

Technical Report on Semantic Similarity in Natural Language Processing

Soham Petkar
soham.petkar@plaksha.edu.in
Plaksha University

December 27, 2024


Abstract

This report provides an in-depth analysis of semantic similarity tasks within Natural Language Processing (NLP). It details the methodologies employed to convert textual data into numerical representations, explores various approaches for measuring semantic similarity at the word, phrase, and sentence levels, and presents findings from multiple experiments conducted. Comprehensive comparisons are made between different models and techniques, including static word embeddings, fine-tuned transformers, and large language models (LLMs).

I have tried to do all the given tasks under the NLP theme and give a reasoning for the results obtained. I would like to thank the team at Precog for coming up with the task, it has surely deepened my understanding about the subject and has let me learn new things.

I also hope I don't get a non-brevity penalty unlike BERTScore! by [Zhang et al., 2020a](#) given the leng

Contents

Contents	1
0.1 Overview of the Task Theme Chosen - NLP	3
0.2 Detailed Tasks	3
0.2.1 Representations for Words, Phrases, Sentences	3
1 Constrained Word to Vector Evaluation	5
1.1 Introduction	5
1.2 Methodology - Cosine Similarity Approach	6
1.2.1 Data Preprocessing and Vocabulary Construction	6
1.2.2 Subsampling of Frequent Words	7
1.2.3 Training Pair Generation	7
1.2.4 Negative Sampling with Unigram Distribution	7
1.2.5 Model Initialization	8
1.2.6 Training Procedure	8
1.2.7 Observed Loss Progression	8
1.3 Evaluation	9
1.4 Methodology - Euclidean Distance approach	9
1.4.1 Normalization of Embeddings	10
1.4.2 Training Procedure	10
1.4.3 Rationale	11
1.4.4 Evaluation	12
1.5 Replicating  's work - Word2Vec	12
2 Unconstrained Word to Vector Evaluation	14
3 Phrase and Sentence Embeddings	18
3.1 Pretrained Embeddings with Aggregation - Phrase	18
3.1.1 Pooling Strategy	18
3.1.2 Averaging Strategy	19
3.1.3 Concatention Strategy	19
3.1.4 Model Formulation	20
3.1.5 Aggregation Strategies	20
3.1.6 Classification and Loss Calculation	21
3.1.7 Backpropagation and Learning	21
3.1.8 Phrase - Weighted learning	22
3.2 Sentence Task	23
3.2.1 TwoBranchModel	23
3.2.2 TwoBranchModel + Cross Attention	24

3.3	Analysis	25
4	BONUS Tasks	27
4.1	Binary classification using LLama 8b	27
4.2	Finetuning for Sentence classification	28
4.3	Finetuning for Phrase classification	29
5	Paper Summary - BERTScore	30
5.1	Three Major Strengths	30
5.2	Three Major Weaknesses	31
5.3	Three Suggested Improvements	31
	Bibliography	32

0.1 Overview of the Task Theme Chosen - NLP

The central theme of this report revolves around semantic similarity in NLP. The tasks addressed include word similarity scores, phrase and sentence similarity, and advanced methodologies involving transformers and large language models (LLMs).

0.2 Detailed Tasks

0.2.1 Representations for Words, Phrases, Sentences

NLP encompasses various tasks such as regression, classification, and generation. A common denominator across these tasks is the question of "how do we convert text into numbers/representations" so that machines can process them. One way to measure a machine's ability to "understand" text is semantic similarity, i.e., determining how similar or dissimilar a given pair of text inputs are. This is the central theme of the tasks addressed in this report.

a. Word Similarity Scores

Given a pair of words, the objective is to predict their similarity score. The focus is on converting a word to its numerical representation and applying learning algorithms such as regression or classification. The dataset provided will be used entirely as a test set, with no supervised training data available. Solutions must be developed under two conditions:

1. **Constraints on Data Resources:** Utilize only the following resources (any one or all):
 - Any monolingual English corpus with a maximum of 1 million tokens.
 - Any curated/structured knowledge-bases or ontologies.
2. **Unconstrained:** Remove the above constraints and allow the use of any data or model.

A comparison of results across these two settings will be conducted to analyze what works, what doesn't, and why.

b. Phrase and Sentence Similarity

Building upon the word representation methods, the next step is to create representations for phrases and sentences by aggregating individual word embeddings. This can be achieved using pretrained static word embeddings like Word2Vec, GloVe, FastText, or by creating custom embeddings. Linguistic features can be computed using tools/libraries such as NLTK, Stanza, or SpaCy.

1. **Phrase Similarity:** Classify whether a given pair of phrases are similar using a provided dataset with train/dev/test splits.
2. **Sentence Similarity:** Similarly, classify sentence pairs for similarity using the corresponding dataset.

Multiple approaches will be experimented with to derive phrase and sentence representations, followed by comparative analysis to identify failure cases and underlying patterns.

c. Bonus Task

1. **Fine-Tuning Transformers:** Fine-tune pre-trained transformer-based models (e.g., BERT, RoBERTa) to solve phrase and sentence similarity tasks.
2. **Prompting LLMs:** Utilize large language models (LLMs) such as ChatGPT and LLAMA to generate similarity scores through commercial and open-source APIs, exploring zero and few-shot settings.
3. **Comparative Analysis:** Compare the performance of static word embeddings, fine-tuned transformers, and LLMs to observe improvements and differences across these methodologies.

d. Paper Reading Task

BERTScore: Evaluating Text Generation with BERT ([Zhang et al., 2020b](#)) is reviewed to understand its application in evaluating text generation tasks.

Chapter 1

Constrained Word to Vector Evaluation

1.1 Introduction

In addressing the task of converting words into vector representations and evaluating them using the SimLex evaluation set, several potential methodologies were considered. To ensure a systematic exploration, a structured approach was adopted, progressing from the most intuitive concepts to more unconventional strategies. The initial strategies focused on utilizing a corpus of one million words with the objective of maximizing cosine similarity between contextually related word pairs while minimizing cosine similarity between unrelated pairs.

This approach can also be interpreted in terms of Euclidean distances. Specifically, for normalized vectors, maximizing cosine similarity is equivalent to minimizing the Euclidean distance between the vectors. Mathematically, this relationship is defined as follows.

The cosine similarity between two vectors \mathbf{u} and \mathbf{v} is given by:

$$\text{cosine_similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \quad (1.1)$$

The Euclidean distance between the same two vectors is defined as:

$$\text{Euclidean_distance}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\| \quad (1.2)$$

When both vectors \mathbf{u} and \mathbf{v} are normalized, i.e., $\|\mathbf{u}\| = \|\mathbf{v}\| = 1$, the equations simplify. The cosine similarity becomes:

$$\text{cosine_similarity}(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} \quad (1.3)$$

Expanding the Euclidean distance for normalized vectors:

$$\text{Euclidean_distance}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\| \quad (1.4)$$

$$= \sqrt{(\mathbf{u} - \mathbf{v}) \cdot (\mathbf{u} - \mathbf{v})} \quad (1.5)$$

$$= \sqrt{\mathbf{u} \cdot \mathbf{u} - 2\mathbf{u} \cdot \mathbf{v} + \mathbf{v} \cdot \mathbf{v}} \quad (1.6)$$

$$= \sqrt{2 - 2\mathbf{u} \cdot \mathbf{v}} \quad (1.7)$$

$$= \sqrt{2(1 - \mathbf{u} \cdot \mathbf{v})} \quad (1.8)$$

$$= \sqrt{2(1 - \text{cosine_similarity}(\mathbf{u}, \mathbf{v}))} \quad (1.9)$$

From the final expression, it is evident that minimizing the Euclidean distance $\|\mathbf{u} - \mathbf{v}\|$ is directly related to maximizing the cosine similarity $\mathbf{u} \cdot \mathbf{v}$ when the vectors are normalized. Specifically, as $\text{cosine_similarity}(\mathbf{u}, \mathbf{v})$ approaches 1, indicating higher similarity, the Euclidean distance approaches 0, indicating closer proximity in the vector space.

This equivalence provides a dual perspective on the optimization objective, allowing flexibility in the choice of distance metrics based on the specific requirements of the task. The first two methods, grounded in these principles of cosine similarity and Euclidean distance optimization, are elaborated upon in the following sections. These methods form the foundation of the constrained approach to word vector generation, setting the stage for more advanced techniques discussed later in this chapter.

1.2 Methodology - Cosine Similarity Approach

The most basic idea for this section is to create a dataframe with center word, its context word and a created label, which should be 1 for the words in fixed context window and 0 for non related word pairs and then updating the random initialized embeddings using the following

```
dot_prods = np.sum(main_embeddings_center * context_embeddings_context)
scores = sigmoid(dot_prods)
errors = (df.label - scores)
updates = diffs*errors*learning_rate
```

But this was quickly ruled out as this update is very inefficient, i.e., memory and training time-consuming. So, a more efficient methodology was obtained by researching and querying large language models (yes, I used ChatGpt and Claude) and employed.

