus we will be using 80 epoch as the fine-tuned parameter for Skip-gram.

As the Skip-gram model stagnates at a MSE value of approximately 0.19 as compared to CBOW's 0.13, we will be using the CBOW model for the rest of the fine tuning.

The next parameter to be tuned is the vector size. We trained the model using vector size ranging from 50 to 300 with increments of 25.

Train		Validation	
Vector Size	MSE	Vector Size	MSE
50	0.251	50	0.2994
75	0.2504	75	0.2977
100	0.2531	100	0.3016
125	0.2529	125	0.3012
150	0.2538	150	0.3022
175	0.2542	175	0.3029
200	0.2548	200	0.3036
225	0.2554	225	0.3044
250	0.2553	250	0.3044
275	0.2565	275	0.3059
300	0.2564	300	0.3059

Figure 13. Results of Vector Size Tuning

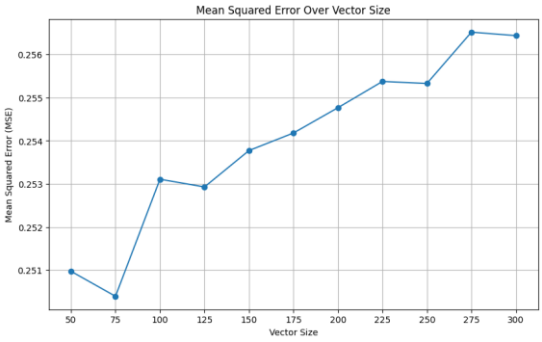


Figure 14. Training Data

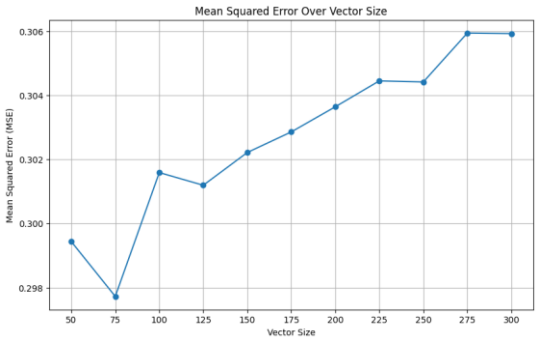


Figure 15. Validation Data

From the figures above, there are no significant changes to the MSE scores whenever the vector size is changed. However, when the vector size is more than 75, we can see that the MSE increases as the vector size increases. Therefore, we decide to use 75 as the vector size moving forward.

Final Model Parameters:

- **Model:** CBOW
- **Epoch:** 80
- **Vector Size:** 75

Evaluation Matrix on Test Dataset

F1 Score	Precision	Recall	MSE
0.7178	0.5598	1.0	0.2847

Table 1. CBOW Evaluation Matrix on Test Dataset

The final model has a recall value of 1.0, which indicates that it is able to detect all true positives in the model. However, it has a precision rate of 0.5598 which means that while the final model is identifying all true positives, it is identifying a lot of false positives as well which is not desirable.

4.1.2 BERT

The key parameters that we will be hypertuning:

- Number of epochs
- Learning rates

By tuning the epochs, we aim to find the point where your model achieves the best performance on unseen data.

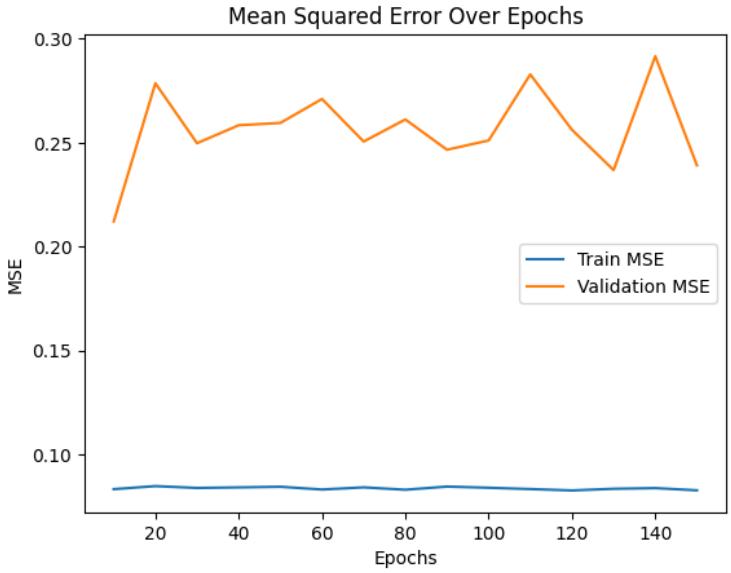


Figure 16. BERT Mean Squared Error Over Epochs

Based on the plot, Epoch **10** seems to be the best choice, as the model's validation performance is at its lowest point there before the MSE begins to rise.

Since we are training the model with a **small number of epochs**, the learning rate becomes crucial for achieving stable and meaningful progress during training.

In the context of a **small number of epochs**, the learning rate controls how quickly the model learns from the data. Fixing an optimal learning rate allows the model to converge within the available time (epochs) without overshooting the minimum or failing to learn meaningful patterns.

For BERT models, the learning rate is typically **much smaller**. To find the range of learning rates, we simply experiment using different learning rates of bigger range — [1e-3, 1e-4, 1e-5]

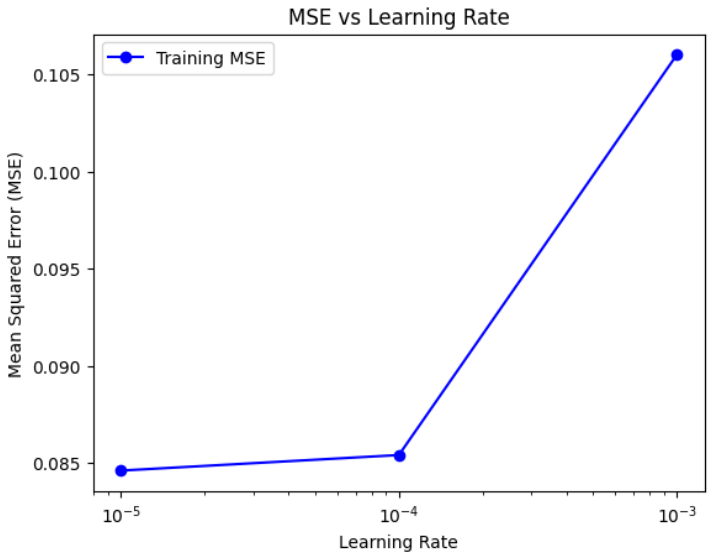


Figure 17. BERT Mean Squared Error Over Learning Rate

After which we further tune within e-5:

- From 3e-5 to 5e-5 (0.00003 to 0.00005) for fine-tuning tasks.

