Pretrained NLP Models

– Pre-trained NLP Models Implementation on Relation Extraction

Abstract

BERT is a pre-trained language model proposed by Google in 2018, it has achieved reliable results on various NLP tasks. It has been widely used in the NLP field since it is convenient to transfer and adapt to other tasks or data. In this assignment, we will explore how to transfer BERT to extract relations on document-level.

Our model with pretrained Bert has its best performance when the batch size is 8 and when all the samples in the train set are used during training. The more samples we used for training, the better generalization the model has.

Another strategy for making predictions is introducing a threshold. The threshold is updated by the relation prediction probability at the time when the model has reached its best performances during training. This is effective to filter out the predicted relations with too small probabilities, avoiding false identifying non-relation to some random relations.

We tested different batch sizes and how many samples were used for training. The hyperparameter we choose for Bert is batch 8, learning rate 4e-5. The output has a score at 0.5699 on the test set, evaluated by the competition website for DocRed. We used the same hyperparameter but switched the pretrained model to Roberta, and the performance was improved with a score at 0.5873.

1 Introduction

Relation Extraction(RE) is the task of predicting attributes and relations for entities in a sentence. It is the key component for building relation knowledge graphs, and it is of crucial significance to natural language processing applications such as structured search, sentiment analysis, question answering, and summarization[1]. The main task here is Document-level Relation Extraction (DocRE). The DocRE task is more challenging than its sentence-level counterpart in the following aspects: (1) The complexity of DocRE increases quadratically with the number of entities; (2) Aside from the imbalance of positive and negative examples, the distribution of relation types for the positive entity pairs is also highly imbalanced[2].

DocRED dataset is the one for training and testing the model. It is a relation extraction dataset constructed from Wikipedia and Wikidata. Each document in the dataset is human-annotated with named entity mentions, coreference information, intra- and inter-sentence relations, and supporting evidence. In total, it contains 132,375 entities and 56,354 relational facts annotated on 5,053 Wikipedia documents. In addition to the human-annotated data, the dataset provides large-scale distantly supervised data over 101,873 documents[3]. Format examples for DocRed Data can reference to [4].

A fully functional code framework and data preparation, training and testing workflow has been provided. Besides tuning hyperparameters and running experiments, we need to answer the questions commented in the code base for butter understanding.

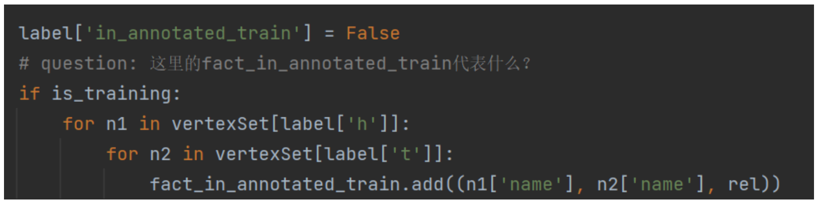
2 Code Understanding

The textual samples are tokenized/splitted to subwords by pretrained BERT model. These subwords are converted to IDs in the preprocessing stage, in addition we also calculate the position encoding for the subwords.

Implementation for gradient accumulation[5] is provided to overcome memory limitations when training large models or processing large batches of data. In the code itself, we could also use random sampling to only train on parts of the data(depending on the hardware).

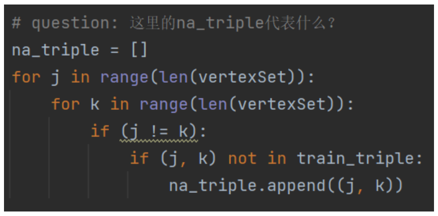
This section answers the questions commented in the code framework. All questions listed below follow the occurrence order in the original code.

2.1 Module gen_data.py



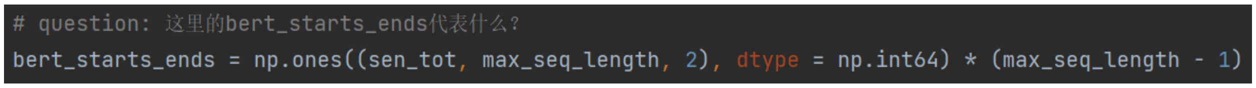
```
label['in_annotated_train'] = False
# question: 这里的fact_in_annotated_train代表什么？
if is_training:
    for n1 in vertexSet[label['h']]:
        for n2 in vertexSet[label['t']]:
            fact_in_annotated_train.add((n1['name'], n2['name'], rel))
```

A: The fact_in_annotated_train is a collection of triples(head_entity, tail_entity, relation) that appear in the annotated training data. It works like a look-up table to process the data in dev and test files.



```
# question: 这里的na_triple代表什么？
na_triple = []
for j in range(len(vertexSet)):
    for k in range(len(vertexSet)):
        if (j != k):
            if (j, k) not in train_triple:
                na_triple.append((j, k))
```

A: The train_triple is a collection to store all the (head_entity, tail_entity) pairs. If such a pair extracted from the given data is not in the train_triple, we will add it to the na_triple.

