COMP90042 Project 2023:
Automated Fact Checking For Climate Science Claims

Student Number: 1336242

1 Introduction

Climate change has drawn a lot of attention from human society, but the surge in unverified assertions about climate science has resulted in a misrepresentation of public opinion, emphasizing the need to verify claims pertaining to climate science. Therefore, the main purpose of the project is to develop an automated fact-checking system, which is able to conduct fact-checks given a claim. There are two main components in the system that aims to handle different tasks. The first component focus on evidence retrieval, where given a claim, the component should search for the most related evidence passages from the evidence dataset. With regard to the second component, its target is to classify the status of the claim given the evidence. In addition, this is a multi-class classification task, containing 4 different classes: [SUPPORTS, REFUTES, NOT_ENOUGH_INFO, DISPUTED]. Hence, a successful system for this project should meet two requirements, one is to retrieve the correct set of evidence passages and the other is to classify the claim correctly.

There are a number of studies about information retrieval systems. For example, multi-hop dense retrieval (MDR) (Xiong et al., 2020) and generation augmented retrieval (GAR) (Mao et al., 2020) are all prevalent information retrieval systems. In this project, because the first task can be regarded as an open-domain question answering (QA) task (Voorhees et al., 1999) that aims to find answers in a large collection of documents (evidence dataset), the idea based on dense passage retrieval (DPR) (Karpukhin et al., 2020) has been utilized to tackle the first component of the project. Meanwhile, to solve the second fact-checking task, BERT (Devlin et al., 2018) with a classifier is implemented and has achieved relatively high accuracy.

2 Method

2.1 Evidence Retrieval

For the evidence retrieval task, two types of models inspired by dense passage retrieval (DPR) (Karpukhin et al., 2020) are developed. Figure 1 presents the architecture of dual-encoder dense passage retrieval. The only difference between the two models implemented in this project is the number of encoders used. In other words, one is a single-encoder architecture that encodes the claim and evidence in the same encoder, while the other is a dual-encoder type, encoding the claim and evidence separately.

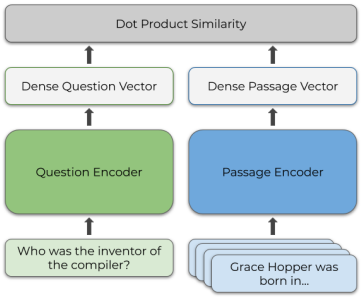

Figure 1: Dual-encoder architecture used in dense passage retrieval

The main idea in dense passage retrieval (DPR) (Karpukhin et al., 2020) is that the system should be able to index all the passages in a low-dimensional and continuous space when given a collection of M text passages. With these passage encodings, the top k passages closest to the problem should be output. The similarity between the question and passage can be defined using the dot product of the encoding vectors, shown as equation 1, as well as the cosine similarity. In my implementation, there is a trick when using cosine similarity to calculate the loss. By scaling up the cosine similarity and processing it to log softmax, we can make the gap in loss more significant and allow the model to con-

verge faster. Moreover, the log softmax function can prevent overflow and accelerate backpropagation at the same time. The lowercase text preprocessing also has been applied in the project, since the pre-trained model is trained using corpus in lowercase.

$$sim(q, p) = \mathbf{E_Q}(\mathbf{q})^{\mathrm{T}} E_P(p) \quad (1)$$

Negative sampling is crucial in this approach. In my implementation, negative sampling is done by running a random choice upon the other evidence that is not related to the certain claim in the dataset. And the random choice process will continue until the total number is equal to the value of a preset hyperparameter **evidence_samples**. Then we concatenate the embeddings of positive evidence and negative evidence together to do the training. The loss function is defined as the negative log-likelihood of the positive passage, where q_i is the question, p_i^+ is the positive sample, and p_i^- is negative:

$$L(q_i, p_i^+, p_{i1}^-, ..., p_{in}^-) = -log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + e^{sim(q_i, p_{ij}^-)}} \quad (2)$$

The encoders used in the system are official pretrained models from Huggingface, including BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019), as these large pretrained models have achieved eminent performance on NLP tasks. During training, the Adam optimizer (Kingma and Ba, 2014) and step learning rate scheduler are utilized to update the model.

It is worth noting that BM25 approach has also been tried in this project. However, if only take advantage of BM25, the model itself is too simple to enable a semantic sensible search for evidence retrieval on large datasets. The experimental F1-score on the test dataset is 0.

2.2 Fact Checking

Regarding the fact-checking task, a simple classifier based on pretrained model is implemented. An illustration of the model architecture is shown in Figure 2. We combine a pretrained model with a simple neural network to do the classification work. Such framework architecture can capture the semantic meaning of the sentences well with the help of pretrained model. Besides, the light neural network classifier can be trained relatively fast and produce outstanding performance.

