Exploration of BERT embedddings in Apache Spark

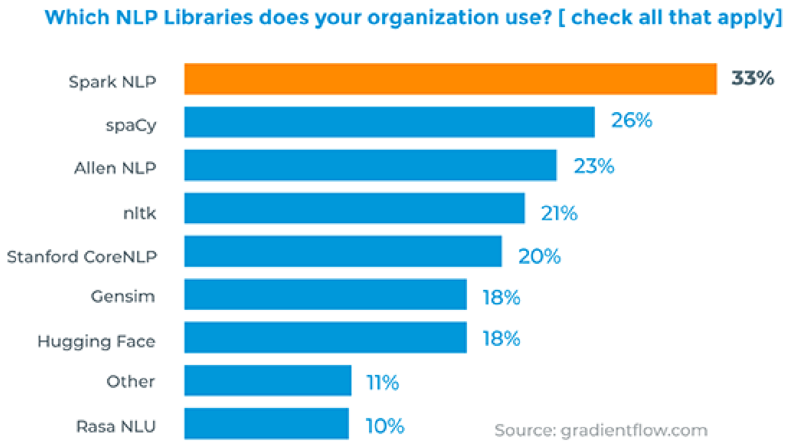
Rahul Sharma | UIUC CS 410 (Text Information Systems)

Abstract

This tech review focuses on BERT embeddings in Apache Spark. It covers the high-level BERT architecture, key terminologies, SparkNLP comparison with other solutions and BERT SentenceEmbeddings code snippets in Python. The BERT stands for Bidrectional Encoder Representation from Transformers and it's deep bidrectional in nature. It's a unsupervised learning technique for pre-training NLP.

BERT architecture is referenced from the many existing NLP models, i.e. GPT, Elmo. BERT achieves the state-of-art nlp results and performance on NLP tasks i.e. Questional Answering, Langauage inferences, NER, Next sentence prediction and analyze sentences holistically.

BERT was published by the Google in 2018 and since then, there're multiple implementations have been released in open-source community. One of early adoptor is SparkNLP and emperically the Spark version gained popularity as it allows to train and perform NLP tasks at the large scale using distributed processing.

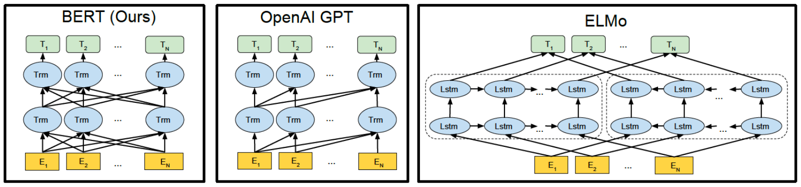


Which NLP Libraries does your organization use? [check all that apply]

Source: gradientflow.com

BERT Architecture

The standard NLP Language Models are unidirectional and extracts context features from left to right (LTR) or Right to Left (RTL) where token can only attend to LTR or RTL tokens in self-attention layers of transformer. For an example- the word `bank` would have the same context-free representation in `bank account` and `bank of the river`. The unidirectional model represent `Bank` based on the previous word `Accessed` in `I accessed the bank account` sentence but BERT resprent it using the both sides- `Accessed & account`. This picture depicts the architectural comparison with previous models:



- BERT is deeply bidirectional and overcome with the limitations of unidirectional LMs using Mask Language Model (MLM).
- The existing LM ELMo follow feature based approach whereas BERT is fine-tuning based approach like GPT
- BERT uses a bidirectional Transformer.
- BERT representations are jointly conditioned on Left and Right contexts in all the layers.

Left to right context:

I

I **accessed**

I accessed **the**

I accessed the **bank**

I accessed the bank **account**

Right to Left context:

account

bank account

accessed bank account

....

BERT joint on Left and Right contexts:

I accessed the bank account

I **accessed** the bank account

I accessed **the** bank account

I accessed the **bank** account

In BERT's Input/Output representation, the first token of every sequence is always a special classification token ([CLS]). Sentence pairs are packed together into a single sequence. It differentiate the sentences in two ways. First, separate them with a special token ([SEP]). Second, It add a learned embedding to every token indicating whether it belongs to sentence A or sentence B.

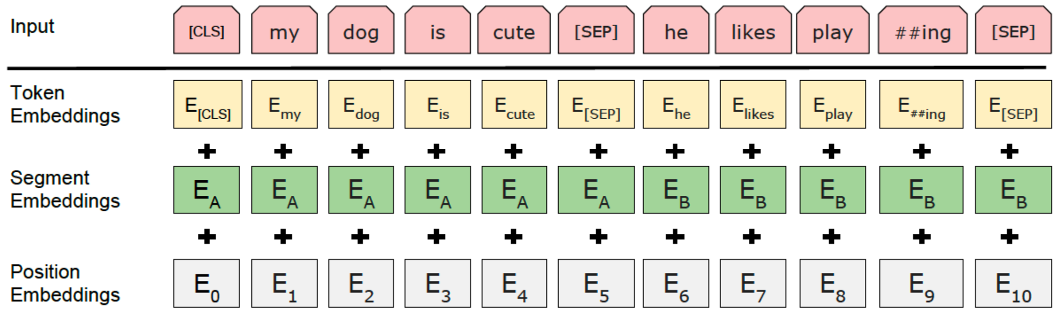


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

BERT- SparkNLP by Johnsnowlabs

SparkNLP provides BertSentenceEmbeddings annotator with 42 available pre-trained models for sentence embeddings used in SentimentDL, ClassifierDL, and MultiClassifierDL annotatorshas implemented NLP models trained several embeddings/pipelines.