With few epochs, the model might not overfit easily, but **overly aggressive learning rates** could still cause fluctuations in performance. A fixed, well-chosen learning rate helps in preventing drastic updates that might harm training stability. Hence we test using learning rates from 3e-5 to 5e-5.

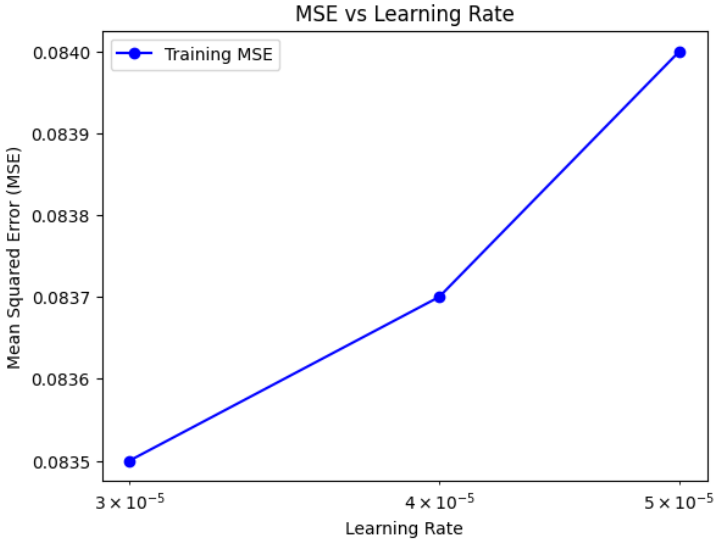


Figure 18. BERT Mean Squared Error Over Learning Rate

We realised that a smaller learning rate leads to a smaller MSE value.

{3e-05: [0.7178056717805672, 0.5598259608411893, 1.0, 0.32190314277012333],
4e-05: [0.7178056717805672, 0.5598259608411893, 1.0, 0.32190314277012333],
5e-05: [0.7178056717805672, 0.5598259608411893, 1.0, 0.32190314277012333]}

Evaluation Matrix on Test Dataset

Accuracy	F1 Score	Precision	Recall	MSE
0.5525	0.7104	0.5570	0.9805	0.3577

Table 2. BERT Evaluation Matrix on Test Dataset

This shows that the model is very good at identifying true positives (high Recall) but struggles with false positives (lower Precision). This leads to a decent F1 Score, but there's a trade-off between Precision and Recall.
The MSE indicates that while the predictions are not perfect, the errors are moderate, which is expected in many models.

4.2 Dataset 2: Text Document Classification Dataset

4.2.1 Sentence2Vec

Base Model Parameters:

- **Epoch:** 5
- **Vector Size:** 100

Confusion Matrix of Base Model

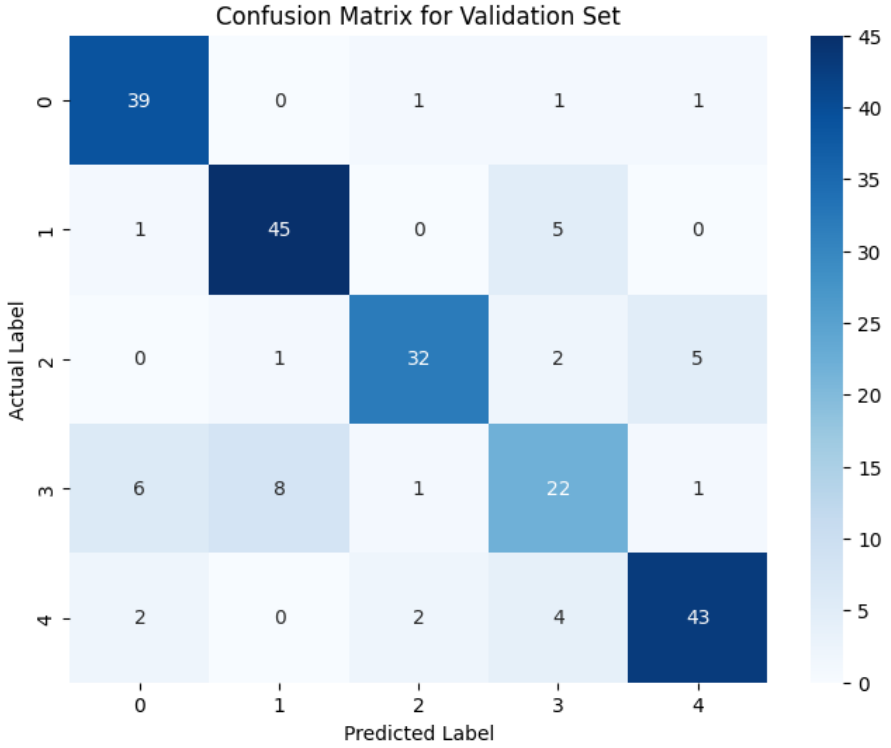


Figure 19. Sentence2Vec Confusion Matrix For Validation Set On Base Model

Evaluation Matrix on Validation Data

	Accuracy	**F1-Score**	**Precision**	**Recall**
CBOW	0.8153	0.8057	0.8084	0.8066
Skip-gram	0.9324	0.9279	0.9276	0.9285

Table 3. CBOW and Skip-gram Evaluation Matrix

The Skip-gram base model has better results in all aspects in comparison to the CBOW base model as such we will be using the skip-gram model for this dataset going forward.

The base model has a precision of 0.9276 and a recall of 0.9285 which indicates that the model is able to identify a true positive relatively accurately. We will be tuning the parameters to increase the accuracy rate to achieve a more confident model.

The first parameter to be tuned is the number of epochs required for the training of the model. We will train the model on a range of epochs, ranging from 10 to 150.