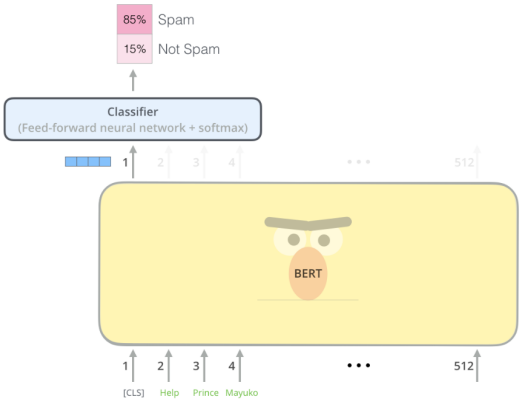

Figure 2: Bert with a Classifier, source: https://jalammar.github.io/illustrated-bert/

The contextual representation of [CLS] is generated by the pretrained model and fed into the classifier. During fine-tuning, the parameters of the whole network are updated. More detailed, the most complex classifier in the experiment has three layers, including a drop-out layer, a linear layer with a Tanh activation function, and another linear layer with a Relu activation function. Here, the Tanh activation function is designed to increase the nonlinearity of the neural network classifier. Some ablation experiments will be discussed in the next section. Since it is a multi-classification task, the loss function defined in the model is cross-entropy loss, which evaluates the effectiveness of a classification model that produces probability values ranging from 0 to 1.

2.3 Data Analysis

Some analysis has been done on the datasets given in this project. Overall, there are 12808827 pieces of evidence, 1288 training data, 154 development data, and 153 testing data. Moreover, the number of relevant evidence for each claim also has been calculated. For the training set, the number of evidence distribution of claims is {5: 477, 2: 223, 1: 210, 3: 191, 4: 127}, while for the development set is {5: 52, 1: 31, 2: 29, 3: 26, 4: 16}.

3 Experiment

3.1 Environment

All the project experiments are conducted on Google Colab. The Pytorch version is 2.0.0+cu118, GPU type is A100. Under this condition, it takes

about 20 minutes per epoch, mainly for the evidence dataset encoding when doing evidence retrieval. While the running time of the classification model is much faster, around 2 minutes for every epoch. The wandb package (Biewald, 2020) is used to record the numerical metric values during the training procedure.

3.2 Hyper Parameter settings

3.2.1 Evidence Retrieval

Due to the running time being too long for each epoch in evidence retrieval model training and limited computational resources, the **epoch number** is set to 6, while the **batch size** number is set to 8. The advantage of mini-batch training is various, including computational efficiency and vectorization benefit. The default **max length** for the tokenizer is 128. Besides, the **evidence samples** number is set to 64 for ransom negative sampling. The default learning rate of the Adam optimizer is set to 2e-5. The default step size of the step learning rate scheduler is set to 5 and the default gamma value of it is set to 0.5. Lastly, the k value of top-k algorithm is 3 according to the result of the data analysis.

3.2.2 Fact Checking

Since the running time is much shorter for each epoch in fact-checking classification model training, the **epoch number** is set to 10 and the batch size is set to 8. Meanwhile, as same as evidence retrieval, the learning rate of Adam optimizer is set to 2e-5. Note that, there is no learning rate scheduler in the classification model since the loss is found low enough when training is finished during the experiments. Besides, the default **max length** of the tokenizer is 512 for language model maximum utilization. And the default dropout rate for the dropout layer is 0.5.

3.3 Method Comparison

3.3.1 Evidence Retrieval

Figure 3 shows the change curve of the f-score corresponding to every epoch in the training procedure. Here, the legend name without **v2** means the single-encoder architecture while **v2** means the dual-encoder. **Bert** and **roberta-base** pretrained language models are tested for comparison. The **different LR** means the hyperparameter of the default step learning rate scheduler is changed, in which the 50 for step size and 0.9 for gamma.

From the chart, we can find that RoBERTa (Liu et al., 2019) model as pretrained language encoder outperforms the original Bert (Devlin et al., 2018). The reason for that is RoBERTa introduces dynamic masking, enabling masked token changes during the training procedure. Besides, RoBERTa utilizes larger batch-training sizes, which is found to be more useful in the training procedure.

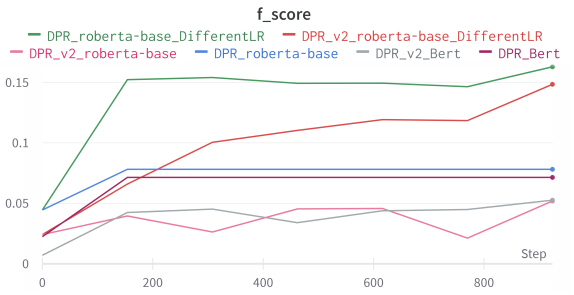

Figure 3: F-score for evidence retrieval model

Moreover, we can find that the f-scores of 'DPR_Bert' and 'DPR_roberta_base' does not change after one epoch. This can be explained compared to the results using different learning rate schedulers. Since the step size of the default scheduler is too small and the gamma value is too large, the learning rate narrowed to meaningless after one epoch. As a result, the parameters in the language model failed to update and the f-scores stay still.

Surprisingly, the performance of dual-encoder DPR is slightly weaker than single-encoder one. This may be because the training dataset is not large enough, as dual-encoder DPR has two encoders for training, and should intuitively reach the convergence more slowly. Perhaps a deeper comparison could be conducted by increasing the number of epochs and the size of the training dataset.

