Training a DistilBERT model from scratch

Sabrina Herbst · Following
Published in Artificial Intelligence in Plain English · 5 min read · Jan 22

LLMs seem to be taking over the world, many people still do not really understand how they seem to work. I worked with Machine Learning for some years now and am absolutely fascinated by Natural Language Processing and the recent progress.

Even though I read most of the accompanying papers, training these models still seemed like a mystery to me, which is why I decided to go ahead and train one myself to really get an idea on how it works. I combined this with training a Q&A model, but will only go into detail on the DistilBERT model here.

To make life easier for you, I have decided to provide a short review on how it works. Please view the `distilbert.ipynb` file in [1] to find the relevant code.

Cedric Yong: pixabay.com

Why DistilBERT

The first question to answer is why I chose DistilBERT over BERT, ALBERT and all other variants of the model. Unfortunately, I do not have unlimited cloud computing access and only a local GPU with limited memory, so I had to optimize for model size and training time rather than performance.

That said, officially DistilBERT has only a performance degradation of 3% compared to BERT, which seemed like a reasonable tradeoff. BERT base has 110 million parameters and was trained for approx. 12 days, whereas DistilBERT has 66 million and was trained for only about 3.5 days. Sanh et al. [3] in the original paper report a 40% reduced size, retaining 97% of the language understanding capabilities and being 60% faster.

I looked at [2] for a short summarization and comparison of BERT, RoBERTA, DistilBERT and XLNet. He provides a great table in his review, comparing all the models.

Data

I used the OpenWebText dataset [4] from HuggingFace (https://huggingface.co/datasets/openwebtext) to train the model. It is an Open-Source version of the WebText dataset from OpenAI. It contains 8013769 paragraphs that were sampled from Reddit.

HuggingFace provides an amazing (!!!) interface for a lot of datasets and models, which I have used throughout this whole project. You can download the entire dataset using just the command below.

```
from datasets import load_dataset

ds = load_dataset("openwebtext")
```

I then went on to store the dataset in chunks of 10 000 locally, because it takes some time and I didn't want to wait every time.

Tokenization

Next, we need to train a tokenizer for the model (as we cannot feed natural language into a model). We can use HuggingFace's BertWordPieceTokenizer. We can just pass the paths of the files, and it will do everything automatically. Additionally, we need to add the special tokens `PAD` (padding), `UNK` (unknown), `CLS` (classification), `SEP` (separator) and the `MSK` (mask) token. For an explanation of these tokens, please refer to a basic BERT model tutorial.

```
from tokenizers import BertWordPieceTokenizer

paths = [str(x) for x in Path('data/original').glob('**/*.txt')]

tokenizer = BertWordPieceTokenizer(
        clean_text=True,
        handle_chinese_chars=False,
        strip_accents=False,
        lowercase=True
)
tokenizer.train(files=paths[:10], vocab_size=30_000, min_frequency=2,
                    limit_alphabet=1000, wordpieces_prefix='##',
                    special_tokens=['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]'
```

When we test it, we get the following tokens and decoding them again shows us that the tokenizer adds a CLS token in the beginning of every input and adds the separator token after the sentence. Also, we see that the tokenized input contains input ids (the ids for every word) and the attention masks (telling the model which tokens are important, i.e. they would be 0 if we padded the sequence to a given length).

```
tokens = tokenizer('Hello, how are you?')
print(tokens)
# {'input_ids': [2, 21694, 16, 2287, 2009, 1991, 35, 3],
# 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}

tokenizer.decode(tokens['input_ids'])
# '[CLS] hello, how are you? [SEP]'
```

Dataset and Dataloader

We can go ahead and prepare our data to be loaded into a model, using a self-defined Dataset class and the DataLoader from PyTorch. The dataset

class can be found here. We basically load the files and encode the input using our tokenizer.

Another thing I did in the Dataset was to load the files one by one. Given the memory constraint, I had to implement it that way. It has some drawbacks, namely you cannot shuffle the data this way, as this will mess everything up. This should not be too much of a problem though, as the dataset is already shuffled according to the dataset description.

