CS 613: NLP Assignment 3

Pretraining and fine-tuning an LLM

Team 8

Assigned Assignment Processing

Aamod Thakur | 23210001

Bhautik Vala | 23120002

Chirag Modi | 23210031

Dhruv Khandelwal | 23120003

Kareena Beniwal | 20110095

Priya Gupta | 20110147

Rakesh Thakur| 20210013

Assumptions and Conclusions:

1. Selected the Bert-base-uncased model

Selected bert-base-uncased model

2. Pretrain model parameters

-> We have selected the bert-base-uncased model for pretraining. We calculated the total parameters of the pretrain model using the following code. BertPretraining.ipynb file contains this code.

```
from transformers import BertForPreTraining, BertConfig,AutoModelForSequenceClassification,AutoModel
num_classes = 5
model = AutoModel.from_pretrained("Bhautiksinh/BertPretrain", num_labels=num_classes)
import torch
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
# Assuming your fine-tuned BERT model is named 'model'
model = model.to(device)  # Move the model to the desired device
params={}
def print_parameters(model):
    total_params = 0
    for name, param in model.named_parameters():
        if param.requires_grad:
            num_params = param.numel()
            params[name]=param.numel()
            total_params += num_params
            print(f"{name}: {num_params}")
    #print(params)
    print(f"Total Parameters: {total_params}")

# Print all parameter names and sizes
print_parameters(model)
```

-> We are getting a total of 109482240 parameters. As per official paper, parameters should be 110M. So it is nearly the same.

```python
param_count = {}
for n,p in model.named_parameters():
    param_count[n] = p.numel()
```

```python
param_count
```

```
{'bert.embeddings.word_embeddings.weight': 23440896,
 'bert.embeddings.position_embeddings.weight': 393216,
 'bert.embeddings.token_type_embeddings.weight': 1536,
 'bert.embeddings.LayerNorm.weight': 768,
 'bert.embeddings.LayerNorm.bias': 768,
 'bert.encoder.layer.0.attention.self.query.weight': 589824,
 'bert.encoder.layer.0.attention.self.query.bias': 768,
 'bert.encoder.layer.0.attention.self.key.weight': 589824,
 'bert.encoder.layer.0.attention.self.key.bias': 768,
 'bert.encoder.layer.0.attention.self.value.weight': 589824,
 'bert.encoder.layer.0.attention.self.value.bias': 768,
 'bert.encoder.layer.0.attention.output.dense.weight': 589824,
 'bert.encoder.layer.0.attention.output.dense.bias': 768,
 'bert.encoder.layer.0.attention.output.LayerNorm.weight': 768,
 'bert.encoder.layer.0.attention.output.LayerNorm.bias': 768,
 'bert.encoder.layer.0.intermediate.dense.weight': 2359296,
 'bert.encoder.layer.0.intermediate.dense.bias': 3072,
 'bert.encoder.layer.0.output.dense.weight': 2359296,
```

```
 'bert.encoder.layer.0.output.LayerNorm.bias': 768,
 'bert.encoder.layer.1.attention.self.query.weight': 589824,
 'bert.encoder.layer.1.attention.self.query.bias': 768,
 'bert.encoder.layer.1.attention.self.key.weight': 589824,
 'bert.encoder.layer.1.attention.self.key.bias': 768,
 'bert.encoder.layer.1.attention.self.value.weight': 589824,
 'bert.encoder.layer.1.attention.self.value.bias': 768,
 'bert.encoder.layer.1.attention.output.dense.weight': 589824,
 'bert.encoder.layer.1.attention.output.dense.bias': 768,
 'bert.encoder.layer.1.attention.output.LayerNorm.weight': 768,
 'bert.encoder.layer.1.attention.output.LayerNorm.bias': 768,
 'bert.encoder.layer.1.intermediate.dense.weight': 2359296,
 'bert.encoder.layer.1.intermediate.dense.bias': 3072,
 'bert.encoder.layer.1.output.dense.weight': 2359296,
 'bert.encoder.layer.1.output.dense.bias': 768,
 'bert.encoder.layer.1.output.LayerNorm.weight': 768,
 'bert.encoder.layer.1.output.LayerNorm.bias': 768,
 'bert.encoder.layer.2.attention.self.query.weight': 589824,
 'bert.encoder.layer.2.attention.self.query.bias': 768,
 'bert.encoder.layer.2.attention.self.key.weight': 589824,
 'bert.encoder.layer.2.attention.self.key.bias': 768,
 'bert.encoder.layer.2.attention.self.value.weight': 589824,
 'bert.encoder.layer.2.attention.self.value.bias': 768,
 'bert.encoder.layer.2.attention.output.dense.weight': 589824,
 'bert.encoder.layer.2.attention.output.dense.bias': 768,
 'bert.encoder.layer.2.attention.output.LayerNorm.weight': 768,
 'bert.encoder.layer.2.attention.output.LayerNorm.bias': 768,
 'bert.encoder.layer.2.intermediate.dense.weight': 2359296,
 'bert.encoder.layer.2.intermediate.dense.bias': 3072,
 'bert.encoder.layer.2.output.dense.weight': 2359296,
 'bert.encoder.layer.2.output.dense.bias': 768,
 'bert.encoder.layer.2.output.LayerNorm.weight': 768,
```

