Data preprocessing vs. Sequence bucketing

Seunghwan Seo

November 5 2024

1 Abstract

Check model training time after using two methods (Data preprocessing, Sequence bucketing) to solve the problem of time-consuming model fine-tuning and check performance against time.

2 Introduction

HuggingFace has multiple pre-trained models. HuggingFace's model fine-tuning process is time-consuming due to the large size of the model. Let's try to solve these shortcomings by using data preprocessing or sequence bucketing. We will also check for differences in model performance over time.

3 Method

Select a model from the hugging face and use data preprocessing or sequence bucketing for fine-tuning. We also changed the model to TensorFlow format to experiment in the TensorFlow environment.

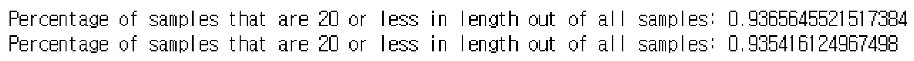
3.1 model

GPU resources are limited, so we had to choose a relatively small model. However, this doesn't seem to have a significant impact on our experiments.

Selected models: klue/bert-base

3.2 Data preprocessing

Sequence of data preprocessing steps. 1) deleting missing data if it exists. 2) check the distribution of data length. 3) check how much data is included after setting max-len. 4) select only data below max-len. 5) convert the data type to Hugging Face's Dataset. See Figure 1 for the maximum length set in step 3. With the exception of the steps above, the rest are designed to follow a typical fine-tuning process. Training is done using the trainer available on HuggingFace.



```
Percentage of samples that are 20 or less in length out of all samples: 0.9365645521517384
Percentage of samples that are 20 or less in length out of all samples: 0.935416124967498
```

Figure 1: After setting max-len, we checked how much data it contains and found that

3.3 Sequence bucketing

Sequence bucketing steps. 1) Separate input and target data from dataset and tokenise them. 2) Define and apply bucketing function Datasets are stored in TensorFlow's tf.Dataset type because it uses the bucket-by-sequence-length() method. That's why you can't use the trainer provided by Huggingface. That's why we use the model.fit() method for training

4 Result

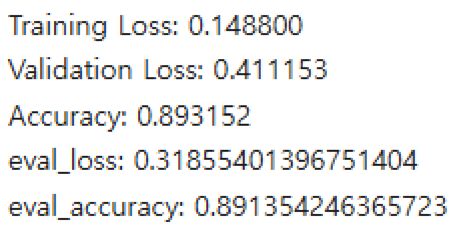
Model training time, model performance description

4.1 training time

1) Data preprocessing: 08:08:21 hours 2) Sequence bucketing: 00:33:08 hours We can see that the Sequence bucketing method is about 16 times faster than the data preprocessing method.

4.2 model performance

We will focus on eval-loss and eval-accuracy of the model performance, see Figure 2 and Figure 3 for the rest. 1) Data preprocessing: 0.3185(eval-loss), 0.891(eval-accuracy) 2) Sequence bucketing: 0.4010(eval-loss), 0.8200(eval-accuracy) In terms of performance, you can see that the Sequence bucketing method performs better than the Data preprocessing method



```
Training Loss: 0.148800
Validation Loss: 0.411153
Accuracy: 0.893152
eval_loss: 0.31855401396751404
eval_accuracy: 0.891354246365723
```

Figure 2: Remaining performance of data preprocessing methods

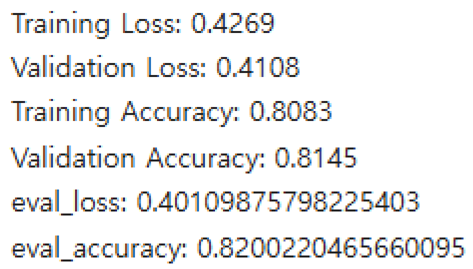

```
Training Loss: 0.4269
Validation Loss: 0.4108
Training Accuracy: 0.8083
Validation Accuracy: 0.8145
eval_loss: 0.40109875798225403
eval_accuracy: 0.8200220465660095
```

Figure 3: Remaining performance of sequence bucketing methods

4.3 Meaning of results

Sequence bucketing method is superior in terms of training time and data preprocessing method is superior in terms of model performance However, you may decide that the Data preprocessing method is better because you value model performance more. Of course, when GPU resources are scarce, the Sequence bucketing method is probably the best option.

5 Conclusion

This study answers the question of what to do when GPU resources are scarce. One thing that could be improved is that we did not use the Data Collator provided by HuggingFace's trainer. In this study, we used the bucket-by-sequence-length() method of TensorFlow, so we believe that improving this method will bring better results.