Appendix: On Finetuning Large Language Models

Yu Wang
Fudan Institute for Advanced Study for Social Sciences
Fudan University

Step-by-Step Replication

In this appendix, we report on the details of each step in the paper and link these details to their corresponding script for replication.

1 How to replicate the original results

Häffner et al. (2023) report an MSE of 1.75 and R^2 of 0.60 for ConfliBERT. To replicate that set of results, we rerun the script(s) in the Harvard Dataverse package. The details are as follows:

1. From code.zip, we get the training script final_bert.py.
2. From data.zip, we get the data file cw_texts_clean_bert.csv. We then put this file into a Google Drive and make it public so that anyone could download.
3. Convert the final_bert.py into an ipynb file, which we call Baseline_Replication.ipynb, so that we could run it on Google Colab using A100 GPUs.

Baseline_Replication.ipynb does not edit final_bert.py. It only adds a few new lines for package installation and data download, as illustrated below.

Package installation:

```
!pip install transformers==4.28.0
!pip install datasets
!pip install accelerate -U
```

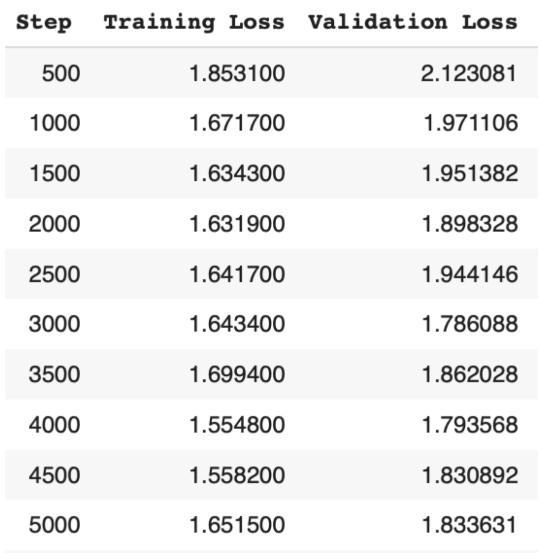
Data download:

```
!gdown 1a1v06PvGaWmTCWvL33o2C133D90SRzMC
!mkdir output
!mkdir /data
!mv cw_texts_clean_bert.csv /data
```

Please see the replication results below. The file is available for download at Baseline_Replication.ipynb. This file generates the results on MSE and R^2, which are the same as reported in Häffner et al. (2023):

```
1.7503021153174287
0.6003749194371616
```

In addition, this file also contains the records of the training process, which we use to create Table 1 (left) in the main paper. The file takes 37 minutes to run.



Step	Training Loss	Validation Loss
500	1.853100	2.123081
1000	1.671700	1.971106
1500	1.634300	1.951382
2000	1.631900	1.898328
2500	1.641700	1.944146
3000	1.643400	1.786088
3500	1.699400	1.862028
4000	1.554800	1.793568
4500	1.558200	1.830892
5000	1.651500	1.833631

Figure 1: The training process when we freeze most layers as is done in Häffner et al. (2023). This forms part of Table 1 in our paper.

2 How to create ConfliBERT Unrestricted (MSE: 0.99, R^2: 0.77)

We need to change the following lines from Baseline_Replication.ipynb. First, we comment out the following two lines. These lines were responsible for freezing layers from finetuning.

```
for param in model.bert.parameters():
    param.requires_grad = False
```

Second, we decrease the learning rate from 2e-3 to 2e-5. We need to decrease the learning rate because 2e-3 is too high for finetuning all the parameters.[1]

```
LEARNING_RATE = 2e-3
LEARNING_RATE = 2e-5
```

With these changes, we create the file ConfliBERT_Unrestricted_V0.ipynb. One drawback of ConfliBERT_Unrestricted.v0.ipynb is that it is training for too long as we are inheriting the number of training epochs from Häffner et al. (2023). To optimize the training process (save some time), we use the learning logs to decide on a more appropriate number of epochs. For simplicity and without too much hyperparameter tuning, we are setting the number of epochs to 10. With this further change, we create ConfliBERT_Unrestricted_V1.ipynb.

```
EPOCHS = 20
EPOCHS = 10
```

3 How to create ConfliBERT Max Length (MSE: 0.87, R^2: 0.80)

We propose to increase the max sequence length from 256 to 512 by changing the following lines from ConfliBERT_Unrestricted_V1.ipynb. This could potentially improve the model's performance because a substantial number of the samples have more than 256 tokens. The resulting file, ConfliBERT_Max_Length.ipynb, gives us the reported MSE of 0.87 and R^2 of 0.80.

```
MAX_LENGTH = 256
MAX_LENGTH = 512
```

[1]For an example of training failure with too high a learning rate, please see ConfliBERT_Unrestricted_V1_Large_Learning_Rate.ipynb.

