Title: Use sentence pooling to identify writing elements

G051 (s2089076)

Abstract

Usually, students learn writing skills through writing their own essays in the school. To produce more powerful writings, it's important to understand how to segment texts and identify argumentative and rhetorical elements in writings. E.g. Which part is the introduction? Where is the thesis statement? Does one sentence belong to an example or a claim? Building an automated feedback tool to identify such elements in writings would be very helpful for students to learn writings, especially for those students who has limited education resources. In this project, we aim to build a Transformer based model to segment writings into different writing elements. We use a data set provided by Georgia State University on Kaggle consisting of 144,293 argumentative essays written by U.S students in grades 6-12. We choose the BIGBIRD Transformer model as the baseline and then investigate the architecture of model to improve the performance. We find that using our sentence pooling method could improve the F1 score from 52.8% up to 58.8%. The basic idea and novelty behind this method is to treat a single word NER problem as a sentence level NER problem, which is more suitable for this identifying writing elements task.

1. Introduction

Writing is an important skill for everyone. However, according to the National Assessment of Educational Progress, less than a third of high school seniors are proficient writers. Usually, students learn writing skills through writing their own essays in the school. To produce more powerful writings, it's important to understand how to segment texts and identify argumentative and rhetorical elements in writings. E.g. Which part is the introduction? Where is the thesis statement? Does one sentence belong to an example or a claim? Building an automated feedback tool to identify such elements in writings would be very helpful for students to learn writings, especially for those students who has limited education resources.

On the other hand, deep neural networks and Transformer based models(Vaswani et al., 2017), such as BERT(Devlin et al., 2018), have been widely used in various tasks nowadays. Named Entity Recognition(NER) is one of the common task in the Natural Language Processing(NLP) filed. The main goal of NER is to predict position and category for important nouns and proper nouns in a text(Mohit, 2014). Transformer based models have been proved to be effective for NER task(Yan et al., 2019).

For our context, if we treat our identifying writing elements task as a NER task, we could define those writing elements labels as a new set of NER labels, assign each word in the essay with such a NER label, and then apply a Transformer based model to it, which extracts the embedding from the last hidden layer of encoder and adds one more linear layer to classify words. This is what we have for the baseline model.

However, it's very intuitive that the performance of model could still be improved. This is because for the NER task, every word is treated as a single entity and has different embedding, which leads those words in one sentence may have different labels. An example of this could be seen in the figure1. In the contrast, for our identifying writing elements task, words in one sentence tend to have same labels. Though there are exceptions when there are conjunctions which directly link two sentences without any punctuation marks and also usually mark the start of a new writing element sentence, such as "because", labels are split by punctuation in most cases. It is worth noting that here what we mean by "words in one sentence" is words between two punctuation marks but not words between two end marks. This is because it's common to see that main clauses and subordinate clauses have different writing elements. An example of identifying writing elements could be seen in figure2.

Based on the above analysis, the research objective of this project is to explore how we could treat a set of individual labels as a whole when each of the individual label in this set has a same label based on some specific rules. To be more specific in our context, we need to find how we could change the network of the baseline Transformer based model for a NER task so that the new model could make use of sentence level information, which is more important for our identifying writing elements task.

We proposed to add a new sentence pooling layer based on our sentence label after the last hidden layer of encoder. We also proposed two methods to construct the sentence label: one is only based on punctuation marks and the other one is based on both punctuation marks and a carefully picked set of conjunctions. The result shows that compared with the baseline, the first method improves the F1 score from 52.8% to 56.1% while the second method improves the F1 score from 52.8% to 58.8%.

Figure 1. Example of NER

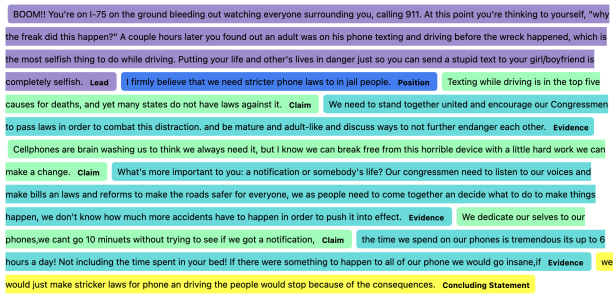

Figure 2. Example of identifying writing elements

2. Data set and task

The data set is provided by Georgia State University on Kaggle consisting of 144,293 argumentative essays written by U.S students in grades 6-12(Kaggle, 2021).

