BERT_NLP_sentiment_analysis_of_movie_reviews_ipynb

March 4, 2023

1 Sentiment Analysis of IMDB Movie Reviews (Part 3: BERT model)

1.0.1 Important: Pre-trained transformer models like BERT from Hugging Face are designed to handle text in its raw form!

Side note: I was using pytorch BERT transfomer(`from transformers import BertModel`, `from transformers import BertTokenizer`) with data processing (e.g. stemming, strip html…). Yet, the model either doesn't run or take more than 1 hr because of the big dataset. Hence, I follow this tutorial to implement a BERT model.

1.1 BERT model

1.1.1 What's special about BERT?

- Context-free models:generate a single word embedding representation for each word in the vocabulary, such as word2vec or GloVe. For example, the word "bank" would have the same representation in "bank deposit" and in "riverbank"
- Contextual models instead generate a representation of each word that is based on the other words in the sentence, such as BERT.

1.1.2 Understand how BERT works

1. Token embeddings: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. Segment embeddings: A marker indicating Sentence A or Sentence B is added to each token. **This allows the encoder to distinguish between sentences.**
3. Positional embeddings: A positional embedding is added to each token to indicate its position in the sentence.

1. Masked LM (MLM) The idea here is "simple": Randomly mask out 15% of the words in the input — replacing them with a [MASK] token. Loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones.

2. Next Sentence Prediction (NSP) In order to understand relationship between two sentences, BERT training process also uses next sentence prediction, BERT separates sentences with a special [SEP] token. During training the model is fed with two input sentences at a time such that:

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a a random sentence from the full corpus.

Example: predict if the next sentence is random or not

Important note: BERT does not try to predict the next word in the sentence!!

1.1.3 Tokenizer for BERT

BERT uses what is called a WordPiece tokenizer. It works by splitting words either into the full forms (e.g., one word becomes one token) or into word pieces — where one word can be broken into multiple tokens.

Word	Token(s)
surf	['surf']
surfing	['surf', '##ing']
surfboarding	['surf', '##board', '##ing']
surfboard	['surf', '##board']
snowboard	['snow', '##board']
snowboarding	['snow', '##board', '##ing']
snow	['snow']
snowing	['snow', '##ing']

By splitting words into word pieces, we have already identified that the words "surfboard" and "snowboard" share meaning through the wordpiece "##board" We have done this without even encoding our tokens or processing them in any way through BERT.

1.1.4 BERT model choice

BERT model we choose **DistilBERT** vs BERT - DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

BERT-base vs BERT-large: BERT-based - BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters - BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

BERT-based-case vs **BERT-base-uncased**: - We don't differentiate between cased and uncased data (english vs English)

1.1.5 BERT input

Input IDs – The input ids are often the only required parameters to be passed to the model as input. Token indices, numerical representations of tokens building the sequences that will be used as input by the model.

Attention mask – Attention Mask is used to avoid performing attention on padding token indices. Mask value can be either 0 or 1, 1 for tokens that are NOT MASKED, 0 for MASKED tokens.

Token type ids – It is used in use cases like sequence classification or question answering. As these require two different sequences to be encoded in the same input IDs. Special tokens, such as the classifier[CLS] and separator[SEP] tokens are used to separate the sequences.

Note: Padding is a special form of masking where the masked steps are at the start or the end of a sequence. Padding comes from the need to encode sequence data into contiguous batches: in order to make all sequences in a batch fit a given standard length, it is necessary to pad or truncate some sequences

1.1.6 BERT tokens

`CLS`: The [CLS] token, short for "classification," is a special token used in BERT to represent the entire input sequence for classification tasks.

When training a classification model using BERT, the [CLS] token is added to the beginning of the input sequence, and the final hidden state corresponding to this token is used as the input to a classifier. This allows the model to make a prediction for the entire input sequence.

`SEP`: The [SEP] token, short for "separator," is used to separate two different segments of a sentence or document.

In BERT, the [SEP] token is used to separate the two segments when performing tasks like question answering or natural language inference, where the model needs to understand the relationship between two different segments of text.

`MASK`: [MASK] is used during pre-training to randomly mask some of the input tokens, forcing the model to learn to predict the masked tokens based on the surrounding context.

1.1.7 Understanding the parameters

`max_length` is a parameter used to define the maximum length of an input sequence.

