Improving the Performance of the BERT model

Author: Satwik Ram K

There are many ways to improve the performances of neural networks.[1] In that fine-tuning, hyper-parameters are one of the best methods to implement.

The hyperparameters which can be tuned for BERT [2] are:

1. Sequence Length (32, 64, 128, 256, and Max sequence length of 512)
2. Batch size
3. Learning Rate Scheduler and Reduce Learning rate over the epochs
4. Optimizers and Loss functions
5. Freezing and Un-Freezing the BERT Embedding Layers

Other methods to improve the performance of the model. (This is very specific to your paper)

1. Try to collect more data and make the dataset balanced.
2. If the dataset is imbalanced, set the class weights for each class.
3. If the dataset is imbalanced, Use the Focal loss function. This will focus on the F1 score while training the model.
4. Feature Engineering

Imbalance dataset

The imbalanced dataset is a dataset with unequal class proportions. The majority classes are those that make up a considerable share of the data set. Minority classes make up a smaller percentage of the population. When we train the model of a neural network, the model will fail to understand the data and it may lead to overfitting towards the majority classes. This will hurt the model performance.

Solutions to overcome the Imbalance dataset

The best solution is to collect a greater number of samples for minority classes and make the dataset balanced.

Class weights

While training the model, we can set a weight for each class. We can set more weights towards the minority class and fewer weights towards the majority class. As a result, the model will "pay greater attention" to samples from a minority group.

A sample formula to calculate weights for two classes.

```
weight_for_0 = (1 / class_0) * (total / 2.0)
weight_for_1 = (1 / class_1) * (total / 2.0)
```

Feature Engineering

Feature engineering aids in the creation of better data, allowing the model to comprehend it and produce reasonable outcomes. To comprehend a natural language, you must first comprehend how we construct sentences, communicate our thoughts using various words, signs, and special characters, and, most importantly, comprehend the context of the sentence to determine its meaning. The model will be able to interpret the sentence better if we can use these contexts as features and feed them to it.

The following are some of the most common qualities that we may extract from a sentence:

1. Number of words

2. Percentage of capital words
3. Percentage of punctuation
4. Percentage of unique words
5. Percentage of stop words
6. Average sentence length
7. Readability Score of the sentence

The hyperparameters tuning:

Learning Rate

Tuning the learning rate is one of the toughest parts of hyperparameter tuning. Unfortunately, the best learning rate for a given model on a particular dataset cannot be calculated analytically. Instead, finding a good (or enough) learning rate requires trial and error.

Some of the methods to find the best learning rate while training the model are:

Add Momentum to the Learning Rate

An exponentially weighted average of the previous modifications to the weights can be provided when the weights are changed. The addition of "momentum" to stochastic gradient descent adds inertia to the update method, leading numerous previous updates in one direction to continue in that direction in the future.

Use a Learning Rate Schedule

The most basic learning rate schedule is to reduce the learning rate linearly from a high beginning value to a low one. This allows for substantial weight adjustments at the start of the learning phase and tiny weight changes or fine-tuning at the conclusion.

Adaptive Learning Rates

The most straightforward application is to reduce the learning rate after the model's performance reaches a plateau, such as by a factor of two or an order of magnitude.

Code:

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = "val_loss", factor
= 0.2,
                              patience = 4, min_lr = 0.001, verbose = 1)

def scheduler(epoch, lr):
  if epoch < 5:
    return lr
  else:
    return lr * tf.math.exp(-0.1)

lr_schedule = tf.keras.callbacks.LearningRateScheduler(scheduler, verbose = 1)
```

```
class_weights = list(class_weight.compute_class_weight("balanced",
                            np.unique(dataset["Y"]),
                            dataset["Y"]))

weights = {}

for idx, weight in enumerate(class_weights):
    weights[idx]=weight
```

Table Reference for BERT model!

Model	**Sequence Length**	**Max Batch Size**
BERT-Base	64	64
…	128	32
…	256	16
…	320	14
…	384	12
…	512	6
BERT-Large	64	12
…	128	6
…	256	2
…	320	1
…	384	0
…	512	0

Batch Size and Learning Rate suggested by Google:

batch sizes: 8, 16, 32, 64, 128
learning rates: 3e-4, 1e-4, 5e-5, 3e-5

Paper Reference:

[1] Kodandaram, Satwik Ram. "Improving the Performance of Neural Networks." (2021).

[2] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (cite arxiv:1810.04805Comment: 13 pages)

Code References

[1] https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#class_weights

[2] https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LearningRateScheduler

[3] https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

[4]

https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html