Question answering (QA) Report

– Judicial Material Reading Comprehension

Abstract

Question answering (QA) is a computer science discipline within the fields of information retrieval and natural language processing (NLP) that is concerned with building systems that automatically answer questions that are posed by humans in a natural language[1]. In this assignment, we investigated the question answering and machine reading comprehension on judgment documents.

A fully functional code framework has been provided, including the pipeline for preprocessing the data, model training and evaluation. The question type is multi-step reasoning. That is, for a given question, it is difficult to get the correct answer through only a single sentence of text. The model needs to combine multiple sentences to derive the answer.

With the provided framework and pipeline, we tested model performances with different pretrained Bert models. We also combine our original data with CAIL2019 data to improve the learning results with larger data.

The original data has its best performance on the dev data when we use pre-trained Bert “hfl/chinese-roberta-wwm-ext-large” to encode the textual tokens to vectors. When we combine the data with CAIL2019, within the 5 epochs, with or without detected supporting facts, have significantly exceeded the performance from the original data. From the loss diagrams, both datasets haven’t finished convergence within the 5 or 10 epochs, that is to say, we could expect even more significant enhancements by increasing the training iterations. Another interesting fact is, when we use CAIL that only has the supporting facts (at maximum 1 for each QA pair), the prediction on the supporting facts becomes weaker. This is potentially due to the lack of labeled supporting facts, which leads to lack of information and the model can identify supporting facts to not-supporting-facts. Below are three metrics for conclusion.

		Answer		Support Facts		Joint		At Epoch
		exact match	f1	exact match	f1	exact match	f1	
with Cail+custom_support_facts	dev	0.6885	0.7749	0.4504	0.6400	0.3472	0.5209	5
Original Data(baseline)	dev	0.6865	0.7897	0.4583	0.7080	0.3651	0.5829	13
with CAIL	dev	0.7163	0.7993	0.5	0.6519	0.3948	0.5366	10

1 Introduction

Judgment documents contain a wealth of case information, such as time, location, person relationships. Intelligent reading and understanding of judgment documents by machines can help judges, lawyers and the general public obtain the required information more conveniently.

The data used in this case is in Chinese, obtaining around 5100 pairs of questions and answers. 1700 pairs within are for civil, criminal, and administrative questions, all of which are question types that require multi-step reasoning. For evaluating the model learning quality, we take 10% of the data as the dev/test set, and the other 90% as the training set. The evaluation is done similar to HotpotQA[2]. Models are evaluated on their answer accuracy and explainability, where the former is measured as overlap between the predicted and gold answers with exact match (EM) and unigram F1, and the latter concerns how well the predicted supporting fact sentences match human annotation (Supporting Fact EM/F1).

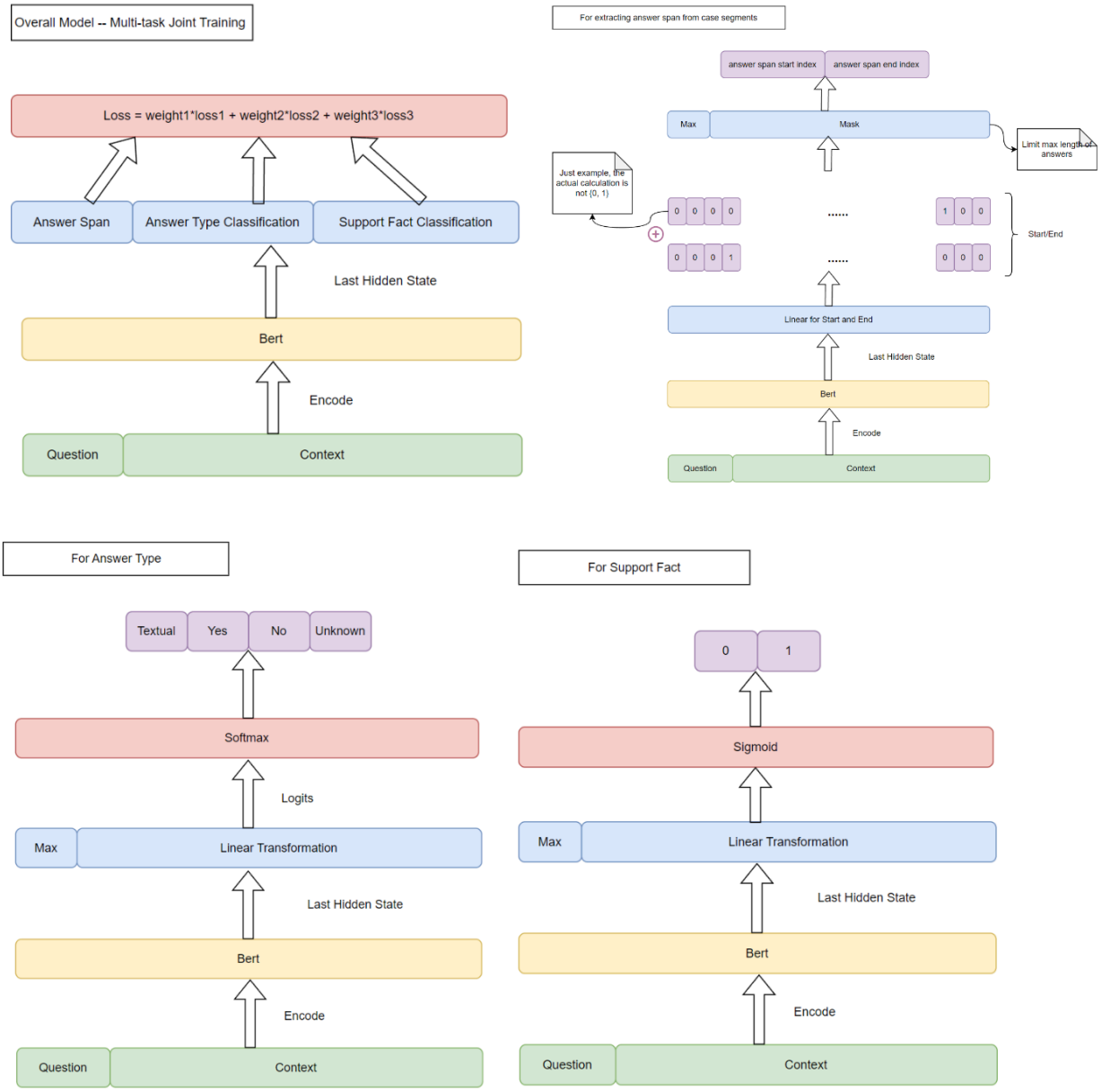
This report has two parts: 1. Code understanding and model explanation; 2. Further exploration on the task and model structure. The code understanding will be reflected by the comments addressing in the code.

