How to Train a BERT Model From Scratch

Meet BERT's Italian cousin, FiliBERTo

James Briggs Jul 6 · 7 min read ★

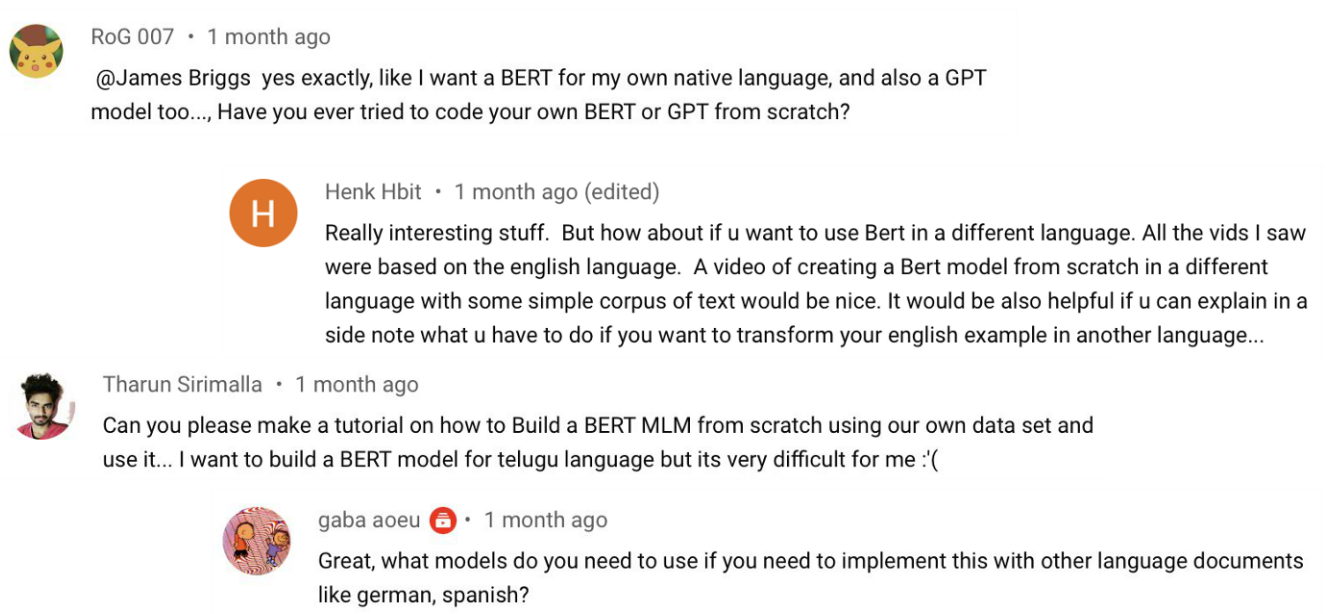
BERT, but in Italy — image by author

Many of my articles have been focused on BERT — the model that came and dominated the world of natural language processing (NLP) and marked a new age for language models.

- `pip install transformers`
- Initialize a pre-trained transformers model — `from_pretrained`.
- Test it on some data.
- *Maybe* fine-tune the model (train it some more).

Now, this is a great approach, but if we only ever do this, we lack the understanding behind creating our own transformers models.

And, if we cannot create our own transformer models — we must rely on there being a pre-trained model that fits our problem, this is not always the case:



A few comments asking about non-English BERT models

So in this article, we will explore the steps we must take to build our own transformer model — specifically a further developed version of BERT, called RoBERTa.

An Overview

There are a few steps to the process, so before we dive in let's first summarize what we need to do. In total, there are four key parts:

- Building a tokenizer
- Creating an input pipeline
- Training the model

Once we have worked through each of these sections, we will take the tokenizer and model we have built — and save them both so that we can then use them in the same way we usually would with `from_pretrained`.

Getting The Data

As with any machine learning project, we need data. In terms of data for training a transformer model, we really are spoilt for choice — we can use almost any text data.

Video walkthrough for downloading OSCAR dataset using HuggingFace's datasets library

And, if there's one thing that we have plenty of on the internet — it's unstructured text data.

The OSCAR dataset boasts a huge number of different languages — and one of the clearest use-cases for training from scratch is so that we can apply BERT to some less commonly used languages, such as Telugu or Navajo.

Unfortunately, the only language I can speak with any degree of competency is English — but my girlfriend is Italian, and so she — Laura, will be assessing the results of our Italian-speaking BERT model — FiliBERTo.

