Comparative Study of Deep Learning Models for Text Classification: LSTM, CNN with GloVe Embedding, and BERT

Hoda Fakharzadehjahromy
(hodfa840)
https://github.com/hodfa840/text-minig-project

Abstract

This project aimed to compare the performance of three deep learning models, namely Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT) for classification of Yahoo Answers dataset. The models were trained and evaluated based on the performance metrics of accuracy, F1-score, and ROC-AUC score. The CNN and LSTM models utilized pre-trained GloVe word embeddings to improve their performance, while the BERT model used its own embedding mechanism. The experimental results demonstrated that the BERT model achieved the best performance with an accuracy of 0.73*, outperforming the CNN and LSTM models. However, it required a significantly longer training time compared to the other models. The LSTM and CNN models achieved comparable performance, with an accuracy of* 0.68 *and* 0.66*, respectively. In conclusion, the BERT model demonstrated superior performance in this task, but at the cost of longer training time, while the LSTM and CNN models could provide a balance between performance and computational efficiency.*

1. Introduction

In recent years, with the rapid growth of digital data, text classification has become a crucial problem in the field of natural language processing (NLP). Deep learning has shown remarkable success in various NLP tasks, including text classification. In this study, we explore the effectiveness of three popular deep learning models, namely Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT), for text classification.

In particular, we use the Yahoo Answers dataset, which consists of over one million questions and their corresponding categories. To obtain vector representations of text data, we use pre-trained word embeddings such as GloVe and BERT embeddings. We then train the models using the vector representations and compare their performance in terms of accuracy, F1 score and ROC AUC score.

Previous studies have shown that deep learning models have outperformed traditional machine learning algorithms in text classification tasks [1]. Moreover, pre-trained embeddings have been found to be effective in improving the performance of deep learning models. For instance, the use of pre-trained BERT embeddings [2] has shown state-of-the-art performance in various NLP tasks.

Therefore, this study aims to compare the effectiveness of different deep learning models for text classification and to investigate the impact of pre-trained embeddings on the performance of these models. Our findings could provide insights for developing more effective text classification models for practical applications.

2. Theory

Deep learning models are known for their ability to learn and extract complex features from raw data. In this study, various deep learning models have been used, such as the CNN with GloVe embeddings, LSTM with GloVe embeddings, and BERT with BERT embeddings. These models have been investigated in this study to evaluate their performance in text classification.

2.1. Convolutional neural networks

Convolutional neural networks (CNNs) have been used for text classification tasks, particularly for tasks that involve sequence-based inputs such as sentences or documents [1]. In CNNs, a filter or kernel slides over the input sequence to generate a feature map, which captures the presence of certain patterns or n-grams in the input. These feature maps are then fed into a pooling layer, which reduces the dimensionality of the input and further extracts important features. The pooling layer also prevents overfitting. Finally, the resulting features are passed through fully connected layers to produce the final classification output.

The convolution operation in CNN can be expressed as follows:

$$y_{i,j} = \sigma(\sum_{k,l} w_{k,l} x_{i+k-1,j+l-1} + b) \quad (1)$$

where x represents the input sequence, w is the filter, b is the bias term, σ is the activation function, i is the index of the output feature map, j is the index of the filter weights, and k is the filter size.

The pooling operation is typically either max pooling or average pooling, and can be expressed as follows:

$$y_{i,j} = \max_{k,l} x_{i+k-1,j+l-1} \quad (2)$$

or

$$y_{i,j} = \frac{1}{n} \sum_{k,l} x_{i+k-1,j+l-1} \quad (3)$$

where n is the size of the pooling kernel.

The fully connected layers at the end of the CNN can be expressed as follows:

$$y = \sigma(Wx + b) \quad (4)$$

where x is the input features, y is the output class probabilities, W is the weight matrix, b is the bias term, and σ is the activation function.

Overall, CNNs have shown to be effective in text classification tasks, especially when combined with pre-trained word embeddings such as GloVe or Word2Vec. An example of a CNN architecture for text classification is shown in Figure 1.

2.2. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that are widely used in natural language processing tasks like text classification. The main advantage of LSTMs over traditional RNNs is their ability to capture long-term dependencies in sequential data, making them particularly effective in tasks that involve analyzing sequences of text [3]. This is because RNNs are vulnerable to the problem of vanishing and exploding gradients as mentioned in [4].