`pad_to_max_length` is a Boolean parameter used to indicate whether sequences shorter than the max_length should be padded with a special token, usually [PAD], to make them the same length as the longest sequence in the batch.

`return_tensors` parameter specifies that we want the encoded data to be returned as TensorFlow tensor

`attention_mask`: 1 indicates a value that should be attended to, while 0 indicates a padded value.

Example:

```
sequence_a = "This is a short sequence."      sequence_b = "This is a rather long sequence. It is at least longer than the sequence A."    len(encoded_sequence_a), len(encoded_sequence_b)
```

(8, 19)

```
padded_sequences = tokenizer([sequence_a, sequence_b], padding=True)
padded_sequences["input_ids"]
```

[[101, 1188, 1110, 170, 1603, 4954, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 1188, 1110, 170, 1897, 1263, 4954, 119, 1135, 1110, 1120, 1655, 2039, 1190, 1103, 4954, 138, 119, 102]]

```
padded_sequences["attention_mask"]
```

[[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

Note: We use huggingface model and setup packeages to have a clear view on the tranining process.

Resources - BERT Explained: A Complete Guide with Theory and Tutorial

1.1.8 BERT vs RoBERTa

RoBERTa model shares the same architecture as the BERT model. It is a reimplementation of BERT with some modifications to the key hyperparameters and minor embedding tweaks.

The key differences between RoBERTa and BERT can be summarized as follows:

- RoBERTa is a reimplementation of BERT with some modifications to the key hyperparameters and minor embedding tweaks. It uses a byte-level BPE as a tokenizer (similar to GPT-2) and a different pretraining scheme.
- RoBERTa is trained for longer sequences, too, i.e. the number of iterations is increased from 100K to 300K and then further to 500K.
- RoBERTa uses larger byte-level BPE vocabulary with 50K subword units instead of character-level BPE vocabulary of size 30K used in BERT.
- In the Masked Language Model (MLM) training objective, RoBERTa employs dynamic masking to generate the masking pattern every time a sequence is fed to the model.
- RoBERTa doesn't use token_type_ids, and we don't need to define which token belongs to which segment. Just separate segments with the separation token tokenizer.sep_token (or).
- The next sentence prediction (NSP) objective is removed from the training procedure.
- Larger mini-batches and learning rates are used in RoBERTa's training.

In the future, we can consider using RoBERTa because it is supposed to have better results.

[ ]:
```
import locale
def getpreferredencoding(do_setlocale = True):
    return "UTF-8"
locale.getpreferredencoding = getpreferredencoding
```

[ ]:
```
#make sure we are using GPU to run

import torch
torch.cuda.is_available()
```

[ ]:
```
True
```

[ ]:
```
!pip install datasets
```

[ ]:
```
from datasets import load_dataset
imdb = load_dataset("imdb")
```

```
WARNING:datasets.builder:Found cached dataset imdb (/root/.cache/huggingface/dat
asets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f74830d1100e171db75bbdd
b80b3345c9c0)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
```

[ ]:
```
#take only 1000 training data but all of the testing data
```

```
small_train_dataset = imdb["train"].shuffle(seed=42).select([i for i in␣
 ↪list(range(1000))])
small_test_dataset = imdb["test"]
```

```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /r
oot/.cache/huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded371
3f1f74830d1100e171db75bbddb80b3345c9c0/cache-9c48ce5d173413c7.arrow
```

[ ]:
```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-
packages (4.26.1)
ERROR: Operation cancelled by user
```

[ ]:
```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
def preprocess_function(examples):
   return tokenizer(examples["text"], truncation=True)

tokenized_train = small_train_dataset.map(preprocess_function, batched=True)
tokenized_test = small_test_dataset.map(preprocess_function, batched=True)
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
```

```
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}

loading file vocab.txt from cache at /root/.cache/huggingface/hub/models--
distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/vocab.txt
loading file tokenizer.json from cache at /root/.cache/huggingface/hub/models--
distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/tokenizer.json
loading file added_tokens.json from cache at None
loading file special_tokens_map.json from cache at None
loading file tokenizer_config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/tokenizer_config.json
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}

WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/
huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f74830d1
100e171db75bbddb80b3345c9c0/cache-916b6147aa954289.arrow

Map:   0%|          | 0/25000 [00:00<?, ? examples/s]
```