So, to download the Italian segment of the OSCAR dataset we will be using HuggingFace's `datasets` library — which we can install with `pip install datasets`. Then we download OSCAR_IT with:

```
In [1]: from datasets import load_dataset
```

Load the **Italian** part of the **OSCAR** (https://huggingface.co/datasets/oscar) dataset. This is a *huge* dataset so download can take a long time:

```
In [2]: dataset = load_dataset('oscar', 'unshuffled_deduplicated_it')

        Reusing dataset oscar (C:\Users\James\.cache\huggingface\datas
        ets\oscar\unshuffled_deduplicated_it\1.0.0\e4f06cecc7ae02f7adf
        85640b4019bf476d44453f251a1d84aebae28b0f8d51d)
```

oscar_it.ipynb hosted with ❤ by GitHub view raw

Let's take a look at the `dataset` object.

```
        num_rows: 28522082
    })
```

```
ing', id=None)}
```

Let's take a look at a single sample:

```
In [7]: dataset['train'][0]

Out[7]: {'id': 0,
         'text': "La estrazione numero 48 del 10 e LOTTO ogni 5 minuti
        e' avvenuta sabato 15 settembre 2018 alle ore 04:00 a Roma, ne
        l Centro Elaborazione Dati della Lottomatica Italia (ora GTech
        SpA), con la supervisione della Amministrazione Autonoma dei M
        onopoli di Stato (AAMS), incaricata di vigilare sulla regolari
        tà delle operazioni di sorteggio.\nIl Montepremi della 48ª est
        razione viene ripartito tra i vincitori delle singole categori
        e di premio.\nRicorda di controllare il Numero ORO 53. E, se l
        o hai giocato, anche il DOPPIO ORO 53 e 66. Se indovini puoi v
        incere premi più ricchi.\nIl nostro sito web impiega cookies p
        er migliorare la navigazione del visitatore. L'utente è consap
        evole che, continuando a visitare il nostro sito web, accetta
        l'utilizzo dei cookies Accetto Informazioni\n(C) Copyright 201
        3-2017 10elotto.biz | Il presente sito è da considerarsi un si
        to indipendente, NON collegato alla rete ufficiale Gtech Sp
```

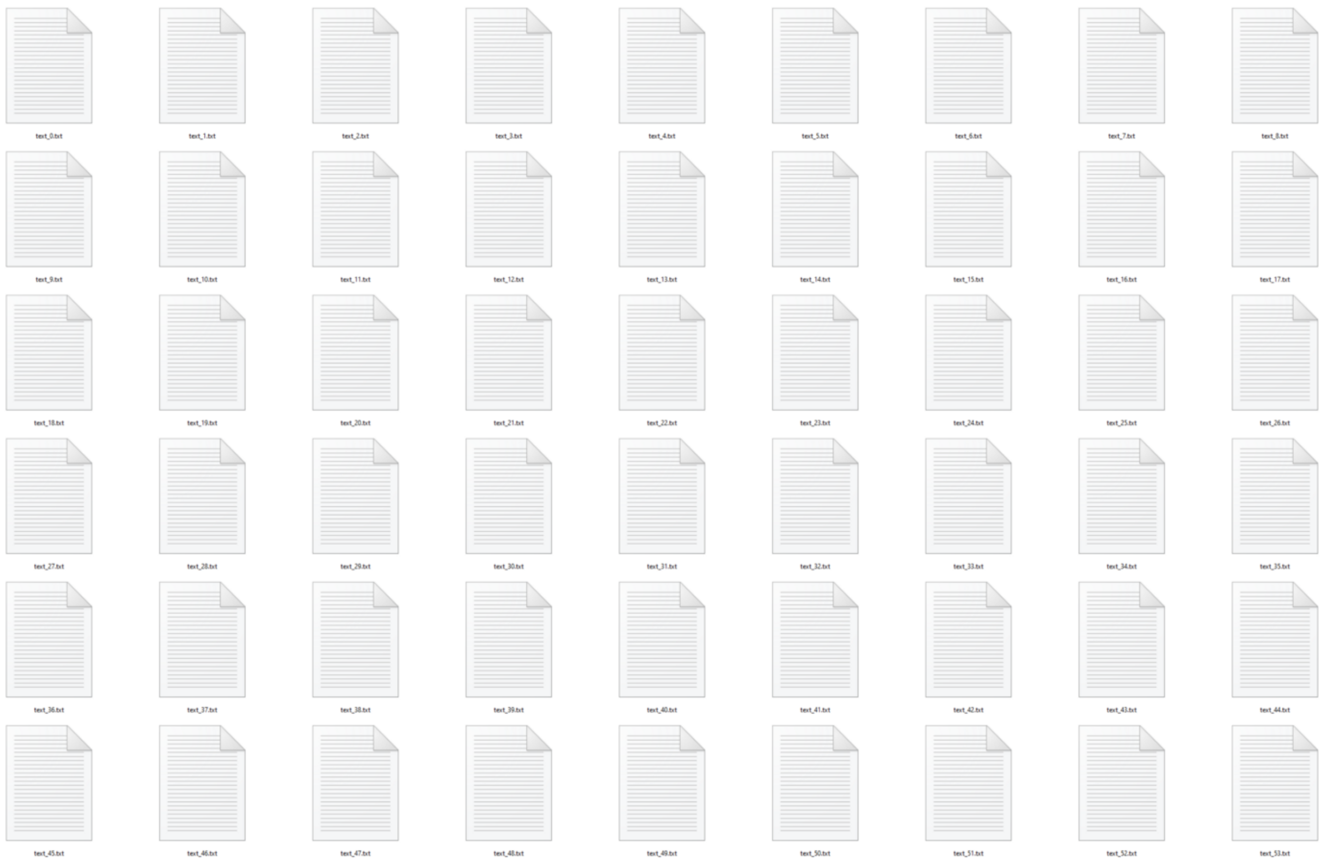
oscar_it_dataset.ipynb hosted with ❤ by GitHub view raw

