How to Apply Neural Network in Evidence Retrieval and Label Classification

Tue11AM Group6

Abstract

This project aimed to develop a deep learning model that could retrieve evidence and classify popular claims related to climate change. We tried cosine similarity with TF-IDF as the baseline model, as well as LSTM and Transformer. By comparing performance, we decided to use Transformer for retrieval and classification. Transformer had an attention mechanism and parallel processing capabilities that could understand contextual information and was very suitable for this task. Our final design retrieved 5 pieces of evidence per claim and assigned a label to the claim, resulting in an f-score of 0.096 for evidence retrieval and a classification accuracy of 0.42 on test dataset.

1 Introduction

The climate change information deluge has had a profound impact on public opinion, leading to widespread misinformation and rumors. While fact-checking is critical to maintaining information integrity, existing methods often rely on time-consuming manual processes that are inadequate when dealing with growing data volumes (Atanasova, 2024). This project proposes an automated fact-checking system based on a deep learning model, aiming to improve the speed and accuracy of information verification through efficient evidence retrieval and claim classification.

The first part is evidence retrieval: using the three models of cosine similarity, Siamese LSTM, and Transformer after preprocessing to filter out the evidence most relevant to the given text content from large data sources. Cosine similarity was used for baseline retrieval, locating the 5 most similar evidences and comparing them with the f-score of the evidence retrieved by Siamese LSTM and Transformer. The second part is claim classification: using zeroR as the baseline, Transformer was used to further improve the accuracy of classification. The system took text and associated evidence as input and predicts a label for the text (supporting, refuting, insufficient information, or controversial).

We finally used local comparison and codalab competition to evaluate the final results. We finally used Transformer to predict two parts.

2 Approach

2.1 Evidence Retrieval

2.1.1 Cosine Similarity (Baseline)

Following the methodology of Iqbal et al. 2021, the baseline model was designed through the following steps: (i) lemmatizing raw data and removing stop words; (ii) converting both claim and evidence texts into vector embeddings using TF-IDF (Term Frequency-Inverse Document Frequency); (iii) calculating the cosine similarity between the claim and evidence embeddings; and (iv) selecting the top-K evidence from the cosine similarity list for each claim.

TF-IDF was a statistical measure used to evaluate the importance of a word in a document relative to a corpus. It captured the relevance of terms within the documents, facilitating more effective similarity comparisons. This vectorization was crucial for handling large datasets and preserving the significance of term frequency across the entire corpus, thereby enhancing NLP tasks such as query retrieval (Abubakar et al., 2022).

To determine the degree of similarity, we computed the similarity scores between each embedded claim and all embedded evidence by performing the dot product between these two normalized matrices, as shown in the formula below (Gao et al., 2021). This method was advantageous as it preserved the directionality and context of the vectors, making it robust for text similarity tasks (Iqbal et al., 2021).

$$sim(c, e) = \cos(e_c, e_e) = \frac{e_c}{||e_c||} \cdot \frac{e_e}{||e_e||} \quad (1)$$

We selected top five evidences that reaching the highest similarity score.

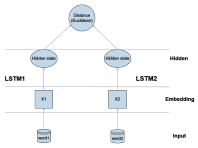
2.1.2 Customized Tokenization

Advanced models such as LSTM and Transformer architectures required special tokens for embedding purposes. Inspired by the tokenization strategies in BERT, we added <CLS>, <PAD>, <SEP>, and <UNK> to the vocabulary to handle specific aspects in generating the query input. This ensured that the text data was converted into normalized indices that the model could understand and process efficiently (Nayak et al., 2020).

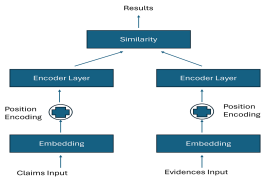
For the tokenization process, we used preprocessed data with stop-words removed as input for the retrieval task since stop-words could introduce noise. Conversely, we used unprocessed text as input for the label classification task, because removing stop-words could compromise the semantic analysis critical for determining labels.