The data set mainly contains two kinds of files: *.txt files and train.csv file. Each of the *.txt file is a full text of an essay. And the train.csv file contains the labeled version of all essays in the form of table. In this table, the main property is *id*, *class*, and *predictionstring*: *id* gives a reference to the *.txt file, *class* gives the name of writings elements for the labeled sentence, and *predictionstring* is the word indices for this labeled sentence in .text file. An example of this could be seen in table1.

id	class	predictionstring
AC594194F01C	Evidence	155 156 157...
4F0E197053FF	Position	0 1 2...
4F0E197053FF	Lead	14 15 16...
4F0E197053FF	Claim	106 107 108...
4F0E197053FF	Evidence	162 163 164...

Table 1. Example of train.csv file

The writings elements could be divided into 7 categories: Lead, Position, Claim, Counterclaim, Rebuttal, Evidence, and Concluding Statement. The meaning of each of them is as following(Kaggle, 2021):

- Lead - an introduction that begins with a statistic, a quotation, a description, or some other device to grab the reader's attention and point toward the thesis
- Position - an opinion or conclusion on the main question
- Claim - a claim that supports the position
- Counterclaim - a claim that refutes another claim or gives an opposing reason to the position
- Rebuttal - a claim that refutes a counterclaim
- Evidence - ideas or examples that support claims, counterclaims, or rebuttals.
- Concluding Statement - a concluding statement that restates the claims

Our task is to predict writing elements category for each words in an essay in a similar form as table1. If one word does not belong to any category, we don't produce a label for that word. As the label in table1 does not directly link to words, which can not be received by a Transformer based model, we need to do preprocessing to the data set. Based on the *id*, *class*, and *predictionstring* and *.txt file, we could link the writing elements category to each word in each of the *.txt file. Considering that each *.txt file is an essay and the labels for words in one essay has logical relationship with each other in the same essay but not has relationship with words in another essay, we construct a new table so that each row stands for one essay as well as one input data, while there are two main columns in each row: one is the text in that essay and another is the label list for each word. Because we also need a label for those words that do not belong any writing elements category so that the Transformer based model could handle, we assign an "O" label to these words. An example of such a table can be seen in table2.

text	labels
Dear Principal,I...	O, O, Position...
Using the facial...	Position, Position, Position...
"In my opinion...	Lead, Lead, Lead...
Luke who was...	Evidence, Evidence, Evidence...

Table 2. Example of input data table

Then, because the computer could not directly receive string data, we need to transfer these string data to numbers. The text data in the first column will be transferred to numbers by Tokenizer through the HuggingFace tool, which assigns each word or each sub-word an index in the vocabulary. The string labels will be directly transferred to numbers through a dictionary in Python. Each of the 7 categories will be assigned a number from 1-7 and the "O" label will be assigned with 0. An example of input data table after such tokenizing is illustrated in table3.

inputs	labels
792,89,45...	0, 0, 2...
465, 71, 125...	2, 2, 2...
95, 43, 641...	1, 1, 1...
876, 457, 310...	6, 6, 6...

Table 3. Example of input data table after tokenizing

Then, 90% original train data set will be split into a train data set and the other 10% will be assigned to a validation data set. We do not choose to do a k-fold validation because it is too computationally demanding. Also, this 10% validation data will have 144,293 * 0.1 = 14,429 data points,

which is enough for validation purpose in normal case. We do not set test data set because we will focus on the network of the model and will not pay too much attention to those hyper parameters, which turns out that the validation set will only be used once for each network but will not be for any hyper parameters selection purpose.

When it comes to evaluate the result, we need first to count the number of true positive, false negative, and false positives example in the results. And then compute recall and precision, and finally use F1 score to evaluate the results because it gives a balance between recall and precision. More details about how to divide examples into true positive, false negative, and false positives could be found in Kaggle(Kaggle, 2021). In brief, if the overlap between the ground truth and prediction is >= 0.5, and the overlap between the prediction and the ground truth >= 0.5, the prediction is a match and considered a true positive(Kaggle, 2021). We adopted the same evaluation method because when the overlap is greater than 0.5 at the same time for both directions, it does mean that there is more than half of the probability that the writing elements of the sentence is correctly predicted.