Similarly we got the same for other layers. We have a total of twelve such layers.

```
 'bert.encoder.layer.11.output.LayerNorm.bias': 768,
 'bert.pooler.dense.weight': 589824,
 'bert.pooler.dense.bias': 768,
 'cls.predictions.bias': 30522,
 'cls.predictions.transform.dense.weight': 589824,
 'cls.predictions.transform.dense.bias': 768,
 'cls.predictions.transform.LayerNorm.weight': 768,
 'cls.predictions.transform.LayerNorm.bias': 768,
 'cls.seq_relationship.weight': 1536,
 'cls.seq_relationship.bias': 2}
```

3. Pretrain ‘bert-base-uncased’ Model

-> For pretraining, we are importing only the architecture of ‘bert-base-uncased’ model with random weight.
-> We have used BertTokenizer for tokenization.
-> Then we used mask language modeling(MLM) and next sentence prediction(NSP). We have used ‘wikitext-2-raw-v1’ data of hugging face dataset.

1. Mask Language Modeling(MLM)

-> For MLM, we randomly selected 15% tokens of training data and among them 80% are replaced with [MASK] tokens and 10% are replaced with random tokens of training data and 10% are kept as it is.

2. Next Sentence Prediction(NSP)

-> For NSP, we created a pair of sentences, in which sentence 2 is the next sentence of sentence 1. 50% of the time sentence 2 is actually next to sentence 1. And 50% of the time sentence 2 is a random sentence of training corpus.

-> We are doing both modeling together.

4. Perplexity of pre trained model

-> We pre-trained the model on 5 epochs and for each epoch, we find perplexity on given test data. We are getting the following perplexity scores.

Epoch 1:- 10726.49
Epoch 2:- 14830.29
Epoch 3:- 15701.38

Epoch 4:- 15030.8
Epoch 5:- 17430.77

-> We are getting perplexed in increasing order after each epoch, but ideally it should decrease as we train, because loss reduces after each epoch. This may occur due to some reasons which are mentioned below.

1. Our training data is noisy and the model is not picking up important patterns in the data. Also we pre trained the model from scratch , so this amount of data might not be sufficient to pretrain models effectively.

2. There might be some overfitting issues, where the model learns training data very well but is not able to work well with testing data.

5. Push the pre-trained model

-> Huggingface hub link:- https://huggingface.co/Bhautiksinh/BertPretrain

6. Fine-tuning

a. Classification

We have taken SST2-Data.
There were two text files:

