GPT2 For Text Classification using Hugging Face Transformer

Abstract:

Hugging Face is very nice to us to include all the functionality needed for GPT2 to be used in classification tasks. Thank you Hugging Face!I wasn't able to find much information on how to use GPT2 for classification so I decided to make this tutorial using similar structure with other transformers models.Our main idea is Since GPT2 is a decoder transformer, the last token of the input sequence is used to make predictions about the next token that should follow the input. This means that the last token of the input sequence contains all the information needed in the prediction. With this in mind we can use that information to make a prediction in a classification task instead of generation task.In other words, instead of using first token embedding to make prediction like we do in Bert, we will use the last token embedding to make prediction with GPT2.Since we only cared about the first token in Bert, we were padding to the right. Now in GPT2 we are using the last token for prediction so we will need to pad on the left. Because of a nice upgrade to HuggingFace Transformers we are able to configure the GPT2 Tokenizer to do just that.

Datasets :

The description provided on the Stanford website This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well. Raw text and already processed bag of words formats are provided.

Installs :

Transformers library needs to be installed to use all the awesome code from Hugging Face. To get the latest version I will install it straight from GitHub.

Ml_things library used for various machine learning related tasks. I created this library to reduce the amount of code I need to write for each machine learning project.

MovieReviewsDataset(Dataset) :

If you worked with PyTorch before, this is pretty standard. We need this class to read in our dataset, parse it and return texts with their associated labels.

In this class I only need to read in the content of each file, use fix_text to fix any Unicode problems and keep track of positive and negative sentiments.

I will append all texts and labels in lists.

There are three main parts of this PyTorch Dataset class:

> init() where we read in the dataset and transform text and labels into numbers.
>
> len() where we need to return the number of examples we read in. This is used when calling len(MovieReviewsDataset()).
>
> getitem() always takes as an input an int value that represents which example from our examples to return from our dataset. If a value of 3 is passed, we will return the example form our dataset at position 3.

Gpt2ClassificationCollator :

I use this class to create the Data Collator. This will be used in the DataLoader to create the bathes of data that get fed to the model. I use the tokenizer and label encoder on each sequence to convert texts and labels to number.

Lucky for us, Hugging Face thought of everything and made the tokenizer do all the heavy lifting (split text into tokens, padding, truncating, encode text into numbers) and is very easy to use!

There are two main parts of this Data Collator class:

> init() where we initialize the tokenizer we plan to use, how to encode our labels and if we need to set the sequence length to a different value.
>
> call() used as function collator that takes as input a batch of data examples. It needs to return an object with the format that can be fed to our model. Luckily our tokenizer does that for us and returns a dictionary of variables ready to be fed to the model in this way: model(**inputs). Since we are fine-tuning the model I also included the labels.

Train(dataloader, optimizer_, scheduler_, device_):

I created this function to perform a full pass through the DataLoader object (the DataLoader object is created from our Dataset* type object using the **MovieReviewsDataset class). This is basically one epoch train through the entire dataset.

The dataloader is created from PyTorch DataLoader which takes the object created from MovieReviewsDataset class and puts each example in batches. This way we can feed our model batches of data!

The optimizer_ and scheduler_ are very common in PyTorch. They are required to update the parameters of our model and update our learning rate during training. There is a lot more than that but I won't go into details. This can actually be a huge rabbit hole since A LOT happens behind these functions that we don't need to worry. Thank you PyTorch!

In the process we keep track of the actual labels and the predicted labels along with the loss.

Validation(dataloader, device_):

I implemented this function in a very similar way as train but without the parameters update, backward pass and gradient decent part. We don't need to do all of those VERY computationally intensive tasks because we only care about our model's predictions.

I use the DataLoader in a similar way as in train to get out batches to feed to our model.

In the process I keep track of the actual labels and the predicted labels along with the loss.

Load Model and Tokenizer

Loading the three essential parts of the pretrained GPT2 transformer: configuration, tokenizer and model.

For this example I will use gpt2 from HuggingFace pretrained transformers. You can use any variations of GP2 you want.

In creating the model_config I will mention the number of labels I need for my classification task. Since I only predict two sentiments: positive and negative I will only need two labels for num_labels.

Creating the tokenizer is pretty standard when using the Transformers library. After creating the tokenizer it is critical for this tutorial to set padding to the left tokenizer.padding_side = "left" and initialize the padding token to tokenizer.eos_token which is the GPT2's original end of sequence token. This is the most essential part of this tutorial since GPT2 uses the last token for prediction so we need to pad to the left.

HuggingFace already did most of the work for us and added a classification layer to the GPT2 model. In creating the model I used GPT2ForSequenceClassification. Since we have a custom padding token we need to initialize it for the model using model.config.pad_token_id. Finally we will need to move the model to the device we defined earlier.

Dataset and Collator :

This is where I create the PyTorch Dataset and Data Loader with Data Collator objects that will be used to feed data into our model.

This is where I use the **MovieReviewsDataset** class to create the PyTorch Dataset that will return texts and labels.

Since we need to input numbers to our model we need to convert the texts and labels to numbers. This is the purpose of a collator! It takes data outputted by the PyTorch Dataset and passed through the Data Collator function to output the sequence for our model.