2.1.3 Siamese LSTM

In order to study further evidence analysis of deep learning models, the first method tried is Siamese Long short-term memory (Siamese LSTM) to predict the relevant evidences of the text. As numerous studies and courses have shown, LSTM is a powerful type of RNN used in deep learning, capable of capturing long-term dependencies and modeling sequence data, and is particularly suitable for processing and predicting important events in time series data(Othman et al., 2019; Nammous and Saeed, 2019). When comparing the similarity between two pieces of text, Siamese LSTM makes it easier to compare the semantic similarity between text and evidences. Siamese LSTM contains an embedding layer and an LSTM layer.



(a) Siamese LSTM



(b) Transformer Retrieval

During training, Siamese LSTM was used negative sampling to enhance the model's ability to learn to distinguish between similar (1) and dissimilar (0) texts. When setting up Siamese LSTM, two LSTM networks were used, each network processes one sentence in the input sentence pair. And in Siamese architecture, both LSTMs shared the same weights. The model would input two inputs each time for learning.

During the training process, text pairs were packed into batches, each batch contains multiple pairs of texts, each pair of texts was processed through the Siamese LSTM to calculate the distance, and then the loss was calculated based on these distances and labels. The Siamese LSTM of this project used Contrastive Loss(see the equation 2) for training. The loss calculation was adjusted based on the distance, and if two samples were similar, their outputs were pushed closer together. Finally, the trained model made predictions based on the corresponding inputs and found the nearest evidences as output objects.

$$(1-Y)\,\frac{1}{2}\,(D_w)^2 + (Y)\,\frac{1}{2}\,\{max\,(0, m - D_w)\}^2 \tag{2}$$

2.1.4 Transformer

A popular way to express the contextual meaning of a sentence was to use the <CLS> token added in front of every input (Devlin et al., 2018). Each token integrated the information from other positions in each encoder layer, and so did the <CLS>. Using the <CLS> token for representation, rather than considering the tokens of each input to calculate similarity, saved a lot of construction and computational costs.

The model transformer required that sentence inputs be converted into embedding inputs and that the inputs be of the same length for computational purposes. As mentioned earlier, <pad> was added to pad the length where the sentence was stop-word removed because the key terms in the sentence were more important in this task. Another key feature was the multi-head attention mechanism. The purpose of single attention was to map queries and key-value pairs and calculate the weighted sum to represent the relative importance of each token. So, multi-head attention worked by running attention in parallel and considering different subspaces at different positions (Vaswani et al., 2017). A mask was introduced in our transformer model to cover those <pad> tokens, to avoid self-attention learning from the irrelevant contents.

Since Transformer required positional encoding to access contextual information of the input query, the periodic position encoding was included in the model (Vaswani et al., 2017).

The similarity results between each embedded claim and embedded evidence were normalized using the log softmax function, and the temperature parameter was utilized to sharpen the results (Guo

et al., 2017). Then, using the results of the correct samples in each claim as the loss, they were accumulated and passed backward with the SGD optimizer to update the parameters of the model.

$$\text{LogSoftmax}(x_i) = \log\left(\frac{e^{x_i/T}}{\sum_j e^{x_j/T}}\right) \quad (3)$$

In Equation 3, x is the similarity score, and T is the temperature parameter uses to scale the input.

2.2 Claim Classification

Unlike the transformer used in the Evidence Retrieval task, there were some differences in both data input and model architectures for the classification task. For instance, the input data did not undergo stopword removal because, in classification, certain stopwords like 'not' could completely reverse the meaning of the evidence, which might have negatively impacted the classification results. To allow the model to comprehensively classify, we decided to merge the claim text and 5 retrieved evidence. Before merging, we used padding techniques to ensure that the length of each claim and evidence was 50, in line with the transformer model's requirement for consistent input lengths. The final input accepted by the model looked like this:

<cls>Claim<sep>Evidence<sep>Evidence

The transformer model processed the input as a sequence of tokens and outputted a classification score list containing four values. These scores indicated the labels that the model predicted with the highest confidence.