1. **sentiment_labels.txt** contains all phrase ids and the corresponding sentiment labels, separated by a vertical line.
   Note that you can recover the 5 classes by mapping the positivity probability using the following cut-offs:
   [0, 0.2], (0.2, 0.4], (0.4, 0.6], (0.6, 0.8], (0.8, 1.0]
   for very negative(4), negative(3), neutral(2), positive(1), very positive(0), respectively.
2. **Sentlex_exp12**: contains phrase ids and the corresponding phrase

sentiment_labels

File Edit View

```
phrase ids|sentiment values
0|0.5
1|0.5
2|0.44444
3|0.5
4|0.42708
5|0.375
6|0.41667
7|0.54167
8|0.33333
9|0.45833
10|0.47222
11|0.59722
12|0.33333
13|0.93056
14|0.80556
15|0.81944
16|0.76389
17|0.5
18|0.5
19|0.69444
20|0.5
21|0.75
22|0.29167
23|0.31944
24|0.73611
```

sentlex_exp12

File Edit View

```
0,!
1,'
2,' (
3,' ( the cockettes
4,' ( the cockettes )
5,' ( the cockettes ) provides a win
6,' ( the cockettes ) provides a win
7,' ( the cockettes ) provides a win
8,' a nightmare on elm street
9,' a nightmare on elm street '
10,' a nightmare on elm street ' or
11,' a nightmare on elm street ' or
12,' a nightmare on elm street ' or
13,' a perfect family film
14,' a perfect family film &#44
15,' a perfect family film &#44 '
16,' a perfect family film &#44 ' be
17,''
18,'ll
19,'ll like it
20,'s
21,'s a certain style and wit to the
22,'s a demented kitsch mess ( altho
23,'s a pale imitation
24,'s a visual delight and a decent
25,'s a visual delight and a decent
26,'s a visual delight and a decent
27,'s about family
28,'s all pretty
29,'s all pretty tame
30,'s another retelling of alexandre
31,'s back-stabbing &#44 inter-racia
32,'s back-stabbing &#44 inter-racia
33,'s back-stabbing &#44 inter-racia
34,'s enough melodrama in this magno
35,'s enough melodrama in this magno
36,'s enough melodrama in this magno
37,'s enough to make one pine for th
38,'s frustrating to see these guys
39,'s frustrating to see these guys
```

We made a csv which looks like:

	A	B	C	D
1	phrase_id	sentiment_	class	phrase
2	0	0.5	2	!
3	1	0.5	2	'
4	2	0.44444	2	' (
5	3	0.5	2	' (the cockettes
6	4	0.42708	2	' (the cockettes)
7	5	0.375	3	' (the cockettes) provides a window into a subculture hell-bent on expressing itself in every way imaginable
8	6	0.41667	2	' (the cockettes) provides a window into a subculture hell-bent on expressing itself in every way imaginable .
9	7	0.54167	2	' (the cockettes) provides a window into a subculture hell-bent on expressing itself in every way imaginable . '
10	8	0.33333	3	' a nightmare on elm street
11	9	0.45833	2	' a nightmare on elm street '
12	10	0.47222	2	' a nightmare on elm street ' or
13	11	0.59722	2	' a nightmare on elm street ' or `
14	12	0.33333	3	' a nightmare on elm street ' or ` the hills
15	13	0.93056	0	' a perfect family film
16	14	0.80556	0	' a perfect family film &#44
17	15	0.81944	0	' a perfect family film &#44 '
18	16	0.76389	1	' a perfect family film &#44 ' because it 's about family
19	17	0.5	2	''
20	18	0.5	2	'll
21	19	0.69444	1	'll like it
22	20	0.5	2	's
23	21	0.75	1	's a certain style and wit to the dialogue

4 features: phrase_id, sentiment_value, class, phrase.

Hugging face hub link for classification task fine-tuned model: https://huggingface.co/KareenaBeniwal/Fine-tuned-bert-model-classification/tree/main

b. Question-Answering

We have taken SQuAD dataset for training and testing of model

- Dataset features: ['id', 'title', 'context', 'question', 'answers']
- We see that for the 'answers' column the dataset contains a dictionary with keys 'text' and 'answer_start', that each contain a list with one element.
- Imported libraries that were used in the notebook, define a seeding function and set the device to cuda if available.
- For the training dataset, I noticed that for each question there is only one answer, so there is no need to keep the values of the answers dictionary in lists. For example: 'answers': {'text': ['singing and dancing'], 'answer_start': [207]}} can be formatted to 'answers': {'text': 'singing and dancing', 'answer_start': 207}}. As for questions that are unanswerable (they look like this:'answers': {'text': [], 'answer_start': []}} we can just have 'answers': {'text': "", 'answer_start': 0}}.
- The training for each epoch took aprox. 2 hours so I couldn't try many epochs and do many runs when using the whole dataset.
- For optimizer, we used AdamW (Adam with weight decay).

