Implementation of a WikiQA Framework Using Sequencing and Natural Language Processing

Robbie Baiuo[1], Amir Andakhs[2], and Colin Melville[3]

[1] University of Western Australia, Australia
23028611@student.uwa.edu.au
[2] University of Western Australia, Australia
22839936@student.uwa.edu.au
[3] University of Western Australia, Australia
23170781@student.uwa.edu.au

Abstract. For this project, we present a novel LSTM model with an attention mechanism for binary classification tasks in the context of sequence question-answering (QA), specifically WikiQA. The model incorporates a scaled dot-product, dot-product, and cosine attention mechanisms to effectively capture semantic relationships between input sequences. We employ the Binary Cross Entropy with Logits Loss and the Adam optimizer for training, achieving improved performance by focusing on relevant parts of the sequences. The flexibility of the attention layer to support different attention functions enhances the model's adaptability to various problem domains, demonstrating the potential of our approach for sequence QA tasks.

Keywords: Bi-LSTM · Text Classification · WikiQA.

1 Dataset

For this project, the Microsoft Research WikiQA Corpus was used. The WikiQA corpus includes a set of question and sentence pairs, which is collected and annotated for research on open-domain question answering. The question sources were derived from Bing query logs, and each question is linked to a Wikipedia page that potentially has the answer[1]. The questions and documents in the dataset had qualitative characteristics, potentially leading to explosive ambiguity in potential answers[8]. This ambiguity was further compounded by the presence of questions with multiple suitable answers or no answer at all. Furthermore, certain documents contained multiple questions, adding another layer of complexity to the dataset.

1.1 Dataset Inspection

Two datasets were provided, one for training and one for testing purposes. Both contained the following attributes: QuestionID, Question, DocumentID, DocumentTitle, SentenceID, These raw datasets required analysis to determine their structure for subsequent modeling. Notably, there were redundancies within the rows, where only the Sentence-ID, Sentence, and Label columns differed while the rest of the data remained the same.

The training dataset had dimensions of (20347, 7), and the datatypes of QuestionID, Question, DocumentID, DocumentTitle, SentenceID, and Sentence were all objects; label was of type int64. The Label column had a severe class imbalance, with 19,308 instances labeled as "0" and only 1,039 instances labeled as "1". This imbalance, where the majority class accounted for approximately 94.9% of the dataset, could introduce majority-favoring bias in the model.Although techniques like resampling or cost-sensitive learning can address this issue, they were not implemented in this model due to time constraints. No missing values were present in the dataset. The first few rows of the raw training WikiQA dataset are represented in Figure 1, found in the **Appendix**.

The testing dataset had dimensions of (6116, 7), and the same data types were maintained. Regarding the "Label" column, 5,825 entries were labeled as "0" and 291 entries were labeled as "1". This resulted in an imbalance of approximately 95.2% in favor of the "0" label. No balancing techniques were applied to address this imbalance, despite the potential impact on model accuracy and bias. The dataset did not contain any missing values.

1.2 Data Wrangling

Text wrangling is converting/gathering/extracting formatted text from raw data. We want to extract only valuable information that will be used in our model.

While keeping in mind the suggest wrangling process (see **Appendix**), due to our prior labwork[5], we knew that the real difficulty in this project would come in the form of the natural language model itself, and decided to choose the model type first and then wrangle the raw data into a usable format. Having been instructed to use a sequence model with attention, we elected to build our architecture with two Bidirectional Long Short-Term Memory (BiLSTM) models with attention: one model would take in the Question-related inputting embeddings, and the other would take in the Sentence-related inputting embeddings. These models' outputs would then be concatenated again and further processed to produce a binary classification output. This model will be discussed in detail under **Sequence QA Model**.