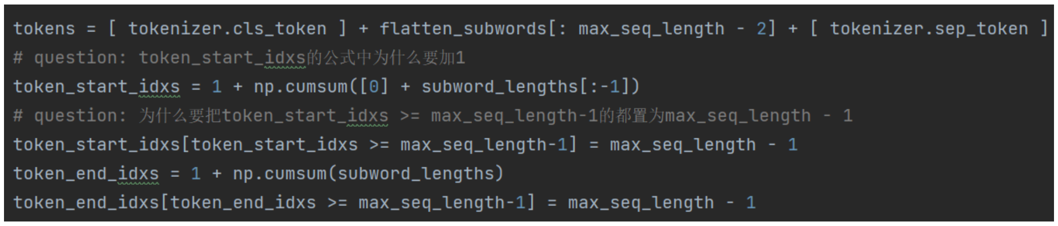


```
# question: 这里的bert_starts_ends代表什么？
bert_starts_ends = np.ones((sen_tot, max_seq_length, 2), dtype = np.int64) * (max_seq_length - 1)
```

A: Each sample in the data set contains a sents list, and the text content is broken down into single words and stored in the sents list. By using a pre-trained tokenizer, we map the words in the list to a subword. A subword may consist of one or more words. These subwords will be stored in the tokens list.

Before position embedding, we need to insert BERT's special characters '[CLS]' and '[SEP]' at the beginning and end of the tokens list. At the same time, in order to avoid the sequence being too long, we introduce max_seq_length to trim the token sequence that is longer than the defined max_seq_length. It should be noted that the first subword of a tokens list must be '[CLS]', and preferably the first subword must be '[SEP]'.

The “bert_starts_ends” is a matrix we initialize for storing position embedding of each subword in the tokens list, and the initial values are all set to the max_seq_length we gave; the reason why it is not initialized to all 1 is that the characters encoded as position 1 in tokens are always '[CLS]' characters.

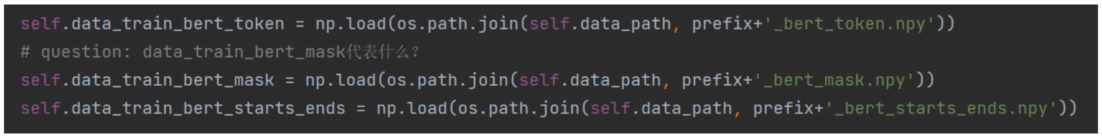


```
tokens = [ tokenizer.cls_token ] + flatten_subwords[: max_seq_length - 2] + [ tokenizer.sep_token ]
# question: token_start_idxs的公式中为什么要加1
token_start_idxs = 1 + np.cumsum([0] + subword_lengths[:-1])
# question: 为什么要把token_start_idxs >= max_seq_length-1的都置为max_seq_length - 1
token_start_idxs[token_start_idxs >= max_seq_length-1] = max_seq_length - 1
token_end_idxs = 1 + np.cumsum(subword_lengths)
token_end_idxs[token_end_idxs >= max_seq_length-1] = max_seq_length - 1
```

A1: Since the tokens will always start with '[CLS]', the first actual token from the tokenization will always have position 1 as the start position index(embedding), and so on, we will need to add 1 to the calculation of the start index.

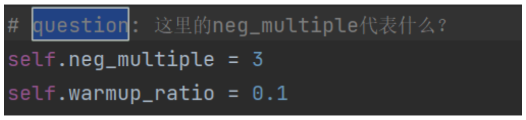
A2: This is a trim policy for avoiding the sequence being too long. If the subword starts at an index that already exceeds the longest length we allowed, we will identify this as the last position of the built sequence, that is “[SEP]” to determine the separation of two sentences in the same input.

2.2 Module Config.py



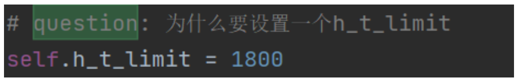
```
self.data_train_bert_token = np.load(os.path.join(self.data_path, prefix+'_bert_token.npy'))
# question: data_train_bert_mask代表什么?
self.data_train_bert_mask = np.load(os.path.join(self.data_path, prefix+'_bert_mask.npy'))
self.data_train_bert_starts_ends = np.load(os.path.join(self.data_path, prefix+'_bert_starts_ends.npy'))
```

A: The “data_train_bert_mask” is a numpy array loaded from a saved file that is generated in the data preprocessing. The array has values 1 at every position that corresponds to a subword, and 0 at every position that is a padding.