During training, the model tries to predict masked tokens, which we need to mask ourselves. I therefore masked (assign `MSK` token) 15% of the input, which worked quite well. Some of this is based on the HuggingFace implementation of DistilBERT, which can be found in [5].

```
dataset = Dataset(paths = [str(x) for x in Path('data/original').glob('**/*.txt'
loader = torch.utils.data.DataLoader(dataset, batch_size=8)

test_dataset = Dataset(paths = [str(x) for x in Path('data/original').glob('**/*
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=4)
```

Model

Next we have to define our model and yes, you guessed it, we use HuggingFace here too. It provides an amazing interface, which makes training very easy.

```
from transformers import DistilBertForMaskedLM, DistilBertConfig

config = DistilBertConfig(
    vocab_size=30000,
    max_position_embeddings=514
```

```
)
model = DistilBertForMaskedLM(config)
```

We use AdamW with a learning rate of 1e-4 as the optimizer and train for 10 epochs (which already takes a lot of time). In the following, you can find my training procedure, which is very basic.

```
epochs = 10

for epoch in range(epochs):
    loop = tqdm(loader, leave=True)

    # set model to training mode
    model.train()
    losses = []

    # iterate over dataset
    for batch in loop:
        optim.zero_grad()

        # copy input to device
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        # predict
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)

        # update weights
        loss = outputs.loss
        loss.backward()

        optim.step()

        # output current loss
        loop.set_description(f'Epoch {epoch}')
        loop.set_postfix(loss=loss.item())
        losses.append(loss.item())

    print("Mean Training Loss", np.mean(losses))
    losses = []
    loop = tqdm(test_loader, leave=True)
```

```
    # set model to evaluation mode
    model.eval()

    # iterate over dataset
    for batch in loop:
        # copy input to device
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        # predict
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)

        # update weights
        loss = outputs.loss

        # output current loss
        loop.set_description(f'Epoch {epoch}')
        loop.set_postfix(loss=loss.item())
        losses.append(loss.item())
    print("Mean Test Loss", np.mean(losses))
```

Testing

Afterwards, we can run some sanity tests to see what the model predicts for some masked tokens. We can again use HuggingFace to create a pipeline, that will handle the predictions for us. We use `fill.tokenizer.mask_token` to add a `MSK` token to the input.

```
from transformers import pipeline

fill = pipeline("fill-mask", model='distilbert', config=config, tokenizer='disti
fill(f'It seems important to tackle the climate {fill.tokenizer.mask_token}.')
```

Furthermore, we get the following predictions with confidence levels, which all seem to be a reasonable next token in this sentence.

1. change: 0.19
2. crisis: 0.12
3. issues: 0.05
4. issue: 0.04

Conclusion

All in all, the results are quite good, given the infrastructure limitations. We obviously do not achieve a performance comparable to the original one, but you can use the pretrained models (see [6]) if you really want to use this in an application.

The main idea was to train a model ourselves and really get an idea of how they work. I hoped it helped!

[1] https://github.com/sabrinaherbst/distilbert_question_answering

[2] https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8

[3] Sanh et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv* abs/1910.01108. 2019.

[4] Gokaslan et al. OpenWebText Corpus. 2019.

[5] https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertForMaskedLM

[6] https://huggingface.co/distilbert-base-uncased

*More content at **PlainEnglish.io**. Sign up for our **free weekly newsletter**. Follow us on **Twitter**, **LinkedIn**, **YouTube**, and **Discord**.*

Build awareness and adoption for your tech startup with Circuit.

Machine Learning | AI | NLP | Artificial Intelligence | Naturallanguageprocessing

Written by Sabrina Herbst

Following

41 Followers · Writer for Artificial Intelligence in Plain English

CS student currently doing a data science master's degree

More from Sabrina Herbst and Artificial Intelligence in Plain English

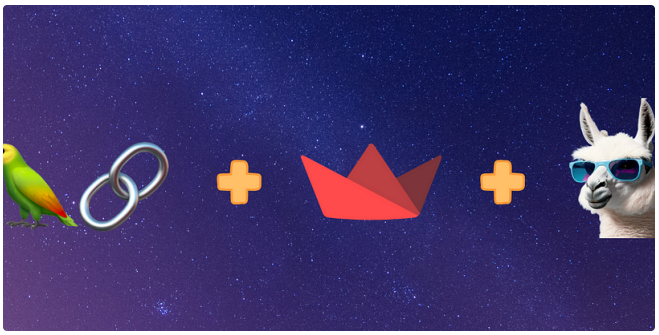
Sabrina Herbst in MLearning.ai

Imputing Missing Values with Machine Learning-Based...

After already diving into data preparation in my previous blog post (check out An...

4 min read · Mar 20, 2022

91



Afaque Um... in Artificial Intelligence in Plain Engli...

🦜 LangChain + Streamlit🔥+ Llama 🦙: Bringing Conversation...