3. Methodology

Our model follows the overall structure of Transformer network(Vaswani et al., 2017). The encoder is to use attention mechanism get the word embeddings while the simplest decoder is to take these embeddings as inputs and add one more feed forward layer and a softmax layer to get the final result.

3.1. Transformer Encoder

Transformer encoder uses attention mechanism as its core. Usually, one Transformer encoder has multiple Transformer blocks and each block includes one attention layer, some layer norm layers and feed forward layers. Residual connections are also used(He et al., 2016). The overall architecture of Transformer encoder and one Transformer block could be seen in figure 3.

Given an input vector $X \in \mathbb{R}^{l \times d}$, the objective of the attention mechanism is to produce an output vector $Y \in \mathbb{R}^{l \times d}$ so that the output vector Y could represent contextual meaning, where l defines the sequence length and d is the feature dimension. And this output vector Y could be called contextual embeddings. We use $x_1, x_2...x_l$ and $y_1, y_2...y_l$ to represent each input and output vector in the sequence length dimension. In the other words, each x or y present a word in the text, which holds a dimension $\mathbb{R}^{1 \times d}$. In order to gain contextual information, each y_i is a weighted sum of $x_1, x_2...x_l$. The formula is given below:

$$y_i = \sum_j x_j w_{ij} \tag{1}$$

In the simplest case, the w_{ij} is a dot product between x_i and x_j:

$$w_{ij} = x_i^\top x_j \tag{2}$$

This dot product could be seen a measure of similarity between x_i and x_j. However, since there are no parameters that makes the model learnable, x_i in equation 2 , x_j in equation 2, and x_j in equation 1 are multiplied by three learnable matrix: W_q, W_k, W_v. Each of them holds a dimension $\mathbb{R}^{d \times d_k}$, where d_k is a hyper-parameter. Then query q_i, key k_j, value v_j are defined by the following three formulas:

$$q_i = x_i W_q \tag{3}$$

$$k_j = x_j W_k \tag{4}$$

$$v_j = x_j W_v \tag{5}$$

The new learnable w'_{ij} is given below:

$$w'_{ij} = q_i^\top k_j \tag{6}$$

In order to limit the value of w'_{ij} not to become too large, map the value to [0,1] and make the sum of them is 1, a scale factor $\sqrt{d_k}$ and the softmax function are applied to w'_{ij}. The reason for the scale factor is that the softmax function is sensitive to very large input values and then produces some very small values elsewhere, which will kill the gradient and slow down learning. The reason for the softmax function is that limiting value in a range from 0 to 1 will speed up the learning. Then, the adjusted learnable w''_{ij} is given below:

$$w''_{ij} = softmax(\frac{q_i^\top k_j}{\sqrt{d_k}}) \tag{7}$$

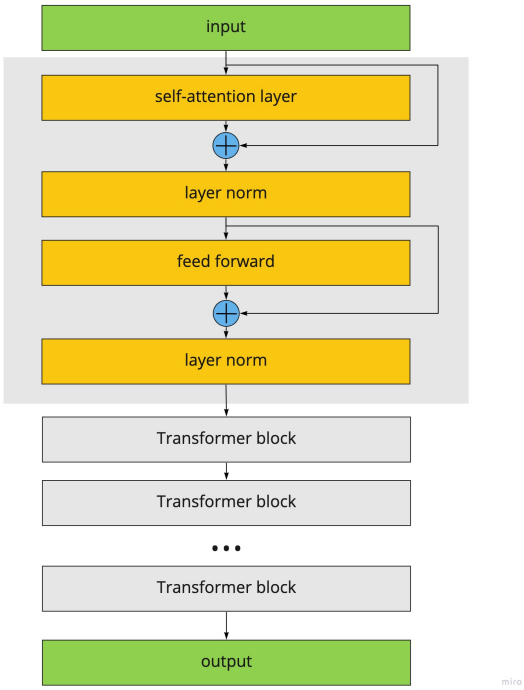

Figure 3. Architecture of Transformer encoder

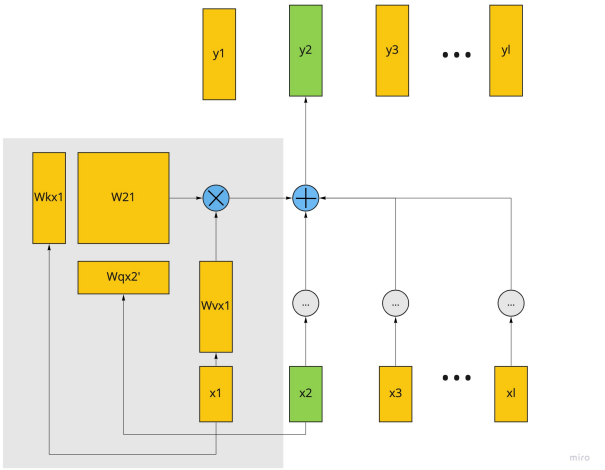

Figure 4. Attention architecture

And the new output y_i is given below:

$$y_i = \sum_j v_j w''_{ij} \tag{8}$$

If we vectorize and combine all the above formulas, the output Y will be given as following:

$$Y = XW_v softmax(\frac{(XW_q)^\top XW_k}{\sqrt{d_k}}) \tag{9}$$

Formula 9 shows the process of one self-attention layer. The architecture could be seen in figure 4. It is also called single-head self-attention. By combining several single-head self-attention mechanisms, we can give the self-attention greater power of discrimination and this is called Multi-head self-attention.

3.2. Baseline Model

In practice, we directly use the encoder of BIGBIRD model as our Transformer encoder (Zaheer et al., 2021). BIGBIRD model uses sparse attention mechanism that reduces the quadratic dependency to linear when computing the matrix of attention weights. It could handle longer text and improve the training speed on various NLP tasks. Considering that the length of essay is long and we just have limited computing resources, BIGBIRD model is very suitable for our baseline model.

After the above BIGBIRD Transformer encoder, input will be transferred to contextual embeddings. Then we pass them to a dropout layer to eliminate overfitting(Srivastava et al., 2014). An then feed the output of the dropout layer to a linear layer to get scores for each of 8 classes, which consist of 7 writing elements labels and 1 "O" label. Softmax or sigmoid layer is not needed because we will combine it in the cross entropy loss function for the numerical stability. The overall architecture of the baseline model could be seen in figure 5. The linear layer has a shape of 768*8 because the feature dimension for each word is 768 after the BIGBIRD encoder and 8 is just the number of output labels.

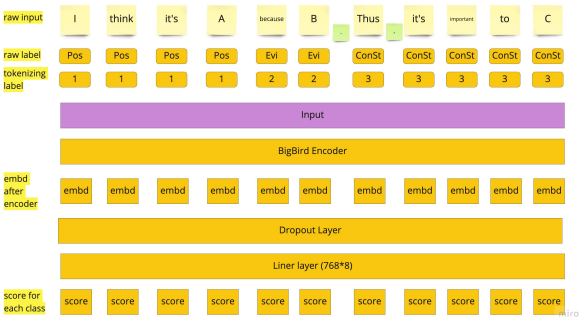

Figure 5. Baseline architecture

3.3. Proposed Model

3.3.1. Sentence pooling based on punctuation marks

Considering the reason that we state in the introduction section, we could use sentence embeddings instead of word embeddings as the input of the following dropout and linear layer after the Transformer encoder. This could be done through an average pooling layer which computes the mean of embeddings that in one sentence. We first need a sentence label to tell the model how we divide sentence. This could be done though the punctuation marks. Before training, we label the first word in sentence with "0" and continue to label words with "0". However, when we encounter a punctuation mark, we switch the sentence label to "1" and continue to label words. And each time we encounter a punctuation mark, we will switch the sentence label from "0" to "1" or from "1" to "0". In this way, all the words in the input will have a sentence label and these "0" and "1" will tell if adjacent words belong to same sentence. At the training time, after the Transformer encoder, those contextual embeddings for each word will be average pooled based on the sentence label. After this pooling layer, each word in the same sentence will have a same embeddings. Then, this output will be passed to a same dropout layer and linear layer as the baseline model. As a result, each word in the same sentence will have same score and have same predicted labels. The overall architecture of this punctuation marks based sentence pooling model could be seen in figure 6.