Therefore the below outlines the methodology employed for generating word vectors under constrained resources, utilizing cosine similarity as the primary metric in a somewhat efficient way. The approach is grounded in the Skip-Gram with Negative Sampling (SGNS) framework introduced by Mikolov et al. (Mikolov et al., 2013), adapted to operate within the specified constraints.

1.2.1 Data Preprocessing and Vocabulary Construction

The initial phase involves the acquisition and preprocessing of textual data. The WikiText-2 dataset is selected for its comprehensive coverage and suitability for training language models. The dataset is tokenized into individual words using a regular expression that

captures lowercase alphabetical sequences, effectively filtering out punctuation, numerical tokens, and other non-alphabetic characters.

To manage computational resources efficiently, a vocabulary of the top one million most frequent words is constructed. Words not present in this vocabulary are mapped to a special token `<UNK>`, representing unknown or rare words.

1.2.2 Subsampling of Frequent Words

High-frequency words, such as stopwords, can dominate the training process, leading to less meaningful vector representations for more informative terms. To address this imbalance, subsampling is applied to reduce the prevalence of these common words. The discard probability for each word is calculated based on its frequency, following the formulation proposed by Mikolov et al. (Mikolov et al., 2013).

Mathematically, the discard probability $P(w)$ for a word w is defined as:

$$P(w) = 1 - \sqrt{\frac{t}{f(w)}} \quad (1.10)$$

where t is a predefined threshold (e.g., 1×10^{-5}), and $f(w)$ is the frequency of word w . Words with higher frequencies have higher discard probabilities, effectively reducing their impact during training. This subsampling technique ensures that the model focuses more on less frequent, semantically rich words, enhancing the quality of the learned embeddings.

1.2.3 Training Pair Generation

The generation of training pairs is executed using a sliding window approach. For each target word in the tokenized corpus, a context window of variable size (up to five words) is defined. To control computational complexity, a maximum of three context words are sampled randomly from this window for each target word. This results in (`target`, `context`) pairs that capture the contextual relationships essential for learning meaningful word vectors.

This method ensures that each word is associated with a diverse set of context words, promoting robust and generalizable vector representations. By limiting the number of context words, the approach balances the need for sufficient contextual information with the practical constraints of processing large datasets.

1.2.4 Negative Sampling with Unigram Distribution

Negative sampling is a critical component for efficiently training the SGNS model by distinguishing true context words from noise. A unigram distribution raised to the $\frac{3}{4}$ power is employed for selecting negative samples, as advocated by Mikolov et al. (Mikolov et al., 2013). This distribution balances the frequency of word selection, providing a better approximation for the noise distribution compared to a uniform distribution.

The unigram distribution $P(w)$ is defined as:

$$P(w) = \frac{f(w)^{\frac{3}{4}}}{\sum_{w'} f(w')^{\frac{3}{4}}} \quad (1.11)$$

where $f(w)$ is the frequency of word w . This adjustment reduces the probability of very frequent words being selected as negative samples, while still allowing moderately frequent words to be sampled effectively. Negative samples are drawn based on this distribution during the training process, enabling the model to learn to differentiate between relevant and irrelevant word pairs efficiently.

1.2.5 Model Initialization

Word vectors are initialized with small random values to facilitate efficient training. Separate embedding matrices are maintained for target and context words, each with a dimensionality of 100. Initializing the embeddings with small random values ensures that the vectors start in a state conducive to gradient-based optimization, preventing symmetry and promoting diverse updates during training.

1.2.6 Training Procedure

The training process iterates through the sampled (**target**, **context**) pairs for a pre-defined number of epochs (e.g., five for my case). For each pair, the cosine similarity between the target and context vectors is computed.

The objective is twofold: to maximize the cosine similarity for true context pairs and to minimize it for negative samples. This dual objective is formalized through the following loss function:

$$\mathcal{L} = -\cos(\mathbf{u}, \mathbf{v}) + \sum_{i=1}^k \cos(\mathbf{u}, \mathbf{v}_i) \quad (1.12)$$

where \mathbf{u} and \mathbf{v} are the target and context vectors, respectively, and \mathbf{v}_i are the negative samples.

The gradients of the loss with respect to the target and context vectors are computed and used to update the embeddings. Specifically, the embeddings are adjusted in the direction that minimizes the loss, thereby increasing the cosine similarity for positive pairs and decreasing it for negative pairs. The learning rate (e.g., 0.025, which was used by the authors of Word2Vec) governs the magnitude of these updates.

1.2.7 Observed Loss Progression

The training log presents the average loss after each epoch as follows:

- **Epoch 1:** -0.0012
- **Epoch 2:** -0.2656
- **Epoch 3:** -0.3118
- **Epoch 4:** -0.3457
- **Epoch 5:** -0.3710

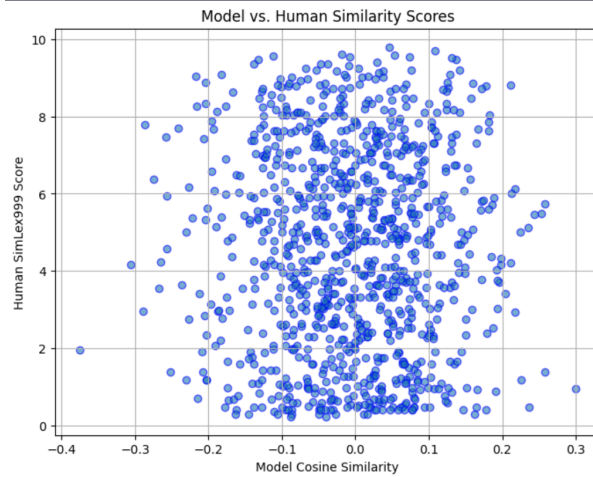


Figure 1.1: SimLex eval of Cosine Sim Model

- Convergence Indicators: The diminishing rate of loss improvement across epochs suggests that the model is approaching convergence. The initial average loss of -0.0012 indicates minimal alignment between target-context pairs and negligible penalties from negative samples. This is consistent with the random initialization where no meaningful semantic relationships have been learned yet. As training progresses, the model adjusts the embeddings to maximize cosine similarity for positive pairs and minimize it for negative samples. This optimization leads to the alignment of related word vectors and the dispersion of unrelated ones, thus leading to a more and more negative loss.

1.3 Evaluation

The performance of the constrained Word2Vec model was assessed using the SimLex-999 evaluation set, resulting in a Spearman's rank correlation coefficient of -0.0074 with a corresponding p-value of 0.8166. These metrics indicate an insignificant and near-zero correlation with human-assigned similarity scores, suggesting that the generated word vectors do not effectively capture the intended semantic relationships. [Which honestly for me was very expected result!]

We observe an average loss of -0.37 in the fifth epoch, and in figure 2.1, we also see that the maximum similarity our model is assigning to two vectors lies nearly around 0.3, which is on point given the structure of our loss function and with the observed loss. Most of the values assigned by our model are concentrated around -0.1 to 0.1, which shows that even though the model has started to learn, it hasn't fully converged. Intuitively training the model till a loss of -0.8 to -0.9 would be considered as convergence as for very similar pairs it should be around these values. [This is absolutely not necessary looking at the noise in the data, this explanation is provided from my side which resides with my intuition]

1.4 Methodology - Euclidean Distance approach

This section elucidates the methodology implemented to achieve this objective, highlighting the key modifications to the training process and the underlying theoretical rationale.

The Loss function in this case would be -

$$\mathcal{L} = \text{Euclidean_distance}(\mathbf{u}, \mathbf{v}) - \sum_{i=1}^k \text{Euclidean_distance}(\mathbf{u}, \mathbf{v}_i) \quad (1.13)$$

where:

- \mathbf{u} and \mathbf{v} are the target and context word vectors, respectively.
- \mathbf{v}_i represents the i^{th} negative sample.
- k is the number of negative samples per positive pair.

The loss function aims to minimize the Euclidean distance between positive pairs (\mathbf{u}, \mathbf{v}) and maximize the distance between negative samples $(\mathbf{u}, \mathbf{v}_i)$.

1.4.1 Normalization of Embeddings

To establish a consistent scale for the word vectors and to facilitate the relationship between Euclidean distance and cosine similarity, embeddings are normalized to unit length. Normalization is performed after each update to ensure that all vectors lie on the unit hypersphere:

$$\mathbf{u} \leftarrow \frac{\mathbf{u}}{\|\mathbf{u}\| + \epsilon}, \quad \mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\| + \epsilon} \quad (1.14)$$

where ϵ is a small constant added for numerical stability. Normalization ensures that the Euclidean distance between two vectors is directly related to their cosine similarity, as previously established.

