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

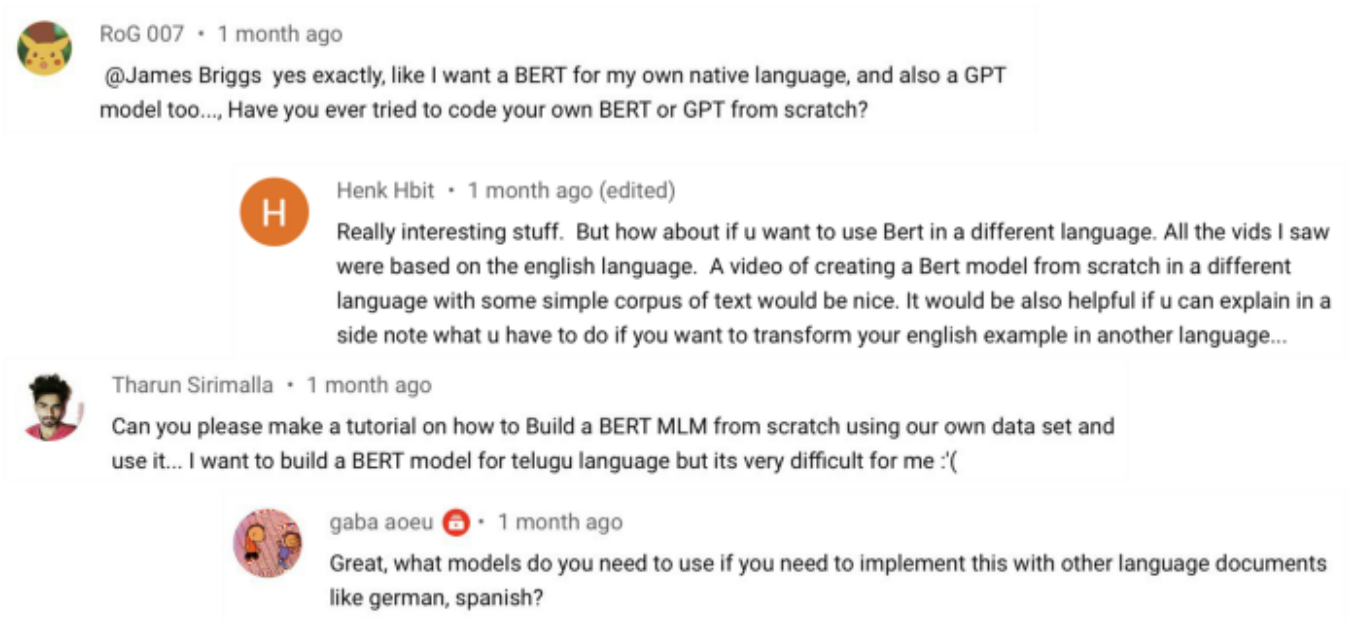
Many of my articles have been focused on BERT — the model that came and dominated the world of natural language processing (NLP) and marked a new

For those of you that may not have used transformers models (eg what BERT is) before, the process looks a little like this:

- `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.

need to do. In total, there are four key parts:

- Getting the data
- 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.

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

One of the largest datasets in the domain of text scraped from the internet is the OSCAR dataset.

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\datasets\oscar\unshuffled_deduplicated_it\1.0.0\e4f06cecc7ae02f7adf85640b4019bf476d44453f251a1d84aebae28b0f8d51d)
```

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

The `dataset` is a `DatasetDict` containing a single `train` dataset.

```
In [3]: dataset
```

```
Out[3]: DatasetDict({
            train: Dataset({
                features: ['id', 'text'],
                num_rows: 28522082
            })
        })
```

We can access the dataset itself through the `train` key. From here we can view more information, like the number of rows and structure of the dataset.