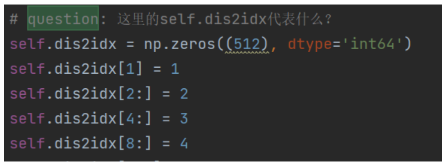
```
# question: 这里的neg_multiple代表什么?
self.neg_multiple = 3
self.warmup_ratio = 0.1
```

A: Used for calculating the sample amount for taking negative samples to training. It is basically specifying the proportion of negative samples(pairs that have “na” relation) to take.



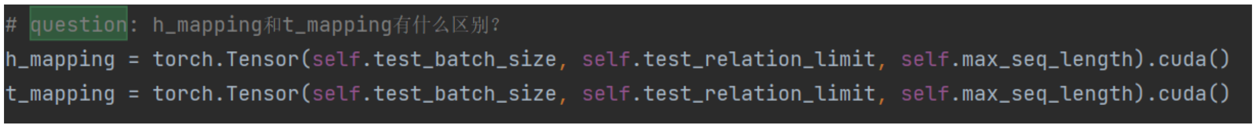
```
# question: 为什么要设置一个h_t_limit
self.h_t_limit = 1800
```

A: To limit the amount of (head entity, tail entity) pairs that appear in one batch of training samples. It is a strategy to capture useful relational semantics from long documents, since most contents in the documents may be irrelevant to the given entities and relations.



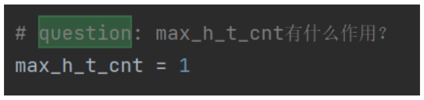
```
# question: 这里的self.dis2idx代表什么?
self.dis2idx = np.zeros((512), dtype='int64')
self.dis2idx[1] = 1
self.dis2idx[2:] = 2
self.dis2idx[4:] = 3
self.dis2idx[8:] = 4
```

A: Mapping the distance to index; distance values falling in the same range will have the same distance index. It is used to calculate the tail entity's relative position embedding to the given head entity.



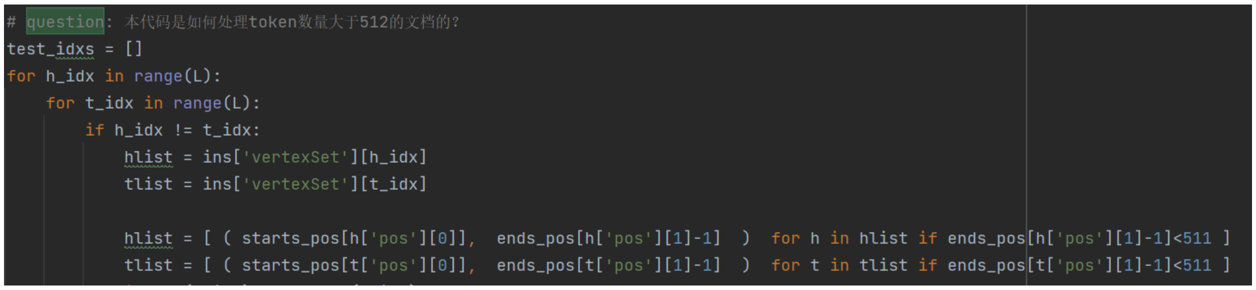
```
# question: h_mapping和t_mapping有什么区别?
h_mapping = torch.Tensor(self.test_batch_size, self.test_relation_limit, self.max_seq_length).cuda()
t_mapping = torch.Tensor(self.test_batch_size, self.test_relation_limit, self.max_seq_length).cuda()
```

A: The "h_mapping" is for storing position embed mapping for head entities. The "t_mapping" is for storing position embed mapping for tail entities. The mapping here is the position embedding of an entity relative to the given long text.



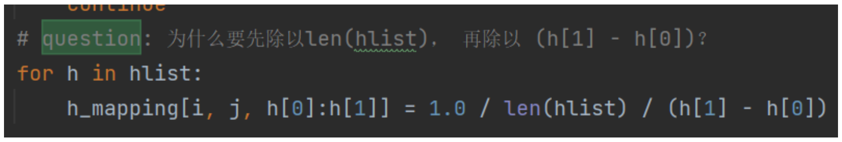
```
# question: max_h_t_cnt有什么作用?
max_h_t_cnt = 1
```

A: Used to return the actual data embedding/mapping matrix after processing the batch data. It is initialized to 1 since we want to avoid returning empty matrices due to there being no (head, tail) entity pair in the given instance.



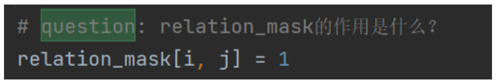
```
# question: 本代码是如何处理token数量大于512的文档的?
test_idxs = []
for h_idx in range(L):
    for t_idx in range(L):
        if h_idx != t_idx:
            hlist = ins['vertexSet'][h_idx]
            tlist = ins['vertexSet'][t_idx]

            hlist = [ ( starts_pos[h['pos'][0]],  ends_pos[h['pos'][1]-1]  )  for h in hlist if ends_pos[h['pos'][1]-1]<511 ]
            tlist = [ ( starts_pos[t['pos'][0]],  ends_pos[t['pos'][1]-1]  )  for t in tlist if ends_pos[t['pos'][1]-1]<511 ]
```

A: Clipped that subwords in the document to the given max length(512). Not considered. More details can be seen from the answers in Section 2.1.



