BME646 and ECE60146: Homework 10

Spring 2024
Due Date: 11:59 pm, Apr 25, 2024
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**.

1 Introduction

The goal of this homework is to introduce you to BERT, an LLM (Large Language Model) from Google. To get an overall sense of where BERT stands in relation to the other LLMs like GPT-2 and GPT-3 that you have surely heard about, please read through at least the Preamble of your instructor's Week 15 slides:

https://engineering.purdue.edu/DeepLearn/pdf-kak/LLMLearning.pdf

This assignment calls upon you to fine-tune a pre-trained BERT model. The focus will be on the downstream task of question answering (Q&A). To accomplish this, we will utilize the Hugging Face transformers framework. Hugging Face plays a pivotal role in the open-source machine learning/artificial intelligence ecosystem, providing developers with libraries that facilitate seamless interaction with pre-trained models. The following sections will help you set up the environment and usage of `transformers` framework.

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Install hugging face `transformers` library. You may use the following command:

```
pip install transformers
```

If you want to install from source or any specific variation, you are welcome to further refer the official documentation at:

https://huggingface.co/docs/transformers/en/installation

2. You will also need to install `datasets` library:

```
pip install datasets
```

For further installation options, please refer the official documentation at:

https://pypi.org/project/datasets/

3. Download the pickle files provided on Brightspace which contain train and test datasets.
4. Review slides 9-26 from Week 15 lecture for an overview of BERT architecture and input to BERT.

2.1 Dataset

The data provided to you (the pickle files) is a subset of the The Stanford Question Answer Dataset (SQuAD) [1]. SQuAD is a reading comprehension dataset, consisting of crowd-sourced questions on a set of Wikipedia articles. The answer to every question could be a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. SQuAD v1.1 contains 100,000+ question-answer pairs on 500+ articles. However, we extracted only 10000 samples and split into 70:20:10 ratio for train, evaluate and test datasets for fine-tuning task.

At this point, the reader may question why only a small portion of data? The notion of fine-tuning is to avoid the necessity for extensive dataset to perform a specific downstream objective.

For this homework, you need to download dataset provided on Brightspace. The zip file contains 6 pickle files, two pickle files corresponding to each split, train, test and, eval.The following snippet shows an example how to load the pickle dictionaries :

```
import pickle

with open('train_dict.pkl', 'rb') as f:
    train_dict = pickle.load(f)
with open('test_dict.pkl', 'rb') as f:
    test_dict = pickle.load(f)
```

```
with open('eval_dict.pkl', 'rb') as f:
    eval_dict = pickle.load(f)

with open('train_data_processed.pkl', 'rb') as f:
    train_processed = pickle.load(f)

with open('test_data_processed.pkl', 'rb') as f:
    test_processed = pickle.load(f)

with open('eval_data_processed.pkl', 'rb') as f:
    eval_processed = pickle.load(f)

print(train_dict.keys())
print(test_dict.keys())
print(eval_dict.keys())

print(train_processed.keys())
print(test_processed.keys())
print(eval_processed.keys())

# dict_keys(['id', 'title', 'context', 'question', 'answers'])
# dict_keys(['id', 'title', 'context', 'question', 'answers'])
# dict_keys(['id', 'title', 'context', 'question', 'answers'])
# dict_keys(['input_ids', 'attention_mask', 'start_positions',
                                    'end_positions'])
# dict_keys(['input_ids', 'attention_mask', 'start_positions',
                                    'end_positions'])
# dict_keys(['input_ids', 'attention_mask', 'start_positions',
                                    'end_positions'])
```

The `*_dict.pkl` contains the raw data with context, question and answers. While the `*_processed.pkl` contains the subword embeddings from BERT tokenizer and embedder. Also, in the preprocessing steps, I have performed offset mapping to gather start and end positions.

What is offset mapping?

You may have noticed change of position or word split when you convert text to tokens. These tokens are used by BERT to understand the start and end of the sentence. However, when dealing with tasks such as Q&A or Named Entity Recognition, we need to know the position of the tokens in the original text. This is where offset mapping comes into play. Offset mapping is a list of tuples that map tokenized words to their character position in the original text. For example, if the original text is "Hello, world!" and the tokenized text is ["Hello", ",", "world", "!"], then the offset mapping would be [(0, 5), (6, 7), (8, 13), (13, 14)]. This means that the word "Hello" starts at position 0 and ends at position 5 in the original text, and so on. This is useful when we want to extract the answer from the original text

using the start and end token positions predicted by BERT. The 'start_positions', 'end_positions' keys denote the start and end token positions of the answer in the tokenized text.