```
In [5]: dataset['train']
```

```
Out[5]: Dataset({
            features: ['id', 'text'],
            num_rows: 28522082
        })
```

```
In [6]: dataset['train'].features
```

```
Out[6]: {'id': Value(dtype='int64', id=None), 'text': Value(dtype='string',
        id=None)}
```

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`.

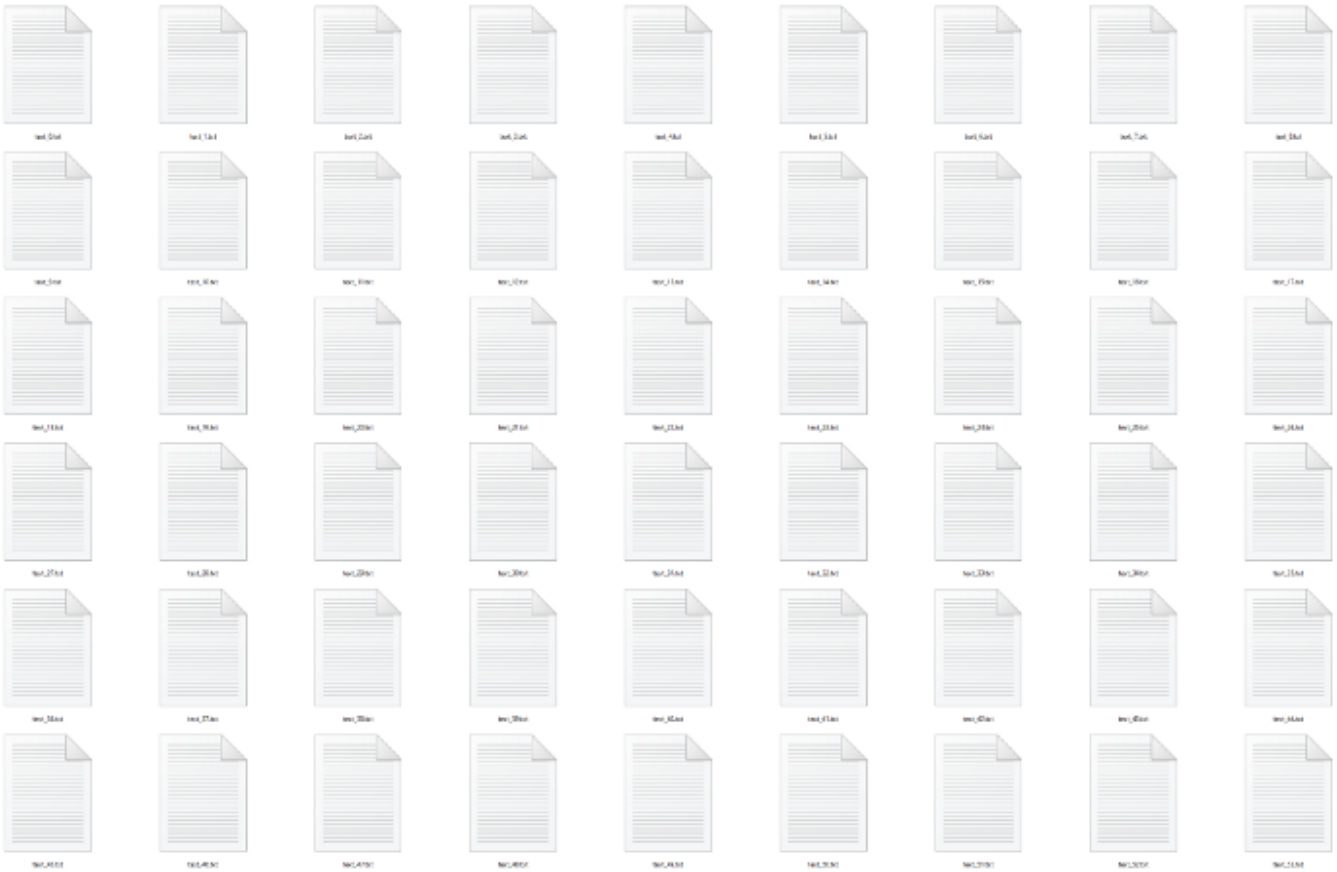
```
In [8]: from tqdm.auto import tqdm

        text_data = []
        file_count = 0

        for sample in tqdm(dataset['train']):
            sample = sample['text'].replace('\n', '')
```

```
        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 samples,
 we save those now too
with open(f'../../data/text/oscar_it/text_{file_count}.txt', 'w', e
ncoding='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