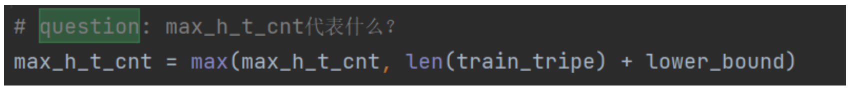
```
# question: 为什么要先除以len(hlist), 再除以 (h[1] - h[0])?
for h in hlist:
    h_mapping[i, j, h[0]:h[1]] = 1.0 / len(hlist) / (h[1] - h[0])
```

A: The "hlist" stores the start position and the end position for all head entity mentions(whose end position does not exceed the max length 512). Here, the division performs a form of mapping from the entity word positions to its relative position in the document. The division by length of the "hlist" is the positional embed mapping relative to the whole document(with max length 512), and the division by (h[1]-h[0]) is to average out the impact of the entity itself’s length, since we see it as an entity, rather than a group of words.



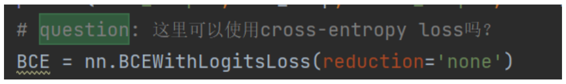
```
# question: relation_mask的作用是什么？
relation_mask[i, j] = 1
```

A: The question here is in the definition of the method to get test batch data. It is not used anywhere for the test batch. But based on its usage in the train batch preparation and actual training process, if the mask is set to 0, the loss from the prediction of the given (head, tail) will not participate in the calculation of the current batch loss. RE suffers from unintended entity bias, i.e. the spurious correlation between entity mentions (names) and relations. Entity bias can mislead the RE models to extract the relations that do not exist in the text. To combat this issue, masking the entity mentions can help prevent the RE models from overfitting entity mentions.



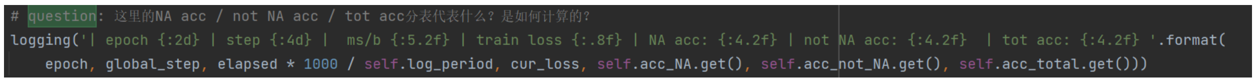
```
# question: max_h_t_cnt代表什么？
max_h_t_cnt = max(max_h_t_cnt, len(train_tripe) + lower_bound)
```

A: If the amount of negative samples(pairs that have no relation) and the positive samples taken into the model for training is more than 1, take all these samples. Else, make sure not to input empty data to training for each batch run.



```
# question: 这里可以使用cross-entropy loss吗？
BCE = nn.BCEWithLogitsLoss(reduction='none')
```

A: Need to do a sigmoid over the model output first, then flush into binary cross entropy loss as the prediction. The BCEWithLogitsLoss is essentially just (binary)cross entropy loss that comes inside a sigmoid function.

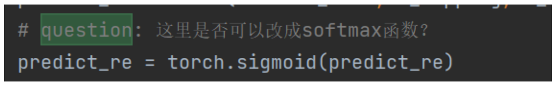


```
# question: 这里的NA acc / not NA acc / tot acc分表代表什么？是如何计算的？
logging('| epoch {:2d} | step {:4d} |  ms/b {:5.2f} | train loss {:.8f} | NA acc: {:4.2f} | not NA acc: {:4.2f}  | tot acc: {:4.2f} '.format(
    epoch, global_step, elapsed * 1000 / self.log_period, cur_loss, self.acc_NA.get(), self.acc_not_NA.get(), self.acc_total.get()))
```

A: This is the evaluation metric for measuring training performance. The prediction is made by choosing the relationship that has the greatest probability and checking if it matches the given label in the data.

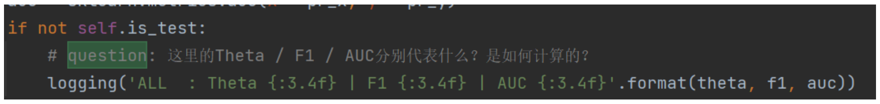
- “NA acc”:
  - Calc: (the amount of no-relation entity pairs that actually predict as no_relation)/(amount of entity pairs that has no relation in between)
  - Evaluates if the model can correctly identify that two entities have no relationship
- “not NA acc”:
  - Calc: (the amount of entity pairs that have relation in between and the relation got predicted correctly)/(amount of entity pairs that has relation in between)
  - Evaluates if the model can correctly identify that two entities have certain relationship and can correctly find the relationship
- “tot acc”:
  - Calc: (the amount of entity pairs whose relation is predicted correctly)/(amount of entity pairs)