This approach represents a departure from conventional methods by training the model solely on sentence correctness rather than focusing on contextual comprehension, enabling a straightforward implementation, computational efficiency, and scalability to accommodate larger corpora. This means that it does not require the suggested data wrangling method, so we developed a different format that included Question, Sentence, and Label, where the sentences are the components of a Document. This would involve the creation of question-sentence pairings, with the correctness of the sentence defined by its Label. [1]

The incorporation of the Label as its own separate column allowed for the model to be trained on whether the sentence was an answer (Label=1) or (Label=0). This training effectively skipped much of the grunt-work in understanding contextual features and reading comprehension[12]. Focusing on training the Label column, rather than on context, allowed for numeric evaluation functions including, but not limited to: precision, recall, and F1-score. These functions facilitated comprehensive model comparison and summative evaluation through the ability to easily determine, present and visualize different aspects of model performance, as was specified[2, 1]. Additionally, as the inputs were single sentences there was no need for EOS or BOS tags[4, 11]. This did not impact our word processing model, which did not require tags to calculate bias. Our wrangled training dataframe can be seen in Figure 2 in **Appendix**.

A 50:50 `'train_test_split'` was performed upon the testing dataframe to obtain a smaller test dataframe and a validation dataframe.

1.3 Preprocessing

Preprocessing plays a crucial role in enhancing the quality and suitability of textual data for subsequent analysis or modeling tasks[5, 10, 11]. In this study, we employed various preprocessing techniques to transform the raw text into a more refined form, adapted from Lab05[5]. This section describes the steps employed and provides a rationale for the selection of specific methods.

To begin, we employed case-folding by converting all text to lowercase. This step helps in achieving consistency by treating words with different cases as identical[10, 5]. For instance, "The" and "the" are treated as the same word after case-folding.

[1] For the purpose of this report, we use the term "Sentence" to describe a component of a document, regardless of label value. In our program, we use the term "Answer". These terms are interchangeable.

We addressed contractions, which are frequently occurring shortened forms of words. To expand these contractions and ensure that they are treated as separate tokens, we utilized a pre-defined dictionary of contractions. This step facilitates a more accurate analysis by considering each word individually, rather than treating contractions as distinct entities[5].

We removed punctuation marks using regular expressions, as punctuation symbols do not typically contribute to the semantic meaning of a sentence, and their removal aids in reducing noise and simplifying later analysis[5].

Tokenization, another critical preprocessing step, involved splitting the text into individual words or tokens. We utilized the `'word_tokenize'` function from the Natural Language Toolkit (NLTK) library to accomplish this task. Tokenization enables the analysis of text at a granular level, allowing for further linguistic processing[4–6, 10].

For the purpose of lemmatization, which involves reducing words to their base or root form, we utilized the WordNetLemmatizer from the NLTK library. By reducing inflected words to their base form, we aimed to consolidate semantically related terms and improve the coherence of the data. This step ensures that words with similar meanings are treated as equivalent during analysis, thus reducing redundancy and enhancing interpretability[5, 10].

The chosen preprocessing methods were based on their effectiveness in preparing the text for subsequent analysis tasks. Case-folding, contraction expansion, punctuation removal, tokenization, and lemmatization collectively contribute to cleaner, more standardized, and semantically consistent textual data[4–6, 10, 11]. These feature extraction functions were applied to all Questions, Sentences, and Labels in our training, testing, and validation sets. Once applied, this new data was loaded into a custom structured dataset for convenience and flexibility, then used in conjunction with DataLoaders to load data in parallel into our model, while also mitigating ordering bias[6].

2 Sequence QA Model

We approached the sequence question-answering (QA) task using a bidirectional Long Short-Term Memory (BiLSTM) model with an attention mechanism. The model is designed to capture the semantic relationships between two input sequences and generate a similarity score. The primary components of the model include an embedding layer, two BiLSTM layers, an attention layer, and a fully connected layer with a sigmoid activation function. This is represented in Figure 3., seen in **Appendix**.

2.1 Input Embedding

Input embedding is crucial in NLP as it converts words or sentences into numerical representations for machine learning models. It captures semantics, contextual information, and relationships, aiding comprehension and interpretation. Embeddings reduce dimensionality, enhance efficiency, and generalize to unseen data, serving as informative features and assist transfer learning[9]. For this project, we were asked to "generate the word vector by using [a] word embedding model and [at least **three**] different types of features"[1].

Word Embedding The word embedding function used combined GloVe and FastText models to provide comprehensive word representations and was based upon the provided code in Lab02 and 05[3, 5]. The function retrieves word vectors from the GloVe model for known words and uses FastText to handle out-of-vocabulary words. GloVe embeddings capture semantic relationships between common words based on global co-occurrence statistics[3]. FastText incorporates subword information and handles out-of-vocabulary words effectively[3]. By combining these models, the function achieves a comprehensive representation of word semantics and addresses challenges posed by rare and unseen words. The output is a list of word vectors, enabling the utilization of semantic information and contextual relationships. This approach enhances the robustness and effectiveness of subsequent natural language processing tasks[3, 5].