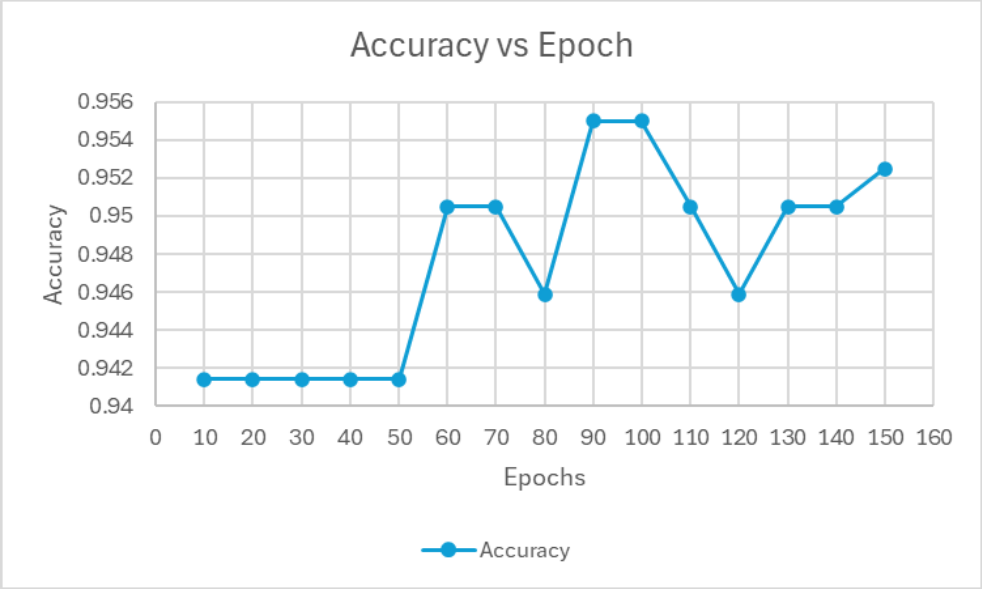


Figure 20. Sentence2Vec Accuracy Over Epochs

The accuracy of the Skip-gram model reaches a peak of 0.9505 at 90 and 100 epochs. After 100 epochs, the accuracy drops and does not reach the peak of 0.9505 again. As such, we decided to go with 90 epochs for the optimal value.

The model is further trained with varying vector sizes ranging from 25 to 300 using increments of 25.

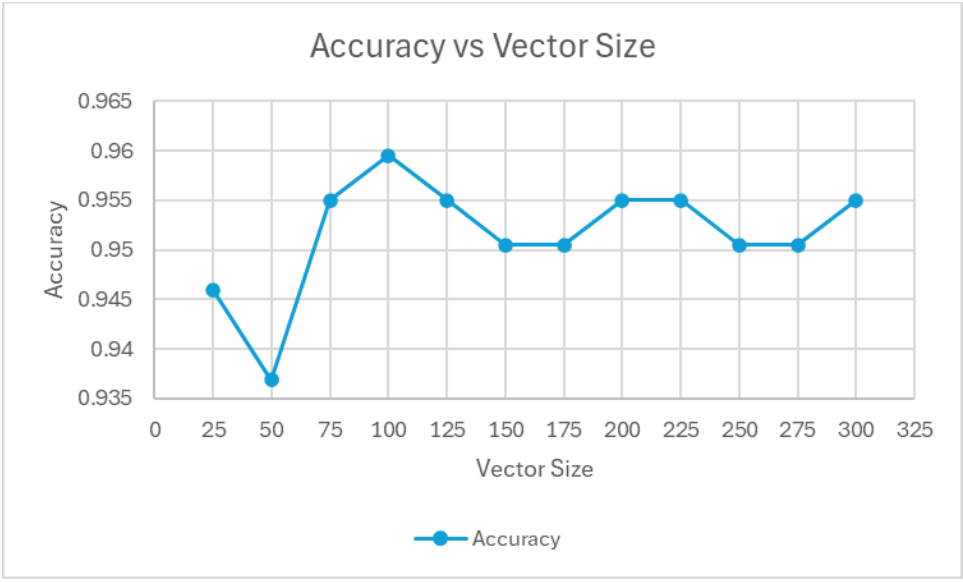


Figure 21. Sentence2Vec Accuracy Over Vector Size

The model's accuracy increases significantly from 0.9369 to 0.9595 when the vector size increases from 50 to 100. As the vector size increases pass 100, the accuracy of the model fluctuates and does not reach the peak of 0.9595 again. Therefore, we can determine that the optimal value for the vector size to be 100.

Final Model Parameters

- **Model:** Skip-gram
- **Epoch**: 90
- **Vector Size:** 100

Running the model on the validation data again after tuning, we obtained the following scores.

Evaluation Matrix on Validation Data

	Accuracy	Precision	Recall	F1-Score
Before Tuning	0.9324	0.9285	0.9279	0.9276
After Tuning	0.9505	0.9527	0.9505	0.9501

Table 4. Skip-gram Before and After Tuning

The tuning of the parameters has increased the confidence level of our model from 0.93 to 0.95 across all 4 evaluation results.

Evaluation Matrix on Test Data

Accuracy	Precision	Recall	F1-Score
0.9865	0.9869	0.9850	0.9857

Table 5. Skip-gram Evaluation Matrix on Test Data

Confusion Matrix for Test Data

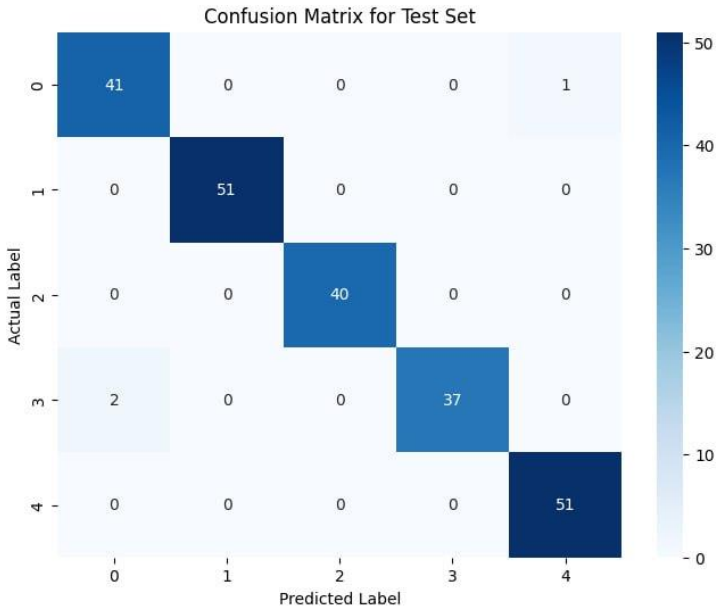


Figure 22. Sentence2Vec Confusion Matrix For Test Set On Tuned Model

The model is able to correctly classify the majority of instances as indicated by its high accuracy value of 0.9865. In addition, it has a low false positive rate as shown by its high precision value of 0.9869. A high recall score of 0.9850 suggests that the model is able to identify a vast majority of true positives which is crucial in sentiment analysis or information retrieval tasks. Overall, the final model is highly effective and well-suited in capturing the semantic relationships in the dataset.