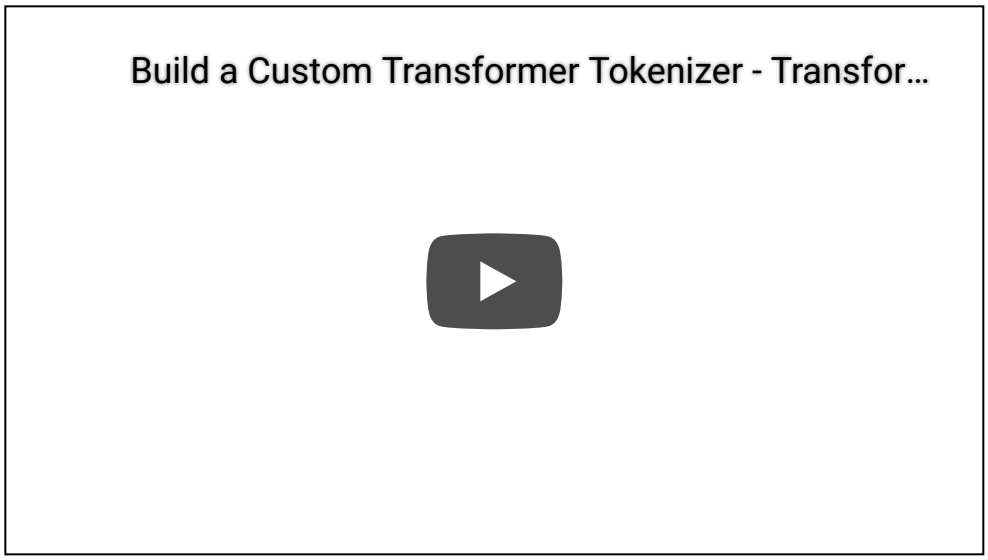
Over in our `data/text/oscar_it` directory we will find:



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.



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

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').glob('**/
        *.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 words will be decomposable into tokens.

```
In [2]: from tokenizers import ByteLevelBPETokenizer

        tokenizer = ByteLevelBPETokenizer()
```

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

tokenizer_build.ipynb hosted with ♡ by GitHub view raw

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']
```

save_model.ipynb hosted with ♡ by GitHub view raw

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

- *merges.txt* — performs the initial mapping of text to tokens
- *vocab.json* — maps the tokens to token IDs

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 s
aved to file
tokenizer = RobertaTokenizer.from_pretrained('filiberto', max_len=5
12)
```

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_mask':
        [1, 1, 1, 1, 1, 1, 1]}

In [11]: tokens.input_ids

Out[11]: [0, 16834, 16, 488, 611, 35, 2]
```

encoding_detail.ipynb hosted with ♡ by GitHub view raw

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.

Video walkthrough of the MLM input pipeline

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', encoding='utf
        -8') as fp:
```

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`.

```
In [4]: batch = tokenizer(lines, max_length=512, padding='max_length', truncation=True)
        len(batch)

Out[4]: 10000
```

encode_batch.ipynb hosted with ♡ by GitHub view raw

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`.

In [6]:
```
import torch

labels = torch.tensor([x.ids for x in batch])
mask = torch.tensor([x.attention_mask for x in batch])
```

In [7]:
```
# make copy of labels tensor, this will be input_ids
input_ids = labels.detach().clone()
# create random array of floats with equal dims to input_ids
rand = torch.rand(input_ids.shape)
# mask random 15% where token is not 0 [PAD], 1 [CLS], or 2 [SEP]
mask_arr = (rand < .15) * (input_ids != 0) * (input_ids != 1) * (in
put_ids != 2)
# loop through each row in input_ids tensor (cannot do in parallel)
for i in range(input_ids.shape[0]):
    # get indices of mask positions from mask array
    selection = torch.flatten(mask_arr[i].nonzero()).tolist()
    # mask input_ids
    input_ids[i, selection] = 3  # our custom [MASK] token == 3
```

```
Out[8]:  torch.Size([10000, 512])
```

create_tensors.ipynb hosted with ♡ by GitHub view raw

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.

```
In [7]: encodings = {'input_ids': input_ids, 'attention_mask': mask, 'label
        s': labels}
```

```
In [8]: class Dataset(torch.utils.data.Dataset):
            def __init__(self, encodings):
                # store encodings internally
                self.encodings = encodings

            def __len__(self):
                # return the number of samples
                return self.encodings['input_ids'].shape[0]

            def __getitem__(self, i):
                # return dictionary of input_ids, attention_mask, and label
        s for index i
                return {key: tensor[i] for key, tensor in self.encodings.it
        ems()}
```

Next we initialize our `Dataset`.

```
In [9]: dataset = Dataset(encodings)
```

And initialize the dataloader, which will load the data into the model during training.

build_dataloader.ipynb hosted with ♡ by GitHub view raw

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.

```
In [11]: from transformers import RobertaConfig

         config = RobertaConfig(
             vocab_size=30_522,  # we align this to the tokenizer vocab_size
             max_position_embeddings=514,
             hidden_size=768,
             num_attention_heads=12,
```

roberta_config.ipynb hosted with ♡ by GitHub view raw

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

```
In [12]: from transformers import RobertaForMaskedLM

         model = RobertaForMaskedLM(config)
```

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.