The chosen word embedding models were selected because they have a proven track record of strong performance in various natural language processing tasks and are widely used in research[9]. They strike a good balance between capturing semantic relationships, handling out-of-vocabulary words, and offering different vector dimensions to cater to diverse use cases:

- 'glove-wiki-gigaword-100': Trained on Wikipedia and Gigaword corpora, this model has a vector dimensionality of 100. It captures rich semantic relationships between words due to the large and diverse text sources it was trained on[3].
- 'word2vec-google-news-100': Trained on a massive dataset from Google News, this model also has a vector dimensionality of 100. It excels at capturing word meanings and relationships from a wide range of news articles[3].
- 'fasttext-wiki-news-subwords-300': Trained on Wikipedia and other news sources, this model has a larger vector dimensionality of 300. It effectively handles out-of-vocabulary words, captures subword-level information, and handles morphological variations and rare words[3].

Feature Extraction In this study, we employed three feature extraction methods to process and extract features from the given text data. These functions operate cumulatively to facilitate subsequent modeling and analysis. The features chosen were Part-of-Speech (POS), Named Entity Recognition (NER), and Term Frequency-Inverse Document Frequency (TF-IDF).

The `'pos_tagging'` function utilizes the NLTK library to perform part-of-speech (POS) tagging on preprocessed sentences. It assigns a POS tag to each word in the sentence, providing valuable linguistic information such as whether a word is a noun, verb, adjective, etc[6, 11].

The `'NER'` function leverages the Spacy library to perform named entity recognition (NER) on preprocessed sentences. It assigns entity labels to words in the sentence, enabling the identification of named entities such as persons, organizations, locations, etc[6, 11].

The `'tfidf'` function calculates the term frequency-inverse document frequency (TF-IDF) score for each word in the preprocessed sentences. This scoring mechanism measures the importance of a word in a specific sentence relative to its frequency in the entire dataset. The function uses the TF-IDF formula to assign a weight to each word, which reflects its significance in the context of the given sentence[6].

Lastly, the `'concatenation_features'` function combines the outputs of the above functions, including word embeddings, POS tags, NER tags, and TF-IDF scores, to create a comprehensive feature representation for each sentence[6]. These features are concatenated and further processed to generate a sentence vector that captures the collective information from different linguistic aspects.

These functions were chosen based on their ability to extract relevant information and enhance the representation of the input text. POS tagging provides linguistic context, NER identifies important entities, TF-IDF highlights the importance of words, and concatenation combines multiple features to create a rich sentence representation. The integration of these functions aims to capture various linguistic properties, improve the interpretability of the model, and potentially enhance performance[6].

The `'get_preprocessed_tokenized_data'` function preprocesses and tokenizes the text data in a DataFrame. It applies the `'concatenation_features'` function to convert the question and sentence text into feature vectors and extracts the label values. This function facilitates efficient data preparation for subsequent analysis and modeling tasks[6].

2.2 Recurrent Modelling with Attention

In this project, we propose a novel approach to binary classification tasks using an LSTM model with an attention mechanism, specifically designed for sequence question-answering (QA) tasks. The LSTM Model architecture comprises an LSTM layer and attention mechanism that can be either dot product, scaled dot product, or cosine. The model is configured with an input embedding dimension of 301 and an LSTM hidden state dimension of 256. The dimension of 301 is a fixable input embedding dimension, based on the final concatenated value. For example, it would be a dimension of 100 if only

word embedding was selected, 200 for word embedding and POS, 300 for word embedding, POS, and NER, and finally our 301 value is derived from the word embedding, POS, NER, and TF-IDF. The hidden state dimension was left as the standard value. To accelerate computation during training, the model is transferred to the GPU using the `'model.to('cuda')'` command.