I'm keeping the tokenizer away from the PyTorch Dataset to make the code cleaner and better structured. You can obviously use the tokenizer inside the PyTorch Dataset and output sequences that can be used straight into the model without using a Data Collator.

I strongly recommend to use a validation text file in order to determine how much training is needed in order to avoid overfitting. After you figure out what parameters yield the best results, the validation file can be incorporated in train and run a final train with the whole dataset.

The data collator is used to format the PyTorch Dataset outputs to match the inputs needed for GPT2.

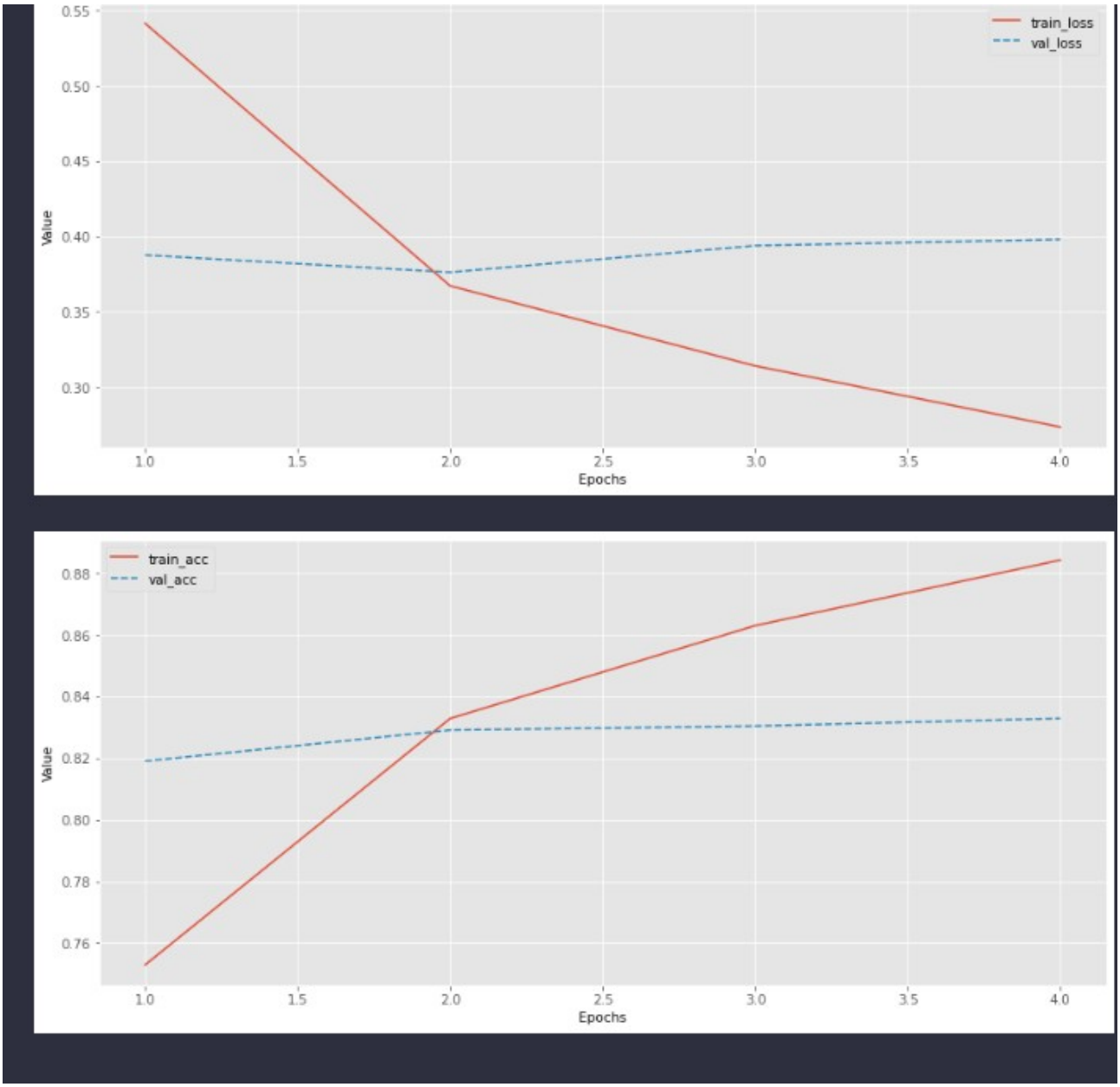
Train :

I created optimizer and scheduler use by PyTorch in training. I used most common parameters used by transformers models.

I looped through the number of defined epochs and call
the **train** and **validation** functions.

I'm trying to output similar info after each epoch as Keras

After training, plot train and validation loss and accuracy curves to check how the training went.



Evaluate

When dealing with classification is useful to look at precision recall and F1 score. A good gauge to have when evaluating a model is the confusion matrix.

```
Training on batches...
100%|██████████████████████████████|782/782 [01:09<00:00, 11.2

              precision    recall  f1-score   support

         neg       0.84      0.83      0.83     12500
         pos       0.83      0.84      0.83     12500

    accuracy                           0.83     25000
   macro avg       0.83      0.83      0.83     25000
weighted avg       0.83      0.83      0.83     25000
```