1.4.2 Training Procedure

The training process unfolds over multiple epochs, iterating through all sampled (**target**, **context**) pairs. For each pair, the following steps are executed:

Positive Pair Optimization

1. Retrieval and Normalization: - Retrieve the target and context embeddings, \mathbf{u} and \mathbf{v} , respectively. - Normalize both embeddings to unit vectors.
2. Euclidean Distance Calculation: - Compute the Euclidean distance between \mathbf{u} and \mathbf{v} :

$$\text{distance_pos} = \|\mathbf{u} - \mathbf{v}\| + \epsilon \quad (1.15)$$

3. Loss Computation: - Define the loss for the positive pair as the Euclidean distance, aiming to minimize this value:

$$\mathcal{L}_{\text{pos}} = \text{distance_pos} \quad (1.16)$$

4. Gradient Calculation: - Compute the gradients with respect to \mathbf{u} and \mathbf{v} :

$$\nabla_{\mathbf{u}} \mathcal{L}_{\text{pos}} = \frac{2(\mathbf{u} - \mathbf{v})}{\text{distance_pos}} \quad (1.17)$$

$$\nabla_{\mathbf{v}} \mathcal{L}_{\text{pos}} = \frac{-2(\mathbf{u} - \mathbf{v})}{\text{distance_pos}} \quad (1.18)$$

Negative Sampling Optimization

1. Sampling Negative Contexts: - Draw k negative samples \mathbf{v}_i from the unigram distribution raised to the $\frac{3}{4}$ power, as described in the methodology section.

2. Euclidean Distance Calculation for Negatives: - For each negative sample \mathbf{v}_i , compute the Euclidean distance to the target vector \mathbf{u} :

$$\text{distance_neg}_i = \|\mathbf{u} - \mathbf{v}_i\| + \epsilon \quad (1.19)$$

3. Loss Computation: - Define the loss for negative pairs as the negative of the Euclidean distance, aiming to maximize these distances:

$$\mathcal{L}_{\text{neg}} = - \sum_{i=1}^k \text{distance_neg}_i \quad (1.20)$$

4. Gradient Calculation: - Compute the gradients with respect to \mathbf{u} and each \mathbf{v}_i :

$$\nabla_{\mathbf{u}} \mathcal{L}_{\text{neg}} = \sum_{i=1}^k \frac{-2(\mathbf{u} - \mathbf{v}_i)}{\text{distance_neg}_i} \quad (1.21)$$

$$\nabla_{\mathbf{v}_i} \mathcal{L}_{\text{neg}} = \frac{2(\mathbf{u} - \mathbf{v}_i)}{\text{distance_neg}_i} \quad (1.22)$$

Parameter Updates

1. Combining Gradients: - Aggregate the gradients from both positive and negative pairs:

$$\nabla_{\mathbf{u}} \mathcal{L} = \nabla_{\mathbf{u}} \mathcal{L}_{\text{pos}} + \nabla_{\mathbf{u}} \mathcal{L}_{\text{neg}} \quad (1.23)$$

$$\nabla_{\mathbf{v}} \mathcal{L} = \nabla_{\mathbf{v}} \mathcal{L}_{\text{pos}} + \sum_{i=1}^k \nabla_{\mathbf{v}_i} \mathcal{L}_{\text{neg}} \quad (1.24)$$

2. Updating Embeddings: - Adjust the target and context embeddings using the computed gradients and the predefined learning rate η :

$$\mathbf{u} \leftarrow \mathbf{u} - \eta \nabla_{\mathbf{u}} \mathcal{L} \quad (1.25)$$

$$\mathbf{v} \leftarrow \mathbf{v} - \eta \nabla_{\mathbf{v}} \mathcal{L} \quad (1.26)$$

3. Normalization: - Normalize the updated embeddings to maintain unit length, ensuring consistency in the vector space:

$$\mathbf{u} \leftarrow \frac{\mathbf{u}}{\|\mathbf{u}\| + \epsilon}, \quad \mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\| + \epsilon}, \quad \mathbf{v}_i \leftarrow \frac{\mathbf{v}_i}{\|\mathbf{v}_i\| + \epsilon} \quad \forall i \in \{1, \dots, k\} \quad (1.27)$$

1.4.3 Rationale

The core intuition behind this methodology lies in adjusting word vectors based on their semantic relationships. By minimizing the Euclidean distance between related word pairs, the model ensures that semantically similar words occupy proximate positions in the vector space. Conversely, by maximizing the distance between unrelated pairs through negative sampling, the model effectively differentiates between semantically related and unrelated words.

Normalization plays a critical role in this process by maintaining consistent vector magnitudes, thereby making Euclidean distance a reliable proxy for cosine similarity. This alignment ensures that the optimization objectives are coherent and that the learned embeddings accurately reflect the intended semantic structures.

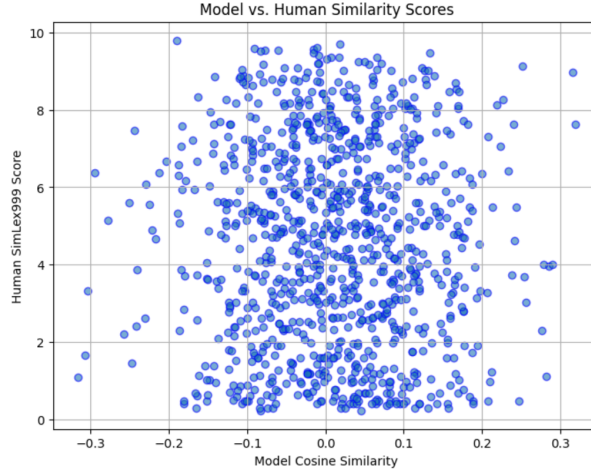


Figure 1.2: Simlex eval for Euclidean Distance Model

1.4.4 Evaluation

The evaluation steps remain same for this case as well where we observe Spearman's rank correlation of -0.0165 and P-value of 0.6050. Which is similar to our cosine approach[No pun intended !].

And the scores also look like the cosine sim model, where we observe the max similarity of around 0.3, which shows that the math is correct !

$$\text{Euclidean_distance}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\| = \sqrt{2(1 - \mathbf{u} \cdot \mathbf{v})} \quad (1.28)$$

Thus, minimizing the Euclidean distance $\|\mathbf{u} - \mathbf{v}\|$ is equivalent to maximizing the cosine similarity $\mathbf{u} \cdot \mathbf{v}$

1.5 Replicating 's work - Word2Vec

There is a fundamental difference in the way I have approached this problem and the authors of (Mikolov et al., 2013) approached it, although the central idea of learning word embeddings through contextual relationships remains the same. While Mikolov et al. employ a probabilistic framework that maximizes the likelihood of true context word pairs and minimizes the likelihood of negative samples using sigmoid-based loss functions, my approach directly optimizes the geometric relationships between word vectors by minimizing Euclidean distances(resp. minimizing cosine sim) for positive pairs and maximizing them for negative pairs(resp. minimizing cosine sim). This shift from a probabilistic to a geometric optimization framework leads to distinct methods of similarity measurement and embedding updates, as detailed in the following pointers.

- **Objective:** Mikolov et al. aim to maximize the probability of true context word pairs and minimize the probability of negative samples through a probabilistic loss function. In contrast, my approach seeks to directly minimize the Euclidean distance between positive pairs and maximize it for negative pairs, focusing on geometric relationships.
- **Similarity Measure:** Word2Vec utilizes the raw dot product as a similarity measure within its probabilistic framework, inherently considering both the direction

and magnitude of word vectors. My method employs cosine similarity by normalizing vectors to unit length, thereby focusing solely on their directional alignment and making the similarity measure scale-invariant.

- **Loss Function:** The loss function in Word2Vec is based on sigmoid functions, treating the task as a binary classification problem where the model distinguishes between true and negative word pairs. Conversely, my loss function is based on Euclidean distance, directly optimizing the spatial relationships between vectors without framing it as a probabilistic classification task.
- **Normalization:** Word2Vec does not normalize word vectors, allowing their magnitudes to influence similarity measures. My approach involves normalizing vectors to unit length after each update, ensuring that cosine similarity accurately reflects the angle between vectors without being affected by their magnitudes.
- **Optimization Strategy:** Word2Vec leverages probabilistic gradients derived from the sigmoid function to adjust word vectors, aligning with its probabilistic loss framework. In contrast, my approach uses gradient updates based on Euclidean distance metrics, facilitating direct geometric adjustments to the word vectors.
- **Theoretical Foundation:** Word2Vec is grounded in the Skip-Gram with Negative Sampling (SGNS) framework, which efficiently approximates the full softmax function by considering only a subset of negative samples. My methodology is based on direct geometric optimization, focusing on the spatial relationships within the embedding space to capture semantic similarities.