The loss function employed in this study is the Binary Cross Entropy with Logits Loss (BCEWithLogitsLoss), which is well-suited for binary classification problems. This loss function combines a sigmoid activation function with binary cross-entropy loss, enabling the calculation of the discrepancy between the model's predictions and the true labels[13]. It takes in the model output and loaded labels during the evaluation period. With each iteration, back-propagation ensures that the loss value is reduced where possible.

For optimization, we utilize the Adam optimizer with a learning rate of 1e-5, a widely adopted choice for training deep learning models, as it dynamically adjusts the learning rate for each parameter during training, resulting in faster convergence and enhanced performance[4].

The proposed Sequence QA Model leverages the power of LSTM networks and attention mechanisms to effectively capture the semantic relationships between input sequences. The attention layer computes attention weights based on the hidden states of the LSTM layer and applies these weights to the LSTM outputs[12, 7], as shown in Figure 3. By incorporating attention, the model can focus on relevant parts of the sequences, leading to improved performance in sequence question-answering tasks[12, 7].The training process is conducted over 20 epochs, iterating through the entire dataset during each epoch.

This architecture is a comprehensive setup for training an LSTM model with attention, employing the Binary Cross Entropy with Logits Loss and the Adam optimizer for an effective binary classification task[13, 6, 12, 7]. The flexibility of the attention layer to support different attention functions further enhances the model's adaptability to various problem domains, making it a promising approach for sequence question-answering tasks.

3 Model Testing

Model testing plays a crucial role in the field of Natural Language Processing (NLP), particularly for architectures such as Bidirectional Long Short-Term Memory (Bi-LSTM) models with attention. It assesses the performance, accuracy, and generalizability of the model on unseen data, ensuring its reliability and suitability for real-world applications. During model testing, the performance of these feature extraction methods is evaluated to determine their impact on the overall model accuracy and effectiveness. This assessment helps to identify the strengths and limitations of each feature and enables researchers to optimize their selection and implementation. By carefully evaluating the contribution of NER, POS, and TF-IDF features in the context of model testing, we can later make informed decisions on how to enhance the representation of linguistic information and improve the model's performance. Furthermore, understanding the impact of feature extraction methods on model testing allows researchers to develop strategies for fine-tuning and optimizing these methods. By iteratively testing different feature combinations and assessing their impact on model performance, we can identify the most informative and effective features for their specific NLP tasks. This iterative process ensures that the chosen feature extraction methods align with our objectives and helps us to enhance the model's ability to accurately process and understand input data.

3.1 Input Embedding Ablation Study

Accuracy measures the proportion of correctly classified instances (both true positives and true negatives) out of the total number of instances, but it can be biased when dealing with imbalanced datasets. Instead, we will use the recall metric, which measures the ability of a model to correctly identify positive instances out of all actual positive instances. In our case of imbalanced data, where the positive instances are fewer, recall can be low as the model may struggle to identify the minority class.

Among the feature combinations tested, the results suggests that the (Word2vec + POS + NER tag) combination using dot product attention yielded the highest recall percentage of 40%. This implies that this particular combination of features and attention mechanism effectively improved the model's ability to identify positive instances. Figure 4 (see **Appendix**) indicates that the added features contribute positively to the model's ability to identify positive instances, which is reflected in the increased recall percentage. However, when we added TF-IDF to the features recall percentage dropped to 37%. This implies that the inclusion of TF-IDF had a negative impact on the model's ability to correctly identify positive instances.

It suggests that the TF-IDF feature may not be informative or relevant for this specific imbalanced dataset, leading to a decrease in recall performance. For future study we recommend to thoroughly analyse and improve the imbalanced data, and perform rigorous evaluation to draw accurate conclusions and justify the observed results.

3.2 Attention Ablation Study

After experimenting with different attention types on the best model feature combination (Word2vec + POS + NER tag), we observed that the Dot Product attention achieved a recall of 40%, whereas the Scaled Dot Product and Cosine attentions resulted in a recall of 37%. The reasons why dot product attention might be giving better results than scale dot product and cosine attention in this specific model can be attributed to a combination of factors:

- Computational properties: Dot product attention is computationally efficient as it involves a simple dot product operation, whereas scale dot product attention requires additional scaling and cosine attention involves computing cosine similarities. This simplicity may also allow the model to converge faster[7].
- Model architecture and task characteristics: The specific architecture of the model can influence the performance of different attention types. The dot product attention might align better with the information flow and dependencies captured by the LSTM layers in this model, leading to improved performance for the given task[6].
- Dataset characteristics: The effectiveness of different attention types can vary depending on the characteristics of the dataset[12]. The dot product attention might be more suitable for capturing relevant patterns or dependencies in the dataset used for training, which could result in better performance in terms of the evaluation metric.

