



Semantic Textual Similarity

Team Player No. 456

Teammates

Abhinav Reddy Boddu (2021101034)

Rohit Gowlapalli (2021101113)

Sasidhar Chavali (2021101111)

PROBLEM STATEMENT

Semantic Textual Similarity (STS) quantifies the extent to which two text snippets share the same meaning. Given a pair of sentences, the model assigns a similarity score on a continuous scale from 0 to 1, where 0 denotes no semantic overlap and 1 indicates full equivalence. The model's performance is evaluated by calculating the correlation between its similarity scores and human judgments.

DATASETS USED

1. [STS Benchmark](#)
2. [Quora Dataset](#)
3. [Paraphrase Corpus \(by microsoft\)](#)
4. [SICK Dataset](#)

SICK dataset

We utilized the Sentences Involving Compositional Knowledge (SICK) dataset, which is specifically crafted to explore compositional distributional semantics. This dataset comprises numerous sentence pairs that illustrate a variety of lexical, syntactic, and semantic features. Each pair is annotated with two labels: relatedness and entailment. The original relatedness scores range from 1 to 5, indicating the degree of semantic similarity between the sentences. Since our task required scores on a 0 to 1 scale, we normalized the original labels accordingly, ensuring all relatedness values lie within the [0, 1] range.

STS Benchmark

We employed the STS Benchmark dataset, which consists of 8,628 sentence pairs categorized into three domains: news, image captions, and forum discussions. These sentences were carefully curated from English datasets used in the Semantic Textual Similarity (STS) tasks at the SemEval competitions between 2012 and 2017. The dataset captures a wide range of linguistic styles, including descriptive captions, news headlines, and casual online discussions, providing a comprehensive snapshot of natural language use. A key strength of this dataset is that similarity scores are based on human annotations, ensuring high-quality, trustworthy labels for model training and evaluation.

Final Dataset

Initially, we intended to utilize all four datasets, including the MSR and Quora datasets. However, both MSR and Quora are binary classification datasets and contain significantly more samples compared to STS and SICK. Incorporating them led to models leaning heavily toward binary classification rather than learning a continuous similarity score. Although we experimented with normalizing their labels to fit a continuous scale, the model performance deteriorated as a result. Consequently, we decided to exclude the MSR and Quora datasets and proceed only with the STS and SICK datasets.

For data preparation, we split each dataset into 70% training, 10% validation, and 20% testing. We then merged the corresponding splits from both datasets—combining the training sets, validation sets, and test sets respectively. These final merged sets were used to train and evaluate our models.

Methodology

N-gram Based Statistical Methods

1. We begin with simple statistical techniques that compare text based on overlapping words or sequences of words (n-grams). These methods provide a straightforward way to measure similarity.
2. Next, we retrieved the synsets (sets of cognitive synonyms) for each token in the first sentence and compared them with the synsets of each token in the second sentence. If any pair of synsets had a similarity score above a defined threshold (set to 0.3), we considered the token pair semantically aligned and added it to the list of alignments.
3. This comparison process was repeated for every token in the first sentence against every token in the second sentence.
4. Here's an illustrative example:
 - a. "She began her journey in the early morning."
 - b. "She started her trip at dawn."
5. This example highlights how different words like "began" and "started" or "journey" and "trip" can be aligned based on semantic similarity.

```
abhinav@asus-x540uar ~/abhi/college/sem8/INLP/project/INLP-Project/Methods/n_gram <main*>
> python alignments.py
[nltk_data] Downloading package stopwords to
[nltk_data] /home/abhinav/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/abhinav/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[('begin', 'start', 1.0), ('morning', 'dawn', 1.0), ('journey', 'trip', 0.5)]
```

6. the similarity score is calculated as follows after the alignments are obtained:

$$Sim(s1, s2) = \frac{n \cdot \sum_{al \in AL_{s1, s2}} al.s}{|T1| + |T2|}$$

Where, $|T1|$ and $|T2|$ are the number of tokens in sentence 1 and sentence 2 respectively. 'n' is the number of tokens contributing to the similarity score 's'.

7. Once we obtain the similarity scores, Pearson's correlation is calculated.

Results:

| N values | Pearsons correlation |
|----------|----------------------|
| 1 | 0.6551536335528231 |
| 2 | 0.6609738837298842 |
| 3 | 0.6609024530747485 |
| 4 | 0.6609024530747485 |
| 5 | 0.6609024530747485 |

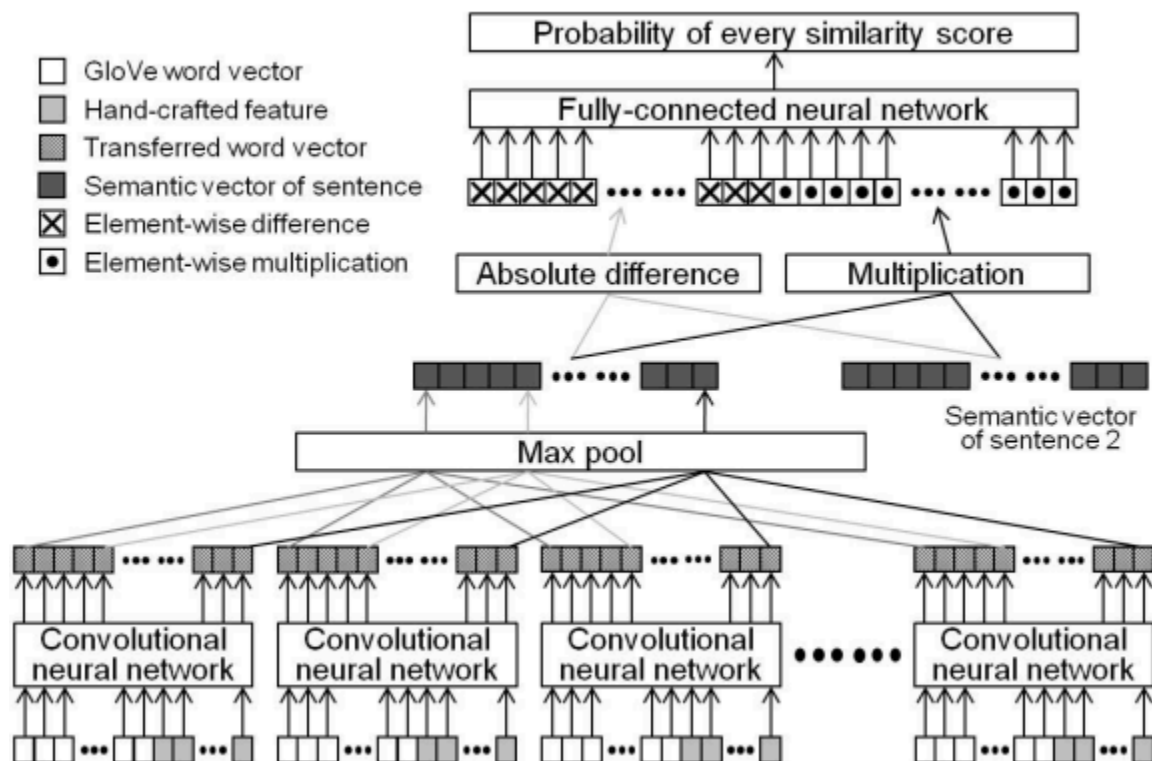
Performance improves slightly from unigrams to bigrams ($n = 1$ to 2): Moving from unigram (single word) tokens to bigrams results in a small but noticeable increase in correlation. This suggests that including short multi-word expressions captures more semantic information—likely due to better handling of phrase-level meaning or idiomatic expressions.