LSTMs operate on individual elements of a sequence, such as words in the context of text classification. In addition to the usual output vector, LSTMs pass the old cell state to the next cell, which is also referred to as the *hidden state*. The next cell's inputs include both the hidden state and the second element of the sequence. By utilizing this hidden state, information can be preserved and transmitted to subsequent cells.

At the core of an LSTM network is the LSTM cell, which is responsible for processing the input sequence and updating the hidden state at each time step. The LSTM cell has several key components: the input gate, forget gate, output gate, cell state, and hidden state. The cell state is responsible for storing long-term information, while the hidden state stores short-term information. The input and forget gates control the flow of information into and out of the cell state, while the output gate controls the flow of information to the output. The forget gate is responsible for deciding how much of the previous cell state should be retained for the current cell. The input gate is responsible for deciding how much new information should be added to the cell state and the output gate is responsible for deciding how much of the cell state should be output at the current time step.

The LSTM cell can be mathematically represented as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (5)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (6)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (7)$$
$$\tilde{c}t = \tanh(W_c x_t + U_c ht - 1 + b_c) \quad (8)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}t \quad (9)$$
$$h_t = o_t \odot \tanh(c_t) \quad (10)$$

where x_t is the input at time t, h_{t-1} is the hidden state at time $t-1$, f_t is the forget gate, i_t is the input gate, o_t is the output gate, $\tilde{c}_t$ is the candidate cell state, c_t is the current cell state, and h_t is the current hidden state. The W and U matrices are the weight matrices, and b is the bias vector. The σ function is the sigmoid function, and $\odot$ denotes element-wise multiplication.

Overall, the LSTM cell is able to selectively forget and retain information from the previous cell state, and add new information to the cell state based on the current input. This allows the LSTM network to learn and remember long-term dependencies in sequential data, making it a powerful tool for text classification tasks.

An example of using LSTM for text classification can be seen in the work of [5]. An unfolded LSTM architecture is illustrated in Figure 2.

Bidirectional LSTM (BiLSTM): is a type of LSTM that processes input sequences in both forward and backward directions, allowing the network to capture contextual information from both past and future time steps. The main advantage of BiLSTMs is their ability to capture long-term dependencies in both directions, making them particularly effective in tasks that involve analyzing sequences of [6].

2.3. GloVe (Global Vectors)

GloVe is a popular word embedding technique used in natural language processing tasks. GloVe embedding is a type of unsupervised learning method that captures the semantic and syntactic information of words based on their co-occurrence in a large corpus of text. GloVe embedding is widely used in tasks such as text classification, sentiment analysis, and machine translation.

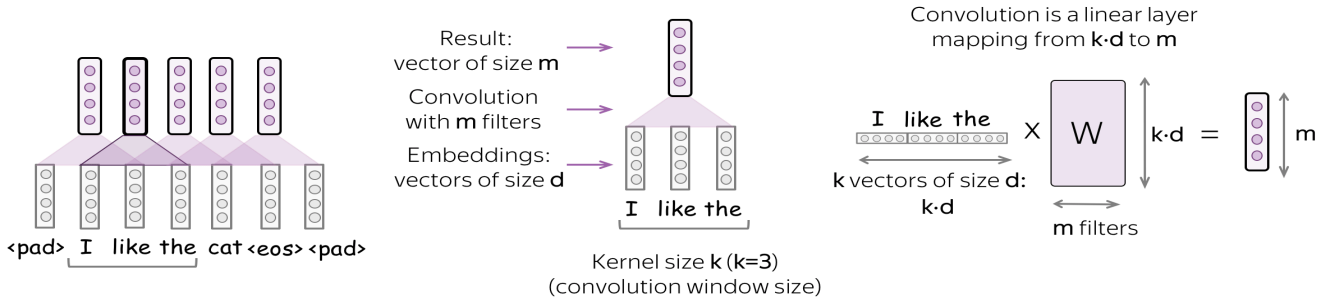


Figure 1. Illustrative Convolutional neural network Source.