Setup GPU/CPU usage.

```
In [13]: device = torch.device('cuda') if torch.cuda.is_available() else tor
         ch.device('cpu')
         # and move our model over to the selected device
         model.to(device)
```

Activate the training mode of our model, and initialize our optimizer (Adam with weighted decay - reduces chance of overfitting).

```
In [15]: from transformers import AdamW

         # activate training mode
         model.train()
         # initialize optimizer
         optim = AdamW(model.parameters(), lr=1e-4)
```

setup_training.ipynb hosted with ♡ by GitHub view raw

Training

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

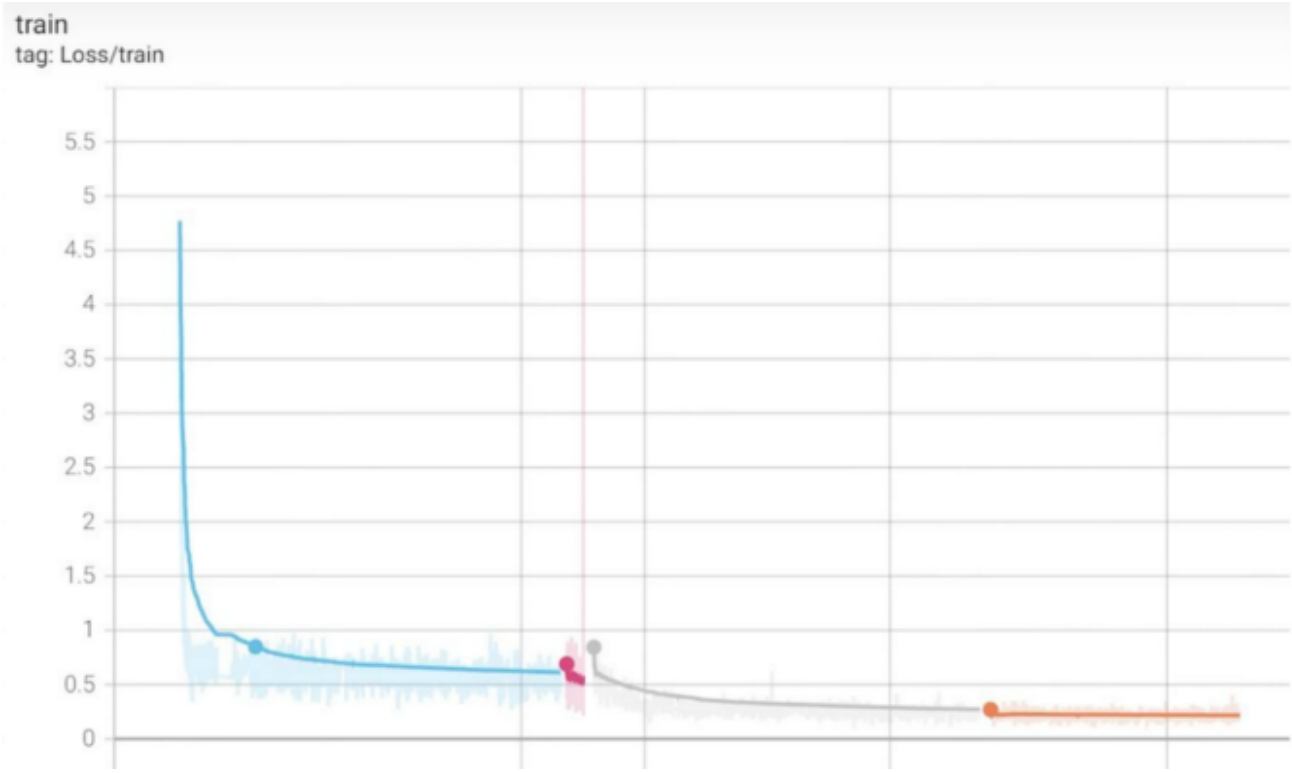
```
In [20]: epochs = 2

         for epoch in range(epochs):
             # setup loop with TQDM and dataloader
             loop = tqdm(loader, leave=True)
             for batch in loop:
```

```
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        # process
        outputs = model(input_ids, attention_mask=attention_mask,
                        labels=labels)
        # extract loss
        loss = outputs.loss
        # calculate loss for every parameter that needs grad update
        loss.backward()
        # update parameters
        optim.step()
        # print relevant info to progress bar
        loop.set_description(f'Epoch {epoch}')
        loop.set_postfix(loss=loss.item())
```

```
Epoch 0: 100%|██████████| 12500/12500 [1:29:47<00:00,  2.32it/s, lo
ss=0.358]
```