Plateau from $n = 2$ onwards: From $n = 2$ to $n = 5$, the correlation score remains nearly constant. This plateau indicates that higher-order n -grams (trigrams and beyond) do not contribute significant additional value. There are a few possible reasons for this

1. **Data sparsity:** As n increases, the frequency of matching n -gram tokens between sentences naturally decreases, reducing the chances of useful alignments.
2. **Diminishing returns:** Most meaningful alignments may already be captured with unigrams and bigrams. Higher-order n -grams might either repeat that information or become too specific.
3. **Fixed expressions are often 2-3 words:** Idiomatic and meaningful phrases that benefit from n -gram modeling are typically captured within bigrams or trigrams.

Optimal balance at $n = 2$: Given the minor improvements after $n = 2$, bigrams seem to offer the best trade-off between capturing meaningful patterns and avoiding data sparsity. Using higher n -grams doesn't hurt performance, but also doesn't help, which could justify sticking to $n = 2$ for efficiency.

CNN - Neural Network Models



- **Preprocessing & Feature Engineering:**

The input sentences were first cleaned by removing all punctuation and converting the text to lowercase. Tokenization was performed using the NLTK toolkit. Each word was then mapped to a pre-trained GloVe embedding with unknown words assigned zero vectors. Sentences were padded to a fixed length of 30 tokens using zero vectors. To enhance the embeddings, we appended handcrafted features: a flag indicating if a word appears in both sentences, a numeric match flag for numbers that occur in both, and one-hot encoded POS tags derived using NLTK.

- **Convolutional Neural Network (CNN):**

Sentence embeddings were generated using a one-layer 1D CNN with 300 filters, where each filter length matched the dimension of the enriched word vectors. ReLU activation was applied to the convolution output. Max pooling was used to extract salient features from the sequence, producing a fixed-length representation for each sentence. No regularization or dropout was applied, but early stopping was used to avoid overfitting based on validation performance.

- **Semantic Similarity Scoring:**

After obtaining the sentence embeddings, we computed a semantic difference vector by concatenating two components: the element-wise absolute difference and the element-wise (Hadamard) product of the sentence embeddings. This vector effectively captures both contrasting and common features between the sentence pair, making it suitable for similarity estimation.

- **Fully Connected Neural Network (FCNN):**

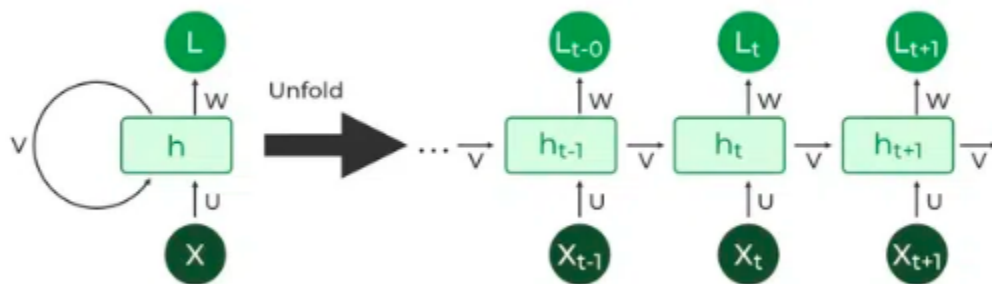
The semantic difference vector (600 dimensions) was passed into a fully connected neural network with two layers. The first layer had 300 units and used the tanh activation function, while the second layer consisted of 6 output units with a softmax activation, providing a probability distribution over the STS similarity labels. No regularization or dropout techniques were employed during training.

Results:

```
X1_train shape: torch.Size([12833, 30, 337])
Epochs:   0%|██████████| 0/10 [00:00<?, ?it/s] Epoch: 1 Train Loss: 0.04950478839090289
Epoch: 1 Validation Loss: 0.04097167384396824
Epochs:  10%|███████| 1/10 [00:35<05:19, 35.49s/it] Epoch: 2 Train Loss: 0.029680807060047762
Epoch: 2 Validation Loss: 0.0420742356735799
Epochs:  20%|███████| 2/10 [01:10<04:41, 35.23s/it] Epoch: 3 Train Loss: 0.022907881450779106
Epoch: 3 Validation Loss: 0.03329779119748208
Epochs:  30%|███████| 3/10 [01:54<04:33, 39.06s/it] Epoch: 4 Train Loss: 0.0190642380748704
Epoch: 4 Validation Loss: 0.03357223007414076
Epochs:  40%|███████| 4/10 [02:29<03:46, 37.77s/it] Epoch: 5 Train Loss: 0.016467753385392882
Epoch: 5 Validation Loss: 0.03365190215926203
Epochs:  50%|███████| 5/10 [03:04<03:02, 36.53s/it] Epoch: 6 Train Loss: 0.013862269397695266
Epoch: 6 Validation Loss: 0.03583112240044607
Epochs:  60%|███████| 6/10 [03:38<02:22, 35.75s/it] Epoch: 7 Train Loss: 0.012280705417923407
Epoch: 7 Validation Loss: 0.03484316682443023
Epochs:  70%|███████| 7/10 [04:21<01:54, 38.17s/it] Epoch: 8 Train Loss: 0.010609422574069962
Epoch: 8 Validation Loss: 0.032870408768455185
Epochs:  80%|███████| 8/10 [04:56<01:14, 37.10s/it] Epoch: 9 Train Loss: 0.009589679153469293
Epoch: 9 Validation Loss: 0.03582658450533119
Epochs:  90%|███████| 9/10 [05:31<00:36, 36.39s/it] Epoch: 10 Train Loss: 0.008974478417887023
Epoch: 10 Validation Loss: 0.04196906886580917
Epochs: 100%|██████████| 10/10 [06:07<00:00, 36.80s/it]
```

Pearson's correlation: 0.7423259183582611

RNN - Neural Network Models



- **Preprocessing & Feature Engineering:**

Text preprocessing followed the same steps as in the CNN approach. Sentences were cleaned by removing punctuation, converting all words to lowercase, and tokenizing using NLTK (Bird et al., 2009). Each token was mapped to a GloVe embedding (Common Crawl, 840B tokens) (Pennington et al., 2014), with unknown words assigned a zero vector. Sentences were padded to a fixed length of 30 tokens (He et al., 2015a). To enrich the embeddings, handcrafted features were appended, including a word match flag, numeric match indicator, and one-hot encoded POS tags.

- **Recurrent Neural Network (RNN):**

Instead of a CNN, a Recurrent Neural Network (RNN) was used to encode sequential information in the sentence. The RNN processed each word in sequence to produce a context-aware embedding for each sentence. The final hidden state was used as the sentence representation, capturing the overall semantic content of the sentence.