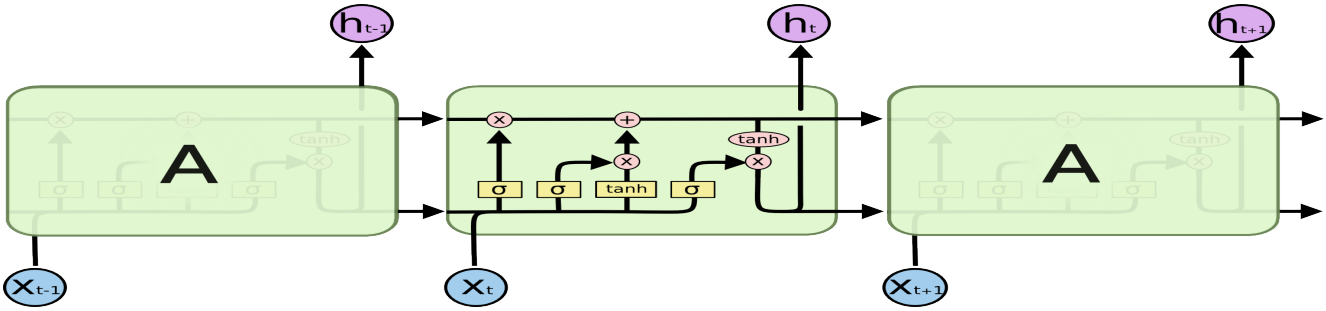


Figure 2. Unfolded LSTM showing the information flow. Source.

At the core of the GloVe algorithm is the co-occurrence matrix, which captures the frequency of word co-occurrence in a corpus. The co-occurrence matrix is then used to calculate the probability distribution of word co-occurrence using the following formula:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}} \tag{11}$$

where P_{ij} is the probability that word j appears in the context of word i, and X_{ij} is the number of times word j appears in the context of word i. The denominator of the formula is the total number of times any word appears in the context of word i. This probability distribution is then used to calculate a weighted least squares objective function:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b_j} - \log(X_{ij}) \right)^2 \tag{12}$$

where V is the vocabulary size, $f(X_{ij})$ is a weighting function that downweights the importance of highly frequent co-occurrences, w_i and $\tilde{w_j}$ are the word vectors for word i and j respectively, and b_i and $\tilde{b_j}$ are the bias terms for words i and j respectively.

Once the GloVe model is trained on a large corpus, the resulting word embeddings can be used in downstream NLP tasks. In this study GloVe embedding has been used withs CNN and LSTM model.

2.4. Bidirectional Encoder Representations from Transformers (BERT)

BERT is a bidirectional transformer encoder used for natural language processing tasks that has achieved state-of-the-art results on several benchmark datasets [2]. It is trained on a large corpus of text using masked language modeling (MLM) and next sentence prediction (NSP). The MLM objective involves predicting masked words based on the surrounding context to learn contextual representations of words for downstream tasks. The NSP objective helps the model learn relationships between sentences and can be useful for tasks like question answering. BERT can handle both sentence-level and token-level tasks with a single model using special tokens [CLS] and [SEP]. It has been successful in a wide range of NLP tasks, including sentiment analysis, question answering, and natural language inference [7].

Figure Figure 3 shows an overview of the BERT architecture.

3. Data

The data set used in this project is The Yahoo! Answers dataset. The Yahoo! Answers dataset is a popular dataset for natural language processing and machine learning research. It contains a large collection of questions and answers posted on the Yahoo! Answers website between 2006 and 2010. The dataset was released by Yahoo! Research

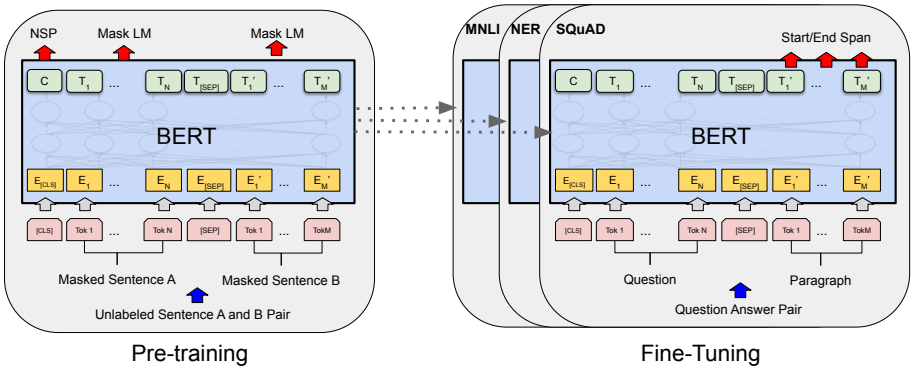


Figure 3. An overview of the BERT architecture: The pre-training and fine-tuning processes of BERT share the same architecture except for the output layers. The pre-trained model parameters are utilized to initialize the models for different downstream tasks. During the fine-tuning process, all the parameters are fine-tuned. Additionally, each input example is pre-pended with a special [CLS] symbol, while a special [SEP] token is used as a separator between questions and answers [2].

and has since been widely used in research projects.

