BERT for Geo Mapping of Entities, Team 3

Sai Charan Akkena

Project Progress Report - III

SECTION 1

Abstract

This project aims to build a model that enables researchers to make quantitative arguments about environmental awareness and cultural change. The dataset has been taken from reddit which has the data from local news of the region for which we are analyzing the environmental changes and also from scholarly articles. We will be using an open-sourced neural network-based technique for Natural Language Processing called BERT (Bidirectional Encoder Representations from Transformers). BERT has been pre-trained on the whole English Wikipedia and can be further fine-tuned on smaller task-specific datasets.

1. Data

1.1. Data Source

This dataset was collected from reddit, searching for articles that contain the keyword 'humanities' . The data is in the format of compressed tar.gz file which has 118,059 files inside it in the format of .txt and has an actual size of 322MB .To ensure that the program doesn't run out of memory, we will train our dataset in multiple batches.

1.2. Data Format

This data is comprised of .txt files which are in natural text language and processed with the Illinois Wikifier. The datasets are all in English except one which is in Norwegian. The data in the text files is comprised of various types of words and sentences, sentences with open and close parenthesis in the beginning and at the end, words in both wikified and non-wikified formats etc.

1.3. Programming Language, packages

Python will be used for fine-tuning and testing the BERT model on Google Colaboratory which allows us to write and execute arbitrary python code through the browser. Google Colab is a Jupyter Notebook environment that doesn't require any setup and runs entirely

on cloud. Since the dataset is huge, we'll be using our local machine's GPU and if required we'll switch to Google Cloud TPU in the future. We will be using many machine learning libraries like PyTorch, Pandas, TensorFlow, Transformers by hugging face etc.

2. Experiments

2.1. Preliminary Results

Since our data is in .txt files which contains different paragraphs with various sentences and has already been pre-processed, we combined all the .txt files into a .csv format and modified it into a .tsv file which is understandable by BERT. We performed tokenization using the Bert Tokenizer using Google Colab on our local GPU. The data has been loaded into a pre-trained BERT Model and a fine-tuned model based on our Vocabulary has been built. The synonymy test has been applied by me to the BERT by finding out the word embeddings and comparing the vector values to find out the cosine similarity between two words.

SECTION 2

1 Data statistics

Since our dataset has extracted from reddit searching for the word 'humanities', and it has already been pre-processed and wikified using the Illinois Wikifier, the wiki words are in the form of 'wiki__XXX__<wikifier_end_parts>'. These words have been separated from the dataset by removing the Wikifier Information at the beginning and also at the end and the frequency of the words has been found out. The words 'arts', 'of', 'liberal','humanities','university','social' etc., are the most frequently occurring words in the dataset whereas the words 'damn', 'reichsbank' etc., have the least occurrence in the dataset.

2. Experiments

2.1 Selected Approaches

We shall be using the Hugging Face's transformer for implementing our BERT model. It is a PyTorch implementation of Google's BERT implementation. We decided to use it due to several advantages it offers us the best results we aim for. It has the same accuracy rate as Google's BERT implementation. There are many pretrained models shared by many people which will allow us to improve our accuracy and precision while not needing us to pre-train a newmodel. It has many functions already implemented which will make our task easy.

Our approach is to infer data from the wiki words and find which words can really have an impact on training the model. So we will be extracting the wiki words, cleaning those words (i.e., removing Wikifier information at the beginning and at the end), finding the frequency of each unique words and making 2 lists from those cleaned words (List 1 – single words after removing Wikifier Information and List 2 – everything else after removing Wikifier Information), comparing the lists with the existing 'BERT vocabulary' to find the matching words from our lists, analyzing the words (i.e., identifying names of cities, universities, geographic locations, etc.,)

2.2 Implementation

This approach helps us in training BERT while finding the frequency of the words after removing Wikifier Information. It will help us decide which words are important and which are not important to the BERT Vocabulary. For example, let us consider two different sets of words with frequency greater than 100 and the other with frequency less than 5. Of these two sets, we can infer that the words which are occurring more times will be more useful to the BERT than the words which occur less time, these words which occur less time shouldn't be omitted and also be added. As our project focuses on environmental awareness using text mining, it is necessary for the BERT to analyze the text in the best way, for which data with a higher number of contexts will help.

We used hashlib package in the process of removing duplicate words after removing Wikifier Information. We used nltk package to find the frequency of the wiki words in the file. After comparing both the lists with the BERT vocabulary, analyzing the matched words from lists we used spacy package's NER (Named Entity Recognition) to identify the category of the words.

For tokenization, since we are using Hugging Face's transformers library, we will be using the provided Bert Tokenizer. It is based upon Google's Word Piece tokenizer. It already has traditional tokenization features such as lower casing which allows us to skip the pre-tokenization of the dataset. We will assign the maximum sequence length of 12 since this was the longest word length in our data. We then convert tokens to the index numbers corresponding in the BERT vocabulary.