4.2.2 BERT

There are many different parameters which can help with fine-tuning BERT models provided by the Trainer API class. However, we will only look at the key parameters that contribute significantly to the model training process. In general, training a neural network model involves tuning model weights such that it is able to generalise well on unseen data. Therefore, the essential parameters for training include:

- **Epochs**: One epoch includes one forward pass and one backpropagation of the training phase.
- **Learning Rate**: controls how much model weights are updated at each time step.
- **Per_device_train_batch_size**: Training input size that is fed into the model for each training iteration. A larger batch size allows for more stable gradient estimates.

These parameters will form the base parameters of the model training process and will be adjusted based on subsequent training outputs. Other parameters would be included but they merely help with logging of the model training process as well as the evaluation of the test dataset. Such parameters include: *load_best_model_at_end*, *per_device_eval_batch_size* and *logging_steps*.

Model 1: Base Parameters

- Learning Rate= 2e-5
- Epochs = 3
- Batch Size = 8

Output of Model 1

<ipython-input-10-4ee56b40929a>:30: UserWarning: To copy construct from a tens
item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
[945/945 08:25, Epoch 3/3]

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
50	1.587200	1.426982	0.449838	0.451885	0.603566	0.464443
100	1.128100	0.727925	0.896440	0.895451	0.907162	0.890432
150	0.425100	0.322937	0.922330	0.920117	0.923828	0.925094
200	0.253500	0.167234	0.961165	0.959312	0.959057	0.961162
250	0.153400	0.177951	0.948220	0.947351	0.947725	0.947064
300	0.176600	0.212643	0.948220	0.945941	0.945446	0.947602
350	0.151200	0.191245	0.948220	0.945789	0.946740	0.946066
400	0.096500	0.154491	0.964401	0.963394	0.963317	0.963778
450	0.098900	0.166429	0.957929	0.957306	0.958692	0.956517
500	0.067100	0.162539	0.957929	0.957040	0.958580	0.955820
550	0.098200	0.159139	0.961165	0.960137	0.962021	0.960231
600	0.100000	0.152099	0.964401	0.963110	0.964437	0.962599
650	0.046800	0.169901	0.954693	0.952177	0.956602	0.951342
700	0.016200	0.160063	0.964401	0.963148	0.962958	0.963654
750	0.073400	0.154785	0.961165	0.960249	0.959979	0.960837
800	0.014200	0.154960	0.961165	0.960203	0.960106	0.960837
850	0.032500	0.158271	0.961165	0.960203	0.960106	0.960837
900	0.022000	0.153076	0.961165	0.960249	0.959979	0.960837

Figure 23: Output of Model 1

Loss Curve of Model 1

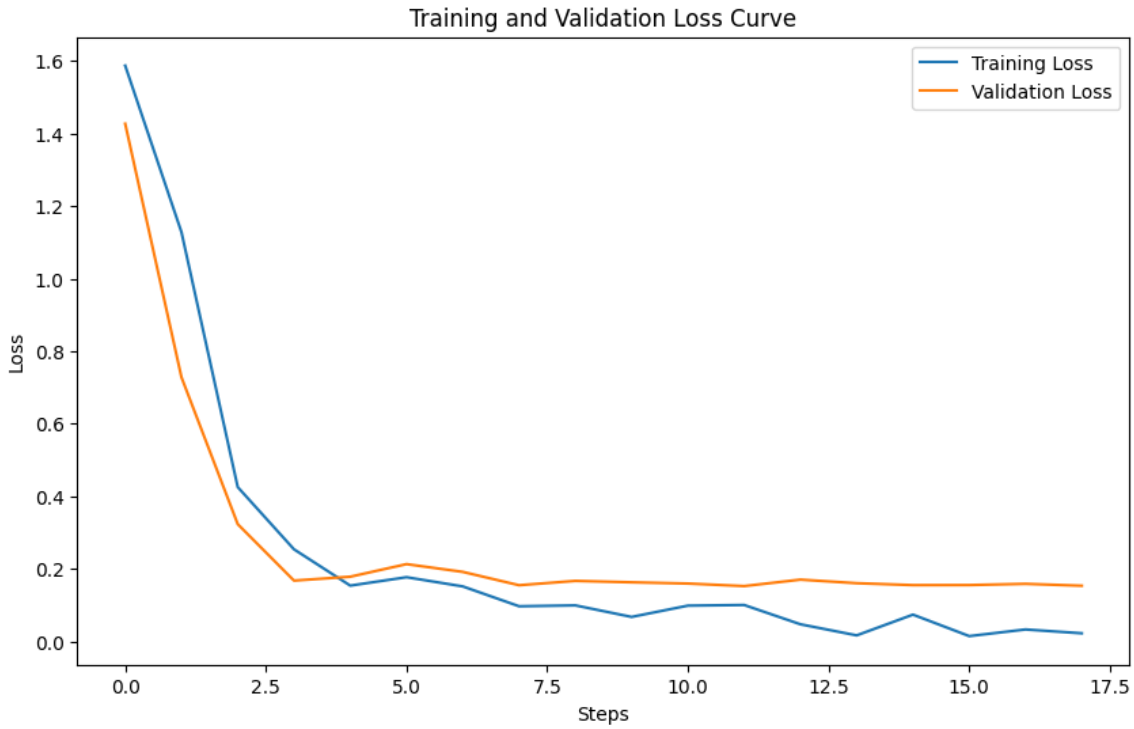


Figure 24: Loss Curve for Model 1

The model seems to perform well with the base parameters introduced and the loss curve affirms this with the training loss continuing to decrease as the training phase progresses. However, the validation loss seems to remain constant even as the training continues to run. This can imply that the model performance on the validation dataset is not improving with the validation loss plateauing over many time steps and showing signs of overfitting on the training dataset. Therefore, we need to introduce some regularisation techniques to the training process and prevent the model from overfitting on the training data. The graph also shows certain spikes in both training and validation losses which

might imply that the batch size is too small or the learning rate is too high. However, with the learning rate set at 2e-5, it might point to the batch size being too small.