The dataset consists of over 1.4 million questions and answers. It consists of 10 categories; Society & Culture, Science & Mathematics, Health, Education & Reference, Computers & Internet, Sports, Business & Finance, Entertainment & Music, Family & Relationships, and Politics & Government.

The dataset is available here. We downloaded the train set and split it into subsets for training, validation, and testing in the ratio $0.7 : 0.15 : 0.15$, respectively. Each sample in the dataset comprises four columns, namely; title, content, best answer, and class. We have decided to exclude the content column from our analysis and to construct our models using the title and best answer columns as features and the class column as the target variable indicating the category to which each sample belongs.

Figure 4 shows the class distribution for each category.

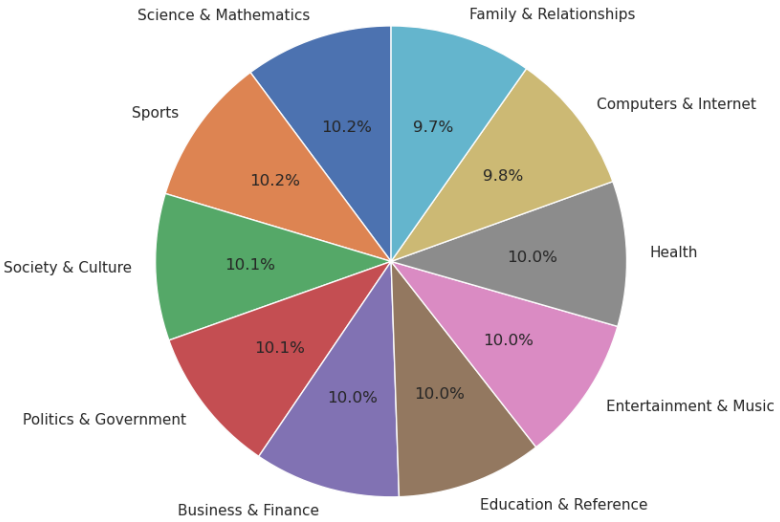


Figure 4. Class distribution.

3.1. Preprocessing

The preprocessing is performed by using the spaCy library [8]. Specifically, each text is tokenized using spaCy's English language model and then lemmatizes each token.

In order to use a CNN and LSTM model for text analysis, the text data is prepared by creating a dictionary with class labels and corresponding question-answer tuples. The next step is encoding, which converts the text data into a numerical format that can be processed by machine learning algorithms. This is achieved by tokenizing the text, mapping each token to a unique integer value, and padding each sequence of integers to a fixed length (maximun length added by 10). The label encoding step converts each class label to a one-hot encoded vector.

After encoding the cleaned text, the next step is to create an embedding matrix for the words in the text using GloVe. Specifically, we use the pre-trained GloVe word embeddings (glove.6B.100d) to create an embedding matrix, which will be used to initialize the embedding layer of the neural network model.

Tokenization for BERT is different from traditional tokenization methods because it uses a technique called WordPiece tokenization. In this method, words are broken down into subwords based on their frequency of occurrence in a large corpus of text. The most common words are kept as single tokens, while less common words are split into multiple subwords.

In this project we will use Ktrain library [9]. Ktrain uses the FullTokenizer from the bert-for-tf2 library to tokenize text for BERT models.

4. Method

These models are designed to be trained on the Yahoo Answers dataset to perform multi-class text classification across 10 categories. The baseline model is a simple model that always predict a specific class in the output.

Firstly, we split the input data into training, validation, and testing sets. For replicating the results the implementation can be found in the corresponding GitHub repository[1].

4.1. CNN

The model is defined using the Keras functional API. The input layer takes in sequences of length 10, which are then passed through an embedding layer to convert them into dense, fixed-length vectors. We initialized the embedding layer with pre-trained word embeddings from GloVe.