- Evaluates the overall performance of the model, the higher the score is, the better the model at extracting and identifying the relationship, and will not mistakenly recognize the small probability relation as the actual relationship(actually no relation).



```
# question: 这里是否可以改成softmax函数?
predict_re = torch.sigmoid(predict_re)
```

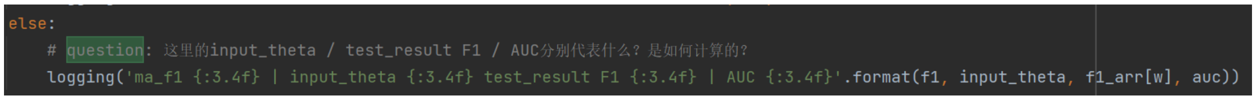
A: No, if we use BCEWithLogitsLoss or cross-entropy loss, it applies Sigmoid not Softmax.



```
if not self.is_test:
    # question: 这里的Theta / F1 / AUC分别代表什么? 是如何计算的?
    logging('ALL  : Theta {:3.4f} | F1 {:3.4f} | AUC {:3.4f}'.format(theta, f1, auc))
```

A: This is the performance evaluation on all the relation facts in the dev set.

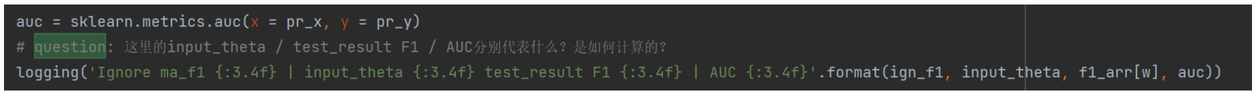
- Theta: Maximizing the F1 is to maximize recall and precision. We detected the time step with the highest F1 score and found its corresponding probability value for the relation in the prediction for this time being, and assigned this probability to theta. It will be the threshold to filter small probability relations that are predicted by the model.
- F1: The F1 here is calculated by precision and recall. The precision(denoted as y in the code) is calculated by the division of (amount of relation facts that correctly predicted) and (amount of all the relation facts inferred by the model), specifying the true positive rate at the current time step. The recall(denoted as x in the code) is calculated by the division of (amount of relation facts that correctly predicted) and (amount of all the relation facts that exist in the dev data), similar to the false positive rate. Then the general F1 formula used this precision and recall value to calculate the F1 score. F1 here is the highest F1 score the model has achieved.
- AUC: AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.We use the precision(calculation method see above) as the y-axis coordinate and the recall as the x-axis coordinate to draw the ROC curve. The area between this curve and the x-axis is our AUC score(sklearn metric is used here).



```
else:
    # question: 这里的input_theta / test_result F1 / AUC分别代表什么? 是如何计算的?
    logging('ma_f1 {:3.4f} | input_theta {:3.4f} test_result F1 {:3.4f} | AUC {:3.4f}'.format(f1, input_theta, f1_arr[w], auc))
```

A: Evaluation for the performances if on the test data.

- Input_theta: When the model is not set to test, we initialize the input theta as -1, and update it with the probability value for the relation in the prediction at the time when the F1 score reaches the highest. This suggests that we are going to find the threshold value based on the learning. Essentially, it functions as a threshold on the test set, to detect the predicted relation that has comparatively small probability.
- test_result_F1: Still it is the highest F1 score the model has achieved.
- AUC: Same as above answer. It is the AUC score for the model performance when in test.



```
auc = sklearn.metrics.auc(x = pr_x, y = pr_y)
# question: 这里的input_theta / test_result F1 / AUC分别代表什么？是如何计算的？
logging('Ignore ma_f1 {:3.4f} | input_theta {:3.4f} test_result F1 {:3.4f} | AUC {:3.4f}'.format(ign_f1, input_theta, f1_arr[w], auc))
```

A: According to the paper for DocRed[6], some relation facts are present in both the training and dev/test sets, therefore the model may memorize these relations during training and achieve a better performance during validation/test, which is undesirable. Scores in this section are calculated similarly to the upper question, the difference would be that we remove the relation facts(entity-relation triples) that present in both the training set and the dev/testing set.

The precision(denoted as y in the code) is calculated by the division of (amount of relation facts that correctly predicted and not exists in the training set) and (amount of all the relation facts inferred by the model and not exists in the training set). The recall(denoted as x in the code) is calculated by the division of ((amount of relation facts that correctly predicted and not exists in the training set) and (amount of all the relation facts that exist in the dev/test data)[7].

- Input_theta: We initialize the input theta as -1, and update it with the probability value for the relation in the prediction at the time when the F1 score reaches the highest. For the predicted relations(triples), we sort in descending order based on the calculated probability. The input_theta here is used here as a threshold to find the last predicted relation that has probability greater than the threshold. When not on test data, it will start with -1 and explore the value based on the training/dev performance. On the test set, the theta from the training/dev will be the input theta for prediction inference. This is to avoid wrongly identify non-relation to relations with small probabilities.
- test_result_F1: This is where we used the input_theta as a threshold, to get the F1 score at the time step when the model's relation probability has reached the bottom line(when the F1 score reaches the highest).
- AUC: AUC for the model without the impact from the relation facts that represented also in the training set. This is a more truthful AUC compared to the AUC that simply measures on all the dev/test data. The calculation is above[7].