- **Semantic Similarity Scoring:**

Once the sentence embeddings were obtained from the RNN, two operations were performed: the embeddings were element-wise **added** to produce one vector (vector 1), and element-wise **multiplied** to produce another (vector 2). These two vectors were then concatenated to form a combined representation that encodes both shared and contrasting semantic features between the sentences.

- **Fully Connected Neural Network (FCNN):**

The combined vector was fed into a two-layer fully connected neural network. The

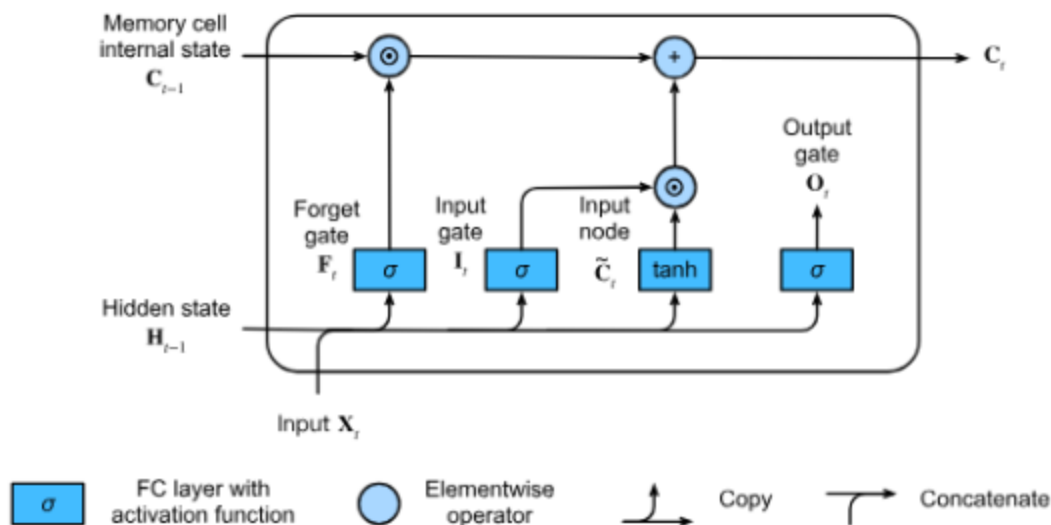
first layer consisted of 300 units with a tanh activation function, while the final layer had 6 output units with a softmax activation to predict a probability distribution over the STS similarity labels. No dropout or regularization was used during training.

Results:

```
Epochs: 0%| | 0/100 [00:00<?, ?it/sEpoch [1/100], Train Loss: 0.0763
Epoch [1/100], Validation Loss: 0.1189
Epochs: 1%| | 1/100 [00:05<09:43, 5.89s/itEpoch [2/100], Train Loss: 0.0697
Epoch [2/100], Validation Loss: 0.0952
Epochs: 2%| | 2/100 [00:11<09:33, 5.85s/itEpoch [3/100], Train Loss: 0.0690
Epoch [3/100], Validation Loss: 0.0935
Epochs: 3%| | 3/100 [00:21<12:01, 7.43s/itEpoch [4/100], Train Loss: 0.0670
Epoch [4/100], Validation Loss: 0.0977
Epochs: 4%| | 4/100 [00:29<12:50, 8.03s/itEpoch [5/100], Train Loss: 0.0523
Epoch [5/100], Validation Loss: 0.0606
Epochs: 5%| | 5/100 [00:38<12:53, 8.14s/itEpoch [6/100], Train Loss: 0.0435
Epoch [6/100], Validation Loss: 0.0653
Epochs: 6%| | 6/100 [00:47<13:11, 8.42s/itEpoch [7/100], Train Loss: 0.0409
Epoch [7/100], Validation Loss: 0.0463
Epochs: 7%| | 7/100 [00:56<13:19, 8.60s/itEpoch [8/100], Train Loss: 0.0378
Epoch [8/100], Validation Loss: 0.0471
Epochs: 8%| | 8/100 [01:03<12:29, 8.15s/itEpoch [9/100], Train Loss: 0.0345
Epoch [9/100], Validation Loss: 0.0534
Epochs: 9%| | 9/100 [01:11<12:29, 8.24s/itEpoch [10/100], Train Loss: 0.0330
Epoch [10/100], Validation Loss: 0.0439
Epochs: 10%| | 10/100 [01:20<12:33, 8.37s/itEpoch [11/100], Train Loss: 0.0328
Epoch [11/100], Validation Loss: 0.0435
Epochs: 11%| | 11/100 [01:28<12:07, 8.18s/itEpoch [12/100], Train Loss: 0.0304
Epoch [12/100], Validation Loss: 0.0417
Epochs: 12%| | 12/100 [01:36<12:03, 8.22s/itEpoch [13/100], Train Loss: 0.0296
```

Pearson's correlation: 0.6666183175554309

LSTM - Neural Network Models



- **Preprocessing & Feature Engineering:**

The preprocessing steps for the LSTM model are identical to those used in the CNN and RNN setups. Sentences are lowercase and cleaned by removing punctuation. Tokenization is carried out using NLTK (Bird et al., 2009). Each token is mapped to a pre-trained GloVe word vector (Common Crawl, 840B tokens) (Pennington et al., 2014), with out-of-vocabulary words replaced by zero vectors. All sequences are padded to a uniform length of 30 tokens (He et al., 2015a). Additionally, handcrafted features such as word overlap flags, numerical match indicators, and one-hot encoded POS tags are concatenated to the word embeddings.

- **Long Short-Term Memory (LSTM):**

To capture long-range dependencies and contextual meaning, a Long Short-Term Memory network (LSTM) was used in place of a vanilla RNN. The LSTM processes each sentence word by word and outputs a final hidden state that serves as a dense, context-rich sentence embedding. Its memory cell mechanism helps retain relevant information across the sentence, making it more robust to varying sentence structures.

- **Semantic Similarity Scoring:**

After generating the two sentence embeddings using LSTM, we compute their

semantic relationship by performing two element-wise operations: addition and multiplication. The resulting vectors — one representing shared features (product) and the other representing cumulative semantic context (sum) — are concatenated into a single vector. This combined representation serves as the semantic difference vector for the sentence pair.