The CNN model consists of multiple convolutional layers, each followed by a max pooling layer and a dropout layer with $rate = 0.15$. The convolutional layers apply a set of learnable filters to the input, which extract local features from the text. The max pooling layers downsample the output of the convolutional layers by taking the maximum value over a small window. The dropout layers help to regularize the model and prevent overfitting.

Finally, the flattened output is passed through two dense layers with ReLU and softmax activation functions, respectively, to make the final prediction. We compiled the model using the categorical cross-entropy loss function, Adam optimizer, and accuracy as the evaluation metric.

Categorical cross-entropy is a commonly used loss function in text mining for multi-class classification tasks. It measures the dissimilarity between the predicted probability distribution and the true probability distribution of the class labels. In other words, it quantifies how well the model's output matches the actual labels. The formula for categorical cross-entropy is given by:

$$CE = -\sum_{i=1}^{N} y_i \log(\hat{y_i}) \tag{13}$$

where N is the number of classes, y_i is the true probability distribution of the classes, and $\hat{y_i}$ is the predicted probability distribution of the classes. The logarithmic function ensures that the loss is higher when the predicted probability is further away from the true probability, and lower when they are close to each other. This loss function is commonly used in conjunction with the softmax activation function in the output layer of the neural network, which ensures that the predicted probabilities sum up to 1.

Model diagram is shown in Figure 9a.

[1] Projet link

4.2. LSTM

After the embedding layer, a Bidirectional LSTM layer with 256 units is added, followed by a dropout layer with a rate of 0.4. The recurrent dropout parameter is set to 0.2 to prevent overfitting. Another LSTM layer with 128 units and a dropout layer with a rate of 0.4 are added. Finally, a unidirectional LSTM layer with 64 units and a dropout layer with a rate of 0.2 are added.

Two dense layers with 64 and 10 units, respectively, are added to the end of the model with sigmoid and softmax activation functions, respectively. The model is compiled using categorical cross-entropy loss and the Adam optimizer with accuracy as the evaluation metric.

Additionally, two Keras callbacks are defined to monitor the model's validation accuracy during training. EarlyStopping is used to stop the training process if the validation accuracy does not improve after two epochs. ReduceLROnPlateau is used to reduce the learning rate by a factor of 0.1 if the validation accuracy does not improve after one epoch. The training is stopped after about 12 epochs. The total number of parameters are 29k. The batch size is 1024.

Model diagram is shown in Figure 9b.

4.3. BERT

We uses the ktrain library to preprocess the text data, generating numerical inputs that can be used to train a BERT-based model. This includes tokenization, padding and converting the text into a sequence of integer IDs. Ktrain is a Python library that provides a simple and intuitive interface to work with deep learning models, especially for text classification tasks. It is built on top of the popular deep learning frameworks TensorFlow and Keras and provides high-level abstractions to perform common tasks such as data preprocessing, model training, and deployment. The batch size is chosen 32.

The ktrain library is specifically designed to make it easy to apply transfer learning techniques to NLP tasks using pre-trained language models such as BERT, RoBERTa, and DistilBERT. It includes a set of built-in functions to perform common NLP preprocessing tasks such as tokenization, stemming, stop-word removal, and data augmentation [9].

5. Results

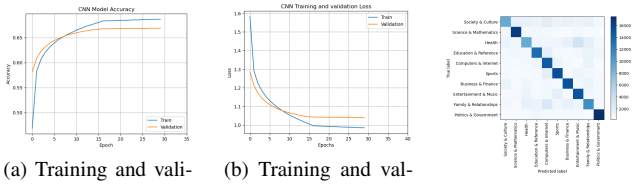
The results of our experiments are presented in Table 1. The CNN model achieved an accuracy of 0.66, an F1 score of 0.66, and a ROC-AUC score of 0.92. The LSTM model achieved an accuracy of 0.68, an F1 score of 0.67, and a ROC-AUC score of 0.93. The BERT model outperformed both CNN and LSTM models, achieving an accuracy of 0.73, an F1 score of 0.73, and a ROC-AUC score of 0.95. The base model, which always predicts one class, has an ac-

curacy of 0.1. In terms of running time, the CNN model was the fastest, completing 40 epochs in 20 minutes. The LSTM model took approximately 1 hour to run for 12 epochs due to the early stopping layer implemented in Keras. The BERT model took the longest, running for 3 hours to complete 3 epochs.