2.3 Approach

We have several tasks to get the data which will help BERT in-text analyzing efficiently. Frist, we counted the total number of words in the entire reddit dataset and extracted all the wiki words from the dataset and placed them in a separate text file (wiki_word_dataset.txt). Then we have counted the total number of wiki words from the dataset. After this we have cleaned the wiki words, i.e. removing Wikifier Information at the beginning and end of each wiki word. After removing the Wikifier Information we will be having words that can be helpful in training the model. But we can't train BERT with all the words we got, as this might end up underfitting the model. To overcome this we found the frequency of every word after removing Wikifier Information. From the output of this task, we could see that occurrence of words is in a range from 1 to 50,000+. Among these words, words that occur more than 50 times in this dataset will have a different type of contexts that will eventually help BERT in analyzing more about that word, whereas words with occurrence less than 30 to 50 times will considerably have different contexts, this can provide more information about the word and its usage, but the words which occurred less than 20 times in this huge dataset is not a big issue, as training BERT on these words will narrow the analyzing of BERT which results in less efficiency (i.e., the error rate may be higher because of these words). After this step, we have

removed the duplicates from the cleaned words file. We then matched these unique words file with the existing BERT vocabulary (which has word count more than 30,000) and wrote them in a new separate file, so that we can know the words that need to be added into the BERT vocabulary. We then made two lists from the unique words file, List 1 will contain all the single word after removing Wikifier Information and List 2 will contain all the other set of words. We then compared these two lists with BERT vocabulary and generated two separate files. We then analyzed these two files using the spacy package, from this analysis we figured out the category each word in both of the files would fall into, this package identified the word's type (for example names of people, nationalities, cities, facilities, etc.,)

The table 1.1 shows the Frequency of the Wiki_Words

	Count
Total Wiki_Words	1,321,967
Wik_Words(After removing duplicates)	78,598
Matching Single_Words	3718
Matching Multiple_Words	14039

Table 1.2 shows the Wiki_Words which highly are occurring

Top 5 Wiki Words	Frequency
Liberal Arts	43,117
Humanities	42,139
Academic Term	19,185
Whitelist	15,910
Professor	11,946

Section 3

3. Experiments:

3.1 Approach

Previously, we had identified the wiki words, that are matching in the BERT Vocabulary after removing the Wikifier Information from the beginning and also at the end of the word, now for each document/wiki word pair we need to get the topic assignment. There are a couple of approaches for this:

1) One is for each occurrence of a wikiword in a document, we match the word with BERT vocabulary and if it is present we get the BERT embedding for the wikiword. We then use the dimension of the Word Embedding which has the highest absolute value as the topic assignment of that particular document.

2) Another approach is to get the Word embedding for the wikiword's matching in BERT Vocabulary and cluster them together and use that particular cluster ID as the topic assignment for that document/wikiword pair.

For easy classification, we found out the word embeddings for the single Wiki_Words only. After extracting the embeddings we started the process of finding the cosine similarity between the word embeddings to see how similar or different the words are from each other.

Since there are many words which are present in BERT Vocabulary and also many words which are not present in BERT Vocabulary we took an approach where we take the wikiword's that are matching with the Vocabulary and get their embeddings by using an encoding service called "Bert as a service " and then find the cosine similarity between them. As required we then calculate the maximum value of particular wikiword embedding and get the index of that value and print it along with the name of the document, wikiword, topic assignment(index of the maximum value) in a text file.

Bert-as-Service is an online BERT encoder python service which makes it easier and less time consuming to find word embeddings. It is a stream-as-you-go service which allows us to extract embeddings in minimal time, using it we can compute similarity or relatedness between query and document, also cluster the words with the same topics together.

3.2 Implementation

As discussed above we first extracted the word embeddings using Bert-as-Service it basically creates a server which we can access using python code in our notebook/terminal, each time we send a word/sentence as a list, it will return the embeddings for the words/sentences. Each sentence/word is translated to a 768-dimensional vector. The Dimension of the embedding depends on the pretrained BERT Model we are using, here we are using "BERT-Base Uncased" which has 12-layers, 768-hidden units, 12-heads, 110M parameters , the dimension of the vector also depends on `pooling_strategy` and `pooling_layer` which can be set.

By default Bert as a service works on the second last layer, i.e. `pooling_layer = -2` . we can change it by setting `pooling_layer` to other negative values, e.g. -1 corresponds to the last layer and -3 corresponds to the 10^{th} layer. A pretrained BERT model has 12/24 layers, each "self-attends" on the previous one and outputs a [batch_size(B), seq_length(T), num_hidden(D)] tensor. Here our goal is to get the word embedding, but if the goal is getting a sentence embedding, we need to pool the [B,T,D] tensor into a [B,D] embedding matrix. There are different pooling strategies such as average pooling, max pooling, hierarchical pooling, even concatenating avg-max pooling can be applied here directly as they do not introduce new parameters (so no extra training).

In BERT two special tokens [CLS] and [SEP] are padded to the beginning and the end of an input sequence, respectively. Once fine-tuned with specific tasks, the embedding of those two tokens can represent the whole sequence, thus, we can include them in as well. However, if the BERT model is only pretrained and not fine-tuned on any specific task, embeddings on those two symbols are meaningless.