3.3 Hyper Parameter Testing

We conducted an evaluation by testing different learning rates [0.000001, 0.00001, 0.0001, 0.001, 0.01] and analysing the corresponding recall performance of our model. The results we obtained [1, 0.4, 0.08, 0, 0] represent the model's performance at each learning rate:

- A learning rate of 0.000001 resulted in a perfect accuracy of 1, suggesting that the model may be overfitting to the training data.
- At a learning rate of 0.00001, the model achieved an accuracy of 0.4, indicating moderate performance.
- With a learning rate of 0.0001, the model's accuracy dropped to 0.08, indicating poor performance.
- At a learning rate of 0.001, the model did not effectively learn and achieved an accuracy of 0, failing to make correct predictions.
- Similarly, a learning rate of 0.01 resulted in an accuracy of 0, indicating that the learning rate was too high, causing the model to overshoot and fail to converge.

Overall, it appears that learning rates within the range of 0.000001 to 0.0001 are more suitable for the model, with a learning rate of 0.00001 demonstrating the best performance/fit ratio among the ones tested. Please see Figure 5. in **Appendix** for visual aide.

3.4 Findings

After conducting further testing, it was observed that the model started to exhibit signs of overfitting after 30 epochs, as seen in Figure 6 (see **Appendix**). To prevent this, the decision was made to stop training at 20 epochs. At this point, the model achieved a test accuracy of 62% and a test loss of 0.021. However, when considering the Weighted Average metric, which takes into account the class distribution, the performance improved to 73%.

The model's performance in terms of the ROC AUC metric is a concern as it is only 51%. The ROC AUC (Receiver Operating Characteristic Area Under the Curve) is a measure of the model's ability to distinguish between positive and negative instances. A value of 51% indicates that the model's predictive power is close to random guessing, which suggests that it may not effectively differentiate between the two classes.

Limitations to the Model

1. Interpretability: Attention mechanisms improve the model's performance by assigning different weights to different parts of the input sequence. However, the interpretability of these attention weights might be challenging. Understanding which parts of the input are crucial for the model's predictions can be difficult, especially in complex scenarios.
2. Limited modeling capability: The model uses only a BiLSTM architecture without additional layers or complex structures. While BiLSTM can capture some temporal dependencies in the data, it may not be sufficient to capture more intricate patterns and dependencies present in the data. The model's limited capacity might result in sub-optimal performance on complex tasks or datasets.
3. Potential information loss: The model concatenates the outputs of the two LSTMs without explicitly considering the interactions or relationships between them. This approach may result in some loss of information or not fully exploiting the interdependencies between the two sequences.
4. Limited feature representation: The model relies solely on the word embeddings obtained from the linear embedding layer. While word embeddings can capture some semantic information, they may not fully capture the context and nuances of the input[9]. Incorporating additional features or using more advanced techniques like pre-trained embeddings or contextualized word representations could potentially enhance the model's performance[9].
5. Generalization to other datasets: The model's performance might vary across different datasets and label classification tasks. It is crucial to evaluate the model's performance on diverse datasets to understand its generalization capabilities and potential limitations in handling different types of label classifications.