Table 1. Performance Metrics for CNN, LSTM, and BERT

Model	Performance Metrics		
	Accuracy	F1 Score	ROC-AUC Score
CNN	0.66	0.66	0.92
LSTM	0.68	0.67	0.93
BERT	**0.73**	**0.73**	**0.95**

The results of the experiments are presented visually in Figure 5 for CNN, Figure 6 for LSTM and Figure 7 for BERT model. The training and validation accuracy curves for CNN are depicted in Figure 5a, while the training and validation loss curves are shown in Figure 5b. The confusion matrix for CNN is presented in Figure 5c. Similarly, the visualization report for LSTM and BERT models can be found in Figure 6 and Figure 7, respectively. These visual results provide a clear understanding of the performance of each model and help in interpreting the results.

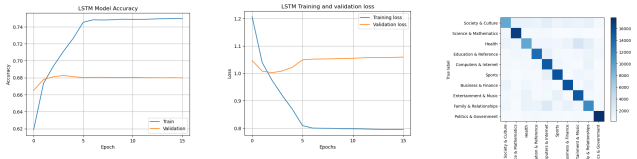


(a) Training and validation accuracy curve for CNN

(b) Training and validation loss curve for CNN

(c) Confusion Matrix for CNN

Figure 5. Visualization Report for CNN



(a) Training and validation accuracy curve for LSTM

(b) Training and validation loss curve for LSTM

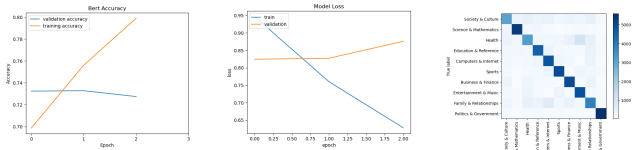
(c) Confusion Matrix for LSTM

Figure 6. Visualization Report for LSTM

The ROC curve for each model is reported in Figure 8.

6. Discussion

The results of our experiments indicate that the BERT-based model outperformed both the CNN and LSTM models in terms of accuracy, F1 score, and ROC-AUC score.

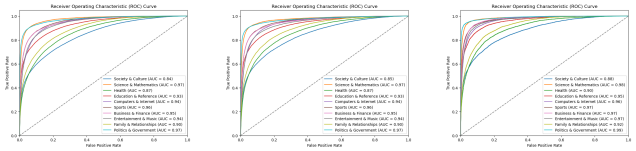


(a) Training and validation accuracy curve for BERT

(b) Training and validation loss curve for BERT

(c) Confusion Matrix for BERT

Figure 7. Visualization Report for BERT



(a) ROC curve for CNN

(b) ROC curve for LSTM

(c) ROC Curve for BERT

Figure 8. ROC curve Results

However, it is important to note that the BERT model also had a significantly longer running time compared to the CNN and LSTM models. The BERT model took three hours to train for just 3 epochs, while the CNN model only took 20 minutes for 40 epochs and the LSTM model took about an hour for 12 epochs. Current state-of-the-art model for Yahoo answer dataset classification is achieved by BERT-ITPT-FiT model [10] and is about 77% accuracy. We achieved 73% accuracy by using a single GPU NVIDIA 3070 RTX. Due to rescorce contraint we dropped the 'content' column. We expect our final model improve by including this feature in the model.

Despite the longer running time, the BERT model had a significant advantage over the other models in terms of the number of parameters it had. The BERT model had 109 million parameters, compared to 29k parameters for both the CNN and LSTM models. This means that the BERT model has a much higher capacity for learning complex patterns in the data.

It is also worth noting that both the CNN and LSTM models used GloVe embeddings, which helped to improve their performance. The GloVe embeddings provided a way to represent words as dense vectors, capturing their semantic meaning and contextual relationships. This allowed the models to learn more meaningful representations of the input data and improve their ability to classify the samples.

In terms of the trade-off between running time and accuracy, the CNN and LSTM models provided a reasonable compromise. Although they did not achieve the same level of accuracy as the BERT model, they still achieved respectable accuracy scores while requiring significantly less time to train. This is especially important for applications where real-time or near-real-time processing is required.

