# Fastai integration with BERT: Multi-label text classification identifying toxicity in texts

abhik jha Follow

Jul 17, 2019 · 6 min read

Photo by Jules D. on Unsplash

. . .

## PROLOGUE

There is no doubt that Transfer learning in the areas of Deep learning has proved to be extremely useful and has revolutionized this field. However, unlike for tasks associated with image recognition and processing, for natural language processing (NLP) tasks which mainly deal with texts & documents, not much success was achieved till recently.

In this article, I will use two recent state of the art Natural Language Processing (NLP) techniques which have sort of transformed the area of

NLP in Deep Learning.

These techniques are as follows:

1. **BERT** (Deep Bidirectional Transformers for Language Understanding)
2. **Fastai ULMFiT** (Universal Language Model Fine-tuning for Text Classification)

Both these techniques are very advanced and very recent NLP techniques (BERT was introduced by Google in 2018 & Jeremy Howard and Sebastian Ruder introduced ULMFiT in 2017–18). Both of these incorporate the methods of Transfer Learning which is quite cool and are pre-trained on large corpuses of Wikipedia and related articles. I wanted to compare the overall performance of these two techniques.

I really like using Fastai for my deep learning projects and can't thank enough for Fastai's amazing community and our mentors and instructors — Jeremy Howard & Rachael Thomas for designing few of the most wonderful courses on the matters pertaining to Deep Learning. However, as till date, BERT is not implemented in Fastai.

Thus one of my aims to work on this project was to **integrate BERT with Fastai**. This means power of BERT combined with the simplicity of Fastai and then compare their respective performances. It was not an easy task especially implementing Discriminative Learning Rate technique of Fastai in BERT modelling.

In my project, below article helped me in understanding few of these integration techniques and I would like to extend my gratitude to the writer of this article:

https://mlexplained.com/2019/05/13/a-tutorial-to-fine-tuning-bert-with-fast-ai/

. . .

# DATA

We will work on an old Kaggle competition dataset which can be found here:

**Toxic Comment Classification Challenge**

Identify and classify toxic online comments

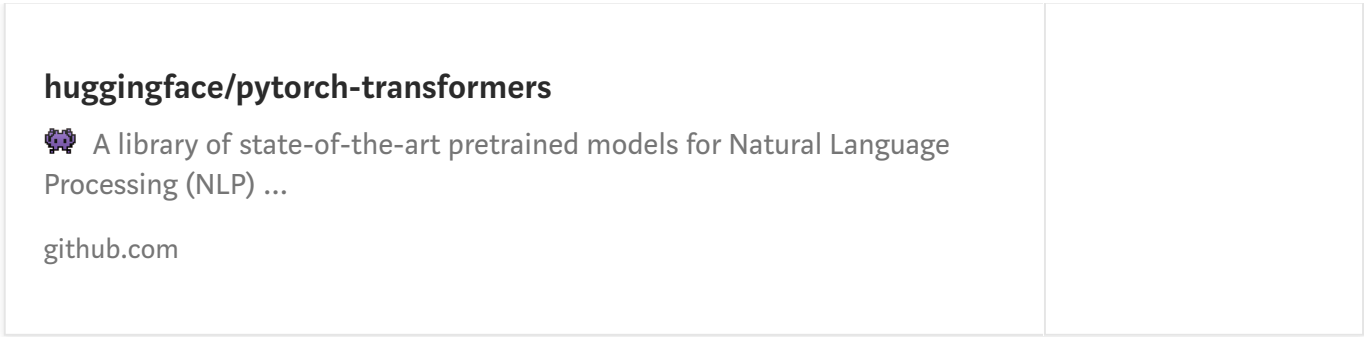
www.kaggle.com

This is a multi-label text classification challenge wherein we need to classify a given string of texts into following classes:

1. Toxic
2. Severe Toxic
3. Obscene
4. Threat
5. Insult
6. Identity Hate

## LIBRARIES AND MAJOR DEPENDENCIES

1. Kaggle Kernel for GPU usage
2. Fastai v 1.0.52
3. Huggingface's pre-trained pytorch models for BERT




Huggingface is a brilliant repository of few of amazing state of the art pre-trained models for NLP. Recently it has been renamed (and upgraded as well) to Pytorch-Transformers.

. . .