References

1. adlnlp, Han, C., Long, S.: Adlnlp/cits4012_2023: 2023 cits4012 assignment. GitHub (Apr 2023), https://github.com/adlnlp/CITS4012_2023
2. Clark, A., Fox, C., Lappin, S.: The Handbook of Computational Linguistics and Natural Language Processing. Blackwell Handbooks in Linguistics, Wiley (2013), https://books.google.com.au/books?id=zBmom42eWPcC
3. Han, C.: Lab 2: Word embeddings and representation. Lab (March 2023), university of Western Australia
4. Han, C.: Lab 4. sequence model and recurrent neural networks. Lab (April 2023), university of Western Australia
5. Han, C.: Lab 5. sequence models and language fundamental. Lab (April 2023), university of Western Australia
6. Han, C.: Lab 6: Part of speech tagging and n-to-n sequence models. Lab (April 2023), university of Western Australia
7. Han, C.: Lab 7. attention. Lab (May 2023), university of Western Australia
8. Han, C.: Lecture 1: Natural language processing. Lecture (March 2023), university of Western Australia
9. Han, C.: Lecture 2: Word embeddings and representation. Lecture (March 2023), university of Western Australia
10. Han, C.: Lecture 5: Sequence model and language fundamental. Lecture (April 2023), university of Western Australia
11. Han, C.: Lecture 6: Part-of-speech tagging and slot/token tagging. Lecture (May 2023), university of Western Australia
12. Han, C.: Lecture 7: Attention and question answering. Lecture (May 2023), university of Western Australia
13. PyTorchContributors: . BCEWithLogitsLoss PyTorch 2.0 documentation (2023), https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

Appendix

Suggested Wrangling Approach Before performing any transformations of the raw data, we kept in mind that the tasksheet suggested a specific wrangling process for model training, consisting of: Question, Document, and Answer[1]. To construct the document data, the rows with the same DocumentID were concatenated, with the row with "label = 1" being designated the answer - if there were no rows with "label = 1", there was no answer[1].

	QuestionID	Question	DocumentID	DocumentTitle	SentenceID	Sentence	Label
7382	Q1038	when did the cold war start	D1001	Cold War	D1001-0	Photograph of the Berlin Wall taken from the W...	0
7383	Q1038	when did the cold war start	D1001	Cold War	D1001-1	The Wall was built in 1961 to prevent East Ger...	0
7384	Q1038	when did the cold war start	D1001	Cold War	D1001-2	It was an iconic symbol of the Cold War and it...	0
7385	Q1038	when did the cold war start	D1001	Cold War	D1001-3	The Cold War, often dated from 1947 to 1991, w...	1
7386	Q1038	when did the cold war start	D1001	Cold War	D1001-4	This began after the success of their temporar...	0

Fig. 1. Raw WikiQA Training Data

Fig. 2. Wrangled Training Data

	Question	Sentence	Label
0	Who sings the rap song that uses spencer davis...	The song is ranked #247 on the Rolling Stone m...	0
1	what is el mate	Tea-bag type infusions of mate (mate cocido)...	0
2	how many presidential terms did fdr serve	With the bouncy popular song " Happy Days Are ...	0
3	how long does a back waxing last	Strip waxing is accomplished by spreading a wa...	0
4	What are the primary components of globalizati...	Advances in transportation and telecommunicati...	0
5	where is jamestown north carolina	The population was 3,382 at the 2010 census .	0
6	what is colonial americans day in usa	Wars between the French and the British—the Fr...	0
7	how many series in hockey before the stanley cup	The Montreal Canadiens have won the Cup a reco...	0

Fig. 3. Model Architecture (Dual Bi-LSTM with Concatenation)