- **Fully Connected Neural Network (FCNN):**

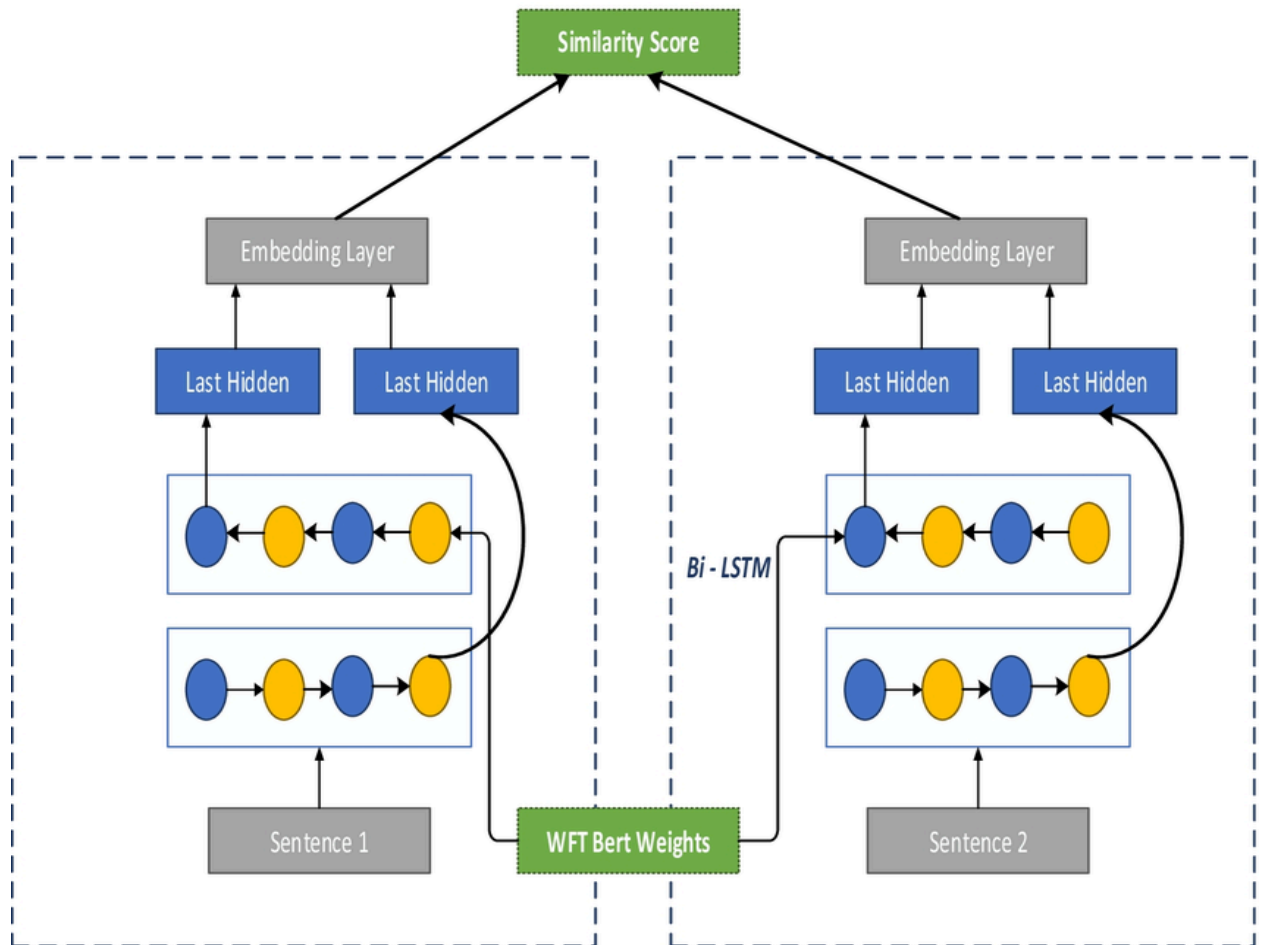
The semantic difference vector is passed into a fully connected neural network with two layers. The first layer has 300 units with a tanh activation, while the output layer consists of 6 units with softmax activation, predicting the similarity score as a probability distribution across six levels. As with previous architectures, the model is trained without dropout or regularization.

Results:

```
Epochs: 0% | 0/100 [00:00<?, ?it/sEpoch [1/100], Train Loss: 0.0762
Epoch [1/100], Validation Loss: 0.0865
Epochs: 1% | 1/100 [00:16<27:28, 16.65s/itEpoch [2/100], Train Loss: 0.0565
Epoch [2/100], Validation Loss: 0.0680
Epochs: 2% | 2/100 [00:37<31:01, 18.99s/itEpoch [3/100], Train Loss: 0.0413
Epoch [3/100], Validation Loss: 0.0511
Epochs: 3% | 3/100 [00:58<32:12, 19.92s/itEpoch [4/100], Train Loss: 0.0336
Epoch [4/100], Validation Loss: 0.0442
Epochs: 4% | 4/100 [01:22<34:34, 21.61s/itEpoch [5/100], Train Loss: 0.0277
Epoch [5/100], Validation Loss: 0.0432
Epochs: 5% | 5/100 [01:46<35:20, 22.32s/itEpoch [6/100], Train Loss: 0.0236
Epoch [6/100], Validation Loss: 0.0415
Epochs: 6% | 6/100 [02:10<35:58, 22.96s/itEpoch [7/100], Train Loss: 0.0205
Epoch [7/100], Validation Loss: 0.0486
Epochs: 7% | 7/100 [02:33<35:47, 23.10s/itEpoch [8/100], Train Loss: 0.0185
Epoch [8/100], Validation Loss: 0.0412
Epochs: 8% | 8/100 [02:58<36:06, 23.55s/itEpoch [9/100], Train Loss: 0.0158
Epoch [9/100], Validation Loss: 0.0417
Epochs: 9% | 9/100 [03:21<35:44, 23.57s/itEpoch [10/100], Train Loss: 0.0135
Epoch [10/100], Validation Loss: 0.0407
Epochs: 10% | 10/100 [03:45<35:38, 23.76s/itEpoch [11/100], Train Loss: 0.0123
Epoch [11/100], Validation Loss: 0.0445
Epochs: 11% | 11/100 [04:09<35:10, 23.71s/itEpoch [12/100], Train Loss: 0.0108
Epoch [12/100], Validation Loss: 0.0413
Epochs: 12% | 12/100 [04:33<34:59, 23.86s/itEpoch [13/100], Train Loss: 0.0102
```

Pearson's correlation: 0.7112907678844448

Siamese Network - BiLSTM



Word Embeddings

- ❖ Embedding Model: Pretrained Google News Word2Vec (300-dimensional vectors).
- ❖ Vocabulary Construction: Built from training sentences, mapping words to indices.
- ❖ Handling OOV Words: Assigned an "unk" token for out-of-vocabulary words.

Model Architecture

The model consists of two identical BiLSTM branches (Siamese structure) with an attention mechanism to weigh important words dynamically.

1. Embedding Layer:

- Initialized with pre-trained Word2Vec embeddings (frozen during training).
- Maps input word indices to 300-dimensional vectors.

2. Bidirectional LSTM (BiLSTM): Hidden Size: 50, Layers: 2

- Processes sequences bidirectionally, capturing contextual dependencies.

3. Attention Mechanism:

- Computes attention weights using a feed-forward layer followed by softmax.
- Produces a weighted sum of BiLSTM outputs, emphasizing relevant words.

4. Similarity Scoring:

- Concatenates the final representations of both sentences.
- Passes through a fully connected layer with sigmoid activation to produce a similarity score in $[0, 1]$.