So after replicating I observe a Spearman's rank correlation of 0.1363 with p-value: 1.7614e-05, just with training it for 5 epochs and it starts to aligns with the actual word2vec [google-news-300] Simlex Scores which are with Spearman's rank correlation of 0.4420 and P-value: 0

This can be easily observed in the figures below



Figure 1.3: Replication [My embeddings alignment]

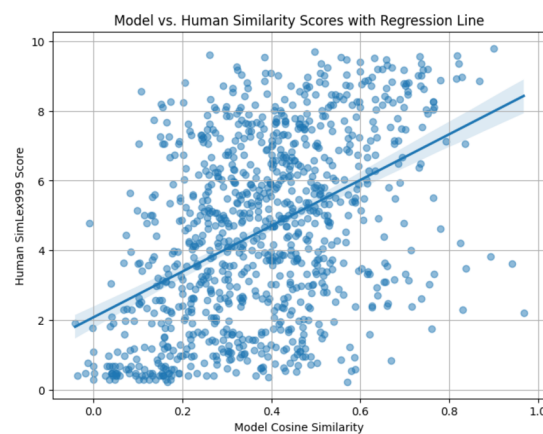


Figure 1.4: Actual Pre-trained Word2Vec alignment

Chapter 2

Unconstrained Word to Vector Evaluation

Now, since the assignment in this part allows the use of any pre-trained model and dataset, we first evaluate existing pre-trained word embedding models, including three open-source models—BERT, RoBERTa, and GloVe—and three closed-source models—text-embedding-3-small, text-embedding-3-large, and text-embedding-ada-002.

The results obtained on the SimLex-999 evaluation set are as follows:

Model Name	Spearman Correlation
BERT [Word Only]	0.15
BERT [with added context]	0.30
RoBERTa [Word Only]	-0.0617 [P-value = 0.0512]
RoBERTa [with added context]	0.4344
GloVe	0.37
text-embedding-3-small	0.50
text-embedding-3-large	0.56
text-embedding-ada-002	0.43

Table 2.1: Spearman Correlation on SimLex-999 Evaluation Set

These metrics demonstrate that the closed-source models generally achieve higher Spearman correlation coefficients compared to their open-source counterparts, indicating a better alignment with human-assigned similarity scores on the SimLex-999 dataset.

- BERT by [Devlin et al., 2019](#) - BERT was chosen because it is simply the first thing that comes to our mind when it comes to creating embeddings. BERT produces contextual embeddings rather than static vectors. When we talk about “BERT [Word Only]” for words in isolation (for our case), we usually extract embeddings by feeding the words in a minimal or blank context (blank in our case). But this is not exactly how BERT was intended to be used. So I use Azure OpenAI credits to fill the words with minimal context which converts the words into phrases (I only perform eval for the first 115 word pairs and not the entire Simlex 999 dataset, because of rate and credit limits)

And yes, we observe what was intended, the BERT eval with added context performs much better than BERT[Word only] on the eval. This is intended because BERT

is inherently designed for giving contextual embeddings and not static embeddings. It is pretrained on a large corpus using Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). It is based on the Transformer encoder. Uses multi-head self-attention and feed-forward layers stacked multiple times.

The observed improvement in Spearman correlation from 0.15 for BERT [Word Only] to 0.30 for BERT with added context underscores the model's reliance on contextual information to generate meaningful embeddings. BERT's architecture is fundamentally designed to understand and utilize the surrounding text to disambiguate word meanings and capture nuanced semantic relationships. Even with minimal added context, BERT can activate its bidirectional attention mechanisms to better infer the intended meaning based on the limited surrounding words. This contextual infusion allows BERT to produce embeddings that more accurately reflect the semantic similarities and differences as perceived by humans, thereby enhancing the correlation with the SimLex-999 evaluation set. In contrast, without any context, BERT struggles to anchor the word embeddings in a meaningful semantic space, resulting in poorer performance compared to static embeddings like word2vec, which are inherently designed to capture global semantic relationships without reliance on context.

Note : I do not rule out the possibility of sampling bias on the 115 pairs.

The prompt used to fill context:

```
"request": "Create a list of two small context phrases using
the given word pairs meaningfully - word1:{word1} and
word2:{word2}.",
"instructions": [
  "Context1 is for using word1 in a sentence, and context2 is
  for using word2 in a sentence.",
  "Ensure that you try to keep the context as same as
  possible, just the word usage should be different.",
  "The context provided should be such that the only
  differentiating factor between them should be the word
  usage.",
  "Do not include any explanations, only provide a RFC8259
  compliant JSON response following this format without
  deviation."
],
"example": {
  "word1": "small",
  "word2": "big",
  "response": [
    {
      "context1": "that fridge is small",
      "context2": "that fridge is big"
    }
  ]
},
"expected_format": [
  {
    "context1": "{sentence with word1}",
    "context2": "{sentence with word2}"
  }
]
```

Listing 2.1: Prompt

- RoBERTa by [Liu et al., 2019](#) - RoBERTa [Word Only] delivered an unexpected correlation of **-0.0617**, initially leaving me surprised. My first conjecture was that the removal of the Next Sentence Prediction (NSP) task had adversely impacted its performance—a potentially tweet-worthy insight. However, the accompanying p-value of **0.0512** suggested that this result was marginally non-significant.

In stark contrast, RoBERTa [With Added Context] achieved a correlation of **0.43**, surpassing BERT [With Added Context]. This improvement can be attributed to RoBERTa's **Robustly Optimized Pretraining Approach**, which fine-tunes various aspects of the training process to enhance performance.

When words are presented in isolation, RoBERTa's self-attention mechanisms lack the surrounding tokens necessary to inform the embeddings, effectively reducing them to static-like representations. This limitation diminishes the model's capacity to capture nuanced semantic relationships. Conversely, even minimal context provides a framework that allows RoBERTa to leverage its self-attention mechanisms effectively, generating embeddings that reflect both the target word and its immediate surroundings. In a context-free setting, positional information becomes less meaningful, potentially disrupting the embedding generation process.

Furthermore, the lower performance of RoBERTa [Word Only] compared to BERT [Word Only] may stem from differences in their tokenization methods. BERT employs **WordPiece tokenization**, which breaks words into subword units based on a fixed vocabulary, whereas RoBERTa utilizes **Byte-Pair Encoding (BPE)**, allowing for more flexible and granular token splits. This variation in tokenization can influence how words are segmented and represented, potentially affecting embedding quality when words are processed in singularity without context.

The overall effectiveness of contextual embedding models hinges on the presence of meaningful input data. In a context-free scenario, the absence of surrounding tokens compromises the model's ability to generate semantically rich embeddings, underscoring the critical role that context plays in leveraging the full potential of models like RoBERTa.

- GloVe by [Pennington et al., 2014](#) and openAI embeddings [Neelakantan et al. \(2022\)](#) - The architectural differences between GloVe and OpenAI's models are pivotal to their performance disparities. GloVe's reliance on global co-occurrence statistics provides robust representations of broad semantic relationships but lacks the flexibility to adapt embeddings based on specific contexts. On the other hand, OpenAI's Transformer-based models leverage their encoder architecture to generate dynamic embeddings that can capture nuanced semantic relationships by considering the broader context in which words appear. The contrastive learning objective further refines these embeddings by explicitly training the model to recognize and encode semantic similarities and distinctions.

OpenAI's embedding models caught my attention because they perform exceptionally well on both word-level and sentence-level tasks. This versatility is likely due to the combination of the Transformer architecture and contrastive learning, which

together enable the models to produce embeddings that are both semantically rich and contextually adaptable.

Note - I am still uncertain about the underlying architecture of the OpenAI embeddings. Reflecting on this, one might wonder why these models do not fully exploit the contextual capabilities that Transformers are renowned for. The very strength of Transformer-based models lies in their ability to generate dynamic, context-aware representations, which adapt based on the surrounding text. This adaptability is a key factor that often makes Transformers superior to traditional static embedding approaches like GloVe. Therefore, the decision to produce static embeddings using a Transformer architecture seems counterintuitive, as it seemingly underutilizes the model's primary advantage—contextual understanding.