3 Programming Tasks

3.1 BERT for Q&A

This homework is an introduction to how to fine-tune a BERT model for downstream tasks, here, specifically Question Answering. Fine-tuning as BERT model for any task involves addition of extra layers to accommodate for the final task of text classification, generation or question-answering. Some methods freeze the backbone BERT entirely or partially or not at all.

Following are the steps to fine-tune your own BERT model for Question Answer task:

1. First, initialize a model. We use BertForQuestionAnswering to initialise the model

```
from transformers import BertForQuestionAnswering
model_name = 'bert-base-uncased'
model = BertForQuestionAnswering.from_pretrained(
                                    model_name)

print(model._modules)

# OrderedDict([('bert',
#                BertModel(
#                  (embeddings): BertEmbeddings(
#                    (word_embeddings): Embedding(30522,
#                                    768, padding_idx=0)
#                    (position_embeddings): Embedding(512,
#                                    768)
#                    (token_type_embeddings): Embedding(2,
#                                    768)
#                    (LayerNorm): LayerNorm((768,), eps=1e-
#                                    12, elementwise_affine=
#                                    True)
#                    (dropout): Dropout(p=0.1, inplace=
#                                    False)
#                  )
#                  (encoder): BertEncoder(
#                    (layer): ModuleList(
```

```
#                     (0-11): 12 x BertLayer(
#                       (attention): BertAttention(
#                         (self): BertSelfAttention(
#                           (query): Linear(in_features=
                                768, out_features=768,
                                bias=True)
#                           (key): Linear(in_features=768,
                                 out_features=768, bias=
                                True)
#                           (value): Linear(in_features=
                                768, out_features=768,
                                bias=True)
#                           (dropout): Dropout(p=0.1,
                                inplace=False)
#                         )
#                         (output): BertSelfOutput(
#                           (dense): Linear(in_features=
                                768, out_features=768,
                                bias=True)
#                           (LayerNorm): LayerNorm((768,),
                                 eps=1e-12,
                                elementwise_affine=True)
#                           (dropout): Dropout(p=0.1,
                                inplace=False)
#                         )
#                       )
#                       (intermediate): BertIntermediate(
#                         (dense): Linear(in_features=768,
                                 out_features=3072, bias=
                                True)
#                         (intermediate_act_fn):
                                GELUActivation()
#                       )
#                       (output): BertOutput(
#                         (dense): Linear(in_features=3072
                                , out_features=768, bias=
                                True)
#                         (LayerNorm): LayerNorm((768,),
                                eps=1e-12,
                                elementwise_affine=True)
#                         (dropout): Dropout(p=0.1,
                                inplace=False)
#                       )
#                     )
#                   )
#                 )),
#                ('qa_outputs',
#                 Linear(in_features=768, out_features=2,
```

```
                                    bias=True))])
```

2. Now, next step would be to set the training arguments for fine-tuning task:

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    use_mps_device=True,
    num_train_epochs=3,              # total number of
                                    training epochs, change
                                    this as you need
    per_device_train_batch_size=8,  # batch size per
                                    device during training,
                                    change this as you need
    per_device_eval_batch_size=8,   # batch size for
                                    evaluation, change this as
                                     you need
    weight_decay=0.01,               # strength of weight
                                    decay
    logging_dir='./logs',            # directory for
                                    storing logs
)
```

3. Finally, to train we use the Trainer class from hugging face

```
from transformers import Trainer
from datasets import Dataset
import pandas as pd

train_dataset = Dataset.from_pandas(pd.DataFrame(
                                    train_processed))
eval_dataset = Dataset.from_pandas(pd.DataFrame(
                                    eval_processed))
test_dataset = Dataset.from_pandas(pd.DataFrame(
                                    test_processed))

trainer = Trainer(
    model=model,                          # the
                                    instantiated Transformers
                                    model to be fine-tuned
    args=training_args,                   # training
                                    arguments, defined above
    train_dataset=train_dataset,          # training
                                    dataset
    eval_dataset=eval_dataset              # evaluation
                                    dataset
)

```

```
trainer.train()

# {'loss': 2.3874, 'learning_rate': 3.3333333333333335e-05, 'epoch': 1.0}
# {'loss': 0.926, 'learning_rate': 1.6666666666666667e-05, 'epoch': 2.0}
# {'loss': 0.4081, 'learning_rate': 0.0, 'epoch': 3.0}
# {'train_runtime': 1347.7488, 'train_samples_per_second': 8.904, '
train_steps_per_second': 1.113, 'train_loss': 1.2405101318359375, 'epoch': 3.0}
```

In your report, dedicate a block showing the train output as above for first 5 epochs.

4. Finally, to test the trained model.