2 Model Explanation and Code Understanding

A code framework has been provided, and will be attached with the report for hand-in, since most of the comments for logic and architecture understanding will be in the code. This section will only go through some of the data preprocess pipelines and model structures.

2.1 Model Structure

Below are the diagrams roughly representing the architecture of the model. Details will be explained in later sections.



2.2 Bert for Encoding

Three core parameters for pre trained Bert from transformer module are:

1. Input_ids*(batch_size, sequence_length=512)*: token ids from the input sequence to the (sub)words in vocabulary, here we input the tokens for both the questions and the contexts.
2. Attention_mask*(batch_size, sequence_length=512)*: to tackle the paddings; here the paddings are used to meet the sequence length we defined.
3. token_type_ids*(batch_size, sequence_length)*: 0 is the token for the sentence, 1 is the sentence. Here, 0 is the question, and 1 stands for the case contexts.

The output of the Bert encoder(by official document) is:

```
( last_hidden_state: FloatTensor = None,
pooler_output: FloatTensor = None,
hidden_states: typing.Optional[typing.Tuple[torch.FloatTensor]] = None,
past_key_values: typing.Optional[typing.Tuple[typing.Tuple[torch.FloatTensor]]] = None,
attentions: typing.Optional[typing.Tuple[torch.FloatTensor]] = None,
cross_attentions: typing.Optional[typing.Tuple[torch.FloatTensor]] = None )
```

Outputs used is:

- last_hidden_state(batch_size, sequence_length, hidden_size): default hidden size is 768; It is the hidden state sequence of the last layer.

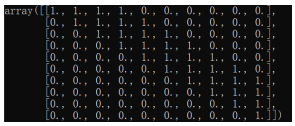
2.3 Question Answering Model Architecture

This assignment is a multimodal task. Therefore, the model architecture can be seen as three separate parts, and they are for: answer type classification, support facts classification and span extraction for answering questions with document contents. All three parts used the last hidden states output by the pretrained Bert. The final loss to update the model is the weighted sum of the losses from these three tasks.

The part for supporting facts detection is a binary classification model. By multiplying the tensor that maps 1 to all tokens in sentences with the output state, and taking the maximum value to retrieve the state of each sentence, we apply a linear transformation to calculate the probability of whether a sentence is a supporting fact. The final criterion is made with a Sigmoid function.

Answer type prediction is a multiclass prediction task, thus Softmax is used for criterion instead of Sigmoid. Simply using the maximum values for each batch from the hidden state, and doing a linear transformation would provide us the probability logits.

Extracting answer spans is the most complex among the three. Two linear transformation layers are used to predict the start position and the end position of the spans separately. The Bert hidden state is flushed into these two layers to get the probability of tokens being the start of the span or the end of the span. The end position is predicted by the sum of the start and end probabilities. Note that we need to mask out paddings and the tokens for the questions, so that the paddings will be eliminated and the question itself will not be predicted as answer spans. We also create a mask in the similar form of symmetrical diagonal matrix. The amount of 1s in each row is the same, indicating the max length of an answer span. The right diagram is an example of the mask we referred to above. The use of it is basically to make sure that the predicted start will not be the start of the next prediction.



