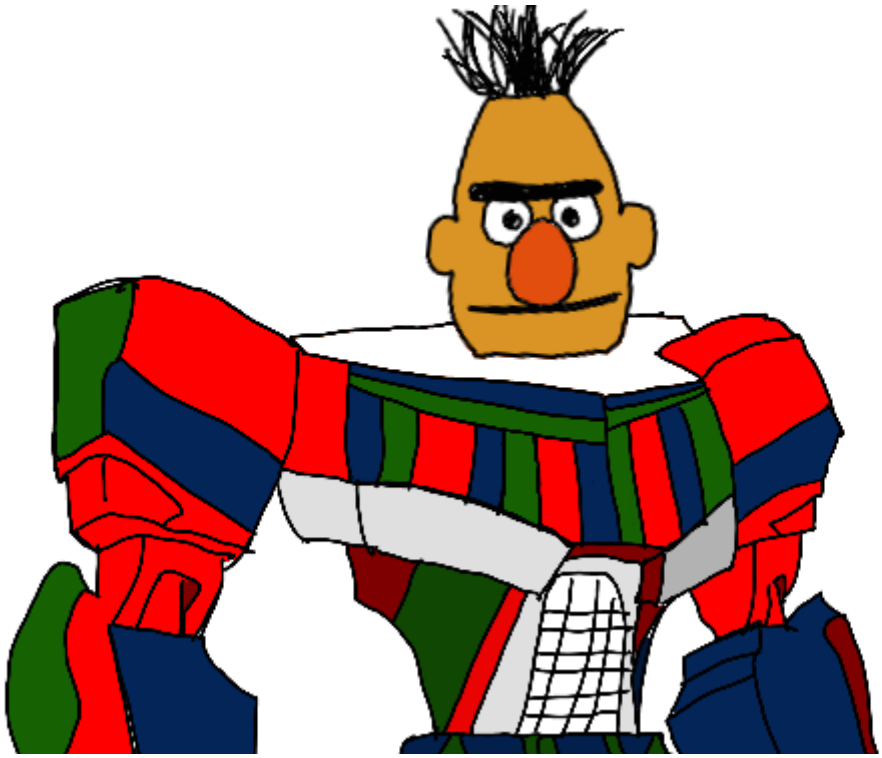
Bert: Step by step by Hugging face

Your guide into Bert model

Abdulelah Alkesaiberi
May 14 · 6 min read



source

In this article, we will know what is BERT and how we can implement it, so let us start.

What is BERT?

Bert stands for Bidirectional Encoder Representations from Transformers. It's google new techniques for **NLP** pre-training language representation. Which means now machine learning communities can use Bert models that have been training already on a large number of words,(some researchers say the Bert model train on the English Wikipedia 2,500 million words) for NLP models to do a wide variety of tasks

such as Question Answering tasks, Named Entity Recognition (**NER**), and Classification like sentiment analysis.

In Bert paper, they present two types of Bert models one is the **Best Base** and the other is **Bert Large.** Both of these models have a large number of encoder layers 12 for the base and 24 for the large. If you understand the concept of transformers. You will see that Bert also trained on the Encoder stacks in the transformers to use the same attention mechanism. But why is it called bidirectional?

What is bidirectional mean?

Because the transformers encoder reads the entire sequence of the words at once which is the opposite of the directional models that read the input sequentially for the left to the right or from the right to the left. The bidirectional method will help the model to learn and understand the meaning and the intention of the word based on its surrounding. Since we will use it for toxic classification, we will explain only the Bert steps for classification tasks only.

What is the input of Bert?

The input of Bert is a special input start with [CLS] token stand for classification. As in the Transformers, Bert will take a sequence of words (vector) as an input that keeps feed up from the first encoder layer up to the last layer in the stack. Each layer in the stack will apply the self-attention method to the sequence after that it will pass to the feed-forward network to deliver the next encoder layer.

What is the output of Bert?