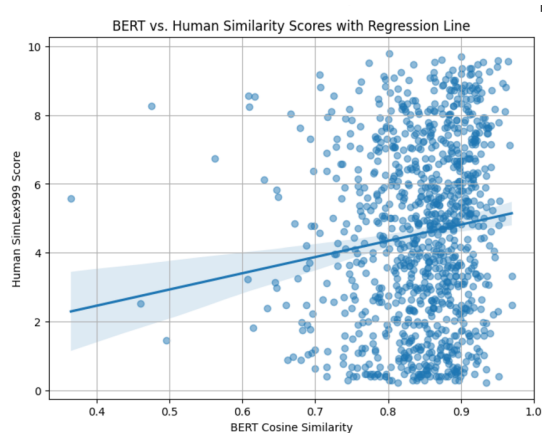


Figure 2.1: BERT [Word Only]

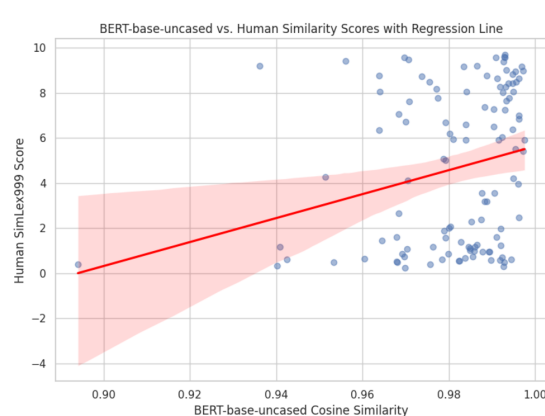


Figure 2.2: BERT [w context]

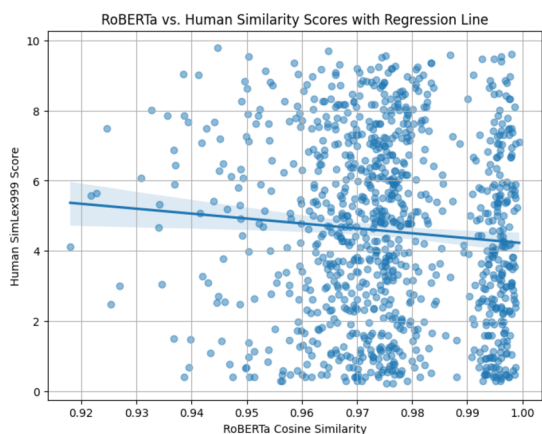


Figure 2.3: RoBERTa [Word Only]



Figure 2.4: RoBERTa [w context]

Chapter 3

Phrase and Sentence Embeddings

When approaching phrase and sentence embeddings, I must admit that my initial repertoire of techniques was limited. My primary considerations revolved around leveraging pre-trained embeddings with various averaging techniques or fine-tuning with task-specific loss functions, depending on the requirements of the downstream evaluation task—in this case, binary classification of phrases and sentences.

Before delving into aggregation and fine-tuning strategies, I took a closer look at the dataset. The tasks predominantly involved sentences like “Something A happened on B because C happened” versus “Something C happened on B because A happened.” While these sentences may appear similar at first glance, their meanings are fundamentally different—the causality is reversed. This highlights the critical importance of directionality and word positioning when designing an approach. The challenge is even more pronounced when dealing with standalone phrases, which are typically just 2-3 words combined. Here, the semantic distinction often only becomes apparent when the phrases are contextualized within sentences.

Initially, I was uncertain whether the provided contextual information for the phrases should be incorporated. However, I ultimately decided to include it, reasoning that the additional context could help the model better grasp the semantic relationships. I also thought of ways to analyze the semantic proximity and distinction between phrases when augmented with varying contexts, using large language models (LLMs).

3.1 Pretrained Embeddings with Aggregation - Phrase

The first approach uses pretrained embedding model of BERT - base -uncased. Models like these tokenize input text into words or subwords and generate embeddings for each token. To create a single embedding for a phrase or sentence, we combine these token embeddings using aggregation [1000 datapoints equally balanced for both classes is used for the phrase task] techniques:

3.1.1 Pooling Strategy

To explore sentence-level representations for classification tasks, I implemented a model leveraging the `bert-base-uncased` architecture as a shared encoder for paired input sequences. This approach combines token-level embeddings produced by BERT into fixed-length representations using max pooling, which were then used for downstream classification.

I began by employing `bert-base-uncased`, a pre-trained BERT model, as the base encoder. Each input sequence, comprising a phrase and its corresponding context sentence, was tokenized using the Hugging Face `transformers` library. The tokenized inputs included `input_ids` and `attention_mask` for both sentence pairs, ensuring compatibility with BERT's input requirements.

To encode the paired inputs, I implemented the **BaseBERTModel**, where each sequence was processed independently through the shared BERT encoder, producing token-level contextual embeddings (`last_hidden_state`) for both sequences. To reduce the variable-length token embeddings into fixed-size representations, I applied **Adaptive Max Pooling** across the token dimension. This pooling strategy extracts the most salient feature across tokens, resulting in sentence-level embeddings of size `hidden_dim`.

The pooled representations of both sequences were concatenated and passed through a fully connected linear layer to produce logits corresponding to the target classification labels. I trained this architecture using the cross-entropy loss function.

Inputs were tokenized to a maximum sequence length of 128 and padded dynamically during batching. The training objective was to classify the relationship between paired inputs, with the final logits mapped to two output classes. The model was evaluated using accuracy and F1-score as metrics, and results were reported after three epochs.

3.1.2 Averaging Strategy

The **BaseBERTModel** remains the same in this scenario as well.

I applied **Mean Pooling**, where each token embedding was weighted by its attention mask before averaging across the sequence.

The mean-pooled embeddings of both sequences were concatenated and passed through a fully connected linear layer for classification. The training objective and evaluation setup mirrored those of the pooling-based model, with inputs tokenized to a maximum sequence length of 128 and dynamically padded during batching. The model was optimized using the cross-entropy loss function.

This architecture retains the simplicity and efficiency of pooling-based methods while leveraging mean aggregation to capture a broader representation of the sequence. Results and comparative analysis are discussed in subsequent sections.

3.1.3 Concatention Strategy

To investigate the utility of sentence-level representations derived from specific tokens, I implemented a model that utilizes the `[CLS]` token embeddings produced by BERT. This approach directly leverages the contextualized embeddings of the `[CLS]` token, which BERT is pre-trained to use as a representation of the entire sequence, for downstream classification tasks.

The architecture employs the **BaseBERTModel** to encode paired input sequences independently through a shared BERT encoder. Tokenized inputs, including `input_ids` and `attention_mask`, were processed to produce token-level contextual embeddings (`last_hidden_state`). From these embeddings, the `[CLS]` token (corresponding to the first position in each sequence) was extracted as the sentence-level representation. This token was designed during BERT's pre-training to capture a global representation of the input.

The `[CLS]` embeddings from both sequences were concatenated to form a combined feature vector. This representation was passed through a fully connected linear layer for

classification. As with the rest aggregation-based models, inputs were tokenized to a maximum sequence length of 128 and dynamically padded during batching. The model was optimized using the cross-entropy loss function, and evaluation metrics such as accuracy and F1-score were computed after three epochs of training.

This strategy explicitly relies on the pretrained capabilities of BERT’s [CLS] token for sequence-level tasks. The simplicity of the concatenation mechanism and the pre-trained nature of the [CLS] token make this approach computationally efficient while preserving the semantic richness of the input sequences. Results and comparisons with other strategies are presented in subsequent sections.

These strategies integrate pretrained embeddings into downstream tasks such as classification, regression, or clustering by keeping the embedding layers fixed during training. In this setup, the embeddings serve as static features, and only the task-specific layers are trained.

- **Frozen Embeddings:** The pretrained embedding weights remain unchanged, preserving the semantic information they contain.
- **Training Downstream Layers:** Only the subsequent layers, classifiers [for our case], are updated to adapt the static embeddings to the specific task.

In all the strategies implemented—Pooling, Averaging, and Concatenation—the primary goal is to generate fixed-length embeddings from token-level representations provided by a pre-trained BERT model and use them for binary classification. Below, I describe the mathematical framework for these strategies, highlighting where learning occurs and how gradients are propagated.

3.1.4 Model Formulation

Let $\mathbf{X}_1 \in \mathbb{R}^{T_1 \times d}$ and $\mathbf{X}_2 \in \mathbb{R}^{T_2 \times d}$ represent the token embeddings for the phrase and sentence, respectively, where T_1 and T_2 are the sequence lengths and d is the hidden dimension of BERT embeddings. These embeddings are generated by passing the input phrase and sentence through a shared pre-trained BERT model:

$$\mathbf{X}_1 = \text{BERT}(\text{phrase}), \quad \mathbf{X}_2 = \text{BERT}(\text{sentence})$$

The goal is to aggregate \mathbf{X}_1 and \mathbf{X}_2 into fixed-length embeddings, concatenate them, and classify the concatenated representation into two classes.