4 How to examine the model's parameters and verify that the pooler layer is randomly initialized and never updated

Readers can refer to ConfliBERT_Parameters.ipynb for the details. In particular, we can check out the number of parameters in each layer using the following lines:

```
print(sum(p.numel() for p in trainer.model.classifier.parameters()))
print(sum(p.numel() for p in trainer.model.bert.pooler.parameters()))
print(sum(p.numel() for p in trainer.model.bert.encoder.parameters()))
print(sum(p.numel() for p in trainer.model.bert.embeddings.parameters()))
print(sum(p.numel() for p in trainer.model.bert.embeddings.parameters()))
print(sum(p.numel() for p in trainer.model.dropout.parameters()))
```

We can confirm that the pooler layer is randomly initialized and never trained by (1) observing that the bias parameters are all set to zero before and after finetuning (2) reading the logs following the trainer.train() step.

1. bias parameters are all zero:

```
print(list(trainer.model.bert.pooler.parameters()))
```

2. training logs:

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at snowood1/ConfliBERT-scr-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'bert.pooler.dense.bias', 'bert.pooler.
dense.weight']
```

5 How to freeze only the pretrained layers

In this section, we show how we can freeze the pretrained layers and exclusively train over the newly initialized parameters (the pooler layer and the classificiation layer).

```
def model_init():
    model = AutoModelForSequenceClassification.from_pretrained(BASE_MODEL,
        num_labels=1, ignore_mismatched_sizes=True)
    for param in model.bert.encoder.parameters():
        param.requires_grad = False
    for param in model.bert.embeddings.parameters():
        param.requires_grad = False
    return model
```

We are able to achieve an MSE of 1.33 and an R^2 of 0.70 by training the pooler layer and the classification layer. This represents a large improvement over ConfliBERT Restricted (Häffner et al., 2023) with an MSE of 1.75 and an R^2 of 0.6. At the same time, the results are considerably weaker than ConfliBERT Unrestricted, which suggests that finetuning all parameters does provide performance improvement for this classification task. Readers could compare the tradeoffs between ConfliBERT + Pooler Layer and ConfliBERT Unrestricted and decide which one better fits their needs. Readers can find the details in Corrected_Baseline.ipynb.

Table 1: Results in Columns 1, 2, and 3 are from Table 2 in Häffner et al. (2023). In Column 4, we make the pooler layer trainable on top of Häffner et al. (2023). In Column 5, we make all the parameters trainable. In Column 6, we further increase the max sequence length to 512. Best results in bold.

Model	OCoDi Random Forest (1)	OCoDi XGBoost (2)	ConfliBERT Restricted (3)	ConfliBERT +Pooler Layer (4)	ConfliBERT Unrestricted (5)	ConfliBERT Max Length (6)
MSE	1.59	1.60	1.75	1.33	0.99	**0.87**
R^2	0.64	0.63	0.6	0.70	0.77	**0.80**

6 Impact on Computation Costs

We observe that running the original script in Häffner et al. (2023) takes 37 minutes on an A100 GPU. If we unfreeze the parameters in the pooler layer, the script will take roughly the same amount of time to run, given this is a relatively small layer with 0.6 million parameters. Once we unfreeze all parameters, the script will take 96 minutes to run, which is still acceptable. However, noticing that we probably do not need to finetune the model for 20 epochs, as a step of optimization, we drop the number of epochs to 10 and reduce the amount of running time to 50 minutes. Lastly, when we increase the max sequence length from 256 to 512 while training for 10 epochs, the script takes 98 minutes to run (Column 5).[2]

[2]All the reported numbers are available from the corresponding running scripts. Interested readers might want to consider using larger batches, among other things, to further speed up the training process.

Table 2: Computation costs as measured in minutes in running the training (and evaluation) script.

Model	ConfliBERT Restricted (1)	ConfliBERT + Pooler Layer (2)	ConfliBERT Unrestricted (3)	ConfliBERT Unrestricted (10 Epochs) (4)	ConfliBERT Max Length (5)
Minutes	37	37	96	50	98

References

Häffner, S., Hofer, M., Nagl, M., & Walterskirchen, J. (2023). Introducing an interpretable deep learning approach to domain-specific dictionary creation: A use case for conflict prediction. *Political Analysis*, 1–19. doi: 10.1017/pan.2023.7