3.3.2. Sentence pooling based on both punctuation marks and conjunctions

As the "because" clause shown in figure 6, sometimes words that are between two punctuation marks may also have different labels. This rarely happens but it will occur when there are conjunctions which directly link two sentences without any punctuation marks. After manually inspecting the training data, words that belong to this set often are "because","so","and","if","but". Indeed, based on our English knowledge, when these words appear, the writing element are different for the sentence before conjunction and the sentence after conjunction. For example, the sentence before "because" is usually a position but the sentence after "because" is usually an evidence. Thus, we propose that

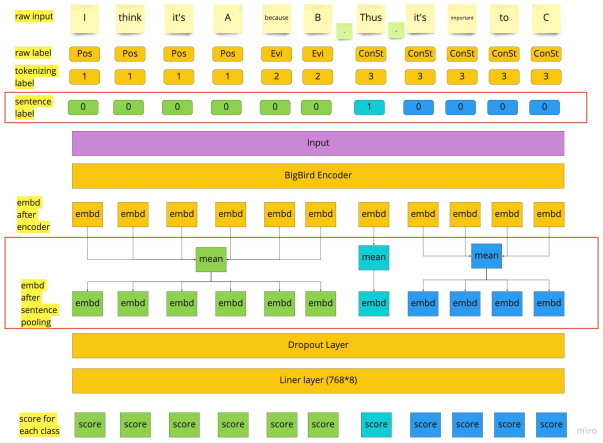

Figure 6. Architecture of punctuation marks based sentence pooling model

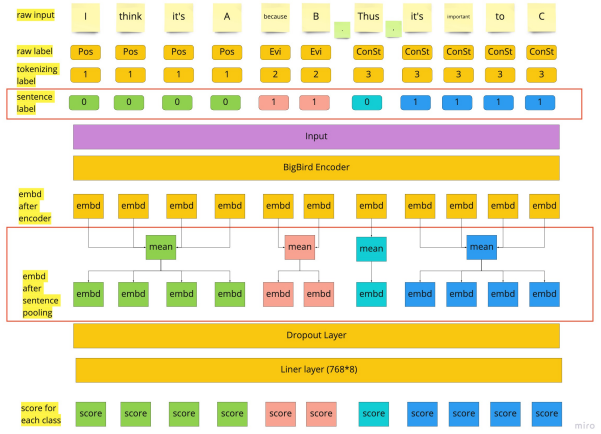

Figure 7. Architecture of both punctuation marks and conjunctions based sentence pooling model

when we build the sentence label, we could also detect if there are these conjunctions. So for this idea, we switch the sentence label not only when there is a punctuation mark but also when there is a conjunction that belongs to the above carefully picked set. In this way, we could further recognize those words that are in one sentence but have different writing elements labels. The overall architecture of this both punctuation marks and conjunctions based sentence pooling model could be seen in figure 7.

4. Experiments

4.1. Motivation

There are three experiments that needed to be carried on: baseline model, punctuation marks based sentence pooling model(PM), punctuation marks and conjunctions based sentence pooling model(PMC). The baseline model experiment aims to research the performance of the basic model where we treat our task as an individual word NER task. The PM model experiment aims to research the impact of sentence pooling based on punctuation marks. The PMC model experiment aims to research the impact of sentence pooling based on both punctuation marks and conjunctions. Based on our analysis, the PM model should have a huge improvement compared with the baseline model and the PMC model should further improve the performance but will not be significant as the former for the reason that the conjunctions case rarely happens.