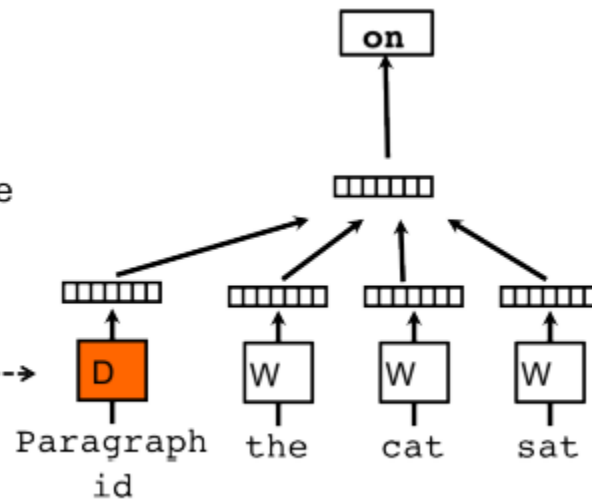
```
Epoch = 0      Training Loss = 0.0023099445907160043
Epoch = 0      Validation Loss = 1.4196899655871675e-06
Epoch = 1      Training Loss = 0.00218376952635891
Epoch = 1      Validation Loss = 8.835387461658684e-07
Epoch = 2      Training Loss = 0.002180990727094291
Epoch = 2      Validation Loss = 0.00015478802379220724
Epoch = 3      Training Loss = 0.0021825124550626965
Epoch = 3      Validation Loss = 1.0324259847038775e-06
Epoch = 4      Training Loss = 0.0021819813249525743
Epoch = 4      Validation Loss = 5.2707004215335473e-05
Epoch = 5      Training Loss = 0.002177644091278232
Epoch = 5      Validation Loss = 5.24942806805484e-05
Epoch = 6      Training Loss = 0.002182439253220819
Epoch = 6      Validation Loss = 6.484144137175463e-07
Epoch = 7      Training Loss = 0.002182599949604528
Epoch = 7      Validation Loss = 9.159506362266256e-07
Epoch = 8      Training Loss = 0.0021794152317392962
Epoch = 8      Validation Loss = 0.0001029971317620948
Epoch = 9      Training Loss = 0.002180796370925275
Epoch = 9      Validation Loss = 1.238501113220991e-06
```

Doc2Vec

Classifier

Average/Concatenate

Paragraph Matrix----->



Model Architecture

The implementation uses **Doc2Vec (Paragraph Vector)** to generate sentence embeddings, followed by three approaches to compute Semantic Textual Similarity (STS) scores:

1. Normalized Cosine Similarity:

$$Score = (1 - cosine(v_1, v_2) + 1) \times 2.5$$

where v_1, v_2 are Doc2Vec embeddings.

2. BiLSTM Regression:

- A bidirectional LSTM processes concatenated embeddings ($v_1 \oplus v_2$) to predict similarity:

$$ht = BiLSTM(xt, ht - 1), Score = FC(ht)$$

- Trained with MSE loss.

2. Key Steps

• Embedding Generation:

- Sentences are tokenized and converted to embeddings using `Doc2Vec` (vector size = 25, window = 6).
Embeddings inferred via `model.infer_vector()`.

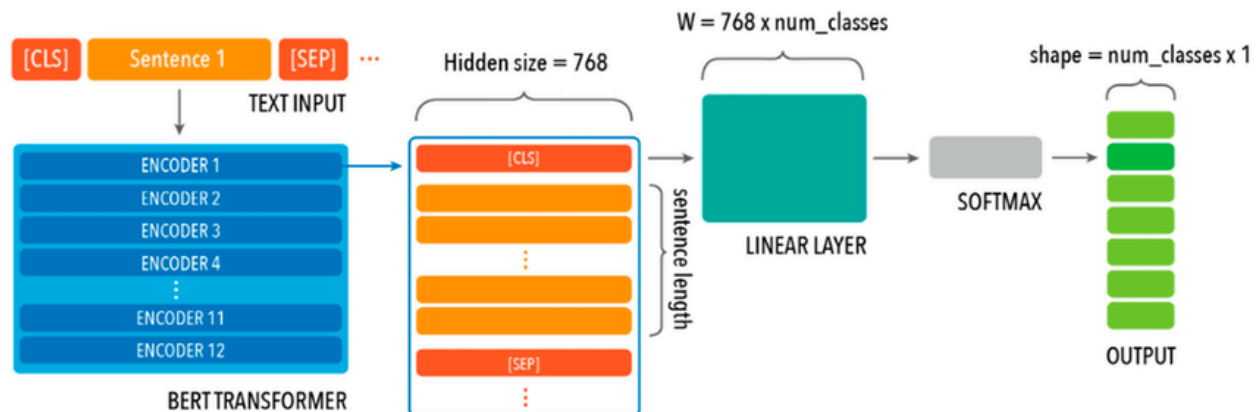
• Training:

- BiLSTM trained for 10 epochs (batch size = 10, LR = 0.001).
Achieved Pearson correlation:
 - Train: **0.82**
 - Val/Test: **~0.40**

| | | |
|-----------|--------------------------------------|---------------------------------------|
| Epoch = 0 | Training Loss = 0.06985956439637371 | Validation Loss = 0.07705373166721653 |
| Epoch = 1 | Training Loss = 0.05266818334216055 | Validation Loss = 0.07576565052727824 |
| Epoch = 2 | Training Loss = 0.0464937524567556 | Validation Loss = 0.07218868434722242 |
| Epoch = 3 | Training Loss = 0.0422714851910711 | Validation Loss = 0.08091851599494947 |
| Epoch = 4 | Training Loss = 0.038915728820697014 | Validation Loss = 0.08016497213263195 |
| Epoch = 5 | Training Loss = 0.03549212360247908 | Validation Loss = 0.08026920217317302 |
| Epoch = 6 | Training Loss = 0.032833071028295406 | Validation Loss = 0.08224525467095363 |
| Epoch = 7 | Training Loss = 0.030669504541752494 | Validation Loss = 0.08439625178177196 |
| Epoch = 8 | Training Loss = 0.028827289421482046 | Validation Loss = 0.08707203664263333 |
| Epoch = 9 | Training Loss = 0.0271283908735017 | Validation Loss = 0.08762856313984815 |

Pearson's correlation: 0.47

BERT



Model Architecture

The model leverages **BERT (Bidirectional Encoder Representations from Transformers)** for sequence classification, fine-tuned to predict continuous similarity scores. Key components include:

- **BERT-Base-Uncased:** Pretrained transformer model with 12 layers, 768 hidden dimensions, and 110M parameters.
- **Fine-Tuning Setup:**
 - Input Processing: Sentences are tokenized and truncated/padded to a maximum length (determined by the 95th percentile of training sentence lengths).
 - Classification Head: A regression layer on top of BERT outputs a single similarity score.
 - Loss Function: **Mean Squared Error (MSE)** for regression-based optimization.
 - Optimizer: **Adam** with a learning rate of **1e-5** and betas **(0.5, 0.99)**.


```
Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"70784f78f5144694aac76d07d1112156","version_major":2,"version_minor":0}

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"c941e9641fcc40a9ac4590b274bb386a","version_major":2,"version_minor":0}

Epoch: 0      Loss: 0.0405695942453924

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"47cd78582a8841c3b4298b2fe872a52a","version_major":2,"version_minor":0}

Epoch: 1      Loss: 0.02640886183090909

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"c0cc5ef937a34b1aaf65f042411b9b8a","version_major":2,"version_minor":0}

Epoch: 2      Loss: 0.02084193215843267

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"b13ec8a3ed1644729ef2df1893bd3486","version_major":2,"version_minor":0}

Epoch: 3      Loss: 0.01721104238838004

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"2b8fcd48c27748a5ba07d3a89f96872f","version_major":2,"version_minor":0}

Epoch: 4      Loss: 0.014451975767381744

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"9923d8d638bf4344a8d9aaf5b3e63c84","version_major":2,"version_minor":0}

Epoch: 5      Loss: 0.012421689305100531

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"2a6bf04a383447d3b1317e60b615d223","version_major":2,"version_minor":0}

Epoch: 6      Loss: 0.010768821553872591

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"6108fdac908247729fb6782eea69d95d","version_major":2,"version_minor":0}

Epoch: 7      Loss: 0.00944712881261419

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"028d7eb6870e47dda874d88fb01de8c6","version_major":2,"version_minor":0}

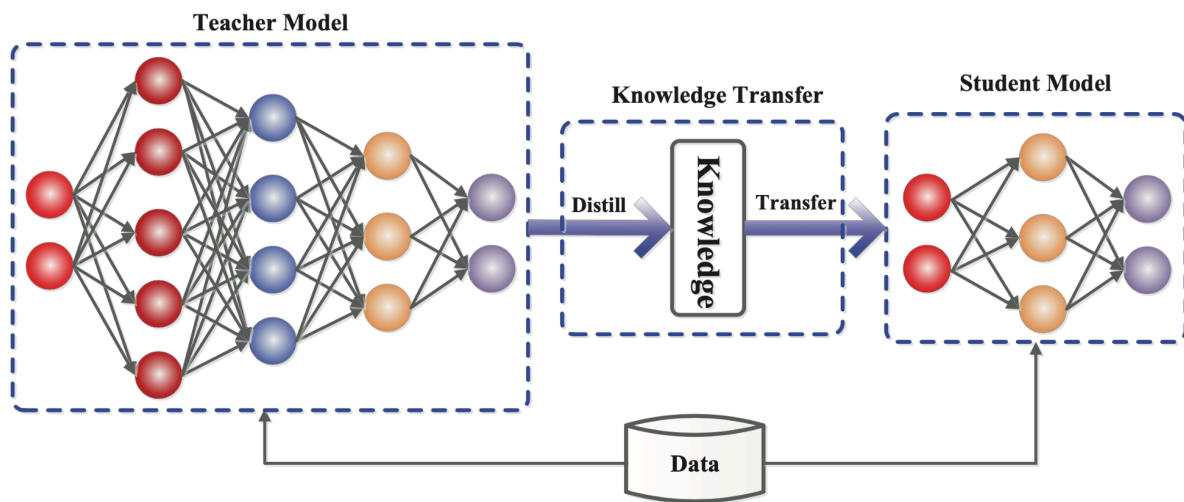
Epoch: 8      Loss: 0.00858722903128899

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"ee7c087e2ea248c1b431ecdd82c1ecba","version_major":2,"version_minor":0}

Epoch: 9      Loss: 0.007735375683671995
```

Pearson's correlation: 0.7902926801810988

Knowledge Distillation



Models used

Teacher model

A large pretrained *BERT* model fine-tuned on the similarity task. BERT provides rich contextual representations but is computationally heavy and slower for inference.

Student model

We used the compact model [microsoft/MiniLM-L12-H384-uncased](#) with sequence classification head (single regression output) as the student.

MiniLM is a **distilled model** designed to be lightweight and fast while retaining much of the teacher's representational power. It has:

- 12 layers
- Hidden size 384

This makes it much smaller and faster compared to BERT.

Loss function

We used a **custom distillation loss** that balances:

1. Hard loss: Mean Squared Error (MSE) between the student predictions and the ground-truth labels.
2. Soft loss: MSE between the student predictions and the soft labels (teacher predictions).

The combined loss:

$$Loss = \alpha * hardLoss + (1 - \alpha) * softLoss$$

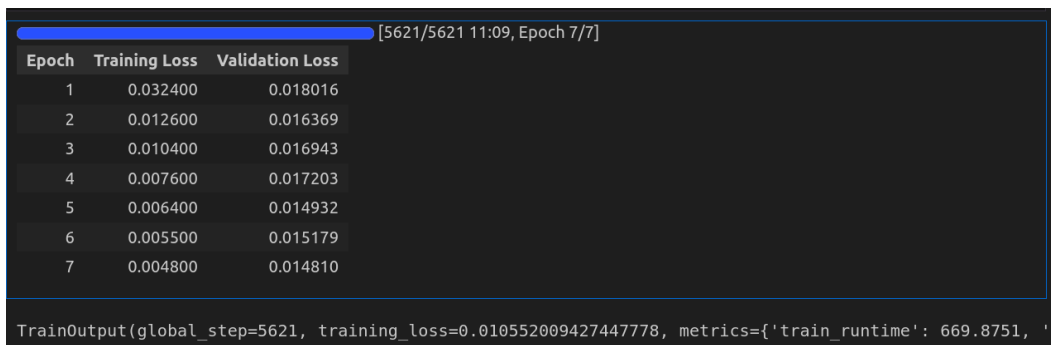
$$hardLoss = \frac{\sum(prediction - ground\ truth)^2}{N}$$

$$softLoss = \frac{\sum(prediction - BERT\ prediction)^2}{N}$$

with $\alpha = 0.5$, giving equal weight to both components. (We have tried and tested other values of α , but **0.5** gave the best results).

Why Distillation Outperforms BERT:

- **Knowledge distillation offers more than just model compression:** It acts as a form of **regularization**. When training a smaller "student" model like MiniLM using the **soft targets** (probability distributions) from a larger "teacher" model (e.g., BERT), the student learns richer relational knowledge rather than merely imitating hard labels. This results in a **smoother loss landscape**, often leading to better generalization on downstream tasks.
- **MiniLM is designed with distillation in mind:** Its lightweight architecture captures the **attention patterns** of larger models, making it both **smaller and smarter**. In tasks like **semantic textual similarity**, which involve smaller datasets and do not require deep linguistic reasoning, such compact models often match or even outperform their larger counterparts.
- **Teacher Bias:** Distillation can help mitigate **teacher bias**. While BERT may retain some noise or overfitting from pretraining, the student model tends to focus on the most relevant patterns, effectively learning a **cleaner, task-optimized representation**.