Great, now let's store our data in a format that we can use when building our tokenizer. We need to create a set of plaintext files containing just the `text` feature from our dataset, and we will split each *sample* using a newline `\n`.

```
In [8]: from tqdm.auto import tqdm

        text_data = []
        file_count = 0

        for sample in tqdm(dataset['train']):
            sample = sample['text'].replace('\n', '')
            text_data.append(sample)
            if len(text_data) == 10_000:
                # once we git the 10K mark, save to file
                with open(f'../../data/text/oscar_it/text_{file_count
        }.txt', 'w', encoding='utf-8') as fp:
                    fp.write('\n'.join(text_data))
                text_data = []
                file_count += 1
        # after saving in 10K chunks, we will have ~2082 leftover sam
        ples, we save those now too
        with open(f'../../data/text/oscar_it/text_{file_count}.txt',
        'w', encoding='utf-8') as fp:
            fp.write('\n'.join(text_data))

        100%|██████████| 28522082/28522082 [33:32<00:00, 14173.48it/s]
```

save_oscar.ipynb hosted with ❤ by GitHub view raw

Over in our `data/text/oscar_it` directory we will find:



The directory containing our plaintext OSCAR files

Building a Tokenizer

Next up is the tokenizer! When using transformers we typically load a tokenizer, alongside its respective transformer model — the tokenizer is a key component in the process.

Build a Custom Transformer Tokenizer - Transformers From Scratch #2

Video walkthrough for building our custom tokenizer

When building our tokenizer we will feed it all of our OSCAR data, specify our vocabulary size (number of tokens in the tokenizer), and any special tokens.

Now, the RoBERTa special tokens look like this:

Token	Use
<s>	Beginning of sequence (BOS) or classifier (CLS) token
</s>	End of sequence (EOS) or seperator (SEP) token
<unk>	Unknown token
<pad>	Padding token
<mask>	Masking token

roberta_tokens.md hosted with ❤ by GitHub view raw

So, we make sure to include them within the `special_tokens` parameter of our tokenizer's `train` method call.

Get a list of paths to each file in our *oscar_it* directory.

```
In [1]: from pathlib import Path
        paths = [str(x) for x in Path('../../data/text/oscar_it').glo
        b('**/*.txt')]
```

Now we move onto training the tokenizer. We use a byte-level Byte-pair encoding (BPE) tokenizer. This allows us to build the vocabulary from an alphabet of single bytes, meaning all

```
tokenizer = ByteLevelBPETokenizer()
```

```
In [3]: tokenizer.train(files=paths[:5], vocab_size=30_522, min_frequ
ency=2,
                special_tokens=['<s>', '<pad>', '</s>', '<unk
>', '<mask>'])
```


Our tokenizer is now ready, and we can save it file for later use:

```
In [4]: import os

        os.mkdir('./filiberto')

        tokenizer.save_model('filiberto')

Out[4]: ['./filiberto\\vocab.json', './filiberto\\merges.txt']
```


Now we have two files that define our new *FiliBERTo* tokenizer:

- *merges.txt* — performs the initial mapping of text to tokens

And with those, we can move on to initializing our tokenizer so that we can use it as we would use any other `from_pretrained` tokenizer.

Initializing the Tokenizer

We first initialize the tokenizer using the two files we built before — using a simple `from_pretrained`:

```
In [1]: from transformers import RobertaTokenizer

        # initialize the tokenizer using the tokenizer we initialized
        and saved to file
        tokenizer = RobertaTokenizer.from_pretrained('filiberto', max
        _len=512)
```

tokenizer_init.ipynb hosted with ❤ by GitHub view raw

Now our tokenizer is ready, we can try encoding some text with it. When encoding we use the same two methods we would typically use, `encode` and `encode_batch`.

```
In [6]: # test our tokenizer on a simple sentence
        tokens = tokenizer('ciao, come va?')

In [7]: print(tokens)

        {'input_ids': [0, 16834, 16, 488, 611, 35, 2], 'attention_mas
        k': [1, 1, 1, 1, 1, 1, 1]}
```

encoding_detail.ipynb hosted with ❤ by GitHub view raw

From the encodings object `tokens` we will be extracting the `input_ids` and `attention_mask` tensors for use with FiliBERTo.