Accuracy	F1-Sore	Precision	Recall
0.94498381877022	0.94484401420025	0.94611308878914	0.94403416014582

Table 6: Evaluation Metric on Validation Dataset

Confusion Matrix

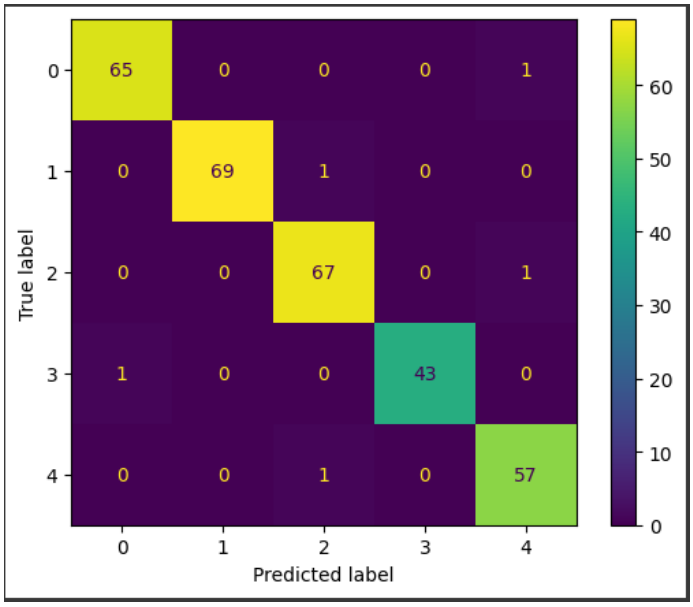


Figure 25. BERT Confusion Matrix For Validation Set

Model 1 performed extremely well on the test dataset achieving a F1-Score of 0.98. A breakdown of the classification result can be seen from the confusion matrix with only 4 data values being misclassified.

Model 2: Adding Weight Decay + Increasing Number of Epochs + Increasing Batch size

To address the problems faced in model 1, weight decay was added to the list of training arguments and also increased the number of epochs to allow the model to converge. Weight decay helps to introduce regularisation into the model and a value of 0.01 implies that the model weights will be penalised by 1% of the current weight value during backpropagation. By adding weight decay, it will take longer for the model to converge since the model weights are now penalised, therefore increasing the number of epochs may help the model converge. The batch size doubled to smoothen the curve and ensure training stability as the model can better calculate gradient estimates with more input data. The updated list of parameters are as follow:

- Learning Rate= 2e-5
- Epochs = **6**
- Batch Size = **16**
- **Weight Decay = 0.01**

Output of Model 2

```
<ipython-input-13-4ee56b40929a>:30: UserWarning: To copy construct from a tens
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
[948/948 13:13, Epoch 6/6]
```

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
50	1.503900	1.157002	0.673139	0.623693	0.797078	0.634802
100	0.577700	0.202379	0.954693	0.952366	0.954293	0.952397
150	0.214700	0.211286	0.948220	0.944529	0.951972	0.942543
200	0.118800	0.205820	0.948220	0.947000	0.949394	0.946066
250	0.126100	0.205312	0.957929	0.956662	0.956652	0.957772
300	0.106300	0.192752	0.957929	0.956263	0.956518	0.957861
350	0.071700	0.186575	0.954693	0.952946	0.953590	0.953327
400	0.065800	0.158276	0.957929	0.956818	0.957630	0.957290
450	0.033800	0.230445	0.951456	0.948801	0.950190	0.949904
500	0.033100	0.217279	0.951456	0.950098	0.950864	0.949400
550	0.023700	0.165849	0.967638	0.965418	0.967892	0.964575
600	0.009600	0.224866	0.954693	0.951383	0.954506	0.951307
650	0.011400	0.232803	0.957929	0.957191	0.959338	0.958758
700	0.012100	0.221961	0.964401	0.963528	0.964990	0.964641
750	0.001100	0.212937	0.967638	0.966602	0.967100	0.967582
800	0.006200	0.225605	0.957929	0.955386	0.957564	0.955752
850	0.000800	0.224260	0.957929	0.955386	0.957564	0.955752
900	0.001100	0.225976	0.957929	0.955386	0.957564	0.955752

Figure 26: Output of Model 2

Loss Curve of Model 2

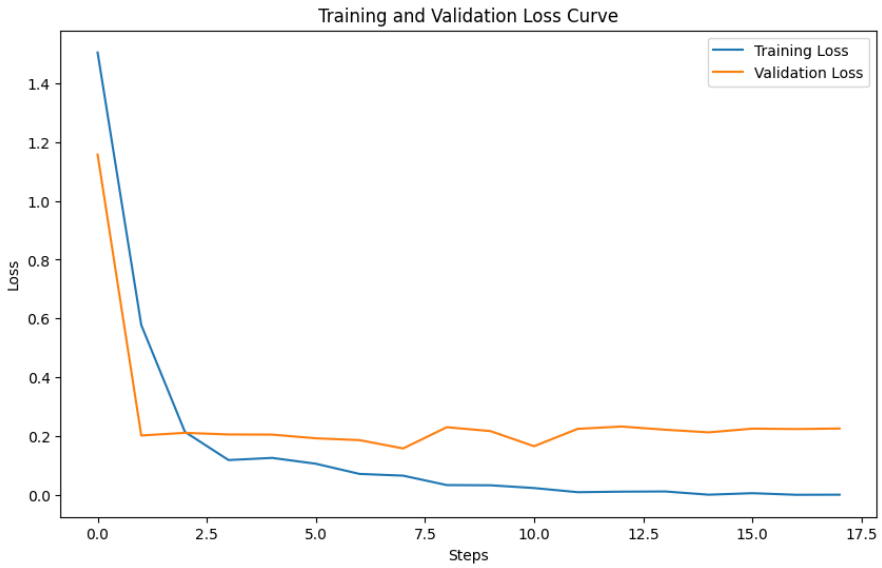


Figure 27: Loss Curve for Model 2

From the loss curve above, the model continues to perform well on the training data with training loss decreasing steadily. The model was also able to generalise well with the validation data at time steps 2.5 to 7.5 with the gap between validation loss and training loss being relatively small. This gap is called the generalisation gap and it indicates how well the model is at generalising on unseen data. However as the training continues, this

gap enlarges which highlights the potential of overfitting on training data and generalising poorly on unseen data. Additionally, the model converges early on in the training process with the loss curve plateauing before spiking during the later time steps. This can imply that our learning rate is too low and the model converges within a low number of epochs. Therefore, we can conclude that the model is able to achieve optimal results with fewer number of epochs even though weight decay was added into the list of parameters.