[ ]:
```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.
 ↪from_pretrained("distilbert-base-uncased", num_labels=2)


import numpy as np
from datasets import load_metric

def compute_metrics(eval_pred):
   load_accuracy = load_metric("accuracy")
   load_f1 = load_metric("f1")

   logits, labels = eval_pred
   predictions = np.argmax(logits, axis=-1)
   accuracy = load_accuracy.compute(predictions=predictions,␣
 ↪references=labels)["accuracy"]
   f1 = load_f1.compute(predictions=predictions, references=labels)["f1"]
   return {"accuracy": accuracy, "f1": f1}
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}
```

```
loading weights file pytorch_model.bin from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/pytorch_model.bin
Some weights of the model checkpoint at distilbert-base-uncased were not used
when initializing DistilBertForSequenceClassification:
['vocab_layer_norm.weight', 'vocab_projector.bias', 'vocab_layer_norm.bias',
'vocab_projector.weight', 'vocab_transform.weight', 'vocab_transform.bias']
- This IS expected if you are initializing DistilBertForSequenceClassification
from the checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing
DistilBertForSequenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification
model from a BertForSequenceClassification model).
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.weight', 'pre_classifier.weight', 'classifier.bias',
'pre_classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:
```python
from transformers import TrainingArguments, Trainer

repo_name = "finetuning-sentiment-model-1000-samples"

training_args = TrainingArguments(
   output_dir=repo_name,
   learning_rate=2e-5,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   num_train_epochs=2,
   weight_decay=0.01,
   save_strategy="epoch",
)

trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_train,
   eval_dataset=tokenized_test,
   tokenizer=tokenizer,
   data_collator=data_collator,
   compute_metrics=compute_metrics,
)
```

```
PyTorch: setting up devices
```

```
The default value for the training argument `--report_to` will change in v5
(from all installed integrations to none). In v5, you will need to use
`--report_to all` to get the same behavior as now. You should start updating
your code and make this info disappear :-).
```

It seems that the defaul loss function is already cross entropy loss: here.

[ ]:
```
trainer.train()
```

```
The following columns in the training set don't have a corresponding argument in
`DistilBertForSequenceClassification.forward` and have been ignored: text. If
text are not expected by `DistilBertForSequenceClassification.forward`,  you can
safely ignore this message.
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
***** Running training *****
  Num examples = 1000
  Num Epochs = 2
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 126
  Number of trainable parameters = 66955010
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.
```

```
<IPython.core.display.HTML object>
```

```
Saving model checkpoint to finetuning-sentiment-model-3000-samples/checkpoint-63
Configuration saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/config.json
Model weights saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/special_tokens_map.json
Saving model checkpoint to finetuning-sentiment-
model-3000-samples/checkpoint-126
Configuration saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/config.json
Model weights saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-
```

```
model-3000-samples/checkpoint-126/special_tokens_map.json


Training completed. Do not forget to share your model on huggingface.co/models
=)
```

[ ]:
```
TrainOutput(global_step=126, training_loss=0.4624747018965464,
metrics={'train_runtime': 105.1295, 'train_samples_per_second': 19.024,
'train_steps_per_second': 1.199, 'total_flos': 263009880425280.0, 'train_loss':
0.4624747018965464, 'epoch': 2.0})
```

[ ]:
```
trainer.evaluate()
```

```
The following columns in the evaluation set don't have a corresponding argument
in `DistilBertForSequenceClassification.forward` and have been ignored: text. If
text are not expected by `DistilBertForSequenceClassification.forward`,  you can
safely ignore this message.
***** Running Evaluation *****
  Num examples = 25000
  Batch size = 16
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
{'eval_loss': 0.308576762676239,
 'eval_accuracy': 0.87956,
 'eval_f1': 0.8788476240292923,
 'eval_runtime': 441.2864,
 'eval_samples_per_second': 56.653,
 'eval_steps_per_second': 3.542,
 'epoch': 2.0}
```

Even though we only run 1000 training data with 2 epochs, when we test on all the testing dataset, we achieve 88% accuracy! We can try to use more datapoints when we have sufficient computational power or customize the BERT model to improve accuracy.

Resources: - Sentiment Analysis of IMDB Movie Reviews - Sentiment Analysis - Cleaning,EDA & BERT(88% Acc) - Text Classification with Movie Reviews - NLP - Data Preprocessing and Cleaning - Getting Started with Sentiment Analysis using Python - BERT Explained: A Complete Guide with Theory and Tutorial