Creating the Input Pipeline

The input pipeline of our training process is the more complex part of the entire process. It consists of us taking our raw OSCAR training data, transforming it, and loading it into a `DataLoader` ready for training.

Building MLM Training Input Pipeline - Transformers From Scratch #3

Preparing the Data

We'll start with a single sample and work through the preparation logic.

First, we need to open our file — the same files that we saved as *.txt* files earlier. We split each based on newline characters `\n` as this indicates the individual samples.

```
In [3]: with open('../../data/text/oscar_it/text_0.txt', 'r', encodin
        g='utf-8') as fp:
            lines = fp.read().split('\n')
```

open_file.ipynb hosted with ❤ by GitHub view raw

Then we encode our data using the `tokenizer` — making sure to include key parameters like `max_length`, `padding`, and `truncation`.

And now we can move onto creating our tensors — we will be training our model through masked-language modeling (MLM). So, we need three tensors:

- ***input_ids*** — our *token_ids* with ~15% of tokens masked using the mask token `<mask>` .
- ***attention_mask*** — a tensor of **1**s and **0**s, marking the position of 'real' tokens/padding tokens — used in attention calculations.
- ***labels*** — our *token_ids* with **no** masking.

If you're not familiar with MLM, I've explained it here.

Our `attention_mask` and `labels` tensors are simply extracted from our `batch` . The `input_ids` tensors require more attention however, for this tensor we mask ~15% of the tokens — assigning them the token ID `3` .

Get started Open in app

In the final output, we can see part of an encoded `input_ids` tensor. The very first token ID is `1` — the `[CLS]` token. Dotted around the tensor we have several `3` token IDs — these are our newly added `[MASK]` tokens.

Building the DataLoader

Next, we define our `Dataset` class — which we use to initialize our three encoded tensors as PyTorch `torch.utils.data.Dataset` objects.

Finally, our `dataset` is loaded into a PyTorch `DataLoader` object — which we use to load our data into our model during training.

Training the Model

We need two things for training, our `DataLoader` and a model. The `DataLoader` we have — but no model.

Initializing the Model

For training, we need a raw (not pre-trained) `BERTLMHeadModel`. To create that, we first need to create a RoBERTa config object to describe the parameters we'd like to initialize FiliBERTo with.

Then, we import and initialize our RoBERTa model with a language modeling (LM) head.

Training Preparation

Before moving onto our training loop we need to set up a few things. First, we set up GPU/CPU usage. Then we activate the training mode of our model — and finally, initialize our optimizer.

Training

Finally — training time! We train just as we usually would when training via PyTorch.

If we head on over to Tensorboard we'll find our loss over time — it looks promising.

Loss / time — multiple training sessions have been threaded together in this chart

The Real Test

Now it's time for the real test. We set up an MLM pipeline — and ask Laura to assess the results. You can watch the video review at 22:44 here:

We first initialize a `pipeline` object, using the `'fill-mask'` argument. Then begin testing our model like so:

We start with *"buongiorno, come va?"* — or *"good day, how are you?"*:

The first answer, "buongiorno, chi va?" means "good day, who is there?" — eg nonsensical. But, our second answer is correct!

Next up, a slightly harder phrase, *"ciao, dove ci incontriamo oggi pomeriggio?"* — or *"hi, where are we going to meet this afternoon?"*:

And we return some more positive results:

```
✅ "hi, where do we see each other this afternoon?"
✅ "hi, where do we meet this afternoon?"
❌ "hi, where here we are this afternoon?"
✅ "hi, where are we meeting this afternoon?"
✅ "hi, where do we meet this afternoon?"
```

Finally, one more, harder sentence, *"cosa sarebbe successo se avessimo scelto un altro giorno?"* — or "what would have happened if we had chosen another day?":

We return a few good more good answers here too:

```
✅ "what would have happened if we had chosen another day?"
✅ "what would have happened if I had chosen another day?"
✅ "what would have happened if they had chosen another day?"
✅ "what would have happened if you had chosen another day?"
❌ "what would have happened if another day was chosen?"
```

Overall, it looks like our model passed Laura's tests — and we now have a competent Italian language model called FiliBERTo!

That's it for this walkthrough of training a BERT model from scratch!

I hope you enjoyed this article! If you have any questions, let me know via Twitter or in the comments below. If you'd like more content like this, I post on YouTube too.

Thanks for reading!

**All images are by the author except where stated otherwise*

Artificial Intelligence Machine Learning Technology NLP Programming