BERT is a model pretrained with two targets: masked language model and next sentence prediction. The last layer is trained in the way to fit this, making it too "biased" to those two targets. Here we can take the second-to-last layer.

There are 2 parts of Bert-as-service - BertServer and BertClient, first we start BertServer which will perform all the embedding functions. Starting Server is a Mandatory step. Next is BertClient(),BertClient() is a client object connected to BertServer, it sends the information to BertServer() that a particular 'word' needs to be encoded. BertServer() sends back the encoding to BertClient() which is then displayed to us.

Next using scikit learn's cosine similarity library we found out the cosine similarity between pairs of highest occurring single words in Bert.

Cosine Similarity is calculated as dot product of X and Y which in our case are 2 tokens:

$K(X,Y) = X.Y/||X||x||Y||$

We started working on our other approach which was grouping on the basis of index of the maximum value of the embedding vector. We found out the max value using .max() and found the index of the maxvalue using .argmax().

3.3 Analysis and Results

The reason we decided to use Bert-as-service is very simple. We looked into many ways to get embeddings. Usually, extracting word embeddings is a tedious task. First we need to fetch models, fetch the hidden states and then get the embeddings from the hidden states. While fetching the model is easy, the main difficulty lies in the step of hidden states. Bert uncased model has four dimensions - layer number(12 layers), batch number(1 for us), word/token(1 for us), hidden unit/feature(768 layers). Reducing all this to the required form of [batch, token, features] takes a considerable amount of time. Therefore, we use Bert-as-Service here to skip all this and directly get the embeddings required.

Different BERT layers capture different information It is said that consecutive layers have similar representation, whereas the first few layers represent considerably different meaning compared to the last few layers. The deepest layer is the one next to the word embedding. It may preserve the very original word information (with no self-attention etc.). On the other hand, we may achieve the very same performance by simply using word embedding. Therefore, using anything in-between the first layer and the last layer is a balanced approach.

While finding the cosine similarity we got some interesting results. Firstly, Cosine Similarity values weren't extreme. As in, we didn't get any value less than 0 or greater than or equal to 1 which can be attributed to the fact that Google's Bert model has been trained on the Whole of Wikipedia data which means that there will be somehow even a little bit of similarity between words.

Our finding was the impact of context on these embeddings. For example, let's take two highest occurring single words which were in Bert - 'humanities' and 'professor'. When finding the cosine similarity of these words, we get a value of 0.6036, but when inserted along with some contextual text , let's say "the professor teaches humanities and he studies humanities " we get a cosine similarity of 0.4756 which is different.

We ran similar tests on other words and found similar findings. It says that the "Word Embeddings" depend on the context of the sentence. We also ran a test including the text from the dataset which contained the Wiki_Words which also showed the same results. Therefore, we feel that grouping on the basis of cosine similarity of the Wiki_Words might not be the best approach.

Moving to the other approach which is group on the basis of the max value of the embedding vector. We got some results which would allow us grouping.

We will take example of 'humanities' and 'court' we found out that both have same index of 105 The output format will look like this:

172244_172244_universitywire_bodypluralhumanitiesorhleadpluralhumanities_1998-01-01_1998-12-31_0_17_0.txt,humanities, 105

For our next steps, we are looking into the approach of the clustering the vectors of the Single Wiki_Words matching with the Bert vocabulary other than grouping them under the same index value.

Tasks	Task Description	Student 1 name	Student 2 name	Student 3 name	Student 4 name
Task 1	Extracting the Wiki Words	Suriya Prakaash	**Sai Charan**		
Task 2	Cleaning the extracted word		**Sai Charan**		Sai Vishal
Task 3	Finding frequency	Suriya Prakaash		Siddhant Jain	

Task 4	Removal of duplicates and Comparison with Bert Vocab			Siddhant Jain	Sai Vishal
Task 5	Separating Single and Multiple Words	Suriya Prakaash			Sai Vishal
Task 6	Named Entity Recognition using Spacy		**Sai Charan**	Siddhant Jain	
Task 7	Word Embedding Extraction			Siddhant Jain	Sai Vishal
Task 8	Cosine Similarity	Suriya Prakaash	**Sai Charan**		
Task 9	Retrieving index of highest vector and grouping by index	Suriya Prakaash	**Sai Charan**		
Task 10	Analysis of embeddings			Siddhant Jain	Sai Vishal

I performed the following tasks in the project so far:

1) Combining all the text files into a format understandable by BERT and then loading the dataset set to a pre-trained BERT Model(uncased) and fine-tune the model based on our Vocabulary.

2) Perform Synonymy test on the word embeddings from the dataset and the BERT Vocabulary

3) Extracting the Wiki words, cleaning the wiki words using python by removing the Wikifier Information from the beginning and at the end and performing the text analysis on the words which are common between the reddit data set and the BERT Vocabulary.

4) Finding the cosine similarity between the vectors and performing analysis on the similarity of different words(Eg: words of same category like Monday, Tuesday etc.)

5) Retrieving the index of highest vector and grouping by index and printing the output file in the required format.