Natural Language Processing using Transformers

Kristoffer Brix

June 18, 2025

Natural Language Processing using Transformers

Kristoffer Brix

June 18, 2025

Cool article, but everyone already read it, so I will mostly skip it for the presentation and instead show some stuff I made.

– Kristoffer Brix (about two weeks ago)

Cool article, but everyone already read it, so I will mostly skip it for the presentation and instead show some stuff I made.

– Kristoffer Brix (about two weeks ago)

- The article is very hands-on and practical. There are 8 case studies.
- There is very little theory – in two sessions, we will be having more theory on language models, I would like to understand transformer architecture a bit better.
- These language models, in essence, are large and very sophisticated neural networks – I think of them as a function.
- I became inspired when I read the article and made some stuff, which I will be presenting today.
- Hopefully, by walking through my program, we can get a better feel about how data is transformed and what a simple data pipeline could look like.

- BERT stands for **Bidirectional Encoder Representations from Transformers**.
- GPT stands for **Generative Pre-Trained Transformer**.
- Optimus Prime is another a Transformer and leader of the Autobots.

Contents

Contents

1. What can we download?

2. Architecture

3. Applying BERT: Demo

4. Applying GPT: Demo

5. Additional Links

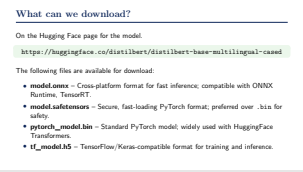
What can we download?

On the Hugging Face page for the model.

```
https://huggingface.co/distilbert/distilbert-base-multilingual-cased
```

The following files are available for download:

- **model.onnx** – Cross-platform format for fast inference; compatible with ONNX Runtime, TensorRT.
- **model.safetensors** – Secure, fast-loading PyTorch format; preferred over `.bin` for safety.
- **pytorch_model.bin** – Standard PyTorch model; widely used with HuggingFace Transformers.
- **tf_model.h5** – TensorFlow/Keras-compatible format for training and inference.



- Would like to go directly to the source and see what is available. Look at the information about parameter size, downloads per month and some general documentation.
- There is also a link the scientific article!
- BERT is a family of models with the same base. The other models vary in parameter size and architecture, they have slightly different heads, inputs and outputs.
- For example, a) the SQuAD model is used for answering a questions using a given context, b) the SST-2 model for (boolean) sentiment analysis, and c) the GPT model is for generating text by predicting a sequence of tokens.
- This is the first trap!

Architecture: Output

Notice that the output are *logits* instead of the *last hidden state* (yellow highlight)!

119547 is the size of the vocabulary (tokens).

512 is the size of the input sequence.

768 is the size of the last hidden layer.

See also `https://netron.app/`.

Architecture: Output
Notice that the output are *logits* instead of the *last hidden state* (yellow highlight)!
119547 is the size of the vocabulary (tokens).
512 is the size of the input sequence.
768 is the size of the last hidden layer.
See also `https://netron.app/`.

- The traps has been laid, let us walk into it! Go to `https://netron.app/`.
- Let us download the model and inspect it.
- `logits` are used to obtain probabilities... but we were supposed to obtain some kind of embedding of the text instead!!
- We need to remove this small neural network (this should be understood as a type of classifier) and we will later replace it something else.
- The website allows us to inspect various weights in the neural network. When we do Masked Language Modeling, it simply updates the weights to better fit our data using some loss function (cross-entropy loss function).
- A simple Python script can convert our model for us, let us look at the script.

Architecture: Input

The BERT model requires two input:

- An `InputIds` object – an array of integers corresponding to token identifiers in the vocabulary file.
- An `AttentionMask` object – an array of boolean values indicating if a token is real or padded (0 = false (padding), 1 = true (real)).

Just download a tokenizer (lol) – a sophisticated algorithm for transforming words into tokens, e.g. [“tokenization”] → [“token”, “##ization”] (there are many tokenizers and they vary in quality).

Architecture: Input

The BERT model requires two input:
• An InputIds object – an array of integers corresponding to token identifiers in the vocabulary file.
• An AttentionMask object – an array of boolean values indicating if a token is real or padded (0 = false (padding), 1 = true (real)).