3 Parameter Tuning and Model Performances

The sample point amount for the data is not as small, and requires certain hardware standards to run the experiments, let alone the time spent for each iteration. Therefore, we will not go through all the parameters and tune them one by one.

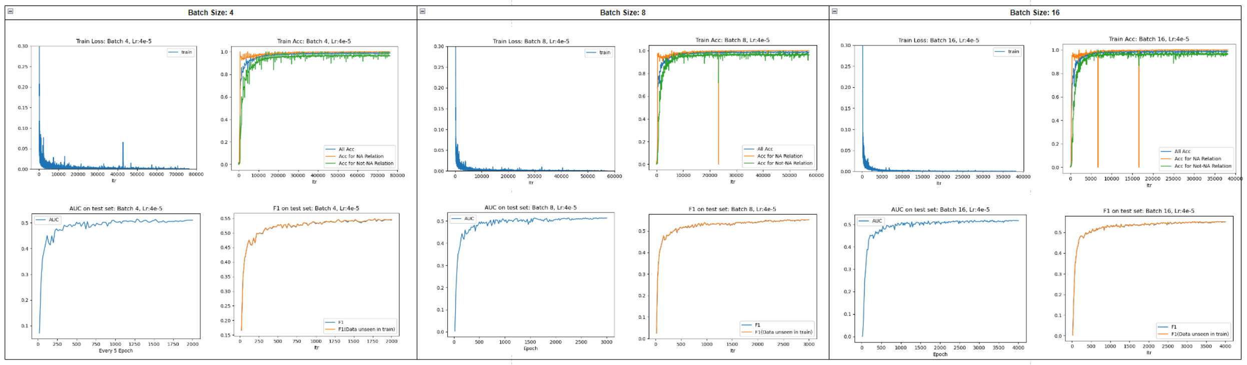
The performance in this section is evaluated on the training and dev sets. We will choose the hyperparameters that produce the best learning results to run on the test set.

3.1 Batch Size

We run batch size 4, 8, 16 on the full training data set. Other hyperparameters used the default values provided by the code. The "bert-base-cased" is used as the pretrained model for all these three batch sizes.

Below diagram is the scores from training and dev set. The “test set” in the curves diagrams is actually referred to as the “dev set”.

Model Type	Model Name	Learning Rate	Batch Size	Theta	F1	Ignore F1	AUC	Ignore AUC	Best Epoch	Best Epoch F1	Best Epoch AUC
Bert	bert-base-cased	4e-5	4	0.2294	0.5682	0.5436	0.5437	0.5107	88	0.5706	0.5077
Bert	bert-base-cased	4e-5	8	0.3779	0.5750	0.5518	0.5452	0.5130	147	0.5755	0.5130
Bert	bert-base-cased	4e-5	16	0.5817	0.5738	0.5517	0.5495	0.5166	189	0.5747	0.5171



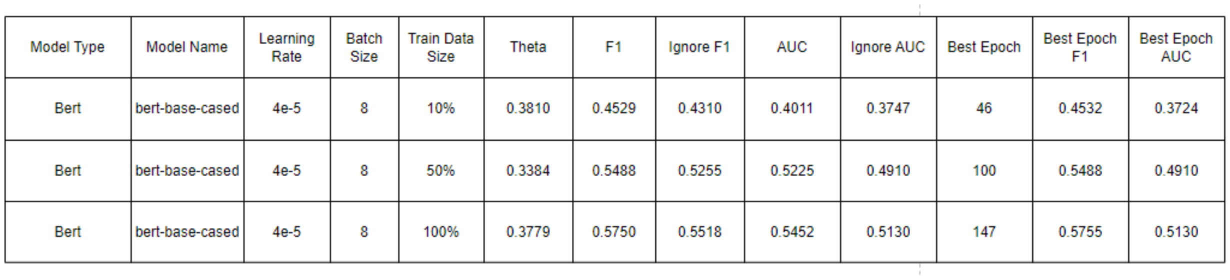
Ignore F1 and AUC refers to the scores when we exclude the relation facts that are represented in both the training and the dev sets during calculation.