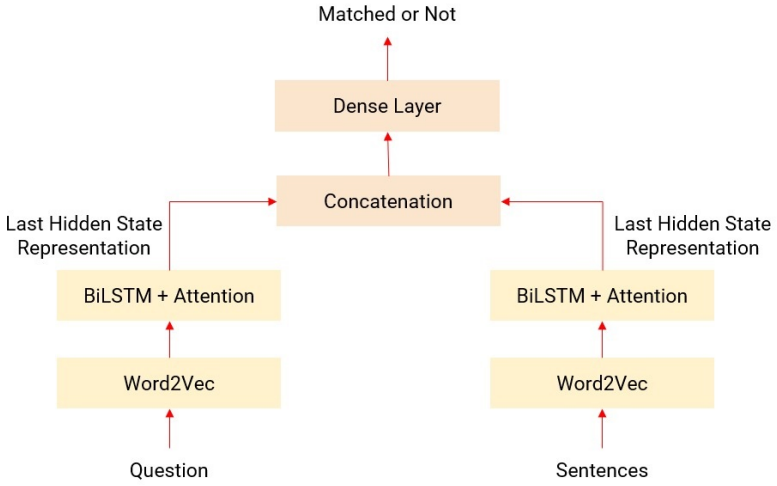

Fig. 4. Recall v Number of Features for Different Models

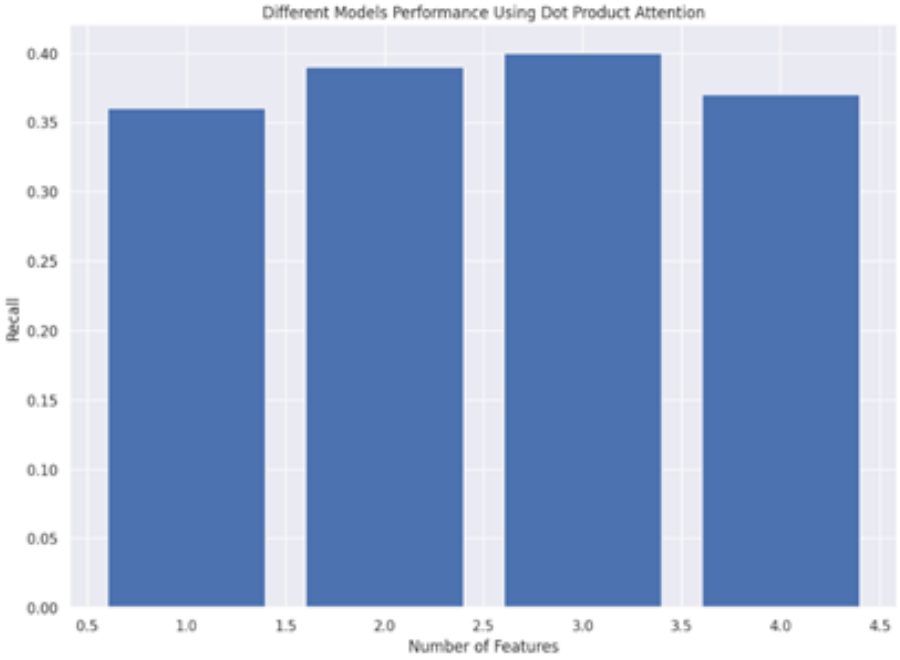

Fig. 5. Recall v Learning Rate for Best Model

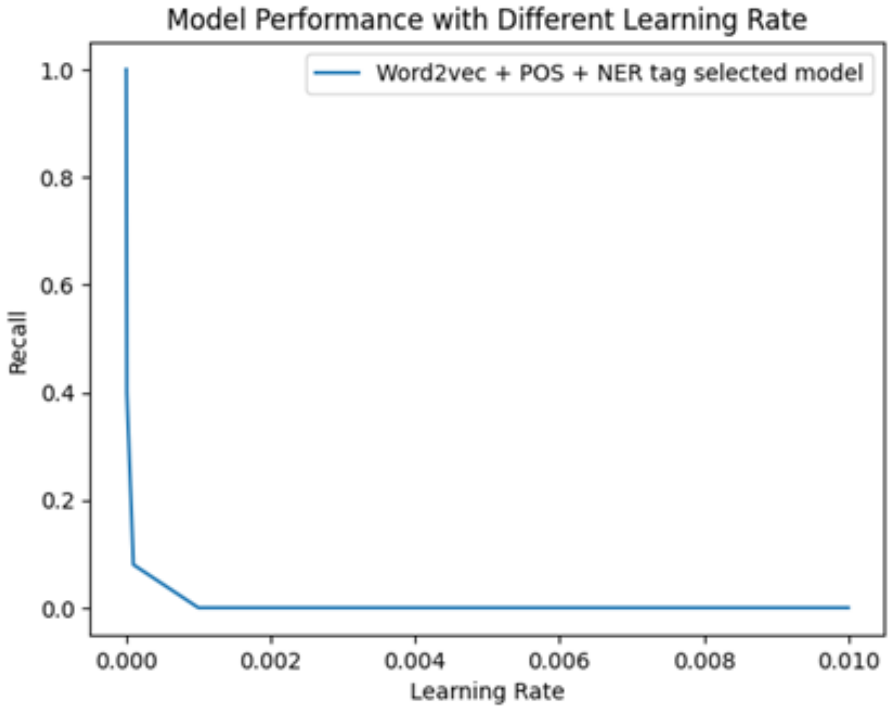

Fig. 6. Training and Validation Loss, Accuracy v Epochs