## INTEGRATION TECHNIQUES

There are basically following techniques I used (with the help of medium article, link of which is given above):

1. Using BERT's Tokenizer
2. Using BERT's Vocab
3. Muting *include_bos* and *include_eos* of Fastai's defaults as False

4. Introducing [CLS] and [SEP] in the beginning and end respectively of each token of BERT
5. A technique to split the model so that discriminative learning can be applied (a novel method being taught in Fastai lectures so that different levels of learning rates and weight decays can be introduced in different parts of the model architecture)

So, let's see how these techniques can be applied:

First we will import BERT Tokenizer from Huggingface's pre-trained BERT model:

```
from pytorch_pretrained_bert import BertTokenizer

bert_tok = BertTokenizer.from_pretrained(
 “bert-base-uncased”,
)
```

There are many tokenizer methods which we can import but we will use the simplest and most common of them all — *“bert-base-uncased”*

Next, we will define a function to create tokenizer depending on above tokenizer model which can be compatible with fastai:

```
class FastAiBertTokenizer(BaseTokenizer):
 """Wrapper around BertTokenizer to be compatible with fast.ai"""

 def __init__(self, tokenizer: BertTokenizer, max_seq_len: int=128,
**kwargs):
 self._pretrained_tokenizer = tokenizer
 self.max_seq_len = max_seq_len

def __call__(self, *args, **kwargs):
 return self

def tokenizer(self, t:str) -> List[str]:
 """Limits the maximum sequence length"""
 return ["[CLS]"] + self._pretrained_tokenizer.tokenize(t)
[:self.max_seq_len - 2] + ["[SEP]"]
```

Here you can see that each token is made to start with [CLS] and end with [SEP].

After that, we will create vocab function:

```
fastai_bert_vocab = Vocab(list(bert_tok.vocab.keys()))
```

and after that, we need to wrap above created tokenizer function in fastai:

```
fastai_tokenizer = Tokenizer(tok_func=FastAiBertTokenizer(bert_tok,
max_seq_len=256), pre_rules=[], post_rules=[])
```

That's it as far as creating BERT tokens and vocab compatible with Fastai library.

We can create our Databunch now as follows:

```
label_cols = [“toxic”, “severe_toxic”, “obscene”, “threat”,
“insult”, “identity_hate”]

databunch_1 = TextDataBunch.from_df(“.”, train, val,
 tokenizer=fastai_tokenizer,
 vocab=fastai_bert_vocab,
 include_bos=False,
 include_eos=False,
 text_cols=”comment_text”,
 label_cols=label_cols,
 bs=32,
 collate_fn=partial(pad_collate, pad_first=False, pad_idx=0),
 )
```

While creating databunch above, please note that we have put *include_bos* and *include_eos* as False. We do this because it somehow interferes with the BERT's [CLS] and [SEP] methods.

I have ignored providing the codes for creation of train and validation data-set here but these can be found at my GitHub account (link will be given below)

In last, for discriminative learning technique, we need to split the model architecture and this can be done as follows:

```
def bert_clas_split(self) -> List[nn.Module]:

 bert = model.bert
 embedder = bert.embeddings
 pooler = bert.pooler
 encoder = bert.encoder
 classifier = [model.dropout, model.classifier]
 n = len(encoder.layer)//3
 print(n)
 groups = [[embedder], list(encoder.layer[:n]),
list(encoder.layer[n+1:2*n]), list(encoder.layer[(2*n)+1:]),
[pooler], classifier]
 return groups
```

Here, the BERT model can be imported as follows:

```
from pytorch_pretrained_bert.modeling import BertConfig,
BertForSequenceClassification, BertForNextSentencePrediction,
BertForMaskedLM


bert_model_class =
BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=6)


model = bert_model_class
```

For the modelling work, we will use following loss function and metrics:

1. Loss function = Binary Cross Entropy with Logistic Loss
2. Metric = accuracy with threshold (with threshold of 25%) considering we can't use simple accuracy as metric here as this is a multi-label classification task

Finally, this is our learner function:

```
from fastai.callbacks import *


learner = Learner(
 databunch_1, model,
 loss_func=loss_func, model_dir='/temp/model', metrics=acc_02,
)
```

We can use above function to split the model in following way:

```
x = bert_clas_split(model)

learner.split([x[0], x[1], x[2], x[3], x[5]])
```

After doing all of these, we can straightforwardly use usual Fastai techniques to train the model such as finding appropriate learning rates range and training post freezing / unfreezing the layers.