Evaluation Metric on Validation Dataset

Accuracy	F1-Sore	Precision	Recall
0.96763754045307	0.96559110231170	0.96684887321470	0.96607843137254

Table 7: Evaluation Metrics of Validation Dataset

Confusion Matrix

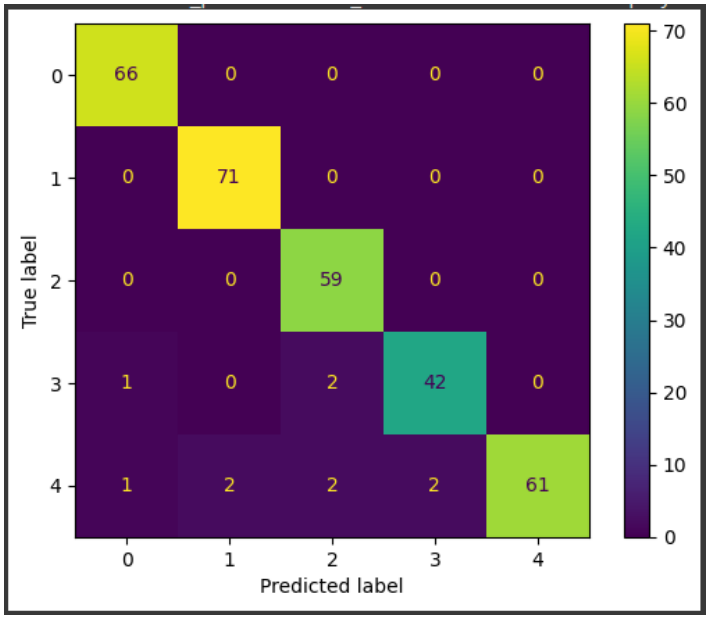


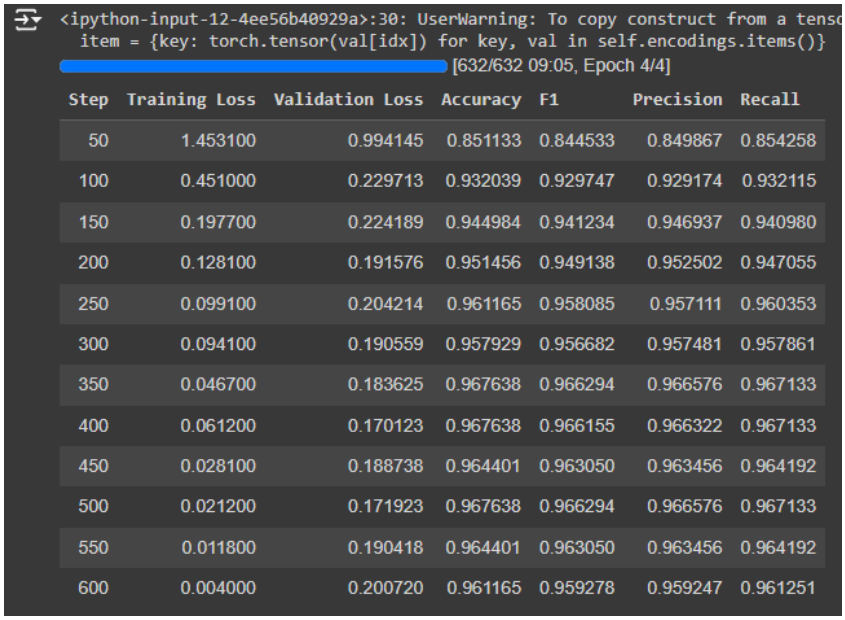
Figure 28. BERT Confusion Matrix For Validation Set

From the results above, the current set of parameters performed better than Model 1 within all aspects of the evaluation metric having increased. By increasing the number of epochs and adding a weight decay value, Model 2 managed to outperform Model 1 which indicates to keep the current list of parameters. Next, we experiment on Model 3 by increasing the learning rate in hopes of producing a different result by forcing the model to converge during the later training phase.

Model 3: Lowering the number of epochs + increasing learning rate

Since the model converges early on in the training phase, we reduce the number of epochs to match this observation. Furthermore, by increasing the learning rate, it can potentially escape the local minima learned by updating the weights in larger steps. The updated list of parameters are listed below:

- Learning Rate= **5e-5**
- Epochs = **4**
- Batch Size = 16
- Weight Decay = 0.01



```
<ipython-input-12-4ee56b40929a>:30: UserWarning: To copy construct from a tenso
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
[632/632 09:05, Epoch 4/4]
```

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
50	1.453100	0.994145	0.851133	0.844533	0.849867	0.854258
100	0.451000	0.229713	0.932039	0.929747	0.929174	0.932115
150	0.197700	0.224189	0.944984	0.941234	0.946937	0.940980
200	0.128100	0.191576	0.951456	0.949138	0.952502	0.947055
250	0.099100	0.204214	0.961165	0.958085	0.957111	0.960353
300	0.094100	0.190559	0.957929	0.956682	0.957481	0.957861
350	0.046700	0.183625	0.967638	0.966294	0.966576	0.967133
400	0.061200	0.170123	0.967638	0.966155	0.966322	0.967133
450	0.028100	0.188738	0.964401	0.963050	0.963456	0.964192
500	0.021200	0.171923	0.967638	0.966294	0.966576	0.967133
550	0.011800	0.190418	0.964401	0.963050	0.963456	0.964192
600	0.004000	0.200720	0.961165	0.959278	0.959247	0.961251

Figure 29: Output of Model 3

Loss Curve of Model 3

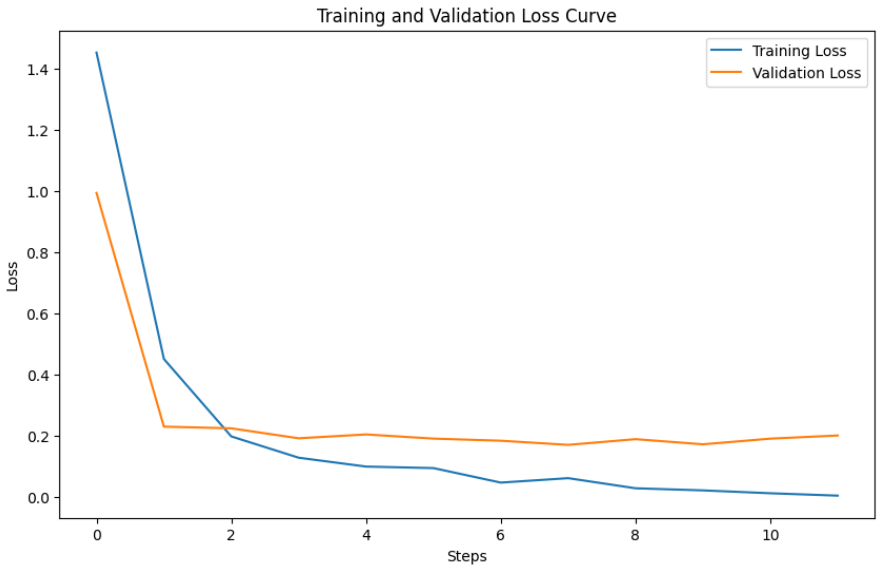


Figure 30: Loss Curve of Model 3

From the loss curve, Model 3 seems to produce similar results as Model 2 with the key difference being in the validation set. Even with an increased learning rate, the model seems to have generalised well on the validation set with the validation loss plateauing.