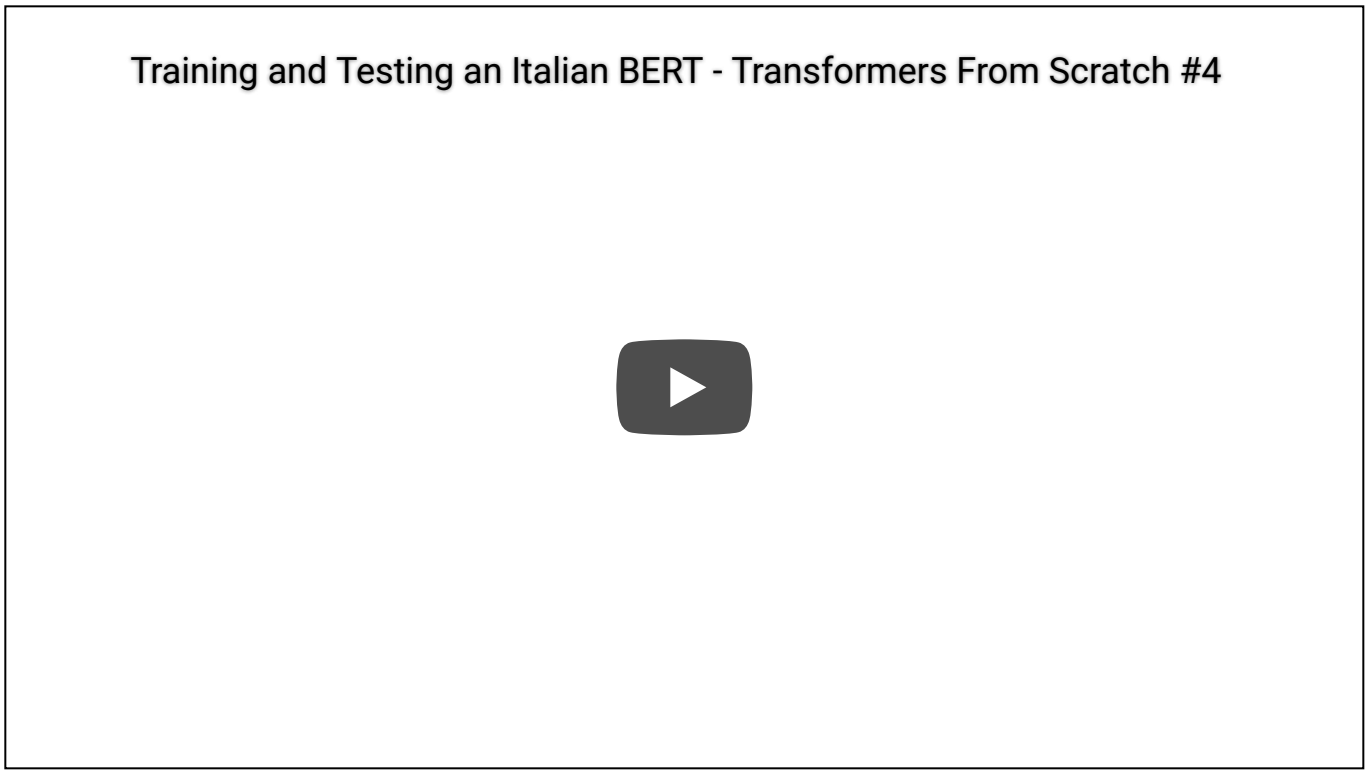
training_loop.ipynb hosted with ♡ by GitHub view raw

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

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:

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

In [2]: fill = pipeline('fill-mask', model='filiberto', tokenizer='filibert
        o')

        Some weights of RobertaModel were not initialized from the model ch
        eckpoint at filiberto and are newly initialized: ['roberta.pooler.d
        ense.weight', 'roberta.pooler.dense.bias']
        You should probably TRAIN this model on a down-stream task to be ab
        le to use it for predictions and inference.