The best classification model utilized the encoder layers of a transformer, complemented by a fully connected linear layer on top of the encoder. Based on the input from the linear layer, the softmax function was used as activation to determine which label had the highest probability.

The model utilized cross entropy as its loss function and the Adam optimizer to allow for an adaptive learning rate.

$$H(p, q) = -\sum_x p(x) \log q(x) \quad (4)$$

In the Equation 4, $p(x)$ is the true probability distribution, and $q(x)$ is the estimated probability distribution.

3 Experiments

3.1 Evidence Retrieval Baseline Evaluation

The main model configurations for evidence retrieval included determining the optimal Top-K and deciding whether to remove stop-words. The optimal performance for Top-K is detailed in the results section.

We observed that the cosine similarity results by TF-IDF embedding achieved higher accuracy when stop-words were removed. This improvement can be attributed to noise reduction and enhanced term weighting, which improve document representation. Consequently, the model focuses on meaningful words that better reflect the text's content.

However, for label classification, we retained stop-words because negation words like 'not,' 'no,' and 'never' are critical for determining the sentiment of a sentence. These words can reverse the sentiment polarity, and retaining them ensures that the model accurately interprets sentences with negation, leading to more precise sentiment classification. This approach is validated not only by our own accuracy improvements but also by experimental data in the Twitter Sentiment Study article (Saif et al., 2014).

3.2 Evidence Retrieval with Siamese LSTM Evaluation

The experimental details of Siamese LSTM mainly include the input and distance calculation requirements of the training process and prediction process. when training, both input will be converted into index. Therefore, for the training of Siamese LSTM, Negative sampling extracted files from 'evidence.json' file and marked them as 0, while the corresponding evidence of each text in the 'train-claim.json' file was marked as 1. Each text would have ten evidences as support provided to Siamese LSTM for learning. In order to avoid error forward, we avoided using cosine similarity and use Euclidean distance to calculate the distance between the hidden states of two inputs. The trained Siamese LSTM used cosine similarity to find the 100 most similar evidences for each text in the 'dev-claim.json' file as input for prediction. The model would calculate the first 5 closest evidences as output.

When evaluating, we used a local test script to detect f-scores using different evidence predicted by development texts for Siamese LSTM.

3.3 Evidence Retrieval with Transformer Evaluation

In training the Transformer model for retrieval, a custom dataset and dataLoader were used to batch the training input. Model parameters were updated by evaluating the loss in each batch, and this process was repeated for several epochs to obtain a trained model. The performance of the model was evaluated by the average loss per 20 batches in the training dataset. Every 100 batches, the model predicted the evidences for the development dataset and evaluated the f-score to find the set of parameters achieving the best performance on the development dataset.

In the final designed model, hyper-parameters included a dimension and hidden units set to 1024, with 6 encoder layers and 8 attention heads. Training was conducted for two epochs using the entire training dataset because there was no improvement in f-score performance on the development dataset, as the model seemed to over-fit the training dataset after two epochs.

3.4 Classification Evaluation

In this experiment, the primary evaluation method involved a 5-fold cross-validation technique to assess the performance of a Transformer-based classification model. Quantitative metrics used included the average training and validation loss, as well as the accuracy of the validation subsets. The model was trained and validated on each fold using DataLoader and a customized collate function for batch handling. The final evaluation of the model was conducted using scores from both the training dataset and development set to ensure the model was robust.

In our experimental setup, we systematically explored various model configurations, including changes in embedding dimensions, the number of attention heads and etc. and adjustments in the optimizer's learning rate. Each experimental run was carefully designed to test the impact of one specific parameter alone while keeping all other variables remained constant. This approach allowed us to directly observe the effects of each configuration change on the model's performance. results from experiments that had the most significant impacts on the model performance were thoroughly detailed in the Results section, providing clear insights into the settings of the optimal model.

4 Result

4.1 Evidence Retrieval with Preprocessing

To study the effect of stopwords in data preprocessing within the same top-K setting of the baseline model, we compared the F-scores. The F-score for stopword removal in preprocessing is 0.088, which is higher than the F-score for retaining stopwords, which is 0.079.