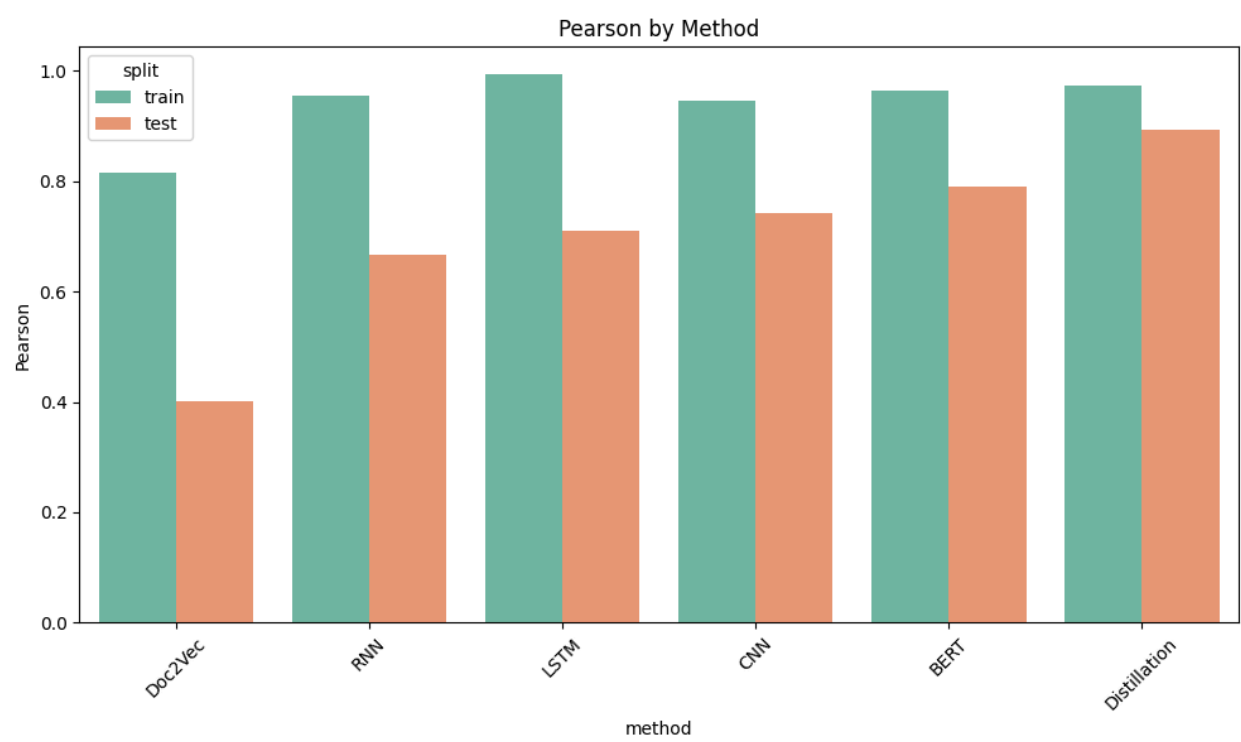
| Epoch | Training Loss | Validation Loss |
|-------|---------------|-----------------|
| 1 | 0.032400 | 0.018016 |
| 2 | 0.012600 | 0.016369 |
| 3 | 0.010400 | 0.016943 |
| 4 | 0.007600 | 0.017203 |
| 5 | 0.006400 | 0.014932 |
| 6 | 0.005500 | 0.015179 |
| 7 | 0.004800 | 0.014810 |

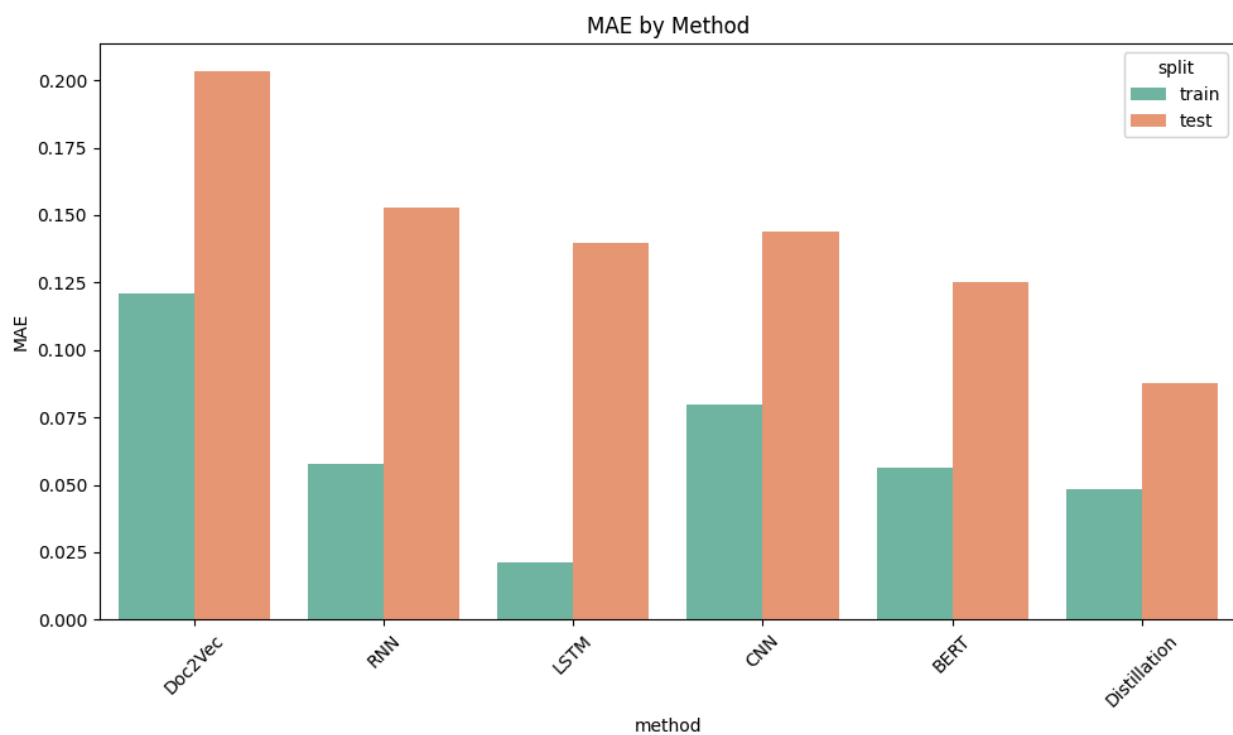
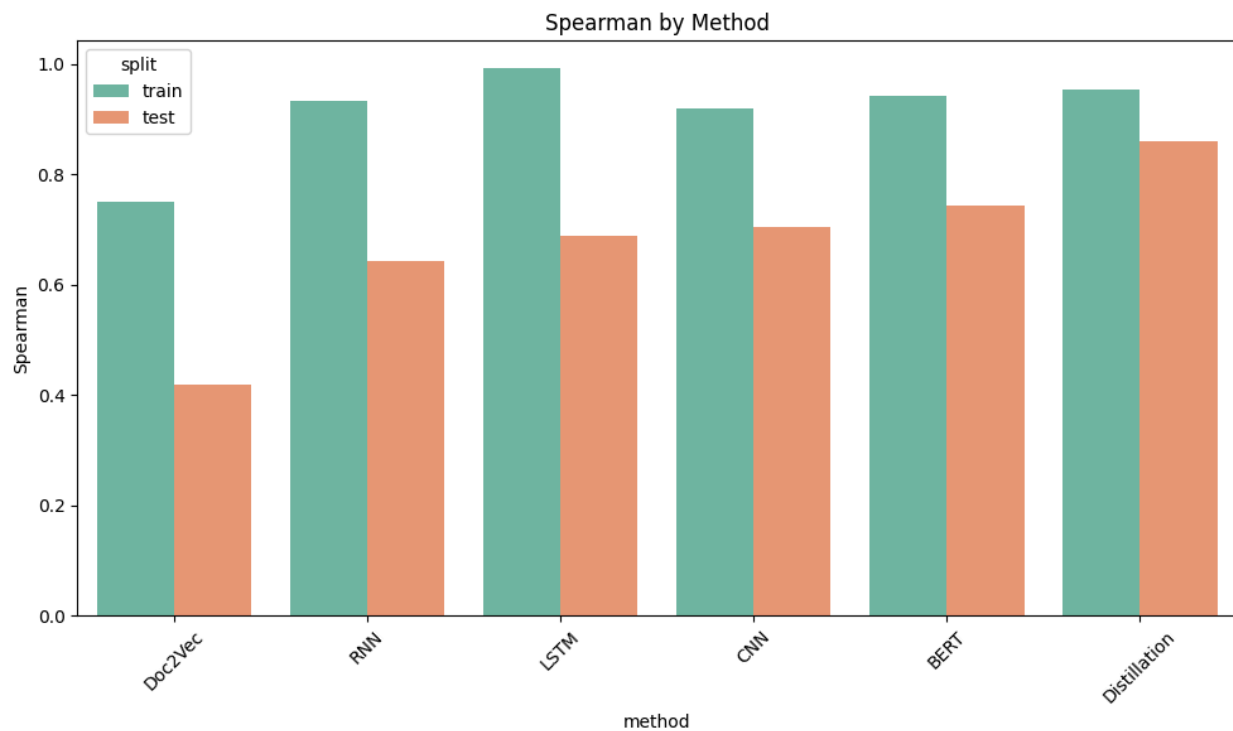
TrainOutput(global_step=5621, training_loss=0.010552009427447778, metrics={'train_runtime': 669.8751, ' '})