3.1.5 Aggregation Strategies

The fixed-length embeddings, denoted by \mathbf{p}_1 and \mathbf{p}_2 for the phrase and sentence, are computed using one of the following aggregation strategies:

- **Pooling:** Apply max pooling across the token dimension:

$$\mathbf{p}_1 = \max(\mathbf{X}_1, \text{dim} = 0), \quad \mathbf{p}_2 = \max(\mathbf{X}_2, \text{dim} = 0)$$

- **Averaging:** Compute the mean of the token embeddings, weighted by the attention mask to exclude padding tokens:

$$\mathbf{p}_1 = \frac{\sum_{i=1}^{T_1} \mathbf{X}_1[i] \cdot \text{mask}_1[i]}{\sum_{i=1}^{T_1} \text{mask}_1[i]}, \quad \mathbf{p}_2 = \frac{\sum_{i=1}^{T_2} \mathbf{X}_2[i] \cdot \text{mask}_2[i]}{\sum_{i=1}^{T_2} \text{mask}_2[i]}$$

- **Concatenation:** Use the [CLS] token embedding from each sequence:

$$\mathbf{p}_1 = \mathbf{X}_1[0], \quad \mathbf{p}_2 = \mathbf{X}_2[0]$$

The resulting fixed-length embeddings $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^d$ are concatenated:

$$\mathbf{z} = [\mathbf{p}_1; \mathbf{p}_2], \quad \mathbf{z} \in \mathbb{R}^{2d}$$

3.1.6 Classification and Loss Calculation

The concatenated embedding \mathbf{z} is passed through a fully connected classification layer to produce logits for the binary classification task:

$$\mathbf{o} = \mathbf{W}_c \mathbf{z} + \mathbf{b}_c, \quad \mathbf{o} \in \mathbb{R}^2$$

where $\mathbf{W}_c \in \mathbb{R}^{2d \times 2}$ and $\mathbf{b}_c \in \mathbb{R}^2$ are the learnable parameters of the classifier. The probabilities for the two classes are computed using the softmax function:

$$\hat{y} = \text{softmax}(\mathbf{o})$$

The cross-entropy loss is then computed as:

$$\mathcal{L} = - \sum_{i=1}^2 y_i \log(\hat{y}_i)$$

where y is the one-hot encoded ground truth label.

3.1.7 Backpropagation and Learning

Gradients of the loss are computed with respect to:

- The classifier parameters \mathbf{W}_c and \mathbf{b}_c :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_c}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}_c}$$

These are updated during backpropagation using gradient descent.

- The embeddings \mathbf{p}_1 and \mathbf{p}_2 :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_1}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{p}_2}$$

The gradients flow through the aggregation step (if applicable) and propagate back to the BERT model. If fine-tuning is enabled, the gradients further update BERT's transformer layers.

The aggregation strategy influences the gradients as follows:

- **Pooling:** Gradients flow back to the tokens contributing to the maximum value.
- **Averaging:** Gradients are distributed proportionally across all tokens, weighted by the attention mask.

- **Concatenation:** Gradients directly update the [CLS] token representation.

When using a hidden size of 768 (typical for BERT) and training for only 3 epochs, We observe similar performance whether or not the BERT embeddings are fine-tuned (`requires_grad = True` vs. `False`). This may occur because BERT's embeddings are already learned from large-scale pretraining, and a short training regimen may not significantly alter or improve these representations.

Furthermore, the three aggregation strategies discussed (max pooling, average pooling, and concatenation of a special token such as [CLS]) are all differentiable or piecewise differentiable:

1. **Max Pooling:** Although the max function is not differentiable at points where multiple values tie for the maximum, deep-learning frameworks (like PyTorch) use subgradients, allowing gradients to flow back through the selected index.
2. **Average Pooling:** The mean operation

$$\frac{1}{n} \sum_{i=1}^n x_i$$

is fully differentiable. During backpropagation, each input receives an equal fraction of the upstream gradient.

3. **Concatenation:** Placing vectors side-by-side is a simple reorganization of the tensor along a given dimension. The gradient splits back into corresponding positions (effectively an identity mapping for each chunk), thus preserving end-to-end differentiability.

Because all these pooling/aggregation methods permit gradients to propagate back through the network, they support fine-tuning. However, in our short, training regime with only 3 epochs, the model may not have enough opportunity to significantly benefit from modifying the pretrained embeddings, thus yielding similar results with or without gradient flow through the embedding layers of BERT.

3.1.8 Phrase - Weighted learning

This **PhraseSimilarityModel** is designed to classify the similarity between two phrases by incorporating context-specific embeddings and learnable transformations

- **BERT Backbone:** Pretrained BERT is used to independently encode:

1. Phrases (P_1, P_2),
2. Phrase1 with Sentence1 context ($P_1 + S_1$),
3. Phrase2 with Sentence2 context ($P_2 + S_2$).

The pooled [CLS] token embedding (`pooler_output`) is extracted for each input.

- **Linear Transformations:**

- **phrase_linear:** Transforms phrase embeddings to emphasize similarity-relevant features.

- **context_linear**: Transforms context-augmented embeddings (e.g., $P_1 + S_1$) to capture phrase-context interactions.
- **Feature Concatenation**: Outputs from the linear layers are concatenated to form a combined representation of size $3 \times \text{hidden_size}$.
- **Classifier**: A fully connected layer maps the combined representation to the similarity score, using cross-entropy loss for optimization.

This architecture explicitly models phrase-context relationships, enhancing representation quality and addressing the limitations of raw aggregation baselines, where embeddings are concatenated without transformation or interaction modeling.

Test Set Performance by Strategy - Phrase

Strategy	Accuracy (%)	F1 Score
Pooling	55.40	0.547
Averaging	60.35	0.589
Concatenation	60.30	0.598
Weighted learning	64.30	0.6403

Table 3.1: Performance of different strategies (Pooling, Averaging, Concatenation) on the test set.

This performance remains the same for both (`requires_grad = True & False`)

3.2 Sentence Task

The same set of approaches can be applied to aggregate over-sentence embeddings as well, but here we turn to more different approaches which are based on two-branch type architectures with the objective to find a good threshold to classify two given sentences.

- **Training Frameworks**:
 - **Triplet Loss with Contrastive Learning**: For scenarios involving three inputs (anchor, positive, and negative), triplet loss helps the model learn to bring similar phrases closer and push dissimilar ones apart. But it wasn't easy to mine for perfect triplets so I did not use this.
 - **Siamese Networks**: When dealing with pairs of inputs, siamese networks process each input through identical subnetworks and learn to measure similarity or difference between the pairs.

3.2.1 TwoBranchModel

The model utilizes a two-branch architecture for sentence-pair classification tasks, with a shared DistilBERT encoder to independently process each sentence in the pair. Given two input sentences, s_1 and s_2 , the tokenized representations are passed through DistilBERT

to obtain contextual embeddings, h_1 and h_2 , which correspond to the CLS token’s output from the last hidden state. These embeddings are then combined using multiple heuristics based out from (Mou et al., 2016) to capture both individual sentence representations and their relational features. The combined representation is subsequently fed into a fully connected classification head to predict the probability of the positive class.

The relational features are constructed using four transformations: concatenation ($[h_1; h_2]$), element-wise difference ($h_1 - h_2$), element-wise product ($h_1 \odot h_2$), and a symmetric combination ($[h_1; h_2; h_1 - h_2; h_1 \odot h_2]$). While $h_1 - h_2$ is theoretically a linear combination of h_1 and h_2 (i.e., $W[h_1; h_2]^\top = [W_1; -W_1][h_1; h_2]^\top$), it uses the findings from word embeddings (Mikolov et al., 2013), where vector offsets capture semantic relationships, such as "king - man + woman = queen". The addition of $h_1 - h_2$ therefore ensures better interpretability and enriches the model’s ability to differentiate sentence-pair relationships.

The fully connected classification head processes the combined representation to generate the final output. This head consists of a linear layer projecting the input into a 128-dimensional space, followed by a ReLU activation, a dropout layer to mitigate overfitting, and a final linear transformation with a sigmoid activation. The overall model is trained using binary cross-entropy loss, with the sigmoid output interpreted as the probability of the positive class.

Despite the design of the model, the results did not meet the expectations I set during its conceptualization. While the use of DistilBERT as a backbone provided a strong foundation for generating contextual embeddings and the inclusion of relational features such as element-wise difference and product aimed to capture nuanced sentence-pair relationships, the model’s performance plateaued. The accuracy remained only marginally better than random guessing across various thresholds, and no significant improvement was observed compared to simpler baselines.

3.2.2 TwoBranchModel + Cross Attention

After the failure of the previous model I thought maybe there is a explicit need to model interactions between the two sentences which a mere difference/offset and multiplication is not able to capture, therefore this architecture incorporates a multi-head cross-attention module to model inter-sentence relationships more effectively.

The first stage of the model involves independent processing of the two input sentences. The sentences are tokenized and passed through a shared DistilBERT encoder to generate contextual embeddings, h_1 and h_2 , with dimensions $[\text{batch_size}, \text{seq_len}, \text{hidden_size}]$. Unlike standard approaches that directly use these embeddings for relational feature computation, the model employs a cross-attention mechanism to capture contextual dependencies between the two embeddings. Specifically, for each sentence embedding, the multi-head attention mechanism computes a new representation by attending to the tokens of the other sentence. This results in two refined embeddings: `attended.h1` (where h_1 attends to h_2) and `attended.h2` (where h_2 attends to h_1).