4.2 Evidence Retrieval Model Comparison

After using preprocessing with stopwords, we tried three models (cosine similarity, Siamese LSTM, Transformer) to predict the evidence retrieval of dev claims (Figure 2 and table 1).

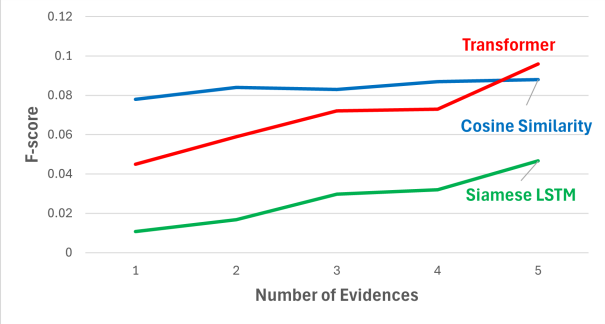


Figure 2: Accuracy of three models in predicting from one to five evidences in dev results

Models	Evidence Num	dev f-score
Cosine Similarity	5	0.088
Siamese LSTM	5	0.0466
Transformer	5	0.096

Table 1: Best dev results in different models

The f-score of all models increases as the number of evidence increases, and the local Transformer achieves the best performance, with F = 0.096. However, it did not meet our expectations for this model. This may be due to the limitations of the Transformer when dealing with highly diverse text data, especially when the length and complexity of the evidence text vary greatly. The performance of Siamese LSTM is relatively stable, but the performance of f-scores for predicting different numbers of evidences is significantly lower than Transformer. The performance of Siamese LSTM is lower than expected (F=0.0466). This is mainly due to insufficient capture of long-distance dependencies, and we only used Euclidean distance for

operation(Othman et al., 2019). The overall performance of Cosine Similarity is relatively stable, but as shown in the Table 1, the best local prediction for evidences is still not higher than that of transformer.

Because Transformer performs better locally, we submitted it to Codalab. The competition results of Transformer did not meet expectations. The ongoing result of 0.0899 is slightly smaller than the local result, which shows that the model is somewhat overfitting to the training data.

For the improvement of this part, Siamese LSTM can use Manhattan distance for distance detection, which is the most suitable measure for high-dimensional text data (Othman et al., 2019). Transformer can perform hyperparameter optimization, systematically search for optimal learning rates, optimizers, etc.

4.3 Claim Classification in differenct models

Previously, we determined to use Transformer to predict 5 evidences for each text, and then we used ZeroR and Transformer to predict labels (Figure 3) for evidences and text.

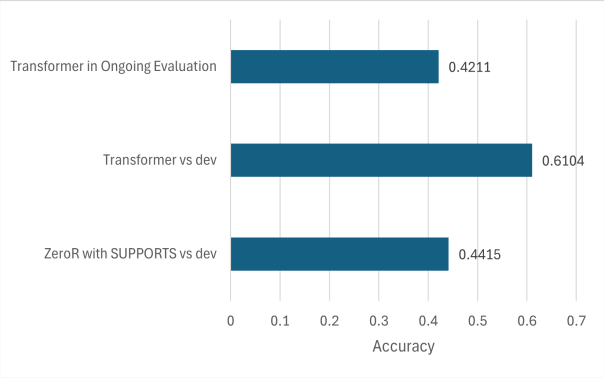


Figure 3: Comparison of ZeroR and Transformer vs dev results and Codalab results

After comparing the local predictions for the development set, the Transformer performed better, with a predicted label accuracy of 0.6104. The accuracy of predicting all labels as SUPPORTS is 0.4415, which provides a performance standard. Transformer's local performance has been greatly improved, and it is better at learning and extracting relevant information from a large amount of evidence. However, when the Transformer was selected to participate in Codalab's ongoing evaluation of the data predicted by the test set, the accuracy rate became 0.4211. This indicates that the model's generalization ability is not as expected

In order to further compare the performance of Transformer under different parameters, different dimensions are used for comparison.