In summary, our experiments demonstrate that transfer learning with pre-trained language models such as BERT can provide significant improvements in the accuracy of text classification tasks. However, the longer running time and higher resource requirements of these models may not always be feasible, especially for applications with strict real-time requirements. The CNN and LSTM models, when using GloVe embeddings, can provide a reasonable alternative, achieving reasonable accuracy while requiring significantly less time to train.

7. Conclusion and Future work

In conclusion, this project focused on the use of deep learning models for text classification tasks on the Yahoo Answers dataset. We implemented three popular deep learning models: CNN, LSTM, and BERT, and compared their performance on the dataset.

Our experiments showed that the BERT model outperformed the other models in terms of accuracy, F1 score, and ROC-AUC score. This is not surprising, as BERT is a powerful pre-trained language model that has achieved state-of-the-art results on various NLP tasks. However, it should be noted that BERT required a much longer training time compared to the other models, which is a trade-off between accuracy and training time.

Both the LSTM and CNN models used GloVe word embeddings, which helped to improve their performance. We used the pre-trained 100-dimensional GloVe embeddings provided by the Stanford NLP Group, which captures semantic relationships between words and is widely used for various NLP tasks. On the other hand, for the BERT model, we used the pre-trained BERT embeddings, which were learned during pre-training and are specific to the BERT model.

Overall, this project demonstrates the effectiveness of deep learning models for text classification tasks and highlights the importance of choosing appropriate pre-trained language models and embeddings. It is worth noting that deep learning models can be computationally expensive and may require substantial computational resources for training. However, with the increasing availability of pre-trained models and libraries such as ktrain, it has become more accessible for researchers and practitioners to apply deep learning techniques to NLP tasks.

It is worth noting that the base model, which always predicts one class, has an accuracy of 0.1. Hence, our models' accuracies are significantly higher than the baseline model, indicating that the models indeed learned some valuable information from the input data.

As a final note, we believe that this project can serve as a starting point for further research in the field of text mining and deep learning. There are various avenues for future work, such as exploring other pre-trained language models, experimenting with different hyperparameters and architectures, and testing the models on other datasets. We can also consider the 'content' column as another feature and source of information to improve model. With the ever-growing amount of data and the rapid advancements in deep learning, there is no doubt that the potential for deep learning in text mining is vast and exciting.

References

[1] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015. 1

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. 1, 3, 4

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. 2

[4] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012. 2

[5] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 207–212, 2016. 2

[6] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018. 2

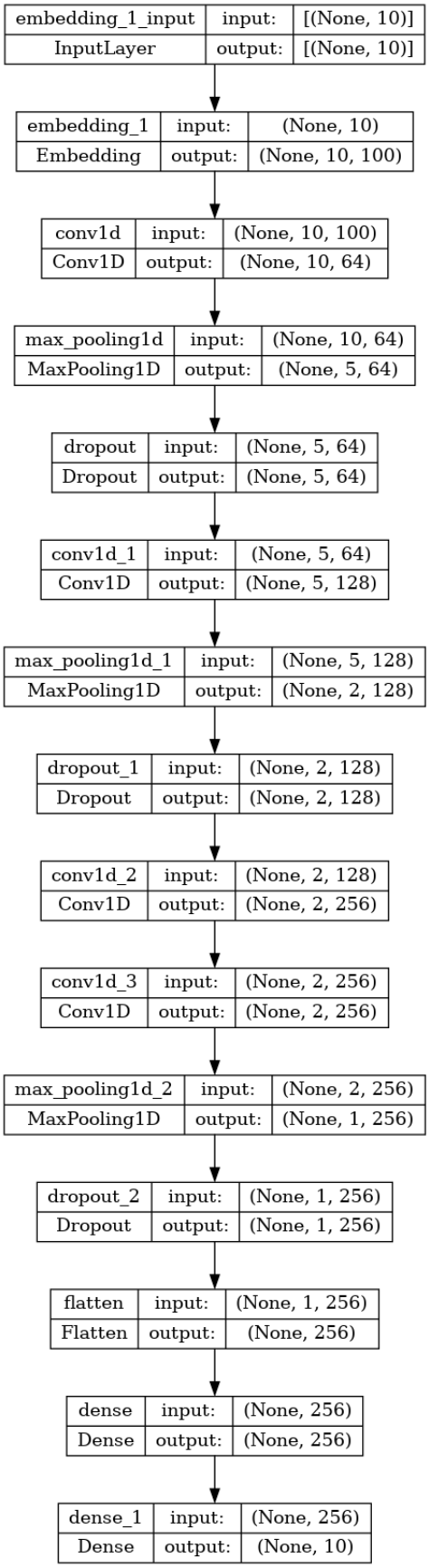
[7] Songwen Pei, Lulu Wang, Tianma Shen, and Zhong Ning. Da-bert: Enhancing part-of-speech tagging of aspect sentiment analysis using bert. In *Advanced Parallel Processing Technologies: 13th International Symposium, APPT 2019, Tianjin, China, August 15–16, 2019, Proceedings 13*, pages 86–95. Springer, 2019. 3