4.2. Description

All of the three experiments are carried on Kaggle with a Tesla P100-PCIE-16GB GPU. The pre-trained BIGBIRD model is downloaded from HuggingFace(Zaheer et al., 2021). For the PM and PMC model, we design a new PoolBySentence class as the pooling layer to compute the average embeddings based on our sentence label, which is produced during enumerating dataloader. Then we add this pooling layer between the encoder and dropout layer. The baseline model will not include this layer. Cross entropy loss is applied for the training because we are doing a classification task. We use Adam as the optimizer for its good training speed. Then each model is fine-tuned for 6 epochs and the learning rate for each epoch is 2.5e-5, 2.5e-5, 2.5e-6, 2.5e-6, 2.5e-7, 2.5e-7. These learning rates are carefully chosen and decrease with epoch. This is because we find that the training loss will keep same for the later epochs if we always keep the learning rate at a e-5 level. By making it decreasing with epoch, it will reach a lower loss at the end. We also apply early stopping because we find that the training loss will even increase a bit at the last epoch. So in practice we use the model at epoch 5 to test. Both training and testing use a batch size of 4 and a max sequence length of 1024, which are limited by the computing resources. At testing time, we compute the overall F1 score as well as the F1 score for each writing elements class because this may help us to find our model behaves not well for which class, which may help to point out future direction.

4.3. Results

The result for three experiments is shown as in table 4.

Class	Baseline	PM	PMC
Lead	0.694	0.788	**0.789**
Claim	0.431	0.348	**0.367**
Evidence	0.561	0.632	**0.650**
Counterclaim	0.405	0.431	**0.468**
Rebuttal	0.296	0.355	**0.384**
Position	0.606	0.565	**0.656**
Concluding Statement	0.705	**0.809**	0.801
Overall	0.528	0.561	**0.588**

Table 4. Result of three experiments

We could observe that the overall F1 score is improved from 52.8% to 56.1% for the PM model while the PMC model improves the F1 score from 52.8% to 58.8%. This result is in line with our theoretical expectations.

We also visualize the result for the example with id

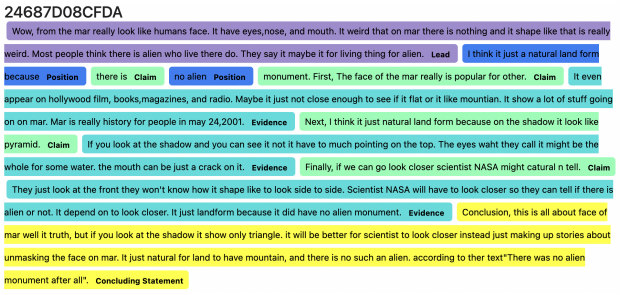

Figure 8. Visualization for a specific example with the baseline model

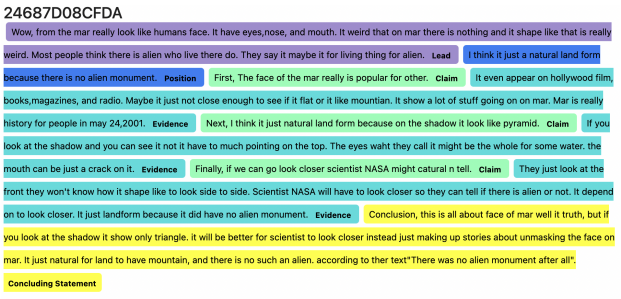

Figure 9. Visualization for a specific example with the PMC model

"24687D08CFDA" to compare the different models in detail. Figure 8 shows the baseline model. Figure 9 shows the PMC model. Figure 10 shows the gold standard.

4.4. Interpretation and Discussion

From the result in table 4, we could find that the PMC model gains the highest F1 score all the time except for concluding statement. But the difference between PM and PMC model for concluding statement class is just 0.008, which is not significant as most of other results, so we treat it just as a random factor. The overall performance for PM and PMC model improves a lot compared with the baseline result, which proves that our sentence pooling method is indeed reasonable and more suitable for our identifying writing elements task, compared with treating the task as a single word NER task.