Integrating Open Source LLMs and LangChain for Free Generative Question...

14 min read · Jun 23

3.3K 39

Anthony Alca... in Artificial Intelligence in Plain En...

Vector Search Is Not All You Need

Introduction

· 6 min read · Sep 18

478 14

Sabrina Herbst

Model Soups for Higher Performing Models

Recently, researchers from various universities and companies published a pap...

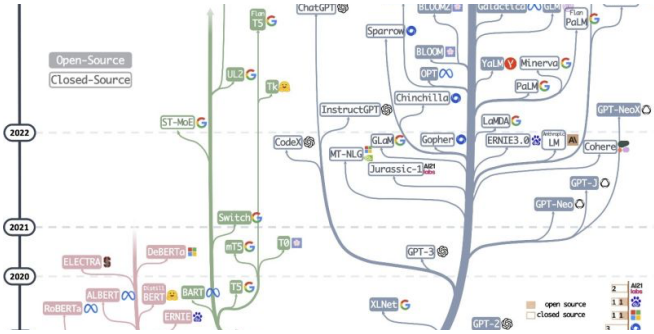
4 min read · Apr 4, 2022

65

See all from Sabrina Herbst

See all from Artificial Intelligence in Plain English

Recommended from Medium



Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372

Ahmet Taşdemir

Fine-Tuning DistilBERT for Emotion Classification

In this post, we will walk through the process of fine-tuning the DistilBERT model for...

8 min read · Jun 14

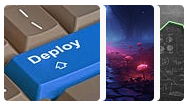
Lists

Natural Language Processing

669 stories · 283 saves

The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves

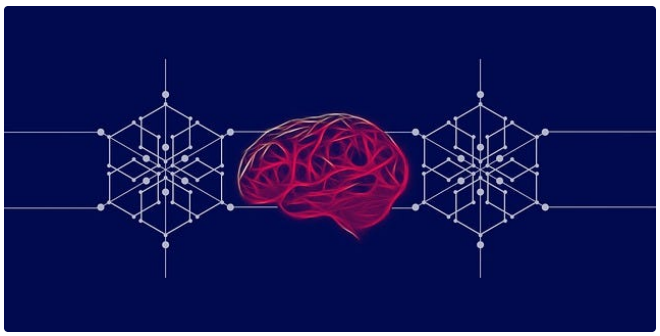


Predictive Modeling w/ Python

20 stories · 452 saves

AI Regulation

6 stories · 138 saves



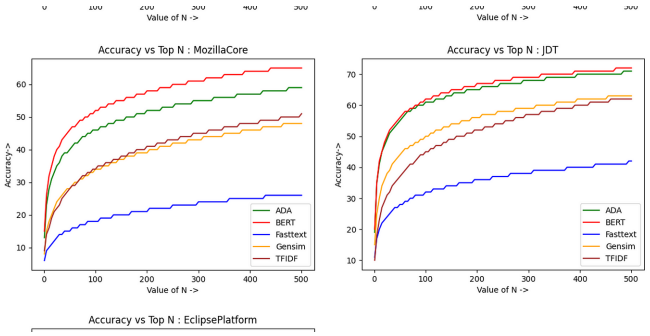
ai geek (wishesh)

Best Practices for Deploying Large Language Models (LLMs) in...

Large Language Models (LLMs) have revolutionized the field of natural language...

10 min read · Jun 26

100 1



Avinash Patil

Embeddings: BERT better than ChatGPT4?

In this study, we compared the effectiveness of semantic textual similarity methods for...

4 min read · Sep 19

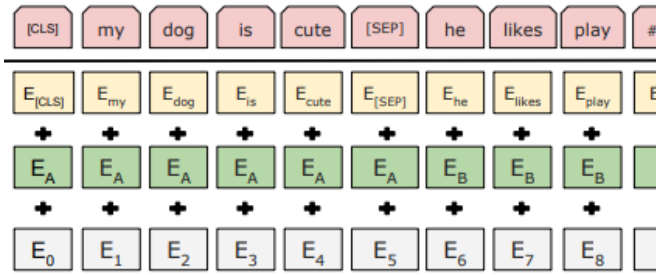
3 1

Nitin Kushwaha in Python in Plain English

A Guide to Build Your Own Large Language Models from Scratch

Introduction

15 min read · Aug 4



Zain ul Abideen

A Comparative Analysis of LLMs like BERT, BART, and T5

Exploring Language Models

6 min read · Jun 26

See more recommendations