Further details of the model are in the code, and can also refer to the model structure diagram at the beginning of Section 2.1.

2.4 Evaluation Metrics

The metrics to evaluate the model performance is inspired by HotpotQA[5]. The scores that most significantly represent the prediction quality would be “exact match” and F1. The metrics are calculated on answers and supporting facts, then a joint of these two metrics is the product of these two.

For answers, the exact match focuses on how many predictions are exactly the same as the ground truths. F1 also takes into account the similarity of the predicted answer spans and the ground true answers. Below diagram is a code snippet. One disadvantage of such calculation is that we only care about if the tokens in the prediction and the ground truth are the same, but do not care about the order of tokens. That is, “I like bananas but hate milk” and “I like milk but hate bananas” would be regarded as the same, but with totally opposite semantic meanings.

```
def f1_score(prediction, ground_truth):
    # prediction and ground truth are strings
    if prediction in ['yes', 'no', 'unknown'] and prediction != ground_truth:
        return 0, 0, 0
    if ground_truth in ['yes', 'no', 'noanswer'] and prediction != ground_truth:
        return 0, 0, 0

    common_tokens = Counter(prediction) & Counter(ground_truth)
    same_token_amount = sum(common_tokens.values())
    if same_token_amount == 0:
        return 0, 0, 0
    precision = 1.0 * same_token_amount / len(prediction)
    recall = 1.0 * same_token_amount / len(ground_truth)
    f1 = (2 * precision * recall) / (precision + recall)
    return precision, recall, f1


def if_exact_match(prediction, ground_truth):
    # if the answer is exactly the same as the ground truth
    if prediction == ground_truth:
        return 1.0
    return 0.0
```

Metrics on the supporting facts use the basic formula for precision and recall.

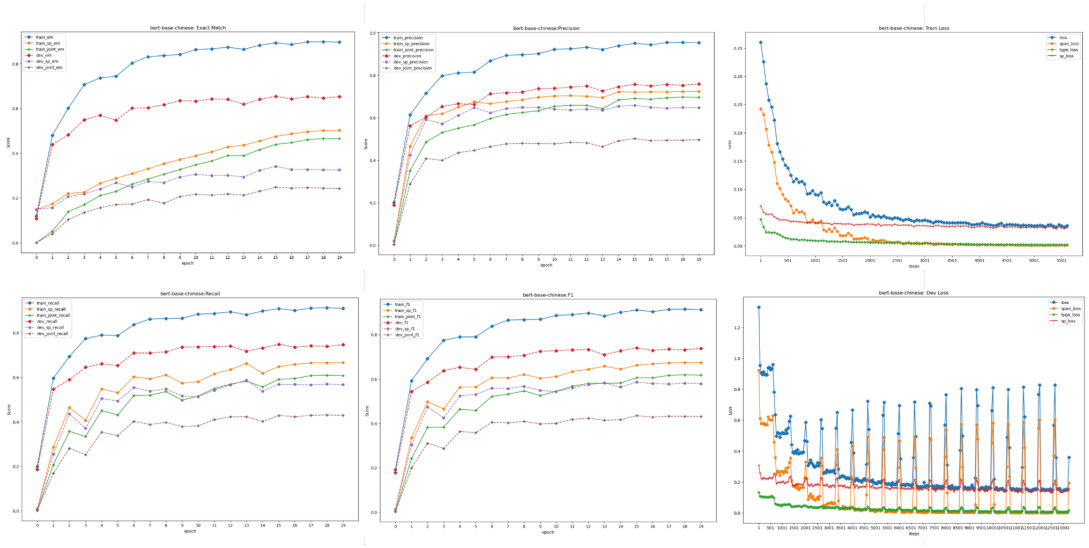
3 Comparison between Pretrained Bert

We set the training batch size to 8 and learning rate 1e-5, to compare the QA model performances within 20 epochs, with different pretrained bert models.