For NLP tasks, as mentioned by Jeremy in his lectures, unlike in Image classification / regression tasks, we will gradually unfreeze the layers.

I am not going into details for training procedures for both BERT and ULMFiT models. These techniques are the same which are being taught in Jeremy's classes and can be learnt in much better way there.

Let's see how these model performed in terms of accuracy and prediction.

. . .

## MODEL PERFORMANCE

BERT's performance (after 2 epochs of training):

| epoch | train_loss | valid_loss | accuracy_thresh | time |
|---|---|---|---|---|
| 0 | 0.031048 | 0.037301 | 0.980604 | 25:57 |
| 1 | 0.028199 | 0.038580 | 0.982683 | 28:36 |

BERT's performance on multi-label classification task

This is great performance! **98.27% accuracy** in just 2 epochs of training.

Here is its prediction on few sample texts (remember, after seeing the text, model should be able to tell whether this contains any of the classes we described above such as whether there are any abusive, threatening language or not):

```
In[36]:
text = 'you are so sweet'
learner.predict(text)

ut[36]:
(MultiCategory ,
 tensor([0., 0., 0., 0., 0., 0.]),
```

```
tensor([1.1777e-04, 2.4234e-05, 1.0298e-04, 9.0122e-06, 3.9043e-05, 1.2749e-05]))
```

+ Code + Markdown

In[37]:

```
text = 'you are pathetic piece of shit'
learner.predict(text)
```

ut[37]:

```
(MultiCategory toxic;obscene;insult,
 tensor([1., 0., 1., 0., 1., 0.]),
 tensor([9.9793e-01, 1.6227e-01, 9.8385e-01, 7.9275e-05, 9.7258e-01, 7.4961e-03]))
```

Model is performing really well! It correctly identified that in first text, there is no abusive slang or threats but in next sentence, it identified, toxicity, obscenity and insult

Fastai ULMFiT's performance (after 2 epochs of training):

| epoch | train_loss | valid_loss | accuracy_thresh | time |
|---|---|---|---|---|
| 0 | 0.058955 | 0.455071 | 0.971549 | 10:35 |
| 1 | 0.053523 | 0.454931 | 0.972103 | 10:12 |

Fastai's performance on multi-label classification task

Fastai's ULMFiT's performance is also great (around **97.2% accuracy**). This could have improved further if we could have run few more epochs as still training error is higher than validation error.

Now, let's see how did it predict on same two pieces of texts:

In[66]:
```
learn.predict('she is so sweet')
```

Out[66]:
```
(MultiCategory ,
 tensor([0., 0., 0., 0., 0., 0.]),
 tensor([0.4081, 0.0108, 0.0850, 0.0199, 0.1322, 0.0290]))
```

In[67]:
```
learn.predict('you are son of a bitch ')
```

Out[67]:
```
(MultiCategory toxic;severe_toxic;obscene;insult,
 tensor([1., 1., 1., 0., 1., 0.]),
 tensor([0.9990, 0.7699, 0.9998, 0.0061, 0.9787, 0.1059]))
```

Pretty neat! It predicted one additional category than what was predicted by BERT

. . .

## CONCLUSION

We have seen in this article that how we can integrate the power of BERT with the simplicity of Fastai and make gains from both worlds.

Both models have performed really well on this multi-label text classification task.

Few important things to note are:

1. Tokenizer and Vocab of BERT must be carefully integrated with Fastai
2. [CLS] and [SEP] needs to be carefully inserted into each token
3. Model architecture splitting is necessary if we would like to take the advantage of discriminative learning which is being taught in Fastai

Here is the GitHub link for my notebook (it can be a bit messy, so kindly excuse me for that)

**abhikjha/Fastai-integration-with-BERT**

Step wise instructions to integrate the power of BERT with Fastai - abhikjha/Fastai-integration-with-BERT

github.com

And same can be found on my Kaggle kernel as well:

https://www.kaggle.com/abhikjha/jigsaw-toxicity-bert-with-fastai-and-fastai/notebook

If you like my article, request you to kindly share and clap for it :)

Machine Learning Deep Learning Fastai Bert NLP