The output of Bert model contains the vector of size (hidden size) and the first position in the output is the [CLS] token. Now, this output can be used as an input to our classifier neural network for classification of the toxicity of the words. In the Bert paper, they achieve a great result by using only a single layer neural network as the classifier.

. . .

Now we understand the concept of Bert, we should dig deep into the implementation phase

Data:

our data is from *Jigsaw Multilingual Toxic Comment Classification* Kaggle competition. In train data, we use just the English language and in the validation and test data we use multiple languages.

files we use:

- jigsaw-toxic-comment-train.csv
- validation.csv
- test.csv

. . .

Here we go to the most interesting part… ***Bert implementation***

1. Import Libraries
2. Run Bert Model on TPU *for Kaggle users*
3. Functions
   3.1 Function for Encoding the comment
   3.2 Function for build Keras model
4. Preprocessing and configuration
   4.1 configuration
   4.2 Import Datasets
   4.3 tokenizer
   4.4 Encode Comments
   4.5 Prepare TensorFlow dataset for modeling
5. Build the model
   5.1 Build the model
   5.2 Training The Model, Tuning Hyper-Parameters
   5.3 Testing The Model
6. Predict and store the result

1.import Libraries

```
import os

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from kaggle_datasets import KaggleDatasets
import transformers
from transformers import TFAutoModel, AutoTokenizer
from tqdm.notebook import tqdm
from tokenizers import Tokenizer, models, pre_tokenizers, decoders,
processors
```

2. Run Bert Model on TPU * for Kaggle users*

```
# Detect hardware, return appropriate distribution strategy
try:
    # TPU detection. No parameters necessary if TPU_NAME environment
variable is
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and
single GPU.
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

3.Functions

3.1 Function for Encoding the comment

Encode job is to convert word into vector encapsulate the meaning of the word, similar word has a closer number.

```
def regular_encode(texts, tokenizer, maxlen=512):
    """
    Function to encode the word
    """
    # encode the word to vector of integer
    enc_di = tokenizer.batch_encode_plus(
        texts,
        return_attention_masks=False,
        return_token_type_ids=False,
        pad_to_max_length=True,
        max_length=maxlen
    )

    return np.array(enc_di['input_ids'])
```

3.2 Function for build Keras model

```
def build_model(transformer, max_len=512):
    """
    This function to build and compile Keras model

    """
    #Input: for define input layer
    #shape is vector with 512-dimensional vectors
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32,
name="input_word_ids") # name is optional
    sequence_output = transformer(input_word_ids)[0]
    # to get the vector
    cls_token = sequence_output[:, 0, :]
    # define output layer
    out = Dense(1, activation='sigmoid')(cls_token)

    # initiate the model with inputs and outputs
    model = Model(inputs=input_word_ids, outputs=out)
    model.compile(Adam(lr=1e-5), loss='binary_crossentropy',metrics=
[tf.keras.metrics.AUC()])

    return model
```

4.Preprocessing

4.1 configuration

```
# input pipeline that delivers data for the next step before the
current step has finished.
# The tf.data API helps to build flexible and efficient input
pipelines.
# This document demonstrates how to use the tf.data
```

```
# API to build highly performant TensorFlow input pipelines.
AUTO = tf.data.experimental.AUTOTUNE

# upload data into google cloud storage
GCS_DS_PATH = KaggleDatasets().get_gcs_path()

# Configuration
EPOCHS = 2
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
MAX_LEN = 192
MODEL = 'bert-base-multilingual-cased'
```

4.2 import dataset

```
train1 = pd.read_csv("/kaggle/input/jigsaw-multilingual-toxic-
comment-classification/jigsaw-toxic-comment-train.csv")

valid = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-
comment-classification/validation.csv')
test = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-
classification/test.csv')
sub = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-
classification/sample_submission.csv')
```

4.3 tokenizer