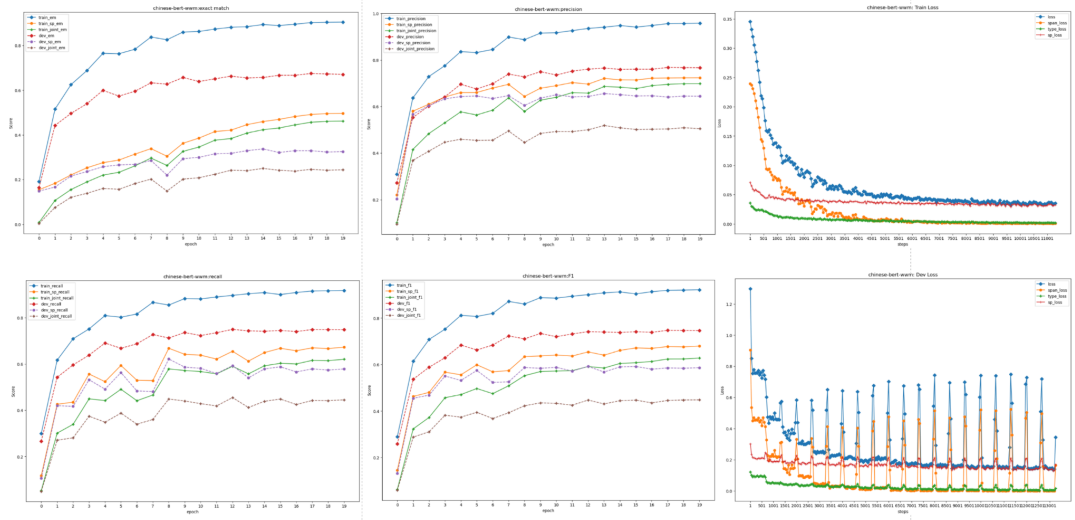
The pretrained Bert models compared are: “bert-base-chinese”, “hfl/chinese-bert-wwm”, “hfl/chinese-bert-wwm-ext”, and “hfl/chinese-roberta-wwm-ext-large”. Above diagram shows the pretrained model behaviors on CJRC, a Chinese machine reading and understanding dataset for the judicial field, from the official website[4], for semantic analysis. The diagram suggests that “hfl/chinese-roberta-wwm-ext-large” has carried out the best performances. However, due to the differences of datasets and tasks, this performance may not be guaranteed.

模型	开发集	测试集
BERT	54.6 (54.0) / 75.4 (74.5)	55.1 (54.1) / 75.2 (74.3)
ERNIE	54.3 (53.9) / 75.3 (74.6)	55.0 (53.9) / 75.0 (73.9)
BERT-wwm	54.7 (54.0) / 75.2 (74.8)	55.1 (54.1) / 75.4 (74.4)
BERT-wwm-ext	55.6 (54.8) / 76.0 (75.3)	55.6 (54.9) / 75.8 (75.0)
RoBERTa-wwm-ext	58.7 (57.6) / 79.1 (78.3)	59.0 (57.8) / 79.0 (78.0)
RoBERTa-wwm-ext-large	62.1 (61.1) / 82.4 (81.6)	62.4 (61.4) / 82.2 (81.0)

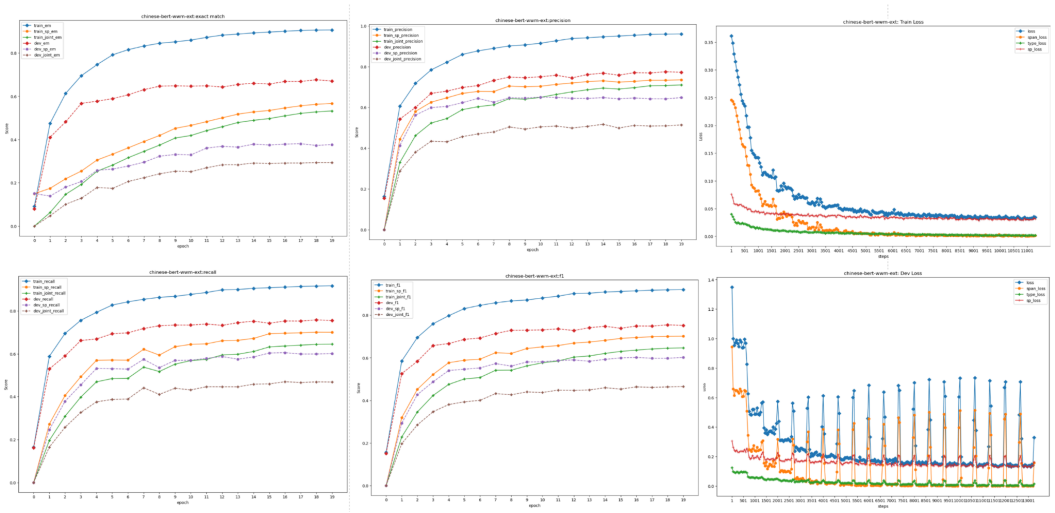
3.1 Bert “bert-base-chinese”