Observations:

1. Models with smaller batch sizes reach their best performances earlier than the models with large epochs. Smaller batches tend to converge faster, they allow the model to begin learning before seeing a large amount of data.
2. The AUC curve for non-relation prediction(inference quality on the entity pairs that have no relations) has a comparatively smooth trend when the batch size is set to 4. When it comes to batch size 16, we could see that there are two large fluctuations on the na-relation AUC curve, almost directly falling to 0 and immediately climbing up. Using a large batch size will create your agent to have a very sharp loss landscape. And this sharp loss landscape is what will drop the generalization ability of the network[8].
3. The F1 and AUC scores are generally better on all the dev data, compared to the scores when we exclude the relation facts that are represented in both the training and dev set. This phenomenon is mentioned in Section 2, the model may memorize these relations during training and achieve a better performance during validation/test.
4. AUC and F1 scores are close with different batch sizes. In order to get better observations on which one is better, we should perform a statistical significance test. Sadly, we do not have the time to do so. We choose batch size 8 for later experiments; it has the highest Ignore F1 and AUC on the dev set after the training is complete, suggesting that it has a good generalization ability.

3.2 Training Data Size

A more rigorous way of comparing the model performances based on how resourceful the training data is should also be involved in tweaking on the batch size and other hyperparameters. However, we only use batch size 8 to test here since we do not have enough time.



Model Type	Model Name	Learning Rate	Batch Size	Train Data Size	Theta	F1	Ignore F1	AUC	Ignore AUC	Best Epoch	Best Epoch F1	Best Epoch AUC
Bert	bert-base-cased	4e-5	8	10%	0.3810	0.4529	0.4310	0.4011	0.3747	46	0.4532	0.3724
Bert	bert-base-cased	4e-5	8	50%	0.3384	0.5488	0.5255	0.5225	0.4910	100	0.5488	0.4910
Bert	bert-base-cased	4e-5	8	100%	0.3779	0.5750	0.5518	0.5452	0.5130	147	0.5755	0.5130

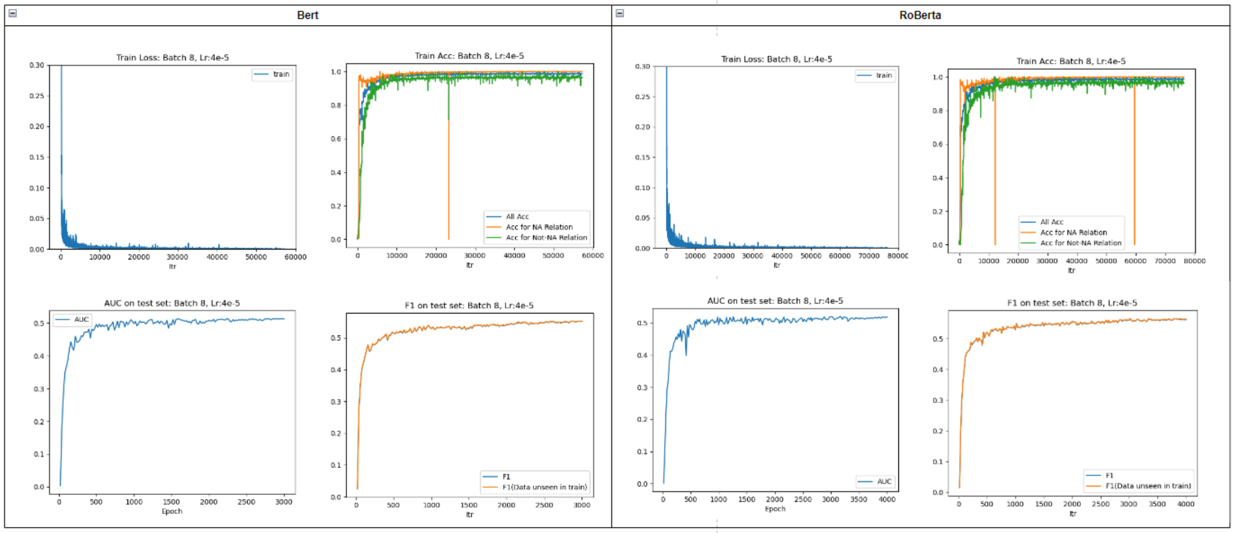
Observations:

1. The model has its worst performance when we only have 10% training samples to train, this represents the situation for low-resources. As the amount of samples increases, the model starts to perform better.
2. Another interesting fact is that the better the model performs, the smaller the theta is. Theta represents the predicted probability at the moment when the model reaches its best F1 score. High F1 scores indicate the strong overall performance of a classification model. It signifies that the model can effectively identify positive cases while minimizing false positives and false negatives. The fact observed here suggests that the model could correctly identify the difference between low probability and no-relation.
3. As appeared here, large training data has better generalization ability. Deep learning can still be effective on small datasets: Even though deep learning typically requires large amounts of data, it can still be effective on small datasets if the data is high-quality and relevant to the problem; however, small datasets it is easier for deep learning algorithms to overfit to the training data and perform poorly on unseen data[9].