Just download a tokenizer (lol) – a sophisticated algorithm for transforming words into tokens, e.g. [“tokenization”] → [“token”, “##ization”] (there are many tokenizers and they vary in quality).

- Let us take a closer look at the input now that we are satisfied with the output – read off the first line and inspect the vocabulary file, this can also be downloaded from HuggingFace.
- Search for the token "hello" and the special token "[MASK]".
- Read off the second line. This is very importing for padding later, search for the special token "[PAD]".
- Tokenizer are pretty sophisticated algorithms, but are quite common and can be downloaded from anywhere – watch out for bad quality, look at "BertTokenizers" on GitHub.

Applying BERT (encoder transformer model): Demo

- Load in data from the parquet file (free and open-source column-oriented data from Apache, efficient compression and encoding).
- Load in a tokenizer and vocabulary file.
- Encode text data using the tokenizer to produce an `InputIds` and `AttentionMask` object (inputs for BERT).
- *Apply* the BERT model (think of it as a function, in essence, BERT is "just" a (highly) sophisticated neural network) (it is similar to what we do in `R` with with the `stats::predict` function on `newdata`).
- Extract the embedding, e.g. the first embedding or the mean embedding.
- Pipe the embedding into a new model.

Applying BERT (encoder transformer model): Demo
- Load in data from the parquet file (free and open-source column-oriented data from Apache, efficient compression and encoding).
- Load in a tokenizer and vocabulary file.
- Encode text data using the tokenizer to produce an InputIds and AttentionMask object (inputs for BERT).
- Apply the BERT model (think of it as a function, in essence, BERT is "just" a (highly) sophisticated neural network) (it is similar to what we do in R with with the stats::predict function on newdata).
- Extract the embedding, e.g. the first embedding or the mean embedding.
- Pipe the embedding into a new model.

- Time for the first demo, read up the steps and then go into the C# code, then look at the two `R` scripts.

Applying GPT (decoder transformer model): Demo

- Load in data from the parquet file (like before).
- Load in "large language model" (e.g. LM Studio, Ollama, the cloud, etc.).
- Build an API to communicate with large language model service (see `https://platform.openai.com/docs/api-reference/chat/create`).
- Create a prompt where the next predicted token will be the class label (hopefully) – this is "classification as generation".

Applying GPT (decoder transformer model): Demo
- Load in data from the parquet file (like before).
- Load in "large language model" (e.g. LM Studio, Ollama, the cloud, etc.).
- Build an API to communicate with large language model service (see https://platform.openai.com/docs/api-reference/chat/create).
- Create a prompt where the next predicted token will be the class label (hopefully) – this is "classification as generation".

- Time for the second demo, read up the steps, go to the two links, execute something in `bash` then go into the C# code. Note that `bash` is not installed on Windows by default, it is a Linux thing. `curl` is installed on Windows, but I can never seem to get it to work, maybe the syntax is wrong.

Additional Links

- ONNX Get Started Documentation: `https://onnxruntime.ai/docs/get-started`.
- ONNX Tutorials: `https://github.com/onnx/tutorials`.
- ModernBert `https://huggingface.co/blog/modernbert`.
- `llama.cpp`: `https://github.com/ggml-org/llama.cpp`.
- Shopping (run e.g. Llama 3.3 70B model at 8-10 t/s): `https://frame.work/dk/en/products/desktop-diy-amd-aimax300/configuration/new`.

Additional Links

- ONNX Get Started Documentation: https://onnxruntime.ai/docs/get-started.
- ONNX Tutorials: https://github.com/onnx/tutorials.
- ModernBert https://huggingface.co/blog/modernbert.
- llama.cpp https://github.com/ggml-org/llama.cpp.
- Shopping (run e.g. Llama 3.3 70B model at 8-10 t/s): https://frame.work/dk/en/products/desktop-diy-amd-aimax300/configuration/new.

- Conceptually, we are using the two types of models very differently, even though they fall into the family of transformers.
- We use BERT to extract covariates and use these covariates for classification (statistical modeling).
- We use GPT for to predict the next token, where the next token happens to be some class – this goes well because the model is VERY big.