3.2 Bert “hfl/chinese-bert-wwm”

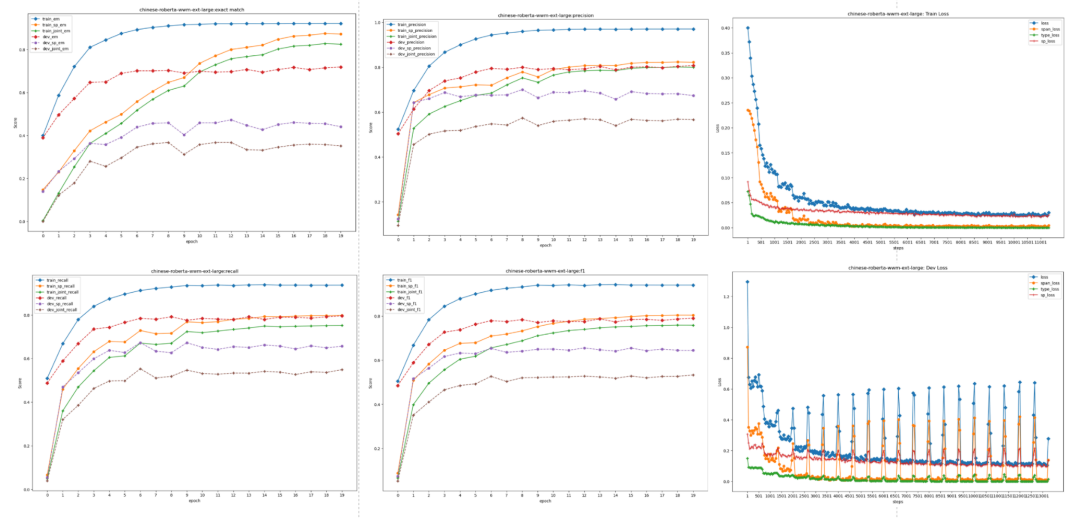


3.3 Bert “hfl/chinese-bert-wwm-ext”



3.4 Bert “hfl/chinese-roberta-wwm-ext-large”

As observed from the training progress, using chinese-roberta-wwm-ext-large has significantly increased the training and inference time.



3.5 Evaluation Metrics

We use the metric at the step when the joint f1 score gets the highest on the dev set to compare.

Bert Model		Answers				Support Facts				Joint				Epoch
		exact match	precision	recall	f1	exact match	precision	recall	f1	exact match	precision	recall	f1	
bert_base-chinese	Train	0.8826	0.9383	0.8966	0.9005	0.4548	0.7222	0.6181	0.6447	0.4157	0.6836	0.5563	0.5820	16
	Dev	0.6409	0.7453	0.7312	0.7273	0.3234	0.6540	0.5371	0.5636	0.2302	0.4905	0.4014	0.4165	16
hfl/chinese-bert-wwm	Train	0.9037	0.9571	0.9170	0.9203	0.4955	0.7233	0.6670	0.6759	0.4608	0.6982	0.6147	0.6241	20
	Dev	0.6726	0.7671	0.7491	0.7468	0.3234	0.6448	0.5741	0.5842	0.2421	0.5088	0.441	0.4479	20
hfl/chinese-bert-wwm-ext	Train	0.9059	0.9592	0.9157	0.9192	0.5632	0.7333	0.7008	0.7009	0.5280	0.7072	0.6441	0.6449	20
	Dev	0.6766	0.7749	0.7581	0.7547	0.3730	0.6419	0.5992	0.5977	0.2936	0.5095	0.4691	0.4632	20
hfl/chinese-roberta-wwm-ext-large	Train	0.9209	0.9706	0.9361	0.9404	0.8743	0.8236	0.7971	0.8043	0.8274	0.8010	0.7506	0.7589	20
	Dev	0.7143	0.8040	0.7916	0.7865	0.4544	0.6814	0.6492	0.6448	0.3571	0.5683	0.5363	0.5267	20

3.6 Observations

1. Using “hfl/chinese-roberta-wwm-ext-large” Bert has produced the best result within the 20 epochs, same as the official website diagrams indicated. We mainly judge the model learning quality based on the exact match and F1 scores.
2. Predictions on answers have better quality than predictions on the support facts. When calculating losses, the loss on the support facts weighted 5 whilst the answer type only weighted 1. The weights here actually specify how we want to trade off between different tasks under a multimodal situation. Theoretically speaking, if we give more weights to support fact losses, the model should be better at finding support facts for a QA pair. However, task imbalances impede proper training because they manifest as imbalances between backpropagation gradients. A way worth exploring is to change the weights and compare how the model behavior changes, or to run adaptive weighted loss calculations[6].
3. In training, all 4 Bert models have their answer related weights fall to 0 after a few epochs, whilst the loss on the support facts falls to the bottom value quickly and remains unchanged. Instead of saying the learning on the support facts has finished convergence, it is more like the model is actually not learning much from the support facts. This is potentially why the performances on supporting facts detection are not as good.
4. Again, due to the reason analyzed above, even the supporting fact’s loss on the train set reaches 0.05, its loss on the dev set is much greater. The model is not learning on supporting facts, and therefore, it has even weaker generalization or interpretation on the unseen data(dev set).
5. Personally I believe that the provided model has limitations on the structure, especially if we want to have the best of both worlds with multitask networks: more efficiency and higher performance. Although we are using the weighted loss for multimodal training, the actual learning for answers and supporting facts are quite independent from each other. The detection of the supporting facts is no more or less than a multiclass criterion task, without much consideration on the entity relationships. According to some experiences shared online, GNN is a much better choice, especially when it comes to supporting facts prediction.