[8] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017. 4

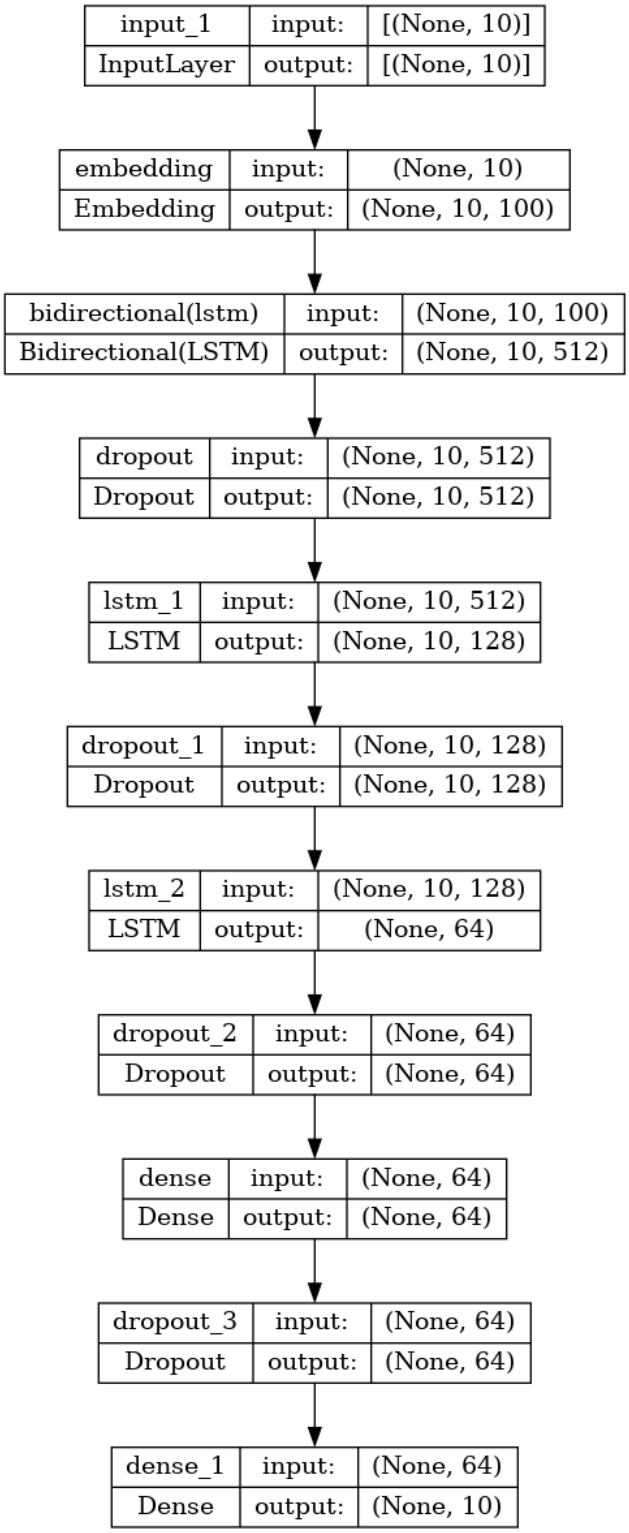
[9] Arun Ammar. ktrain: A python library for deep learning with keras. https://github.com/

amaiya/ktrain, 2021. Accessed: March 13, 2023. 4, 5

[10] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019. 6



(a) CNN Diagram



(b) LSTM Diagram

Figure 9. Model architecture