The training loss also reaches close to 0 which indicates the model is training efficiently. Based on this observation, the model seems to have generalised well on unseen data with the gap between validation and training being small and there are no spikes in the training phase to indicate training instability.

Evaluation Metric on Validation Dataset

Accuracy	F1-Sore	Precision	Recall
0.97411003236245	0.97322905035411	0.97394239220614	0.97346405228758

Table 8: Evaluation Metric for Validation Dataset

Evaluation Metric On Test Dataset Using Model 3

Accuracy	F1-Sore	Precision	Recall
0.99346405228758	0.99305833265249	0.99216572504708	0.99411764705882

Table 9: Evaluation Metric on Test Dataset

Confusion Matrix for Test Data

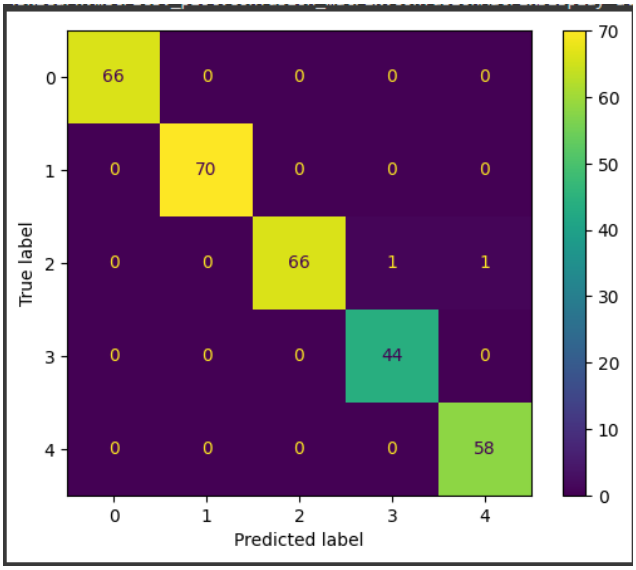


Figure 31. BERT Confusion Matrix For Test Set

Based on the evaluation metrics on the test dataset, the model achieved a F1-Score of 0.99. The model generalised extremely well on the test dataset with only 2 data values being misclassified. Therefore, we can conclude that this set of parameters produces the best result as seen from the performance of the model on the test dataset.

4.3 Comparison of Sentence2Vec and BERT

4.3.1 Dataset 1: STS-Benchmark Dataset

	Accuracy	F1 Score	Precision	Recall	MSE
Sentence2Vec	0.5663	0.7178	0.5598	1.0	0.2847
BERT	0.5525	0.7104	0.5570	0.9805	0.3577

Table 10. Comparison Table for Sentence2Vec and BERT on Dataset 1

Based on the scores above, we have observed Sentence2Vec and BERT performed similarly on Dataset 1, with Sentence2Vec performing slightly better in terms of accuracy and F1-Score than BERT.

Although we had expected BERT to perform better than Sentence2Vec, we believe that the simple sentence structure and lack of complex semantics could account for the similar performance in Dataset 1.

In the first dataset, the sentences used were short and simple. Below are some examples of the sentences provided in the dataset:

```
1: ['A girl is styling her hair.', 'A girl is brushing her
hair.']
2: ['A group of men play soccer on the beach.', 'A group of boys
are playing soccer on the beach.']
3: ["One woman is measuring another woman's ankle.", "A woman
measures another woman's ankle."],
4: ['A man is cutting up a cucumber.', 'A man is slicing a
cucumber.']
5: ['A man is playing a harp.', 'A man is playing a keyboard.']
```

BERT is particularly strong for tasks where context and nuance play a key role. However, given the short sentences given in dataset 1, this could be a possible reason as to why Sentence2Vect performed similarly to BERT.

Data imbalance could be another reason for why the performance of BERT and Sentence2Vec appears similar across metrics like accuracy, F1 score, precision, and recall. When the dataset is heavily skewed towards a dominant class, even models with different architectures (like BERT and Sentence2Vec) may tend to predict the majority class more often, leading to similar evaluation metrics.

4.3.2 Dataset 2: Text Document Classification Dataset

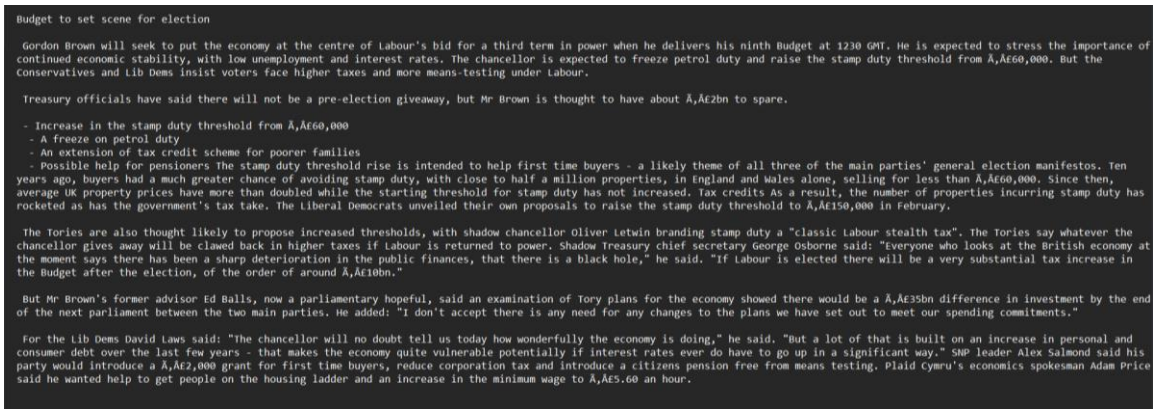
The final evaluation of both models on the test dataset can be seen below:

	Accuracy	Precision	Recall	F1-Score
Sentence2Vec	0.9865	0.9869	0.9850	0.9857
BERT	0.9934	0.9930	0.9921	0.9941

Table 11. Comparison Table for Sentence2Vec and BERT on Dataset 2

Comparing the results of both models, BERT managed to slightly outperform Sentence2Vec with BERT achieving a F1-score of 0.99 compared to Sentence2Vec of 0.95. This differs significantly from Dataset 1. A few reasons can be attributed to BERT performing better than Sentence2Vec in the second dataset.