4.4 Dimensions in Classification

When predicting labels for Transformers with different dimensions, it has been verified that 512 dimensions should be used as the Transformer's prediction for labels. Table 2 shows that on the development set, the accuracy peaked at 0.64 at 512, showing the best performance across all dimensions. Although 512 dimensions is not the highest value on Codalab (the highest is 0.434 in 1024 dimensions), the values are very different. 512 dimensions can better balance performance and computing efficiency. While maintaining performance, efficiency of computing resources and training time also need to be considered.

Dimensions	**128**	**256**	**512**	**1024**	**2048**
Dev Acc	0.42	0.49	0.64	0.6	0.55
Codalab	0.32	0.33	0.42	0.434	0.31

Table 2: Accuracy with different dimensions in Transformer

5 Conclusion

In this automated fact checking, we found out that Transformer is a suitable model for this data set. In evidence retrieval, preprocessing containing stopwords will achieve a higher f-score. After comparing the performance of the three models in 1-5 evidences, Transformer with 5 evidences was predicted to have a higher f-score. Based on these 5 evidences for label prediction, using a 512-dimensional Transformer can achieve higher accuracy and lower computing resources. We finally compared it with the development set and obtained an f-score (F) of 0.096 and an accuracy (A) of 0.64. On Codalab Ongoing we achieved an F of 0.0899 and an A of 0.42. We need to further study about hyper parameter tunnings (n-head, number of layers) and better word embeddings.

Team Contribution

References

Haisal Dauda Abubakar, Mahmood Umar, and Muhammad Abdullahi Bakale. 2022. Sentiment classification: Review of text vectorization methods: Bag of words, tf-idf, word2vec and doc2vec. *SLU Journal of Science and Technology*, 4(1 & 2):27–33.

Type	Content	Surname
C	Data peprocessing	All
C	Cosine similarity model	Tan, Wang
C	LSTM model	Yan
C	Transformer model	Liang,Yan
R	Abstract	Liang
R	1. Introduction	Wang
R	2.1.1 Cos similarity	Tan
R	2.1.2 Tokenization	Tan
R	2.1.3 Siamese LSTM	Yan, Wang
R	2.1.4 Transformer	Liang
R	2.2 Claim cls	Yan
R	3.1 Evi ret baseline	Tan
R	3.2 Evi ret LSTM	Wang,Yan
R	3.3 Evi ret transformer	Liang
R	3.4 Classification	Yan
R	4.1 Preprocessing	Tan
R	4.2 Model comparison	Wang
R	4.3 Claim cls	Wang,Yan
R	4.4 Dimensions	Wang
R	5. Conclusion	Wang

Table 3: Contribution table for code(C) and report(R)

Pepa Atanasova. 2024. Generating fact checking explanations. In *Accountable and Explainable Methods for Complex Reasoning over Text*, pages 83–103. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. *CoRR*, abs/1706.04599.

MD Asif Iqbal, Omar Sharif, Mohammed Moshiul Hoque, and Iqbal H Sarker. 2021. Word embedding based textual semantic similarity measure in bengali. *Procedia Computer Science*, 193:92–101.

Mohammad K Nammous and Khalid Saeed. 2019. Natural language processing: Speaker, language, and gender identification with lstm. *Advanced Computing and Systems for Security: Volume Eight*, pages 143–156.

Anmol Nayak, Hariprasad Timmapathini, Karthikeyan Ponnalagu, and Vijendran Gopalan Venkoparao. 2020. Domain adaptation challenges of bert in tokenization and sub-word representations of out-of-vocabulary words. In *Proceedings of the first workshop on insights from negative results in NLP*, pages 1–5.

Nouha Othman, Rim Faiz, and Kamel Smaïli. 2019. Manhattan siamese lstm for question retrieval in community question answering. In *On the Move to Meaningful Internet Systems: OTM 2019 Conferences: Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21–25, 2019, Proceedings*, pages 661–677. Springer.

Hassan Saif, Miriam Fernandez, Yulan He, and Harith Alani. 2014. On stopwords, filtering and data sparsity for sentiment analysis of twitter.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.