After the cross-attention step, the [CLS] token from the attended embeddings is extracted as a sentence-level representation, `h1_cls` and `h2_cls`. These embeddings are combined using four relational heuristics: concatenation ($[h1_cls; h2_cls]$), element-wise difference ($h1_cls - h2_cls$), element-wise product ($h1_cls \odot h2_cls$), and a symmetric combination ($[h1_cls; h2_cls; h1_cls - h2_cls; h1_cls \odot h2_cls]$). The resulting vector is passed through a fully connected neural network, comprising three layers with intermediate non-

linearities and dropout, to produce a single probability score using a sigmoid activation function.

But this model also turns out to be just better than random chance results

3.3 Analysis

The inclusion of cross-attention mechanisms or a two-branch architecture does not improve performance. Cross-attention, intended to capture token-level interactions between the two sentences, may amplify irrelevant features or noise in the embeddings. This results in the [CLS] token representations clustering in a narrow band, producing logits that converge near the sigmoid midpoint. Consequently, the model tends to predict the same class for most samples, flipping only when thresholds cross certain extremes. The averaged embeddings lose variance, further limiting the model’s capacity to differentiate between the two classes.

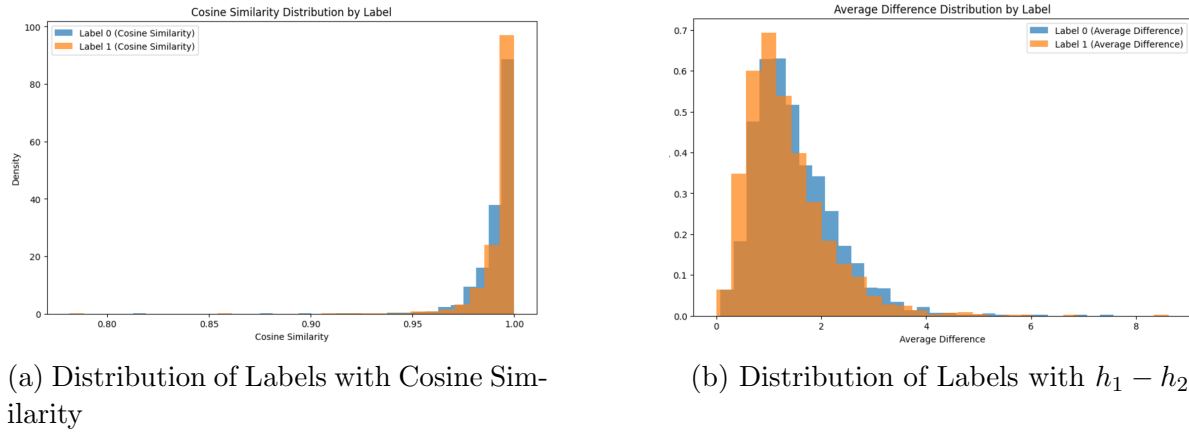
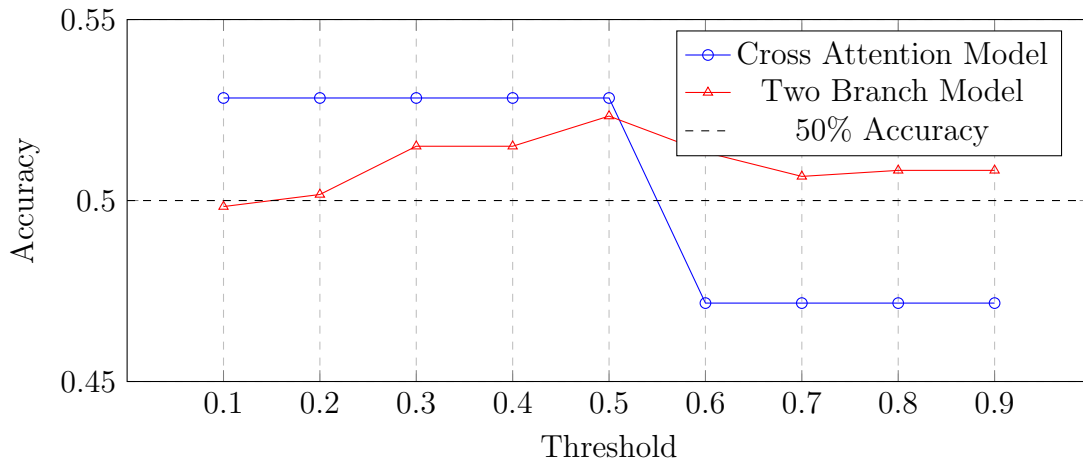


Figure 3.1: Comparison of Label Distributions

The embeddings alone do not provide a clear gradient signal to the classifier, rendering architectural enhancements like cross-attention or pairwise feature combinations ineffective. To address this, there is a need for fine-tuning more sophisticated models directly on the dataset, enabling the representations to adapt to the specific semantic distinctions required for the task. This challenge aligns with the inherent complexity of sentence similarity tasks, where even with contextual embeddings, binary classification objectives struggle to capture subtle relational differences



The results highlight significant challenges in leveraging the current dataset for the sentence-pair classification task. The semantic cues that distinguish positive labels (1) from negative labels (0) are subtle, making it difficult for the model to learn meaningful patterns. The use of frozen embeddings from DistilBERT exacerbates this issue, as they fail to capture the nuanced semantics required for effective classification. This is evident from the minimal variance observed in the representations of the two labels.

Chapter 4

BONUS Tasks

4.1 Binary classification using LLama 8b

I select 1000 entries from each test dataset provided, and using the free tier Groq API Llama 8b model. I run the classification task using the below prompt [One shot]

```
f"Determine if the following two pairs of phrases have the same
  meaning given the usage of these phrases in corresponding
  sentences.\n\n"
f"Pair 1:\nPhrase: \"{phrase1}\"\\nSentence: \"{sentence1}\"\\n\n"
f"Pair 2:\nPhrase: \"{phrase2}\"\\nSentence: \"{sentence2}\"\\n\n"
f"Respond with '1' if they mean the same and '0' if they mean
  different in the given sentence. JUST OUTPUT 1 or 0."
```

Listing 4.1: Prompt for Phrase Classification Task

$$\begin{bmatrix} 189 & 311 \\ 144 & 356 \end{bmatrix}$$

ConfusionMatrix

Accuracy: 54.50%

Class	Precision	Recall
0	0.57	0.38
1	0.53	0.71

```
f"Determine if the following two sentences have the same meaning.\n
  \n"
f"Sentence 1: \"{sentence1}\"\\n\n"
f"Sentence 2: \"{sentence2}\"\\n\n"
f"Respond with '1' if they mean the same and '0' if they mean
  different. DONT GIVE ANYTHING ELSE, JUST OUTPUT 1 or 0"
```

Listing 4.2: Prompt for Sentence Classification Task

$$\begin{bmatrix} 76 & 424 \\ 7 & 493 \end{bmatrix}$$

ConfusionMatrix

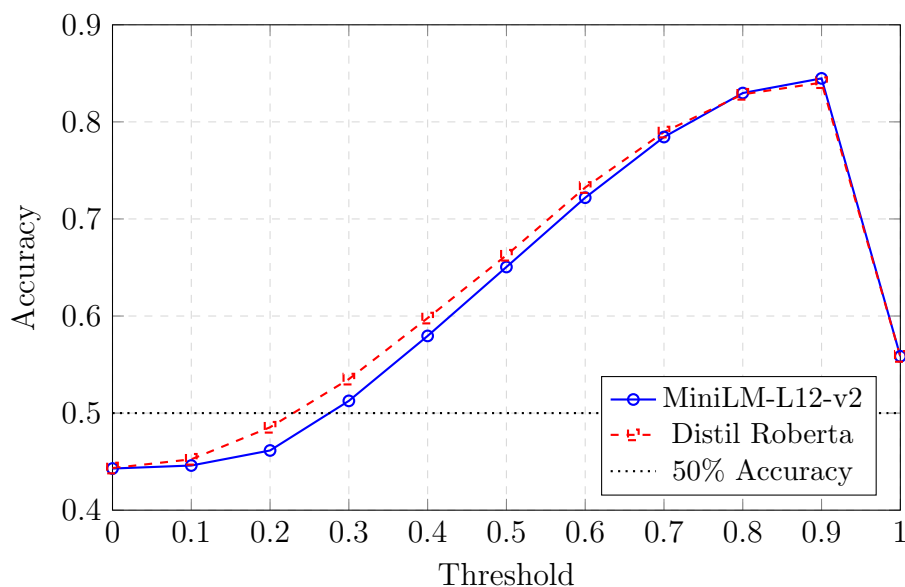
Accuracy: 56.90%

Class	Precision	Recall
0	0.92	0.15
1	0.54	0.99

4.2 Finetuning for Sentence classification

I opted for a contrastive loss because it fits naturally with our binary paraphrase detection setup. By taking pairs of sentences labeled as “similar” or “not similar,” contrastive loss nudges the model to place genuine paraphrases closer in the embedding space and push non-paraphrases further apart. This direct pairwise supervision ensures that each pair’s label is used transparently, avoiding the potential confusion that arises when all other sentences in a batch are treated as negatives (as in many ranking-based losses).

