Nicholas Hamilton

Cognitive Generative AI Externship

June 2 2025

The Process of Fine-tuning the BERT model

I start with importing the necessary libraries and modules. I import torch for the calculations for my AI model. I import datasets to retrieve IMDb from HuggingFace. I also import several key tools from transformers, along with my actual model, a smaller version of BERT. I take in tokenizers, which is kind of like the dictionaries and rules for my model. Numpy of course for working with numbers in certain ways. Sklearn for its metrics to see how my model is performing.

I start with a global configuration set of variables that I store into memory. The model I am using, the dataset, a small sample for training and a smaller sample for evaluating, a max length for our AI when reviewing a piece of data, and locations for directories. I have historically not used a variable for some of these things, but I see it so commonly now that I have adjusted.

I store my dataset in memory, and shuffle reviews into my testing and training sets. I load my tokenizer with auto tokenizer, using its from_pretrained method. This breaks the english language down into tokens, so that my AI model can understand. I have a function under this inside of my load and prepare data function so that I can help pre process my examples. I batch these functions together and use the '.map' method. I go ahead and remove the original text as it is no longer needed. Finally, I set the format of both training and eval sets, to torch for PyTorch tensors!

Moving on to the second major part, I start the real work. Compute metrics function contains the code that Trainer will use during evaluation to see how the model is performing. I am storing logits and labels from eval predictions. **Logits** are the outcome of the models predictions. It does not say something simple like positive or negative when determining this sentiment analysis. It returns raw scores. The **labels** are simply the correct answers for our eval dataset. 0 for negative, 1 for positive. I retrieve my predictions, which is done through converting the logits into the actual predicted class. I create space in memory for accuracy, which I simply do the number of correct predictions divided by the total number of predictions. This is done through the Sklearn library, as mentioned earlier. Finally, I retrieve my F1 score. This is a metric that looks into the precision and recall of the model. I format these into a dictionary and return them out of the function. The dictionary format is for the Trainer.

Finally, we reach the control hub for the program. In the main function, I retrieve my datasets and tokenizer. I then load my model, and create my two labels, positive and negative.

The model, I tell HuggingFace I need one for sequence classification, then I further ask for specifically the pre-trained model I want. Given I asked for these two labels at the head of my

model, the weights are random. I take this model that already has a ton of knowledge, and further refine it throughout this process.

I make several training arguments. The number of training epochs (how many times we go through the entire training dataset. I batch the IMDb reviews into sets of 8. I set my learning rate that determines how big of an adjustment the model makes after an error. This is sort of like easing into a workout, I don't want my model to change drastically early on in the process. I set the weight decay at 0.01, which is to prevent the model from becoming too complex and memorizing the training data. This is to help prevent overfitting. It adds a small penalty to large weights in the model. This is sort of like discouraging the model from creating complex answers that would only serve to solve the training set.

I then created my trainer. This is sort of like a tutor, training my model. I give it the model, the training arguments, the training and eval datasets, the tokenizer (rules for the trainer), and the compute metrics function so that the trainer can grade the model. Then with one line, I start the fine-tuning process (trainer.train()).

After adjusting the hyperparameters, I have been able to achieve the following results:
The evaluation metrics on the test set are:

eval_loss: 0.3779
eval_accuracy: 0.8600
eval_f1: 0.8594
eval_runtime: 5.9219
eval_samples_per_second: 84.4320
eval_steps_per_second: 10.6380
epoch: 1.0000

Sample predicted labels: [1 1 0 0 0 1 1 0 0 1]
Sample true labels: [1 1 0 1 0 1 1 0 0 1]

Writing a simple script, here are results that I have gathered testing against more data:
-Making Predictions-

Review: "This movie was absolutely fantastic! The acting was superb and the plot was gripping."
Probabilities: [[0.04476716 0.9552328]]
Predicted Sentiment: LABEL_1 (Class ID: 1)

Review: "I was really disappointed with this film. It was boring and the ending made no sense."
Probabilities: [[0.976727 0.02327302]]
Predicted Sentiment: LABEL_0 (Class ID: 0)

Review: "An okay movie, not great but not terrible either. Had some good moments."
Probabilities: [[0.24341966 0.75658035]]
Predicted Sentiment: LABEL_1 (Class ID: 1)

Review: "The special effects were incredible, but the story was a bit weak."
Probabilities: [[0.71557534 0.28442472]]
Predicted Sentiment: LABEL_0 (Class ID: 0)

Review: "I loved every minute of it, a true masterpiece!"
Probabilities: [[0.03906078 0.96093917]]
Predicted Sentiment: LABEL_1 (Class ID: 1)

Review: "Worst movie I've seen all year. A complete waste of time."
Probabilities: [[0.97590125 0.02409872]]
Predicted Sentiment: LABEL_0 (Class ID: 0)