3.3 Bert vs RoBerta

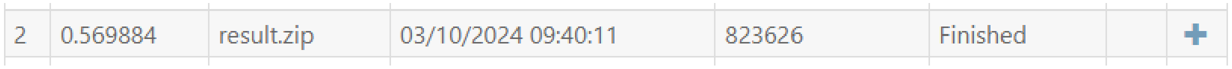
We use the full training data and batch size 8 to compare the performances with pretrained Bert and Roberta.

Model Type	Model Name	Learning Rate	Batch Size	Theta	F1	Ignore F1	AUC	Ignore AUC	Best Epoch	Best Epoch F1	Best Epoch AUC
Bert	bert-base-cased	4e-5	8	0.3779	0.5750	0.5518	0.5452	0.5130	147	0.5755	0.5130
RoBerta	FacebookAI/roberta-base	4e-5	8	0.4267	0.5854	0.5623	0.5512	0.5178	189	0.5872	0.5175



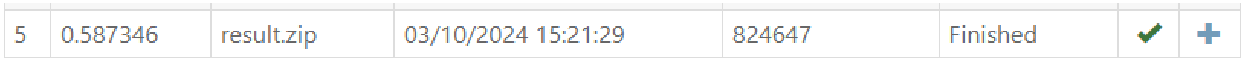
We also submitted the model output on the test set to the CodaLab-Competition[10] website, below are the results.

For Bert:



2	0.569884	result.zip	03/10/2024 09:40:11	823626	Finished		+

For Roberta:



5	0.587346	result.zip	03/10/2024 15:21:29	824647	Finished	✓	+

With Roberta as the pretrained model, the relation extraction is slightly better than with Bert.

RoBERTa is pretrained on a combination of five massive datasets resulting in a total of 160 GB of text data. In comparison, BERT large is pre-trained only on 13 GB of data. Finally, the authors increase the number of training steps from 100K to 500K. As a result, RoBERTa outperforms BERT large on XLNet large on the most popular benchmarks[11].

This is the reason why using Roberta tokenizer has better scores and generalization. The Roberta uses a more aggressive BPE algorithm compared to BERT, leading to a larger number of sub-word units and a more fine-grained representation of the language. It makes RoBERTa more acceptable compared to BERT on a variety of NLP tasks.

4 Summary

We tested different batch sizes and how many samples were used for training. Our model with pretrained Bert has its best performance when the batch size is 8 and when all the samples in the train set are used during training. The more samples we used for training, the better generalization the model has.

Our model is evaluated by two popular metrics, AUC and F1. Since some of the relation facts are presented in both the training and the dev set, we also get the AUC and F1 with these relation faces excluded from the calculation. This is to avoid fake high scores due to the model remembering these relation facts from training.

Another strategy is introducing a threshold. The threshold is updated by the relation prediction probability at the time when the model has reached its best performances during training. This is effective to filter out the predicted relations with too small probabilities, avoiding false identifying non-relation to some random relations.

The hyperparameter we choose for Bert is batch 8, learning rate 4e-5. The output has a score at 0.5699 on the test set, evaluated by the competition website for DocRed. We used the same hyperparameter but switched the pretrained model to Roberta, and the performance was improved with a score at 0.5873.

We also looked into a few other algorithms for relation extraction on document level, such as “DeepKE: A Deep Learning Based Knowledge Extraction Toolkit for Knowledge Base Population”[12]. We do not have the time to explore these ideas in this assignment, but they are ideas that are worth exploring in future.

5 References

[1] Relation Extraction https://paperswithcode.com/task/relation-extraction

[2] Tan, Q. et al. (2022) ‘Document-level relation extraction with adaptive focal loss and knowledge distillation’, Findings of the Association for Computational Linguistics: ACL 2022 [Preprint]. doi:10.18653/v1/2022.findings-acl.132.

[3] DocRED https://paperswithcode.com/dataset/docred

[4] https://zhuanlan.zhihu.com/p/356077381

[5] Gradient Accumulation and Gradient Checkpointing https://aman.ai/primers/ai/grad-accum-checkpoint/

[6] DocRED: A Large-Scale Document-Level Relation Extraction Dataset. ACL 2019

[8] Why Small Batch sizes lead to greater generalization in Deep Learning https://medium.com/geekculture/why-small-batch-sizes-lead-to-greater-generalization-in-deep-learning-a00a32251a4f

[9] What are the advantages and disadvantages of using Deep Learning to train on small datasets? https://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-Deep-Learning-to-train-on-small-datasets

[10] DocRED https://codalab.lisn.upsaclay.fr/competitions/365#participate-submit_results

[11] Large Language Models: RoBERTa — A Robustly Optimized BERT Approach https://towardsdatascience.com/roberta-1ef07226c8d8#:~:text=RoBERTa%20is%20pretrained%20on%20a,steps%20from%20100K%20to%20500K.

[12] Zhang, N. et al. (2022) 'Deepke: A deep learning based knowledge extraction toolkit for knowledge base population', Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations [Preprint]. doi:10.18653/v1/2022.emnlp-demos.10.