```
#use the pre-trained model bert as a tokenizer
#bert tokenizer has vocabulary for emoji. this is the reason we
don't need to remove emoji from
#datasets, for more details see the (EDA & data cleaning) notebook

tokenizer = AutoTokenizer.from_pretrained(MODEL)
```

4.4 Encode Comments

```
%%time
#call the function regular encode on for all the 3 dataset to
convert each words after the tokenizer
#into a vector
#x_train,x_test, and x_validation will have the comment text column
only,(in test called "content")

x_train = regular_encode(train1.comment_text.values, tokenizer,
maxlen=MAX_LEN)
x_valid = regular_encode(valid.comment_text.values, tokenizer,
```

```
maxlen=MAX_LEN)
x_test = regular_encode(test.content.values, tokenizer,
maxlen=MAX_LEN)

#y_train,y_valid will have te target column "toxic"
y_train = train1.toxic.values
y_valid = valid.toxic.values
```

4.5 Prepare TensorFlow dataset for modeling

```
# Create a source dataset from your input data.
# Apply dataset transformations to preprocess the data.
# Iterate over the dataset and process the elements.

train_dataset = (
    tf.data.Dataset # create dataset
    .from_tensor_slices((x_train, y_train)) # Once you have a
dataset, you can apply transformations
    .repeat()
    .shuffle(2048)
    .batch(BATCH_SIZE)# Combines consecutive elements of this
dataset into batches.
    .prefetch(AUTO) #This allows later elements to be prepared while
the current element is being processed.
)

valid_dataset = (
    tf.data.Dataset # create dataset
    .from_tensor_slices((x_valid, y_valid)) # Once you have a
dataset, you can apply transformations
    .batch(BATCH_SIZE) #Combines consecutive elements of this
dataset into batches.
    .cache()
    .prefetch(AUTO)#This allows later elements to be prepared while
the current element is being processed.
)

test_dataset = (
    tf.data.Dataset# create dataset
    .from_tensor_slices(x_test) # Once you have a dataset, you can
apply transformations
    .batch(BATCH_SIZE)
)
```

5.Build the model

5.1 Build the model

```
%%time
# in the TPU
with strategy.scope():
    #take the encoder results of bert from transformers and use it
as an input in the NN model
    transformer_layer = TFAutoModel.from_pretrained(MODEL)
    model = build_model(transformer_layer, max_len=MAX_LEN)
model.summary()
```

5.2 Training The Model, Tuning Hyper-Parameters

```
#train the model
# training the data and tune our model with the results of the
metrics we get from the validation dataset
n_steps = x_train.shape[0] // BATCH_SIZE
train_history = model.fit(train_dataset, steps_per_epoch=n_steps,
validation_data=valid_dataset,
                epochs=EPOCHS)
```

5.3 Testing The Model

```
#test the model on validation
n_steps = x_valid.shape[0] // BATCH_SIZE
train_history_2 = model.fit(valid_dataset.repeat(),
steps_per_epoch=n_steps,epochs=EPOCHS*2)
```

6. Predict and store the result

```
sub['toxic'] = model.predict(test_dataset, verbose=1)
sub.to_csv('submission.csv', index=False)
```

. . .

Conclusion

Bert is a powerful pre-trained model makes a huge effect on NLP world today. You can use Bert in many different tasks like language translation, question and answer, and predict the next word in addition to text classification.

Acknowledgment

I must say Thanks for my teammates (Sarah and Norah), and for my instructors in DSI7:(Irfan, Husain, Yazied, and Amjad) for helping us to finish this journey :)

Useful references

- https://www.kaggle.com/xhlulu/jigsaw-tpu-distilbert-with-huggingface-and-keras
- http://jalammar.github.io/illustrated-bert/
- http://jalammar.github.io/illustrated-transformer/
- https://arxiv.org/pdf/1810.04805.pdf
- https://www.youtube.com/watch?v=xI0HHN5XKDo
- https://www.youtube.com/watch?v=KN3ZL65Dze0&feature=emb_title
- https://www.youtube.com/watch?v=zMxvS7hD-Ug&feature=youtu.be

Contact detail

- Linkedin
- Twitter

Data Science | Bert | NLP | Deep Learning | Machine Learning