3.3.2 Fact Checking

The accuracy change curve is presented in Figure 4. Based on the results, we can conclude that there is no significant difference between using Bert as an encoder with using RoBERTa. The accuracies of the convergent model are both around 99.7%, achieving great performance over the development set.

3.4 Ablation Study

3.4.1 Evidence Retrieval

For the evidence retrieval model, the ablation study is focused on stop words because it is a prevalent preprocess trick, even applied in Google Search Engine. Note that, here the model is trained using

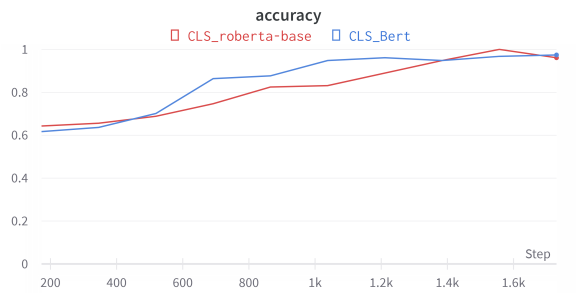

Figure 4: Accuracy for text classification model

both training and development datasets. Thus the f-scores are higher than the above results. Shown as Figure 5, we can find that whether or not the stop words are deleted has no impact on the performance of the model in this scenario. Intuitively, we should not delete the stop words when constructing a QA system since it will change the semantic meaning of the sentences, which may worsen the performance. However, the stop words have no influence on the metrics, probably because the training time was not long enough or the training data was too small.

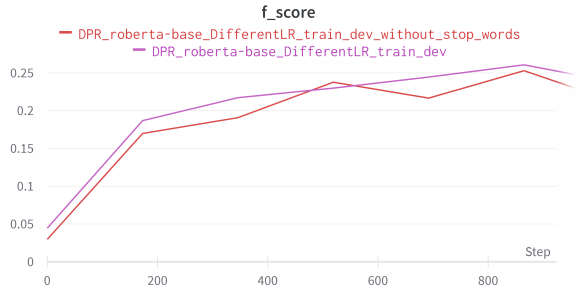

Figure 5: with vs without stop words

3.4.2 Fact Checking

For the fact-checking model, both Relu activation function and dropout layer are tested for ablation study. In Figure 6, we can find that the final performances of the three models are on par with each other, while there are some slight differences during the training procedure. Obviously, the model without the dropout layer converges slower than the other two models. This may be because the dropout layer can alleviate the overfitting problem by turning certain neuron values into zero, reducing the training time to some extent. Therefore, if a model has no dropout layer, the process of convergence takes a longer time. When comes to the piecewise linear function Relu, we can find that Relu slower the convergence. In general, Relu can increase the non-linear relationships between neural networks, allowing for more complex model structures. However, given that the classifier in the model is shallow, Relu may not play its main role.

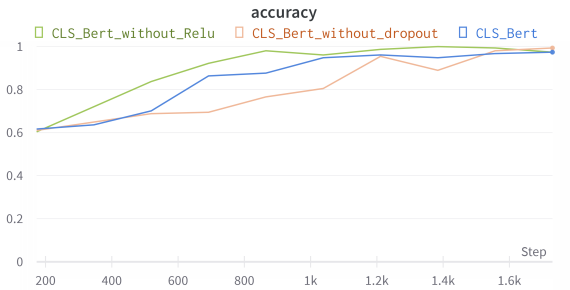

Figure 6: Relu and Dropout Ablation Study

3.5 Conclusion and Future Direction

In this project, we build two different models for evidence retrieval and one for fact-checking. Combing these two components, an automated fact-checking system for climate science claims is constructed. Some experiments have been conducted on the system, including several ablation studies. The following Table 1 presents the ongoing and final performances on the Codalab leaderboard.

Time	Harmonic Mean	F-score	Accuracy
Ongoing	0.197	0.131	0.395
Final	0.206	0.137	0.416
Baseline	0.082	0.048	0.286

Table 1: Ongoing and final performance of the system

From the table, we can conclude that the performance of the final evaluation is slightly higher than the ongoing evaluation. Since the change is small, it can be inferred that the modest effect improvement is mainly due to differences between the ongoing and final test sets. Meanwhile, it also indicates that the system is not over-tuned based on the ongoing test set.

Overall, the system has achieved a good performance in the competition and beaten the baseline, but there are still some limitations that can be conquered by improvements, especially for the evidence retrieval part. For instance, the epoch number can be increased and the negative sampling approach can be more sophisticated, e.g., using BM25 to filter the closest negative samples.

References

Lukas Biewald. 2020. Experiment tracking with weights and biases. Software available from wandb.com.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2020. Generation-augmented retrieval for open-domain question answering. *arXiv preprint arXiv:2009.08553*.

Ellen M Voorhees et al. 1999. The trec-8 question answering track report. In *Trec*, volume 99, pages 77–82.

Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, et al. 2020. Answering complex open-domain questions with multi-hop dense retrieval. *arXiv preprint arXiv:2009.12756*.