Why this implementation really matters:

- Polyglot: Python, Scala, Java
- Natively Scalable: Backend on Apache Spark project
- Pretrainined NLP pipelines: It covers wide range of NLP pipelines and embedding

List of pre-trained models in Spark:

BERT LaBSE, BioBERT Clinical, BioBERT Discharge, ,BioBERT PMC,BioBERT Pubmed,BioBERT Pubmed Large,BioBERT Pubmed PMC,BioBERT Clinical,BioBERT Discharge,BioBERT PMC,BioBERT Pubmed,BioBERT Pubmed Large,BioBERT Pubmed PMC

Usecase exploration:

- Named Entity recognition (NER): I am using SparkNLP BERT word embeddings via Pyspark. As a first step we need to declare the pipeline which consist a uniform set of operations built on top of Spark Dataset API.

  `assembler->sentence->token->bert->loaded_ner_model->converter`

Example:

```
import os
# Install java
! apt-get update -qq
! apt-get install -y openjdk-8-jdk-headless -qq > /dev/null

os.environ["JAVA_HOME"] = "/Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home"
os.environ["PATH"] = os.environ["JAVA_HOME"] + "/bin:" + os.environ["PATH"]
! java -version

# Install pyspark
! pip install --ignore-installed pyspark==2.4.4

# Install Spark NLP
! pip install --ignore-installed spark-nlp

from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.sql.functions import *
import sparknlp
from sparknlp.annotator import *
from sparknlp.common import *
from sparknlp.base import *
```

```
    spark=sparknlp.start()

    text = [["Peter Parker is a nice guy and lives in New York."],["Learning Text Information System"]]
    spark_df = spark.createDataFrame(text).toDF("text")
    doc_assembled_df=assembler.transform(spark_df)
    #doc_assembled_df.head(1)
    train_df = spark.read.csv("train.csv")

    assembler = DocumentAssembler().setInputCol("text").setOutputCol("document")

    sentence = SentenceDetector().setInputCols(['document']).setOutputCol('sentence')
    doc_sentenced_df=sentence.transform(doc_assembled_df)
    #doc_sentenced_df.head(1)

    token = Tokenizer().setInputCols(['sentence']).setOutputCol('token')
    converter = NerConverter().setInputCols(["document", "token", "ner"]).setOutputCol("ner_span")

    bert = BertEmbeddings.pretrained('bert_base_cased', 'en').setInputCols(["sentence",'token']).setOutputCol("bert").

    bert_base_cased download started this may take some time.
    Approximate size to download 389.1 MB
    [OK!]

    loaded_ner_model = NerDLModel.pretrained('ner_dl_bert').setInputCols(["sentence", "token", "bert"]).setOutputCol("

    ner_dl_bert download started this may take some time.
    Approximate size to download 15.4 MB
    [OK!]

    ner_prediction_pipeline = Pipeline(
    stages = [document
    ,sentence
    ,token
    ,bert
    ,loaded_ner_model
    ,converter])

    model=ner_prediction_pipeline.fit(train_df)
    predicted= model.transform(spark_df)
    predicted.show()
```

- Word Embedding:

  Pipeline- `df->sentence->token->bert`

```
word_embeddings = BertEmbeddings.pretrained('bert_base_cased', 'en').setInputCols(["sentence", "token"]).setOutputCol(


    bert_base_cased download started this may take some time.
    Approximate size to download 389.1 MB
    [OK!]


pipeline = Pipeline().setStages(
  [
    assembler,
    sentence,
    token,
    word_embeddings
  ]
)

result=pipeline.fit(spark_df).transform(spark_df)
result.select('embeddings').show()
```

References

- https://arxiv.org/pdf/1810.04805.pdf
- http://jalammar.github.io/illustrated-bert/
- https://www.quora.com/What-is-a-masked-language-model-and-how-is-it-related-to-BERT
- https://nlp.stanford.edu/seminar/details/jdevlin.pdf
- https://arxiv.org/abs/1907.11692
- https://johnsnowlabs.github.io/spark-nlp-workshop/databricks/index.html
- https://colab.research.google.com/github/JohnSnowLabs/spark-nlp-workshop/blob/master/tutorials/blogposts/3.NER_with_BERT.ipynb?authuser=1#scrollTo=8IR30lxz_foj