  Hugging face hub link for classification task fine-tuned model: https://huggingface.co/KareenaBeniwal/fine-tune-qna

7. Metrics on test split

a. Classification

Accuracy: 0.6665

Precision: 0.6647

Recall: 0.6665

b. Question-answering

F1: 0.31

METEOR: 0.45

BLEU: 0.43

ROUGE: 0.7

Exact-match: 0.75

8. Parameters after fine-tuning

a. Classification

```
bert.embeddings.word_embeddings.weight: 23440896
bert.embeddings.position_embeddings.weight: 393216
bert.embeddings.token_type_embeddings.weight: 1536
bert.embeddings.LayerNorm.weight: 768
bert.embeddings.LayerNorm.bias: 768
bert.encoder.layer.0.attention.self.query.weight: 589824
```

```
bert.encoder.layer.1.attention.self.query.weight: 589824
```

```
bert.encoder.layer.2.attention.self.query.weight: 589824
```

```
bert.encoder.layer.3.attention.self.query.weight: 589824
```

```
bert.encoder.layer.4.attention.self.query.weight: 589824
```

```
bert.encoder.layer.5.attention.self.query.weight: 589824
```

```
bert.encoder.layer.6.attention.self.query.weight: 589824
```

```
bert.encoder.layer.7.attention.self.query.weight: 589824
```

Similarly, there are weights corresponding to layer 8, 9, 10, 11 [For more detailed information about parameters and weights, it is in the ipynb file for classification].

9. Pushed the fine-tuned model to hugging face

We pushed the fine-tuned model for classification and question-answering on hugging face.

10. Comments about

a. Poor/good performance

➡For classification task, no of epochs taken were 4, and the evaluation metrics came as follows:

```
Accuracy: 0.6665
Precision: 0.6647
Recall: 0.6665
F1 Score: 0.6607
```

Since we took the dataset with 5 classes, the evaluation metrics are on a somewhat lower side. The metrics value with no of epochs as 2 were around 63%.

Also the data was imbalanced, which can lead to lower values of these metrics.

➡For question/answering task, the F1 score is not good.
One of the reason for this can be explained through this example:

Context: You can find the github link for this video in the description.
Question: Where is the github link?
Predicted answer: description
Correct answer: in the description

Even though the predicted answer is almost correct, this will give an exact match score as 0 and a low F1 score.

b. Understanding from the number of parameters between pretraining and fine-tuning of the model.

Pre-training Model: Parameters: 109,482,240 Task: a general language model.
Fine-tuning Model for Classification Task: Parameters: 109,486,085 Task: Classification Note: Parameters related to 'CLS' flags are not included in the fine-tuning model.
Fine-tuning Model for QnA Task Parameters: 109,484,548 Task: Question-Answering (QnA)

- The fine-tuning process involves optimization and regularization techniques to adapt the model to the target task. These techniques, such as dropout or weight decay, may introduce additional parameters to control the model's behavior during training.Fine-tuning often involves transferring knowledge from a pre-trained model to a target task.
- The target task may require domain-specific information not present in the original pre-training data. As a result, additional parameters are introduced to capture domain-specific features during fine-tuning.
- Clearly, new layers are added during the fine-tuning, leading to increase in no of parameters.

Contribution

1. Aamod thakur - Task 2, Task 3, Task 4, Task 5,Task 10
2. Bhautik vala - Task 2, Task 3, Task 4, Task 5,Task 10
3. Chirag modi- Task 2, Task 3, Task 4, Task 5,Task 10
4. Dhruv Khandelwal - Task 6, Task 7, Task 8, Task 9, Task 10
5. Kareena beniwal- Task 6, Task 7, Task 8, Task 9, Task 10
6. Priya gupta- Task 6, Task 7, Task 8, Task 9, Task 10
7. Rakesh thakur-Task 6, Task 7, Task 8, Task 9, Task 10