Greeshma Chandra Shekar C20084146 greeshc@g.clemson.edu

CPSC-8430
Deep Learning
Homework 3
Extractive Question answering

GitHub link: https://github.com/Greeshma-C20084146/hw3.git

Introduction

This involves posing questions about a document/paragraph and identifying the answers as spans of text in the document itself.

We will fine-tune a BERT model on the Spoken-SQuAD: A spoken question answering dataset on SQuAD, the first large-scale spoken question answering dataset. This is an English dataset.

Preparing the dataset

We download the dataset from their GitHub link and load the dataset using Hugging Face library after flattening the dataset in python.

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 37111
    })
    test: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 15875
    })
})
```

The context and question fields are very straightforward to use. The answers field contains the following: the text field is rather obvious, and the answer_start field contains the starting character index of each answer in the context.

Greeshma Chandra Shekar C20084146 greeshc@g.clemson.edu

Preprocessing Train Data

We convert the text in the input into IDs the model can make sense of, using a tokenizer. We use BERT model to do this.

Some of the questions in the dataset might have extra spaces at the beginning and the end that doesn't add anything, so we remove those extra spaces.

We need to generate labels for the question's answer, which will be the start and end positions of the tokens corresponding to the answer inside the context.
The dataset provides us with the start character of the answer in the context, and by adding the length of the answer, we can find the end character in the context.

we don't need to generate labels for the test data.

Fine-tuning the model

We make use of ID column in the dataset to match each created feature to the original example it comes from.
we set some hyperparameters (like the learning rate, the number of epochs we train for, and some weight decay) and indicate that we want to save the model at the end of every epoch, skip evaluation, and upload our results to the Model Hub

- We masked the start and end logits corresponding to tokens outside of the context.
- We then converted the start and end logits into probabilities using a softmax.
- We attributed a score to each (start_token, end_token) pair by taking the product of the corresponding two probabilities.
- We looked for the pair with the maximum score that yielded a valid answer (e.g., a start_token lower than end_token).

We'll look at the logit scores for the n_best start logits and end logits, excluding positions that give:

- An answer that wouldn't be inside the context

- An answer with negative length
- An answer that is too long

we use the classic AdamW, which is like Adam, but with a fix in the way weight decay is applied. Once we have all those objects like, model, optimizer, train_dataloader, eval_dataloader, we can send them to the accelerator.prepare() method.

Once the training is complete, we can finally evaluate our model (and pray we didn't spend all that compute time on nothing). The predict() method of the Trainer will return a tuple where the first elements will be the predictions of the model (here a pair with the start and end logits). We send this to our compute_metrics() function.