In [3]: fill(f'ciao {fill.tokenizer.mask_token} va?')

Out[3]: [{'sequence': '<s>ciao come va?</s>',
          'score': 0.33601945638656616,
          'token': 482,
          'token_str': 'Ġcome'},
         {'sequence': '<s>ciao, va?</s>',
          'score': 0.13736604154109955,
```

```
 'token': 474,
 'token_str': 'Ġmi'},
{'sequence': '<s>ciao chi va?</s>',
 'score': 0.04759546741843236,
```

test.ipynb hosted with ♡ by GitHub view raw

*"ciao **come** va?"* is the right answer! That's as advanced as my Italian gets — so, let's hand it over to Laura.

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

```
In [3]: fill(f'buongiorno, {fill.tokenizer.mask_token} va?')

Out[3]: [{'sequence': '<s>buongiorno, chi va?</s>',
          'score': 0.299,
          'token': 586,
          'token_str': 'Ġchi'},
         {'sequence': '<s>buongiorno, come va?</s>',
          'score': 0.245,
          'token': 482,
          'token_str': 'Ġcome'},
         {'sequence': '<s>buongiorno, cosa va?</s>',
          'score': 0.116,
          'token': 1021,
          'token_str': 'Ġcosa'},
         {'sequence': '<s>buongiorno, non va?</s>',
          'score': 0.041,
          'token': 382,
          'token_str': 'Ġnon'},
         {'sequence': '<s>buongiorno, che va?</s>',
          'score': 0.037,
          'token': 313,
          'token_str': 'Ġche'}]
```

test_1.ipynb hosted with ♡ by GitHub view raw

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

```
In [3]: fill(f'ciao, dove ci {fill.tokenizer.mask_token} oggi pomeriggio? '
        )

Out[3]: [{'sequence': '<s>ciao, dove ci vediamo oggi pomeriggio? </s>',
          'score': 0.400,
          'token': 7105,
          'token_str': 'Ġvediamo'},
         {'sequence': '<s>ciao, dove ci incontriamo oggi pomeriggio? </s>',
          'score': 0.118,
          'token': 27211,
          'token_str': 'Ġincontriamo'},
         {'sequence': '<s>ciao, dove ci siamo oggi pomeriggio? </s>',
          'score': 0.087,
          'token': 1550,
          'token_str': 'Ġsiamo'},
         {'sequence': '<s>ciao, dove ci troviamo oggi pomeriggio? </s>',
          'score': 0.048,
          'token': 5748,
          'token_str': 'Ġtroviamo'},
         {'sequence': '<s>ciao, dove ci ritroviamo oggi pomeriggio? </s>',
          'score': 0.046,
          'token': 22070,
          'token_str': 'Ġritroviamo'}]
```

test_2.ipynb hosted with ♡ by GitHub view raw

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?”:

```
In [3]: fill(f'cosa sarebbe successo se {fill.tokenizer.mask_token} scelto
```

```
iorno?</s>',
  'score': 0.251,
  'token': 6691,
  'token_str': 'Ġavesse'},
 {'sequence': '<s>cosa sarebbe successo se avessi scelto un altro g
iorno?</s>',
  'score': 0.241,
  'token': 12574,
  'token_str': 'Ġavessi'},
 {'sequence': '<s>cosa sarebbe successo se avessero scelto un altro
giorno?</s>',
  'score': 0.217,
  'token': 14193,
  'token_str': 'Ġavessero'},
 {'sequence': '<s>cosa sarebbe successo se avete scelto un altro gi
orno?</s>',
  'score': 0.081,
  'token': 3609,
  'token_str': 'Ġavete'},
 {'sequence': '<s>cosa sarebbe successo se venisse scelto un altro
giorno?</s>',
  'score': 0.042,
  'token': 17216,
```

test_3.ipynb hosted with ♡ by GitHub view raw

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