From the case study in figure 8, 9, and 10, we could observe that for the baseline model, it predicts "there is" and "monument" as a claim. This might be caused by for most of "there", "is" and "monument" in the training set, their contextual embeddings will be aimed to be closer to a claim label in the later step. However, in an essay, if such words belong to a specific writing element is not only decided by its own contextual embeddings but also decided by the other words that in the same sentence. In the other words, the self-attention mechanism makes every word in the text be aware of other words in the whole text but does not ask each word to keep a same pace with those words that in the same sentence. On the other hand, our sentence pooling method force words in one sentence have a same embeddings, which results in that words in one sentence must yield the same prediction. This seems to be quite

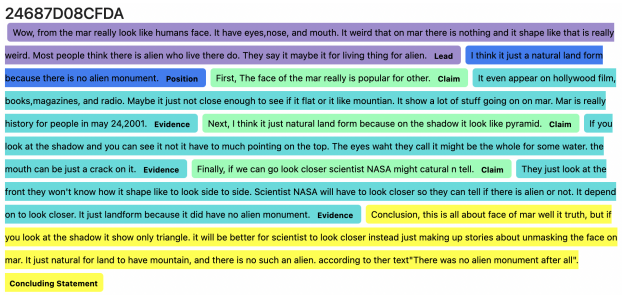

Figure 10. Visualization for a specific example with the gold standard

simple. However, we have the freedom to carefully design the sentence label, which makes it clear how we define if words are in one sentence. This freedom allows the original immutable pooling method to be flexible according to the characteristics of the task, thereby adapting to more context. For example, in a new task context, if the first word always has a same label with the last word in the text, we could assign these two words with a same specific label in the new sentence label.

However, the short of this sentence pooling method is that we have to design the sentence label according to the characteristics of the task and our common knowledge, like the common knowledge about conjunctions in our task. If the data is not readable or we don't have such common knowledge for this task, it is more likely that we could not design a good sentence label and thus this sentence pooling method will not become more meaningful than the baseline. On the other hand, after the average pooling layer, the output is directly passed to the later layer. But will it be more helpful to feed these sentence embeddings to another attention layer or Transformer block, which computes the attention between sentences, so that they become more aware of the existence of other sentences but not individual words like before? This is one of the future directions.

5. Related work

Due to this is a very specific task in the NLP filed, we didn't find much related published work that holds a similar goal. However, some intuition of the sentence pooling is derived from Reimers and Gurevych 's work(Reimers & Gurevych, 2019), which uses the sentence pooling method to predict the similarity between sentences. Another view for this task is to consider methods of encoding the position information because the position also decides the writing elements in some degree. Kitaev and Klein pointed out a method to separate positional and other information in the encoder (Kitaev & Klein, 2018). On the other hand, considering that this a Kaggle competition, many people also give their solutions on the website(Kaggle, 2021). But most of them are focusing tricks on training and stacking models, which may not contribute too much to our work.

6. Conclusions

In this identify writing elements project, we proposed a new sentence pooling method based on the designed sentence label. For now, just using punctuation marks and using both punctuation marks and a carefully picked set of conjunctions are two methods to construct the sentence label. Our two models improve the F1 score from 52.8% up to 58.8% compared with the baseline, which treats this task as a single word NER task. We also analyze deeply why our models perform better than the baseline model through analyzing the characteristics of this task and some case study. The carefully designed sentence label gives our model a freedom to adapt to other tasks while it also brings shorts for the reasons stated in the Discussion section. A future direction could be research on how to construct such sentence label in a more automatic manner. Another future direction is to explore if some following attention layers which compute attention across sentences could help to further improve the performance.

References

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

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaggle. Feedback prize - evaluating student writing, 2021. URL https://www.kaggle.com/c/feedback-prize-2021/data.

Kitaev, Nikita and Klein, Dan. Constituency parsing with a self-attentive encoder. *CoRR*, abs/1805.01052, 2018. URL http://arxiv.org/abs/1805.01052.

Mohit, Behrang. *Named Entity Recognition*, pp. 221–245. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-45358-8. doi: 10.1007/978-3-642-45358-8_7. URL https://doi.org/10.1007/978-3-642-45358-8_7.

Reimers, Nils and Gurevych, Iryna. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

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

Yan, Hang, Deng, Bocao, Li, Xiaonan, and Qiu, Xipeng. Tener: adapting transformer encoder for named entity recognition. *arXiv preprint arXiv:1911.04474*, 2019.

Zaheer, Manzil, Guruganesh, Guru, Dubey, Avinava, Ainslie, Joshua, Alberti, Chris, Ontanon, Santiago, Pham, Philip, Ravula, Anirudh, Wang, Qifan, Yang, Li, and Ahmed, Amr. Big bird: Transformers for longer sequences, 2021.