4 Parameter Tuning

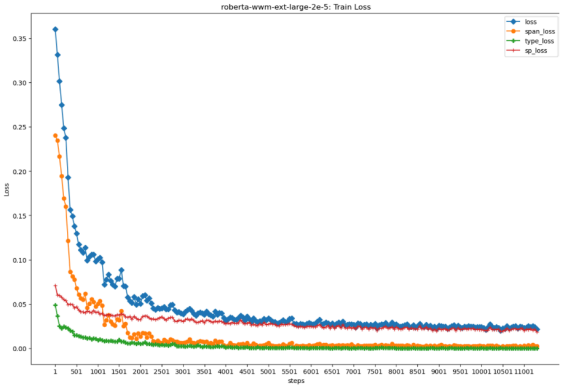
According to the evaluation metrics above, we choose “hfl/chinese-roberta-wwm-ext-large” as the pretrained model. In this section, we are going to explore how the model performs with different batch sizes and learning rates. Due to the time limit, we are still iterating at a maximum 20 epochs.

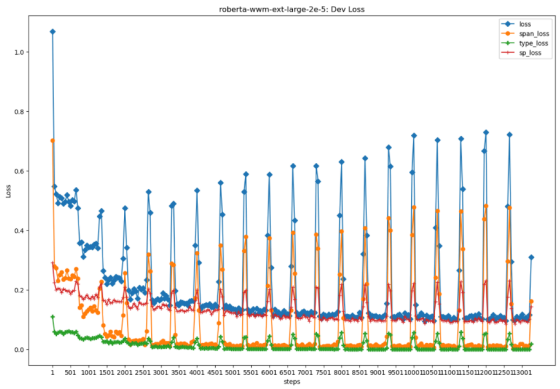
Bert Model		lr	batch size	Answers				Support Facts				Joint				Epoch
				exact match	precision	recall	f1	exact match	precision	recall	f1	exact match	precision	recall	f1	
hfl/chinese-roberta-wwm-ext-large	Train	1e-6	8	0.7171	0.8042	0.7780	0.7809	0.2878	0.6518	0.5795	0.5853	0.2214	0.5543	0.4788	0.4846	18
	Dev			0.5615	0.6724	0.6564	0.6530	0.2520	0.6330	0.5582	0.5640	0.1706	0.457	0.4011	0.3998	
	Train	1e-5		0.9209	0.9706	0.9361	0.9404	0.8743	0.8236	0.7971	0.8043	0.8274	0.8010	0.7506	0.7589	20
	Dev			0.7143	0.8040	0.7916	0.7865	0.4544	0.6814	0.6492	0.6448	0.3571	0.5683	0.5363	0.5267	
	Train	2e-5		0.9140	0.9647	0.9348	0.9395	0.7017	0.7795	0.7552	0.7569	0.6615	0.7539	0.7120	0.7150	8
	Dev			0.6806	0.7794	0.7707	0.7663	0.4544	0.6984	**0.6562**	**0.6571**	0.3532	0.5551	**0.5222**	0.5169	
	Train	4e-5		0.9195	0.9691	0.9380	0.9429	0.8274	0.8247	0.7754	0.7916	0.7815	0.8020	0.7331	0.7506	13
	Dev			**0.6865**	**0.7897**	**0.7728**	**0.7721**	**0.4583**	**0.7080**	0.6212	0.6409	**0.3651**	**0.5829**	0.5054	0.5169	
	Train		4	0.9198	0.9717	0.9418	0.9470	0.8738	0.8206	0.7983	0.8036	0.8259	0.7997	0.7994	0.7650	16
	Dev			0.6825	0.7877	0.7694	0.7691	0.4524	0.6727	0.6556	0.6442	0.3492	0.5460	0.5214	0.5100	

The loss and metrics diagrams are similar to Section 3, so will be omitted here.

Observations:

1. With batch size set to 8 and learning rate set to 4e-5, the model has the best performances on both answer and supporting facts predictions. Changing the batch size to 4 has close quality, but the performance on supporting facts is significantly worse.
2. Learning rate 1e-6 has the worst performances, that is the learning rate is too small for the model to learn.
3. When the learning rate is 2e-5, the model's prediction on supporting facts reaches the highest F1 scores. However, its performances on answer predictions are weakened.





Judging by the peaks of the answer span losses, we could assume that the model starts to overfit at around 2500 steps.

5 Combine with CAIL2019 Data

Through Section 3 and 4, we already know that the model has the best learning quality with "hfl/chinese-roberta-wwm-ext-large" Bert, batch size 8 and learning rate 4e-5. We will use these hyperparameters to compare the model performances when we combine the CAIL2019[7] data with the original data provided with this assignment.

5.1 Supporting Facts from CAIL2019

CAIL2019 data formatted differently from the data provided with the code. An important preparation is to reformat the data so that our current pipeline could still be used. Another major difference is that no supporting facts are provided in the CAIL2019 QA pairs, which means we need to extract supporting evidence from the associated judicial documents.