```
import numpy as np
x = trainer.predict(test_dataset)
start_pos, end_pos = x.predictions
start_pos = np.argmax(start_pos, axis=1)
end_pos = np.argmax(end_pos, axis=1)

for k, (i, j) in enumerate(zip(start_pos, end_pos)):
    tokens = tokenizer.convert_ids_to_tokens(
                                test_processed['input_ids'][k])

    print('Question:', test_dict['question'][i])
    print('Answer:', ' '.join(tokens[i:j+1]))
    print('Correct Answer:', test_dict['answers'][i]['text'])
    print('---')
```

How are the outputs ? Qualitatively look at 10-20 answers and express in your own words how bad or relevant they are. You may need to run more epochs if the sentences make no sense.

3.2 Evaluation Metrics

Now for quantitative metrics, we will use Exact Match and F1 score. You may use the following snippets:

```
def compute_exact_match(prediction, truth):
    return int(prediction == truth)
```

```
3
4  def f1_score(prediction, truth):
5      pred_tokens = prediction.split()
6      truth_tokens = truth.split()
7
8      # if either the prediction or the truth is no-answer then
                                  f1 = 1 if they agree, 0
                                  otherwise
9      if len(pred_tokens) == 0 or len(truth_tokens) == 0:
10         return int(pred_tokens == truth_tokens)
11
12     common_tokens = set(pred_tokens) & set(truth_tokens)
13
14     # if there are no common tokens then f1 = 0
15     if len(common_tokens) == 0:
16         return 0
17
18     prec = len(common_tokens) / len(pred_tokens)
19     rec = len(common_tokens) / len(truth_tokens)
20
21     return 2 * (prec * rec) / (prec + rec)
```

Calculate the average and median EM and F1-score and report them. For this, you may first calculate individual output in the test set and collect them in a list.

3.3 Comparison

Let us now compare out fine tuned model with another open-source fine-tuned model. For this, we will use `distilbert-base-cased-distilled-squad` model from Hugging Face. This model is a distilled version of BERT fine-tuned on SQuAD dataset.

To load and extract outputs from the model, you may use the following snippet:

```
1  from transformers import pipeline
2  question_answerer = pipeline("question-answering", model='
                                  distilbert-base-cased-distilled
                                  -squad')
3
4  for i in range(len(test_dict['question'][:2])):
5      result = question_answerer(question=test_dict['question'][i
                                  ], context=test_dict['context']
                                  [i])
6      print('Question:', test_dict['question'][i])
7      print('Answer:', result['answer'])
```

```
    print('Correct Answer:', test_dict['answers'][i]['text'][0])
    print('Exact Match:', compute_exact_match(result['answer'], test_dict['answers'][i]['text'][0]))
    print('F1 Score:', f1_score(result['answer'], test_dict['answers'][i]['text'][0]))
    print('---')


# Question: Who does Beyonce describe as the definition of inspiration?
# Answer: Oprah Winfrey
# Correct Answer: Oprah Winfrey
# Exact Match: 1
# F1 Score: 1.0
# ---
# Question: Who is still looking for compensation and justice?
# Answer: the many families
# Correct Answer: many families
# Exact Match: 0
# F1 Score: 0.8
# ---
# Question: Discrepancy in what spec brought about a class action suit against Apple in 2003?
# Answer: College of Science
# Correct Answer: the College of Science
# Exact Match: 0
# F1 Score: 0.8571428571428571
# ---
```

Similar to Section 3.2, compute and report average and median EM and F1 scores.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. **Make sure your submission zip file is under 10MB.** Compress your figures if needed. **Do NOT submit your network weights nor dataset.**
2. Your pdf must include a description of
    - The figures and descriptions as mentioned in Sec. 3.

- Your source code. Make sure that your source code files are adequately commented and cleaned up.

3. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw9_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.
4. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.
5. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.
6. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**
7. To help better provide feedbacks to you, make sure to **number your figures**.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.