Accuracy vs. Threshold for MiniLM-L12-v2 and Distil Roberta



Model	Loss Function	Accuracy
Distil Roberta	Contrastive Loss	0.84
MiniLM	Contrastive Loss	0.84

Table 4.1: Best Performance of models on sentence binary classification fine-tuning task.

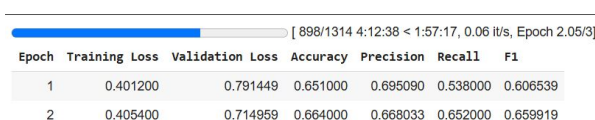
Interestingly, our highest classification accuracy occurs when we set the threshold at 0.9, which indicates that even these fine-tuned models only reliably discriminate paraphrases above a very high similarity score. Additionally, updating the entire model

during fine-tuning yields substantially better results than earlier, more “vanilla” methods—underscoring the complexity and non-triviality of the sentence classification task.

Further evidence of this complexity is seen in the high false-positive rate of large language models on similar data: even an 8B parameter pre-trained model struggles, suggesting that massive pre-training alone isn’t enough for capturing the nuances of the this task (or that my prompt engineering is terrible and could use improvement).

4.3 Finetuning for Phrase classification

I also attempted to train DeBERTa for phrase classification, but the process did not fully complete. While an interim evaluation indicated a validation accuracy of 66%, it remains uncertain how well DeBERTa ultimately performs on this task—this uncertainty is partially due to the fact that these are my first experiments fine-tuning such architectures



A screenshot of a Colab training progress bar and a table showing training metrics. The progress bar is at the top, showing 898/1314 steps, a time of 4:12:38, and a speed of 0.06 it/s. Below it is a table with 7 columns: Epoch, Training Loss, Validation Loss, Accuracy, Precision, Recall, and F1. The table contains two rows of data for epochs 1 and 2.

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.401200	0.791449	0.651000	0.695090	0.538000	0.606539
2	0.405400	0.714959	0.664000	0.668033	0.652000	0.659919

Figure 4.1: 4 Hours training of Deberta

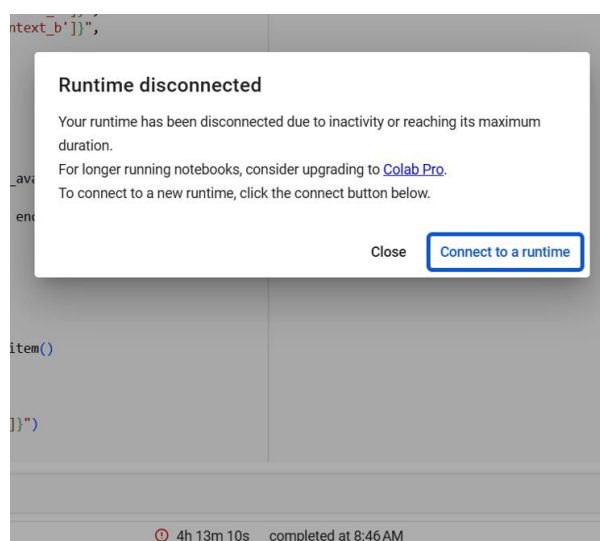


Figure 4.2: Disconnected Runtime after 4 hours

Chapter 5

Paper Summary - BERTScore

The BERTScore paper by [Zhang et al., 2020a](#) presents a metric for evaluating text generation tasks by using BERT’s contextual embeddings. Rather than relying on literal word overlaps, it compares tokens in a candidate text with tokens in a reference text at the embedding level. Each token in the candidate is matched to its closest semantic neighbor in the reference, and vice versa, and these similarity scores are averaged into a final evaluation score.

This approach goes beyond traditional n-gram metrics like BLEU or ROUGE by accounting for synonyms and rephrasings. Because modern language models capture rich contextual information, BERTScore can identify similar meaning even when wording differs. As a result, it often aligns more closely with human assessments of quality, particularly in cases where flexibility in expression is desirable. One example to explain this would be the usage of word **bank** differs in the two sentences - "The robber robbed the **bank**" and "The river has flooded its **bank**"

The paper demonstrates BERTScore’s range by testing it on multiple tasks. It also analyzes how changing model size or embedding layers affects results. These experiments suggest that BERTScore can capture subtle meaning shifts across diverse domains and is more forgiving of variations in wording, offering an alternative to purely lexical metrics.

5.1 Three Major Strengths

1. **Rich Contextual Interpretation** BERTScore taps into the nuanced embeddings learned by BERT, reducing the need for extensive tuning. It evaluates token meaning in context, which allows it to spot semantic similarities that simpler metrics often miss.

2. **Tolerance of Paraphrases and Synonyms** Because tokens are matched based on meaning rather than exact form, the metric remains stable even when candidate and reference texts look different on the surface. This is especially valuable in creative writing or open-ended tasks, where multiple phrasings can convey the same idea.

3. **Extensive Empirical Testing** The paper demonstrates BERTScore’s effectiveness across various datasets and tasks. This helps show that the metric is not a one-off tool but rather a robust measure that consistently correlates with human judgment.

5.2 Three Major Weaknesses

1. **Reliance on Model Variants and Layers** BERTScore’s performance can fluctuate depending on which pretrained model or which embedding layers are used. Although the paper offers some analysis, users may still struggle to pick the right model setup for their specific task. Changing the model size (for example, from BERT base to BERT large) and deciding which embedding layers to use can significantly influence BERTScore results. Larger models often learn richer patterns and subtle semantic nuances, which can boost correlation with human judgments. However, they also come with higher computational costs and might introduce diminishing returns if your text is straightforward or if you have limited computational resources. Earlier layers in BERT generally capture more syntactic and surface-level details, while later layers tend to encode deeper semantic information. Choosing only the final layer might give strong semantic signals but risk overlooking useful cues encoded in the intermediate layers. Some tasks benefit from combining embeddings across multiple layers, yielding a more balanced representation of syntax and semantics.

2. **High Computation Costs** Generating BERT embeddings for every token in a large corpus can be expensive and slow. This overhead may be prohibitive in contexts where quick evaluation is essential or where computational resources are limited.

3. **Emphasis on Local Alignment** BERTScore matches tokens on a local scale, which might overlook broader text organization or flow. Although it excels at capturing fine-grained semantics, it does not explicitly measure whether a text is cohesive at the paragraph or document level. This limitation means that while BERTScore is effective at ensuring that the content is semantically accurate and relevant, it does not evaluate whether the text is well-organized, logically structured, or easy to read as a whole. Consequently, a text could score highly on BERTScore by having all the right pieces but still fail to communicate effectively due to poor overall coherence and flow. This highlights the need for complementary evaluation metrics that can assess higher-level aspects of text quality alongside BERTScore’s token-level assessments.

5.3 Three Suggested Improvements

1. **Adaptive Layer Selection** Instead of forcing a single choice of layer, the paper could propose a process that **picks the best combination of layers for each task**. A small network could learn this layer-weighting scheme to improve consistency and reduce trial-and-error. This network could be either a **supervised network** or could use **Reinforcement Learning to learn policy**, or it could just be a **Multi-arm bandit** scenario where the embedding models and BERT models are the actions and the rewards would be the BERTScores the chosen setting may receive.

2. **Lightweight Approximation for Large Evaluations** To ease the computational burden, a streamlined method could approximate BERTScore without running full embeddings on every pair. Techniques like caching, token pruning, or distillation might significantly reduce compute time while preserving accuracy. But this said, I am not very well versed in proposing an optimization to this

3. **Enhanced Discourse-Level Analysis** Finally, extending the metric to capture broader coherence and structure would make it more holistic. This could involve combining paragraph-level embeddings with the current token alignment, giving a clearer picture of how a text flows as a whole. But said this, it may defeat the idea of **Lightweight Approximation for Large Evaluations** given above.

Bibliography

- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.
- Mou, L., Men, R., Li, G., Xu, Y., Zhang, L., Yan, R., and Jin, Z. (2016). Natural language inference by tree-based convolution and heuristic matching.
- Neelakantan, A., Xu, T., Puri, R., Radford, A., Han, J. M., Tworek, J., Yuan, Q., Tezak, N., Kim, J. W., Hallacy, C., Heidecke, J., Shyam, P., Power, B., Nekoul, T. E., Sastry, G., Krueger, G., Schnurr, D., Such, F. P., Hsu, K., Thompson, M., Khan, T., Sherbakov, T., Jang, J., Welinder, P., and Weng, L. (2022). Text and code embeddings by contrastive pre-training.
- Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.
- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020a). Bertscore: Evaluating text generation with bert.
- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020b). Bertscore: Evaluating text generation with bert.