Below are code snippets to extract supporting evidence.

```python
def _getSentenceIndexByTokenIndex(token_index, sentences):
    length_tokens = 0

    for i, sent in enumerate(sentences):
        length_tokens += len(sent)
        current_index = i+1
        if current_index > len(sentences)-1:
            return -1
        if length_tokens <= token_index <= length_tokens+len(sentences[current_index]):
            return i+1


def _getSupportFactForOneQA(qa_pair, sentences, threshold=0.3):
    """
    The original CAIL2019 data does not contain support facts;
    We are going to extract such info by measuring the similarity and choose the
    top 3 as support facts;
    """
    support_facts = []
    top_sp_index = -1

    if qa_pair[1] == "unknown":
        return support_facts

    sent_similar_dict: Dict[int, float] = {}  # stores the sentence index and its similarity to the target
    if qa_pair[2] == -1:  # if the answer start index is -1
        # compare the similarity between question and sentences
        for i, sent in enumerate(sentences):
            similarity_score = SequenceMatcher(None, qa_pair[0], sent)
            sent_similar_dict[i] = similarity_score.ratio()
    else:
        top_sp_index = _getSentenceIndexByTokenIndex(qa_pair[-1], sentences)
        if top_sp_index != -1:
            support_facts.append(top_sp_index)
        for i, sent in enumerate(sentences):
            # compare the similarity between question+answer and sentences
            similarity_score = SequenceMatcher(None, qa_pair[0]+qa_pair[1], sent)
            sent_similar_dict[i] = similarity_score.ratio()

    # sort thr similarity score by values in descending order
    sorted_sent_similar_dict = {
        k: v for k, v in sorted(sent_similar_dict.items(), key=lambda item: item[1], reverse=True)
    }
    # get the max similarity score, if it's beyond threshold, we will not put anything into support facts
    max_sp = 3
    current_sp = len(support_facts)

    for key, value in sorted_sent_similar_dict.items():
        if top_sp_index != key and value > threshold:
            support_facts.append(int(key))
            current_sp += 1
            if current_sp >= max_sp:
                break

    return support_facts
```

An “answer start” is provided in the CAIL2019 for every query-answer pair. It specifies the starting index of the sentence token, if the answer is a sentence extracted from the case document. The index will be -1 if the answer is not a span.

For answers are spans, we reused this token index to retrieve the segment(sentence) index as the top-1 support facts. Then we detected the similarity scores between the segments in the document and the (query, answer) pair. The higher the score is, the more likely the segment will be a supporting fact. For the candidates we detected, we then introduced a threshold system to filter out the facts that have low similarity to the query and answer. This is a noise-reduction strategy.

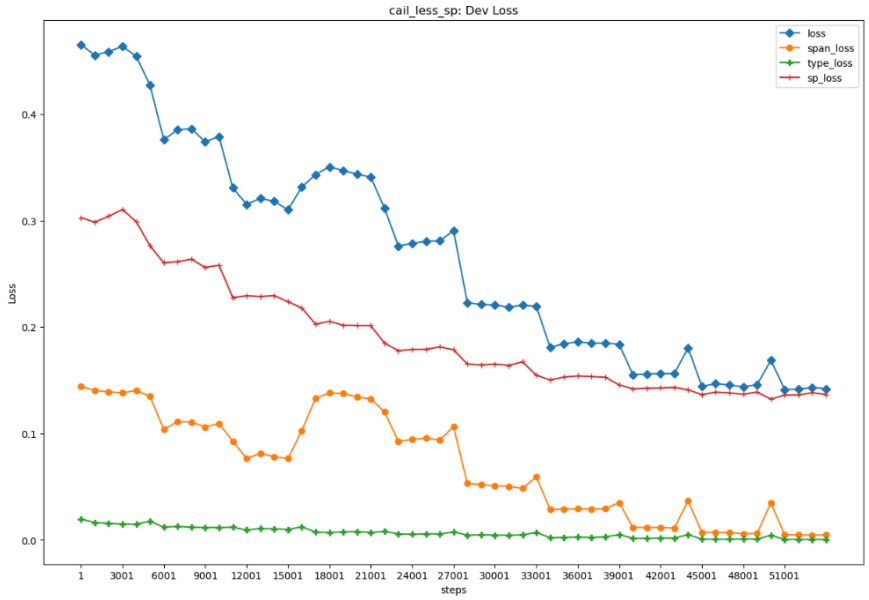
For answers that are not spans, the major difference is that we calculate the similarity scores between the segments and the query, without taking answers into comparisons.

However, this method is not bullet-proof, especially when 1. the answer span crosses multiple segments; 2. The query uses synonymous expressions rather than the tokens from the most relative segment, such as the segment “经局领导审批后从账内套取民政专项资金” cannot be detected as a supporting evidence for query “被告人袁某挪用专项资金是否被领导知晓？”. Such impact is fatal for tasks such as semantic understanding, due to the noises introduced. A more proper way would be to use the word2vec or other NLP techniques that can represent the relations and entities, instead of some simple sequential occurrence similarity. We tested two ways of extraction: 1. The method described above; 2. We only care about the answers are spans and use the starting index associated segment as the supporting evidence, and leave the other answer types corresponding evidence empty.

5.2 Model Comparison

CAIL2019 data formatted differently from the data provided with the code. An important preparation is to reformat the data so that our current pipeline could still be used. We only run limited epochs.