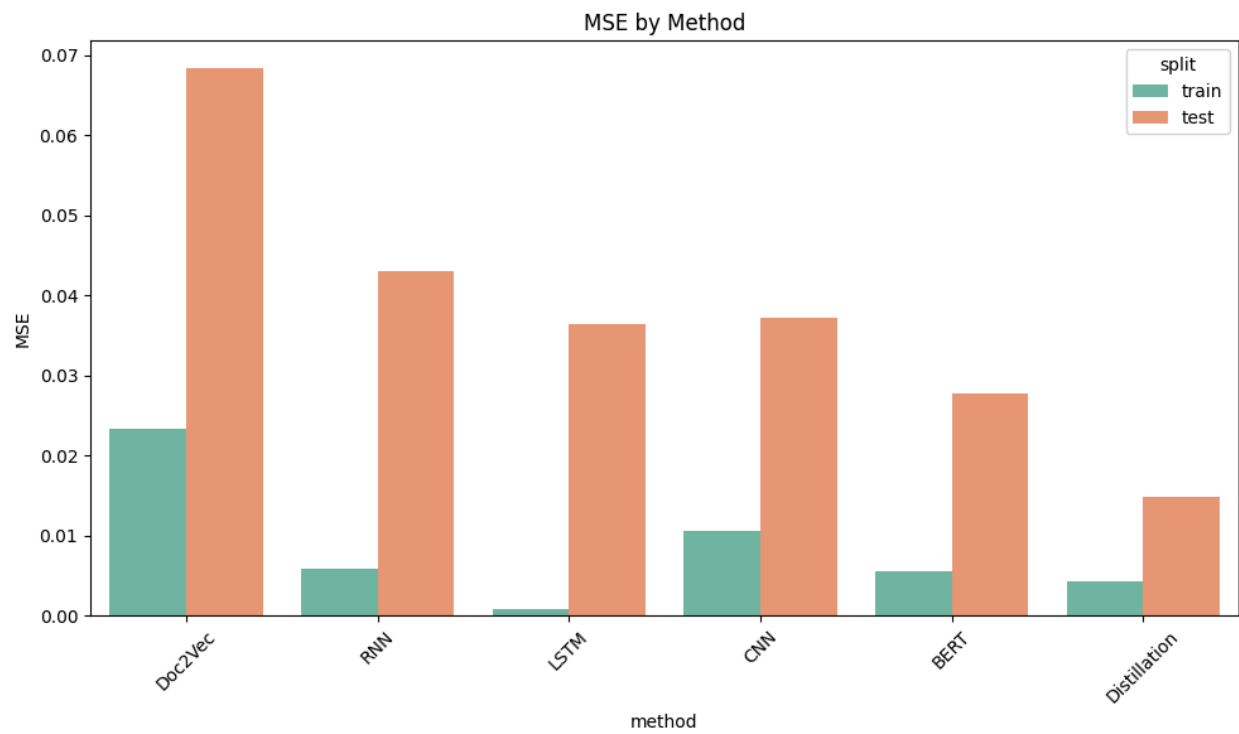
Pearson's correlation: 0.8943392931119134

Results

| method | # train | # test |
|--------------|--------------------|--------------------|
| Doc2Vec | 0.8159071519769091 | 0.401701737067514 |
| RNN | 0.9559883618081264 | 0.6666183175257938 |
| LSTM | 0.9938051201054627 | 0.7112907683050844 |
| CNN | 0.9449980636807448 | 0.7423259190714153 |
| BERT | 0.9635309027564101 | 0.7902926801810992 |
| Distillation | 0.9727508036421509 | 0.8943392931119134 |







Conclusion

- **Neural Network Models vs. Statistical Methods:**

Classical approaches like **N-Gram (N=2)** and **Doc2Vec with BiLSTM** perform poorly on the semantic textual similarity task, as they primarily rely on surface-level token matching or fixed semantic representations. In contrast, neural architectures — **RNN**, **LSTM**, and **CNN** — show considerably higher Pearson correlation coefficients, confirming their superior ability to learn contextual and sequential relationships within text.

- **LSTM: High Capacity for Context Modeling:**

Among the neural methods, **LSTM** achieves the highest correlation on training data. This highlights its capacity to model long-term dependencies and capture sentence structure effectively. However, the slight drop in test performance suggests it may be more prone to overfitting compared to other architectures.

- **CNN: Robust Generalization:**

Although **CNN** slightly trails LSTM in training performance, it surpasses both **RNN** and **LSTM** on the test set. This indicates better generalization capability, possibly due to its localized pattern recognition and reduced sensitivity to sequence length and structure variability.

- **BERT: Strong Performance from Deep Contextual Understanding:**

BERT, a large pretrained transformer model, performs well on both training and test data. Its deep bidirectional attention mechanism captures fine-grained semantic relationships and contextual nuances, making it a reliable but computationally intensive solution.

- **Knowledge Distillation: MiniLM Offers the Best Trade-off:**

The **distilled MiniLM** model outperforms all others on the test data, achieving the **highest Pearson correlation coefficient**. This indicates excellent generalization, despite its significantly reduced model size. The student model benefits from a dual-loss strategy — incorporating both Hard loss and Soft loss

- This dual supervision enables the student model to approximate the behavior of BERT while learning from data-driven corrections, resulting in a model that is both **efficient** and **highly accurate**.

Summary

Distillation strikes the optimal balance, achieving **near BERT-level accuracy** with **faster inference and lower memory usage**, making it the most practical model for real-world deployment in semantic similarity tasks.