Firstly, BERT is able to better comprehend nuanced and complex sentences better than Sentence2Vec with the transformer architecture of BERT capturing nuanced relationships between words in a given input sequence. The second dataset contains 2225 text sequences that have multiple long sentences in each sequence as observed in the example below.


Budget to set scene for election

Gordon Brown will seek to put the economy at the centre of Labour's bid for a third term in power when he delivers his ninth Budget at 1230 GMT. He is expected to stress the importance of continued economic stability, with low unemployment and interest rates. The chancellor is expected to freeze petrol duty and raise the stamp duty threshold from Ã‚Â£60,000. But the Conservatives and Lib Dems insist voters face higher taxes and more means-testing under Labour.

Treasury officials have said there will not be a pre-election giveaway, but Mr Brown is thought to have about Ã‚Â£2bn to spare.

- Increase in the stamp duty threshold from Ã‚Â£60,000
- A freeze on petrol duty
- An extension of tax credit scheme for poorer families
- Possible help for pensioners The stamp duty threshold rise is intended to help first time buyers - a likely theme of all three of the main parties' general election manifestos. Ten years ago, buyers had a much greater chance of avoiding stamp duty, with close to half a million properties, in England and Wales alone, selling for less than Ã‚Â£60,000. Since then, average UK property prices have more than doubled while the starting threshold for stamp duty has not increased. Tax credits As a result, the number of properties incurring stamp duty has rocketed as has the government's tax take. The Liberal Democrats unveiled their own proposals to raise the stamp duty threshold to Ã‚Â£150,000 in February.

The Tories are also thought likely to propose increased thresholds, with shadow chancellor Oliver Letwin branding stamp duty a "classic Labour stealth tax". The Tories say whatever the chancellor gives away will be clawed back in higher taxes if Labour is returned to power. Shadow Treasury chief secretary George Osborne said: "Everyone who looks at the British economy at the moment says there has been a sharp deterioration in the public finances, that there is a black hole," he said. "If Labour is elected there will be a very substantial tax increase in the Budget after the election, of the order of around Ã‚Â£10bn."

But Mr Brown's former advisor Ed Balls, now a parliamentary hopeful, said an examination of Tory plans for the economy showed there would be a Ã‚Â£35bn difference in investment by the end of the next parliament between the two main parties. He added: "I don't accept there is any need for any changes to the plans we have set out to meet our spending commitments."

For the Lib Dems David Laws said: "The chancellor will no doubt tell us today how wonderfully the economy is doing," he said. "But a lot of that is built on an increase in personal and consumer debt over the last few years - that makes the economy quite vulnerable potentially if interest rates ever do have to go up in a significant way." SNP leader Alex Salmond said his party would introduce a Ã‚Â£2,000 grant for first time buyers, reduce corporation tax and introduce a citizens pension free from means testing. Plaid Cymru's economics spokesman Adam Price said he wanted help to get people on the housing ladder and an increase in the minimum wage to Ã‚Â£5.60 an hour.

538 words present in this text sequence

Figure 32. Example Row in Dataset 2

The longer and complex sentences are likely the reason behind BERT's better performance. Sentence2Vec does not perform as well since it averages word vectors, which can dilute the meaning of longer or more complex sentences. In addition, it struggles to capture intricate patterns or the importance of specific word order in text.

Secondly, the bidirectional context in BERT could also result in better performance. Unlike BERT, Sentence2Vec does not consider the entire context of a sentence when generating embeddings which could affect the performance of the Sentence2Vec model.

5 Conclusion

In this report, we experimented with Sentence2Vec and BERT on 2 text datasets to perform sentence similarity search. In the first dataset, we found that both models demonstrated comparable results in terms of accuracy, precision, recall, and F1-score. This could be accounted for by data imbalance or simple sentences in the first dataset that resulted in similar performance from the two methods.

In contrast, BERT performed better than Sentence2Vec on the second dataset. This could be attributed to BERT's ability to leverage bidirectional context, dynamic word embeddings, and its attention mechanism, which allows it to capture nuanced relationships within the text. While Sentence2Vec can capture some contextual relationships between words, it relies on simplistic aggregation of individual word vectors within a sentence, limiting its ability to fully understand complex contextual nuances.

Overall, Sentence2Vec provides a foundational approach to sentence representation by utilising a shallow neural network while BERT utilises a complex transformer model with multiple layers and attention mechanisms. For simpler data consisting of shorter texts, it would be more efficient to run Sentence2Vec, as it runs faster than BERT. However, BERT would provide improved performance in tasks requiring a deep semantic understanding.

In conclusion, the choice between Sentence2Vec and BERT should be driven by the specific requirements of the task at hand. Sentence2Vec is a viable option for tasks that demand speed and simplicity. On the other hand, BERT is a more suitable solution for tasks involving more complex textual data or where higher performance is critical.

6 References

1. GeeksforGeeks. (2024, January 10). *Explanation of Bert Model - NLP*. https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/ (GeeksforGeeks, 2024)
2. Kingma, D. P., & Ba, J. (2017, January 30). *Adam: A method for stochastic optimization*. arXiv.org. https://doi.org/10.48550/arXiv.1412.6980 (Kingma & Ba, 2017)
3. Stanleyfok. (2017). *Stanleyfok/sentence2vec: Sentence2Vec based on word2vec, written in Python*. GitHub. https://github.com/stanleyfok/sentence2vec
4. Suri, M. (2022, January 21). *A dummy's guide to word2vec*. Medium. https://medium.com/@manansuri/a-dummys-guide-to-word2vec-456444f3c673 (Suri, 2022)
5. *What is sent2vec*. ActiveLoop. (n.d.). https://www.activeloop.ai/resources/glossary/sent-2-vec/
6. Yadav, S. (2023, July 13). *What is Bert? how it is trained ? A high level overview*. Medium. https://medium.com/@Suraj_Yadav/what-is-bert-how-it-is-trained-a-high-level-overview-1207a910aaed (Yadav, 2023)