Cail(only supporting facts that detected by the starting index provided with data) – Loss on Dev



Due to the time restriction and we ran the experiments on different machines, the loss diagram for Cail(with sequence similarity detected supporting facts) was not reachable at the time we finished this report. However, the missing diagram was drawn and has a similar shape to the one above, that is, the model is still converging, especially for supporting facts, at the 10th epoch.

Below is a diagram to shoe model performances with different dataset. Again, same reason as above, we only logged to epoch 5 since the Cail(with sequence similarity detected supporting facts) results are on another machine.

		Answer		Support Facts		Joint		At Epoch
		exact match	f1	exact match	f1	exact match	f1	
Original Data	dev	0.6409	0.7181	0.3829	0.5733	0.2817	0.4184	5
with CAIL	dev	0.6806	0.7707	0.3948	0.6317	0.3075	0.5038	
with Cail+custom_support_facts	dev	0.6885	0.7749	0.4504	0.6400	0.3472	0.5209	
Original Data(baseline)	dev	0.6865	0.7897	0.4583	0.7080	0.3651	0.5829	13
with CAIL	dev	0.7163	0.7993	0.5	0.6519	0.3948	0.5366	10

Observations:

1. With the Cail data combined to the training, the model significantly improved, thanks to the large amount of data. Usually, the larger the data set, the more we can extract insights that we trust from that data set.
2. Within the 5 epochs, both datasets that combined with CAIL2019 have significantly exceeded the performance without combining CAIL2019.
3. At epoch 10, the exact match score on both answer prediction and supporting fact prediction are much better than with the CAIL2019 data introduced. However, the f1 score for supporting facts is lower. One potential reason is that many of the QA pairs in CAIL2019, especially the ones when the answer is not a span, do not have supporting facts labeled. And this leads to lack of information and the model can identify supporting facts to not-supporting-facts. Since at epoch 5, with the similarity detected supporting facts, the model has its best performances on detecting supporting facts, we could assume that more labeled supporting facts are required if we want to have better learning results.
4. Due to the lack of logged data, it is too early to conclude if our supporting facts detection has brought positive or negative facts. If time applies, we could run more training iterations to verify.

6 Summary

With the provided framework and pipeline, we tested model performances with different pretrained Bert models. We also combine our original data with CAIL2019 data to improve the learning results with larger data.

The original data has its best performances on the dev data when we use pre-trained Bert “hfl/chinese-roberta-wwm-ext-large” to encode the textual tokens to vectors. When we combine the data with CAIL2019, within the 5 epochs, with or without detected supporting facts, have significantly exceeded the performance from the original data. From the loss diagrams, both datasets haven’t finished convergence within the 5 or 10 epochs, that is to say, we could expect even more significant enhancements by increasing the training iterations. Another interesting fact is, when we use CAIL that only has the supporting facts (at maximum 1 for each QA pair), the prediction on the supporting facts becomes weaker. This is potentially due to the lack of labeled supporting facts, which leads to lack of information and the model can identify supporting facts to not-supporting-facts. Below are three metrics for conclusion.

		Answer		Support Facts		Joint		At Epoch
		exact match	f1	exact match	f1	exact match	f1	
with Cail+custom_support_facts	dev	0.6885	0.7749	0.4504	0.6400	0.3472	0.5209	5
Original Data(baseline)	dev	0.6865	0.7897	0.4583	0.7080	0.3651	0.5829	13
with CAIL	dev	0.7163	0.7993	0.5	0.6519	0.3948	0.5366	10

In this assignment, we also explore the provided code base and try to understand the architecture of the joint modal. A limitation of our architecture is that, the detection on answer types, extraction on answer spans and supporting facts are updated in a comparatively independent manner. However, these three,

by the nature of how human beings interpret and answer questions, are closely related to each other. Due to the time limit, there is a method that worth exploring in the future would be QA-GNN, using the knowledge graphs for QA tasks, and updating the learning on QA and entities and relations more synchronously.

7 References

[1] Question answering https://en.wikipedia.org/wiki/Question_answering#:~:text=Question%20answering%20(QA)%20is%20a,humans%20in%20a%20natural%20language.

[2] HotpotQA https://hotpotqa.github.io/

[3] Qian, Y. et al. (2022) 'Question-driven graph fusion network for visual question answering', 2022 IEEE International Conference on Multimedia and Expo (ICME) [Preprint]. doi:10.1109/icme52920.2022.9859591.

[4] https://github.com/ymcui/Chinese-BERT-wwm?tab=readme-ov-file

[5] HotpotQA https://hotpotqa.github.io/

[6] GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks https://arxiv.org/pdf/1711.02257.pdf

[7] https://github.com/china-ai-law-challenge/CAIL2019

[8] Yasunaga, M. et al. (2021) 'Qa-GNN: Reasoning with language models and knowledge graphs for question answering', Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies [Preprint]. doi:10.18653/v1/2021.naacl-main.45.